MULTISCALE MODELING OF DIFFUSION PROCESSES IN DENDRITES AND DEDRITIC SPINES

by

Fredrik Eksaa Pettersen

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2014

Contents

List of Figures

List of Abbreviations

BE Backward Euler

CNS Central Nervous System

DSMC Direct simulation Monte Carlo

FDM Finite Difference Methods

FE Forward Euler

FLOPs Floating Point Operations Per second

LTD Long-term depression

LTP Long-term potentiation

MC Monte Carlo

PDE Parital Differential Equation

PKCγ protein kinase Cγ

PSD Post Synaptic Density

RNG (pseudo) random number generator

Chapter 1

Introduction

This chapter describes both the scope of this project and the various places where a lot of time was spent.

1.1 The project

In the fall of 2014 a new interdisciplinary research project on brain plasticity will start at the University of Oslo and will include people from biology, neuroscience, statistical mechanics and applied mathematics among others. There are many processes in the brain which are difficult to describe in any methodology, and in that manner behave as meso-scale processes. Processes which are on a length scale between continuum and statistical mechanics. Today, there are some alternatives which might be plausible alternatives for modeling such processes, namely molecular dynamics and various offspring of this like direct simulation Mote Carlo. Both of which possess a few problems.

Molecular dynamics seek to model systems of atoms or molecules by Newtonian mechanics in the sense that each particle is affected by some well known potential which in turn lets us calculate the forces on the particle, and integrate the position. Though these calculations are a viable alternative in many applications, like nanoporous fluid flow, they suffer from the weakness of immense computational cost and can only effectively model something like a few cubic micrometers. More importantly, they are limited to small molecules like SiO_2 or water, and to my knowledge no potential is known which might describe macromolecules like proteins or enzymes.

Direct simulation Monte Carlo is a more plausible candidate than normal molecular dynamics. (To my knowledge) it replaces the computationally costly potential calculations with Monte Carlo simulation by drawing some number of random particle-pairs which collide and give each particle a new velocity by some rule (like conservation of temperature). This is done in combination with calculating collisions with any walls which might be present in the system. Although DSMC is a sort of scaled up version of MD it too suffers from some problems which might make it unsuitable for modeling processes in the brain. DSMC is mainly tailored to recreate the properties of Argon gas. This does not have to be as big a weakness as it sounds; it only means that charge neutrality is assumed.

This thesis enters as the first attempt at trying to combine a well known continuum model of diffusion processes with stochastic model of the same diffusion process. Although this might seem a bit unnecessary at first there are quite a few diffusion processes in the brain where this might be the only reasonable model. For example, the extracellular space in gray-matter is so narrow in some places that it is problematic to assume continuity. Transport

processes in and out of dentritic spines are also assumed to be diffusion governed, but only consist of some ten or less particles (this converts to $\frac{\text{nMol}}{\text{L}}$ concentrations).

1.2 Progress of the project

The combination of stochastic and continuum models immediately raise the question of where the limit between the two scales is. As was done by Plapp and Krama [10] one way to postpone answering this question is by introducing some conversion parameter, which will define how many stochastic units one unit of concentration is equivalent to. In addition to this, the initial model was built with testing in mind, which makes the exact limit rather uninteresting.

The stochastic model chosen in this project is random walks because of its mathematical equivalence to the diffusion equation (see sections 2.3.1 and 2.3.2), its conceptual simplicity and because of how easy it is to add complexity to the model. Seeing as there is a mathematical equivalence between the two models an initial goal was to force the error-term from the random walk part of the solver to smaller than the error term from the numerical solution of the partial differential equation. A lot of time went into making this work by experimenting with the required conversion ratio, the required number of time-steps in the random walk model per time-step on the PDE solver, various curve-fitting possibilities and a lot of (needless) debugging. At one point it seemed that everything behaved as expected, but this turned out to be a bug which meant doing even more testing an needless debugging. Finally, an explanation to the difficulty of this seemingly simple problem was found (see section **??**) and no more attempts were done.

After forcing the error terms to behave properly was abandoned, an attempt at combining the diffusion equation solver with DSMC code developed by Anders Hafreager during his masters thesis was made. Although this combination to some extent works, it suffers from a difference in dimensionality. The DSMC code runs in three spatial dimensions, whereas the code developed in this thesis at the moment is limited to two spatial dimensions because more were not required. The extension to three spatial dimensions should not be too hard, however, and both the "backened" linear algebra solver and the random walk module should already support 3D modeling.

Finally, to prove that the developed software can be used to something reasonable it was slightly modified to fit a real life diffusion problem in which an enzyme diffuses through a wide dendrite into a very narrow dendritic spine ($< 1\mu$m). A thorough description of this problem is found in section **??**. The

formalism of the software also allows for testing of diffusion processes where spines are added and/or removed from the dendrite by some conservation rule as observed in some real life systems [].

1.3 What is computational neuroscience

Neuroscience is the scientific study of the central nervous system, but in the traditional sense it focuses very much on what parts of the brain are responsible for what. Computational neruroscience is more focused on the physics and chemistry involved in the different parts of the brain and nervous system. An example of the power of this approach is the classical work done by Hodgkin and Huxley in 1952, earning them the Nobel price in Physiology or medicine in 1963. Through four non-linear coupled differential equations they were able to predict the propagation speed of signals along the squid giant axon to a quite high precision.

The central nervous system

All animals except sponges and radially symmetric animals (like jellyfish) have a central nervous system (CNS) tasked with gathering and processing information about nerve impulses from the rest of the body. For vertebrates (animals with a backbone), the CNS is made up of a brain and a spinal cord. The spinal cord gathers information from the peripheral nervous system and relays (most of) this information to the brain where it is processed.

The human brain is an immensely complicated structure which we will only barely begin to describe here. It consists of two main types of cells; the neurons and neuroglia as well as blood vessels. The neurons are located (primarily) in the neocortex, which is what one would call the folded structure of gray-matter. Neurons are tasked with signal processing and transport, while the glial cells are thought to have more janitorial tasks like metabolic support and guidance of development. There are some different classifications of neurons, mostly by their location in the neocortex or by their geometry. This thesis focuses on pyramidal neurons, named after their triangular cell body, which are found in the hippocampus and the cerebral cortex (among others) in most (all?) mammals. Pyramidal neurons are chosen because of their large apical dendrite and the presence of dendritic spines.

Neurons and how they work

Neurons are bathed in a salt solution that is mainly Na^+ and Cl^-, but some other ions like potassium and calcium are also present and seem fairly important. Inside the neurons, a highly regulated salt solution of mainly K^+ sets up a potential difference relative to the outside of the cell of approximately -65mV. A common trait for pyramidal neurons is the presence of a single axon and a large dendrite at the apex of the cell body (Soma) called the apical dendrite. The axon is where the outgoing signals from the cell go, after they are gathered by the dendrites and integrated in the Soma. Axons can vary in length from millimeters to several centimeters, but are usually rather thin. They branch out towards the end to connect with several other neurons via synapses.

The activity level of a neuron varies a lot, some continuously fire action potentials at a frequency of 10-100 per second, while others are mostly quiet and fire a lot of action potentials given a certain input. These action potentials are generated in the body of the cell, called the soma, from where they propagate down the axon without loss of amplitude. This is achieved by constantly amplifying the signal using ion pumps (see the Hodgin-Huxley model of the action potential [4]). After propagating down the axon, the action potential reaches a synapse which is a gate to another neuron. If the action potential is of significant strength, vesicles carrying neurotransmitters merge with the synapse membrane, letting the neurotransmitters diffuse to the Post Synaptic Density (PSD of a spine on the receiving dendrite. If enough neurotransmitters reach the post-synaptic side, the signal continues propagating to the soma of this neuron, and the entire process starts over again.

Spines and memory storage/learning

The storage of memories on a sub-cellular level is thought to partly lie in the strengthening or weakening of the receiving synapse end, the PSD []. Familiar impressions will cause similar firing patterns in the sense that the same neurons tend to fire action potentials, which in turn are received by the same neurons each time. If the receiving synapses are strengthened or weakened the resulting integrated signals sent to the cell body will be hastened or held back accordingly and the response is strengthened and more time-efficient (DEFINETELY CITE SOMETHING HERE, OR JUST REWRITE... []). The strengthening of synapses is known as long-term potentiation (LTP, and the weakening of synapses is known as long-term depression (LTD).

LTP has been coupled directly with the enzyme protein kinase Cγ (PKCγ. The dynamics of the actual process is, in part, what will be modeled in this project. According to Craske et.al [2] PKCγ is released into the intracellular plasma of a neuron after being triggered by an increased concentration of calcium ions (Ca^{2+}).

Chapter 2

Basic Theory

This chapter will deal with random walks in general and the transition from the statistical view to PDEs. Different algorithms to produce random walks will be discussed, highlighting their pros and cons in light of this project along with some details which prove either problematic or helpful. Numerical solution of PDEs will also be discussed, also with emphasis on what is relevant for this project. Some words on (computational) neuroscience are needed, and can be found in section **??** along with a description of the physical application of this project. Finally an algorithm for combining random walk diffusion solvers and normal PDE diffusion solvers will be introduced and discussed.

2.1 Notation

There will be some mathematics in this chapter, and so it might be useful to clarify on the notation used throughout the project.

In the following section the Gaussian distribution will be derived from the Bernoulli distribution through some expectation values such as the expected total displacement of a random walker and the expected root mean square displacement of a walker. Expectation values are denoted by brackets. Equation (2.1) shows the expectation value of a quantity a.

$$\langle a \rangle = \sum aP(a) \tag{2.1}$$

Note the difference between the expectation value of a quantity squared and the square of the expectation value of the same quantity, stated in equation 2.2.

$$\langle a \rangle^2 \neq \langle a^2 \rangle \tag{2.2}$$

For the section on discretization of partial differential equations 2.4 some sub- and super- scripts will be used. To clarify; a superscript t^n does not mean t to the n'th power, but $n \cdot \Delta t$ meaning the n'th time-step. Similarly, the subscripts denote positions on the mesh. This means, finally, that u_{ij}^n denotes the numerical solution to the discretized PDE evaluated at time-step n and in the i'th point in x-direction and the j'th point in y direction:

$$u_{ij}^n = u(n \cdot \Delta t, i \cdot \Delta x, j \cdot \Delta y)$$

Throughout the thesis, the use of exponential functions might seem a bit inconsistent notation-wise. There are two different notations in use, e^C and $\exp(C)$. The latter is mainly used when the expression C is large, like a

fraction, to improve readability. However, this is abandoned if there might be room for misconception. As an example, the calculations in section 2.3.2 are done using e^C rather than $\exp(C)$ even though C might be a fraction. This is done because the exponential at some point is multiplied by a parenthesis, and where this multiplication takes place might have been unclear otherwise.

2.2 Physical scope

Say something about neuroscience, neurons, spines

2.3 Introduction to random walks

The most basic random walk (*is it though?*) is a walker on the x-axis which will take a step of a fixed length to the right with a probability p, or to the left with a probability $q = 1-p$. Using (pseudo-) random numbers on a computer we can simulate the outcomes of a random walk. For each step (of which there are N) a random number, $r \in [0, 1]$ is drawn from some distribution (say a uniform one) which will be the probability. If $r \leq p$ the walker will take a step to the left, otherwise it will take a step to the right. After the N steps the walker will have taken R steps to the right, and $L = N - R$ steps to the left. The net displacement from the origin will be $S = R - L$.
This simple approach is easily generalizable to two and three dimensions by having $2d$ possible outcomes from the random number, where d is the dimensionality. In two dimensions the walk can be split in two; first randomly choosing what dimension to perform the step in and then randomly choosing the sign of the step ($(-1)^r \cdot l$).

2.3.1 Random walkers and Gaussian distribution

The following derivation is borrowed from a compendium in statistical mechanics by Finn Ravndal.
If sufficiently many walks are done, the net displacement will vary from $S = +N$ to $S = -N$ representing all steps to the right and all steps to the left respectively. The probability of all steps being to the right is $P_N(N) = p^N$. Should one of the steps be to the left, and the rest to the right the resulting net displacement will be $S = N - 2$ with the probability $P_N(R = N-1) = Np^{N-1}q$. This can be generalized to finding the probability

of a walk with a R steps to the right as

$$P_N(R) = \binom{N}{R} p^R q^{N-R} \tag{2.3}$$

where $\binom{N}{R} = \frac{N!}{R!(N-R)!}$ is the number of walks which satisfy the net displacement in question, or the multiplicity of this walk in statistical mechanics terms. Equation 2.3 is the Bernoulli probability distribution, which is normalized.

$$\sum_{R=0}^{N} P_N(R) = (p+q)^N = 1^N = 1$$

From this distribution important statistical properties of a walk consisting of some N steps can be calculated. For example, the average number of steps to the right is

$$\langle R \rangle = \sum_{R=0}^{N} R P_N(R) = \sum_{R=0}^{N} \binom{N}{R} R p^R q^{N-R} =$$

$$p\frac{d}{dp}\sum_{R=0}^{N} \binom{N}{R} p^R q^{N-R} = p\frac{d}{dp}(p+q)^N = Np(p+q)^{N-1} = Np$$

The average value of the net displacement is found by using $S = R - L = R - (N - R) = 2R - N$.

$$\langle S \rangle = \langle 2R \rangle - N = 2Np - N(p+q) = N(2p - p - q) = N(p-q)$$

We notice that the average net displacement is greatly dependent on the relationship between p and q, and that any symmetric walk will have an expected net displacement of zero. In many cases the mean square displacement is of more interest than the displacement itself, because many important large scale parameters can be related to the root-mean-square displacement. This can also be calculated rather straightforwardly.

$$\langle R^2 \rangle = \sum_{R=0}^{N} R^2 P_N(R) = \sum_{R=0}^{N} \binom{N}{R} R^2 p^R q^{N-R} =$$

$$\left(p\frac{d}{dp}\right)^2 \sum_{R=0}^{N} \binom{N}{R} p^R q^{N-R} = \left(p\frac{d}{dp}\right)^2 (p+q)^N$$

$$= Np(p+q)^{N-1} + p^2 N(N-1)(p+q)^{N-2} = (Np)^2 + Np(1-p) = (Np)^2 + Npq$$

Like before, the average net displacement is given as $S^2 = (2R - N)^2$ and we obtain

$$\begin{aligned}\langle S^2\rangle &= 4\langle R^2\rangle - 4N\langle R\rangle + N^2 = 4((Np)^2 + Npq) - 4N^2p + N^2 \\ &= N^2(4p^2 - 4p + 1) + 4Npq = N^2(2p-1)^2 + 4Npq = N^2(p-q)^2 + 4Npq.\end{aligned}$$

which for the 1D symmetric walk gives $\langle S^2\rangle = N$ and the variance, denoted $\langle \Delta S^2\rangle = \langle \langle S^2\rangle - \langle S\rangle^2\rangle$, is found by insertion as

$$\langle \Delta S^2\rangle = \langle N^2(p-q)^2 + 4Npq - (N(p-q))^2\rangle = 4Npq \tag{2.4}$$

When the number of steps gets very large the Bernoulli distribution (eq. 2.3) can to a very good accuracy be approximated by the Gaussian distribution. This is most easily done in the symmetric case where $p = q = \frac{1}{2}$, but it is sufficient for the step-lengths to have a finite variance (*find something to refer to*). The Bernoulli distribution then simplifies to

$$P(S,N) = \left(\frac{1}{2}\right)^N \frac{N!}{R!L!} \tag{2.5}$$

on which we apply Stirling's famous formula for large factorials $n! \simeq \sqrt{2\pi n}\cdot n^n e^{-n}$.

$$\begin{aligned}P(S,N) &= \left(\frac{1}{2}\right)^N \frac{N!}{R!L!} \\ &= \exp\left(-N\ln 2 + \ln\sqrt{2\pi N} + N\ln N - \ln\sqrt{2\pi R} - R\ln R - \ln\sqrt{2\pi L} - L\ln L\right) \\ &= \sqrt{\frac{N}{2\pi RL}}\exp\left(-R\ln\frac{2R}{N} - L\ln\frac{2L}{N}\right)\end{aligned}$$

Where we have used $R + L = N$. Inserting $\frac{2R}{N} = 1 + \frac{S}{N}$ and $\frac{2L}{N} = 1 - \frac{S}{N}$ and expanding the logarithms to first order results in

$$\begin{aligned}P(S,N) &= \sqrt{\frac{N}{2\pi RL}}\exp\left(-\frac{N+S}{2}\frac{S}{N} + \frac{N-S}{2}\frac{S}{N}\right) \\ &= \sqrt{\frac{N}{2\pi RL}}\exp\left(-\frac{S^2}{2N}\right)\end{aligned}$$

Then inserting $RL = \frac{N^2 - S^2}{4}$ in the prefactor, and approximating $1 - \frac{S^2}{N^2} \simeq 1$ results in a discrete Gaussian distribution (eq. 2.6) with $\langle S\rangle = 0$ and $\langle S^2\rangle = N$.

$$P(S,N) = \sqrt{\frac{2}{\pi N}} \exp\left(\frac{-S^2}{2N}\right) \tag{2.6}$$

A continuous final position can be derived by assuming that the walker is moving on the x-axis, and letting the step-length, a, get small. The final position is now the continuous variable $x = Sa$. Similarly the time interval between each step, τ is assumed to be small, and the walker now runs for a continuous time-variable $t = N\tau$. This changes the distribution 2.6 to

$$P(x,t) = \frac{1}{2a}\sqrt{\frac{2\tau}{\pi t}} \exp\left(-\frac{x^2\tau}{2a^2 t}\right). \tag{2.7}$$

The prefactor $\frac{1}{2a}$ is needed to normalize the continuous probability distribution since the separation between each possible final position in walks with the same number of steps is $\Delta x = 2a$. Introducing the diffusion constant

$$D = \frac{a^2}{2\tau} \tag{2.8}$$

makes the distribution

$$P(x,t) = \sqrt{\frac{1}{4\pi D t}} \exp\left(-\frac{x^2}{4Dt}\right) \tag{2.9}$$

Introducing x also gives us the expectation value and variance of x on a form which will be useful later.

We have $x = Sa$ which means

$$\langle x \rangle = a\langle S \rangle$$

and

$$\langle x^2 \rangle = a^2\langle S^2 \rangle$$

Finally, the variance is found by insertion $\langle \Delta x^2 \rangle$

$$\langle \Delta x^2 \rangle = \langle \langle x^2 \rangle - \langle x \rangle^2 \rangle = \langle a^2\langle S^2 \rangle - a^2\langle S \rangle^2 \rangle = 4Npqa^2 \tag{2.10}$$

2.3.2 More general Random Walks

In the more general case, the position of a random walker, $\mathbf{r}$ at a time t_i is given by the sum

$$\mathbf{r}(t_i) = \sum_{j=0}^{i} \Delta\mathbf{x}(t_j) \tag{2.11}$$

where $\Delta\mathbf{x}(t_j) = (\Delta x(t_j), \Delta y(t_j), \Delta z(t_j))$ in 3D. Each $\Delta x, \Delta y, \Delta z$ is a random number drawn from a distribution with a finite variance $\sigma^2 = \langle \Delta x^2 \rangle$. By the central limit theorem, any stochastic process with a well defined mean and variance can, given enough samples, be approximated by a Gaussian distribution. This means that the probability of finding the walker at some position x after M steps is

$$P(x, M) \propto e^{-\frac{x^2}{2M\sigma^2}} \tag{2.12}$$

Recall that the actual Gaussian distribution is

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(n-\mu)^2}{2\sigma^2}\right)$$

Introducing an Einstein relation $\sigma^2 = 2dD\Delta t$ and the obvious relation $t = M\Delta t$ results in a more desirable exponent.

The introduction of the Einstein relation might put some restrictions on the model. Normalizing the expression gives

$$P(x, t) = \sqrt{\frac{1}{4Dt}} \exp\left(-\frac{x^2}{4Dt}\right) \tag{2.13}$$

A large number, N, of walkers can be described by their concentration $C(x, t) = NP(x, t)$. The concentration is conserved, so any amount that flows out of an area must reflect as a decrease in concentration. This is expressed by the flow of concentration

$$\frac{\partial C}{\partial t} - \nabla \cdot \mathbf{J} = S \tag{2.14}$$

where $\mathbf{J}$ is the flow vector and S is a source term which for now is zero. Through Fick's first law the diffusive flux is related to the concentration gradient $\mathbf{J} = -D\nabla C$. Inserting this gives

$$\frac{\partial C}{\partial t} = \nabla \cdot (D \cdot \nabla C) \tag{2.15}$$

which is the diffusion equation. Insertion of the Gaussian distribution (2.13)) verifies that the Gaussian distribution fulfills the diffusion equation. Starting with only the time derivative gives

$$\begin{aligned}
\frac{\partial P}{\partial t} &= -\frac{4\pi D e^{-\frac{x^2}{4Dt}}}{2\sqrt{(4\pi Dt)^3}} + \frac{x^2 e^{-\frac{x^2}{4Dt}}}{4Dt^2\sqrt{4\pi Dt}} \\
&= e^{-\frac{x^2}{4Dt}} \left(\frac{8Dx^2}{2\sqrt{\pi}(4Dt)^{5/2}} - \frac{(4D)^2 t}{2\sqrt{\pi}(4Dt)^{5/2}}\right) \\
&= \frac{4De^{-\frac{x^2}{4Dt}}(x^2 - 2Dt)}{\sqrt{\pi}(4Dt)^{5/2}}
\end{aligned}$$

Which is balanced by the spatial derivative

$$\begin{aligned} D\frac{\partial^2 P}{\partial x^2} &= \frac{D}{\sqrt{4\pi Dt}}\frac{\partial}{\partial x}\left[-e^{\frac{-x^2}{4Dt}}\left(\frac{-2x}{4Dt}\right)\right] \\ &= \frac{2D}{4Dt\sqrt{4\pi Dt}}e^{\frac{-x^2}{4Dt}}\left[1-x\left(\frac{2x}{4Dt}\right)\right] \\ &= \frac{4De^{-\frac{x^2}{4Dt}}(x^2-2Dt)}{\sqrt{\pi}(4Dt)^{5/2}} \end{aligned}$$

meaning that the diffusion equation is satisfied.

2.3.3 Choosing random walk algorithm

The simplest random walk model, which places walkers on discrete mesh points and uses a fixed step length, has been used with great success to model diffusion processes. Farnell and Gibson discuss this in their article [3]. In this project we will be torn between choosing a realistic algorithm to advance the random walkers, like Brownian motion or to go for simplicity. That being said, by the central limit theorem both models will after some time-steps be described by a Gaussian distribution meaning that on the PDE scale we will not know the difference. Hence it will make no sense to not use the simplest random walk model. Note that it will be quite easy to change the algorithm used for random walks, and so we have not locked ourselves to anything yet.

2.3.4 Random walks and anisotropy

Although the self diffusion problem, which is what we are looking at with diffusing particles in and out of dendritic spines is not anisotropic on its own, there might be some possible applications which require describing anisotropic diffusion of random walkers. There is reason to believe that an anisotropic diffusion process on the PDE level will lead to an anisotropic random walk model as well, but how should this be modeled. Simply replacing the diffusion constant by a function $D = D(\mathbf{x})$ (see eq (2.51)) is a natural first approximation, but this will not quite be sufficient as Farnell and Gibson point out [3]. Through their experiments they found that only adjusting the step-length will not improve the error noticeably and reasoned that this is because the walkers are still as likely to jump in both directions (right or left in 1d), and that the stepsize is the same in both cases, hence the model does not resemble anisotropy. They went on to introduce an adjusted step-length and an adjusted step probability, a solution they landed on after

trial and error. The expressions they proposed are listed in equations (2.16) to (2.20).

$$\Delta_p(x) = \frac{1}{2}\left(L(x) + L(x + \Delta_p(x))\right) \to L(x) + \frac{1}{2}L(x)L'(x) \tag{2.16}$$

$$\Delta_m(x) = \frac{1}{2}\left(L(x) + L(x - \Delta_m(x))\right) \to L(x) + \frac{1}{2}L(x)L'(x) \tag{2.17}$$

where $L(x)$ is defined in equation 2.18 and $\Delta_p(x)$ and $\Delta_m(x)$ are the adjusted step-lengths to the right and left, respectively.

$$L(x) = \sqrt{2D(x)\Delta t} \tag{2.18}$$

The adjusted jump probabilities $T_r(x)$ and $T_l(x)$ which are the probabilities for a walker at position x to jump right or left, respectively are defined in equations (2.19) and (2.20)

$$T_r(x) = \frac{1}{2} + \frac{1}{4}L'(x) \tag{2.19}$$

$$T_l(x) = \frac{1}{2} - \frac{1}{4}L'(x) \tag{2.20}$$

Notice that the adjusted step-length first proposed is still a part of the final expressions.

2.3.5 Random walks and drift

Another point which should be addressed is diffusion with a drift term, $\frac{\partial u}{\partial x}$.

Initially one thought that diffusion in the ECS of the brain was governed by a drift term, but the modern perception is that this drift term is in the very least negligible [9]. Though it is unlikely that we will include a drift term in our model, we will say a few words about it here since it is of importance in other applications and might be a natural extension at some point, should someone else use this work.

We model random walkers with drift by simply adding some vector to the Brownian motion model, thus forcing all walkers to have a tendency to walk a certain direction. This approach can also be used in the fixed steplength (or variable steplength in the anisotropic case) if we express the new step, $\mathbf{s}$, as

$$\mathbf{s} = (\pm l \text{ or } 0, \pm l \text{ or } 0) + \mathbf{d}$$

where **d** denotes the drift of the walker.
We can set up the continuity equation for a concentration, $C(x,t) = NP(x,t)$ of random walkers which are affected by a drift.

$$\frac{\partial C}{\partial t} + \nabla \cdot \mathbf{j} = S \tag{2.21}$$

Where **j** denotes the total flux of walkers through some enclosed volume and S is a source/sink term. Since the walkers are affected by drift the flux will consist of two terms; $\mathbf{j} = \mathbf{j}_{diff} + \mathbf{j}_{drift}$. From Fick's first law we know that $\mathbf{j}_{diff} = -D\nabla C$. The second flux term is the advective flux which will be equal to the average velocity of the system; $\mathbf{j}_{drift} = \mathbf{v}C$. Inserting this in the continuity equation gives us the well known convection diffusion equation (2.22).

$$\frac{\partial C}{\partial t} = \nabla \cdot (D\nabla C) - \nabla \cdot (\mathbf{v}C) + S \tag{2.22}$$

Which in many cases will simplify to

$$\frac{\partial C}{\partial t} = D\nabla^2 C - \mathbf{v} \cdot \nabla C \tag{2.23}$$

In order to determine all the parameters of the convection diffusion equation (2.22) we will need to go through some of the calculations from section 2.3. The situation is the same, a walker in one dimension which can jump left or right, but this time will also move a finite distance d each time-step. This will make the expected net displacement

$$\langle S \rangle = R - L + Nd = N(p-q) + Nd$$

and the expected mean square displacement

$$\langle S^2 \rangle = (2\langle R \rangle - N)^2 + (Nd)^2 = N^2(p-q)^2 + 4Npq + (Nd)^2$$

which in turn gives us the variance

$$\begin{aligned}\langle \Delta S^2 \rangle &= \langle \langle S^2 \rangle - \langle S \rangle^2 \rangle \\ &= N^2(p-q)^2 + 4Npq + (Nd)^2 - N^2(p-q)^2 - (Nd)^2 \\ \langle \Delta S^2 \rangle &= 4Npq\end{aligned}$$

This shows us that the variance is untouched by the drift term, but not the mean which for the symmetric case is $\langle S \rangle = Nd$. When we convert this to the continuous variables x and t we get the solution shown in equation 2.24.

$$C(x,t) = \frac{N}{\sqrt{4\pi Dt}} \exp\left(-\frac{(x-vt)^2}{4Dt}\right) \tag{2.24}$$

Where $v = \frac{d}{\Delta t}$ is the velocity of the concentration and D is the well known diffusion constant, inserted from the Einstein relation $\sigma^2 = 2D\Delta t$.

2.3.6 Pseudo-random numbers

This is also a large mathematical field which will only be touched in this thesis. Further reading can be found in the 1990s review by James [6].

Although modern technology has made true randomness available, we do not currently have access to this and are limited to pseudo-random numbers. For all purposes in this thesis, pseudo-randomness is presumed adequate.

A pseudo random number generator (RNG) is essentially a function which produces seemingly random numbers through a series of operations. These operations often rely on other, engineered numbers which will maximize the RNGs period. A period is the number of function calls which can be performed before the RNG starts producing the same sequence of numbers. Naturally this period should be as large as possible. There are quite a lot of methods to produce pseudo-random numbers [6], but only two are implemented in this project (not counting the built in RNGs in the "cmath" and "armadillo" modules).

ran0

This is a simple RNG borrowed from [5]. It is very fast, and simple to implement and use given that you know the engineered unlikely numbers it uses. The period of ran0 on the other hand is rather small, only $\sim 10^8$. For short simulations this will not be a problem, however.

Five seeded xor-shift

This is a more complicated algorithm developed by George Marsaglia. It uses five seeds which are all updated every time it is called to make a pseudo-random 64-bit unsigned integer (*check this*), which can be converted to a double. Implementing this algorithm is intricate, but it is found written out by Marsaglia [8]. As opposed to the ran0 algorithm this uses a series of bitshifts to produce the random numbers and update the seeds. It is not as fast as ran0, but only some $10 - 30\%$ slower. Because of the five seeds used, however, this algorithm has a period of $\sim 10^{48}$ [8] making it more than good enough for most MC-simulations I have heard about, and certainly for this project.

2.4 Some words about partial differential equations

2.4.1 Finite Difference Methods

Although there are a few methods for solving PDEs numerically, this project will focus on finite difference methods (FDM). This is done both for simplicity and because, as will be argued later, there is no need for very accurate solvers due to the error terms arising from random walk solvers.

Solution of PDEs through FDMs is done by approximating a continuous axis by a discontinuous mesh, and likewise the continuous PDE is approximated by its value at the discontinuous mesh-points. The derivatives in the PDE are then approximated by finite differences through the definition of the derivative (2.25), replacing the limit of h with the finite discretization parameter h.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{2.25}$$

The terms explicit and implicit scheme will also be used. An explicit scheme is a numerical scheme which solves a PDE numerically by using values which have already been calculated. A typical example of this would be to evaluate the spatial derivative at the previous time-step seeing as this has been calculated already. Implicit schemes typically tries to use values which have yet to be calculated in order to calculate new values. This can for example be achieved by evaluating the spatial derivative at the time-step which is being calculated, leading to a system of linear equations (see section 2.4.2 to see how this is done).

Numerical solution of PDEs is an enormous and important field, which cannot be done justice by a presentation here. For more on numerical solution of PDEs in general, consult [] or [].

2.4.2 Discretizing

To maintain a bit of generality, the (potentially) anisotropic diffusion equation in 2d will be discretized. The extension to 3d is trivial, as is the 1d version. The drift term will be omitted in the beginning because this also results in a simple addition to the numerical scheme. The equation to be discussed is

$$\frac{\partial u}{\partial t} = \nabla(D \cdot \nabla u) + f \tag{2.26}$$

where f is some source term. The final expression and scheme will depend on how the time derivative is approximated, but the spatial derivative will have the same approximation.
The innermost derivative is done first in one dimension, generalization to more dimensions is trivial, and will consist of adding the same terms for the y and z derivatives.

$$\left[\frac{d}{dx}u\right]^n \approx \frac{u^n_{i+1/2} - u^n_{i-1/2}}{\Delta x}$$

Where the derivative is approximated around the point x_i. Inserting $\phi(x) = D\frac{du}{dx}$ simplifies the outermost derivative, which will also be done in one dimension.

$$\left[\frac{d}{dx}\phi\right]^n \approx \frac{\phi^n_{i+1/2} - \phi^n_{i-1/2}}{\Delta x}$$

Inserting for ϕ gives the important intermediate step

$$\frac{\phi^n_{i+1/2} - \phi^n_{i-1/2}}{\Delta x} = \frac{1}{\Delta x^2}\left(D_{i+1/2}(u^n_{i+1} - u^n_{i+1}) - D_{i-1/2}(u^n_i - u^n_{i-1})\right)$$

Since the diffusion constant can only be evaluated at the mesh points (or strictly speaking since it is a lot simpler to do so), an approximation $D_{i\pm1/2} \approx 0.5(D_{i\pm1} + D_i)$ is used. Inserting this results in

$$\begin{aligned}\nabla D \nabla u \approx &\frac{1}{2\Delta x^2}\left((D_{i+1,j} + D_{i,j})(u_{i+1,j} - u_{i,j}) - (D_{i,j} + D_{i-1,j})(u_{i,j} - u_{i-1,j})\right) \\ &+\frac{1}{2\Delta y^2}\left((D_{i,j+1} + D_{i,j})(u_{i,j+1} - u_{i,j}) - (D_{i,j} + D_{i,j-1})(u_{i,j} - u_{i,j-1})\right)\end{aligned}$$

The discretization of the time-derivative is where a difference between the two schemes used in this project can be seen. When ordinary differential equations are discretized one can clearly see how this difference arises, and so the PDE is written in a new notation using some general operation, P like the double spatial derivative

$$D_t u = Pu \tag{2.27}$$

Introducing the general discretization of the time derivative gives equation (2.28) known as theta-rule. Setting $\theta = 0$ yields the Forward Euler (FE) discretization, and $\theta = 1$ the Backward Euler (BE) discretization.

$$\frac{u^{n+1}-u^n}{\Delta t} = P\left(\theta u^{n+1} + (1-\theta)u^n\right) \tag{2.28}$$

From the theta rule it is clear that the only difference between the FE and BE scheme is at what time-step the right-hand-side of the equation is evaluated. The theta-rule can give other schemes as well, using some weighted average of the right-hand-side at t^n and t^{n+1} but these are considered irrelevant in this project. Equation 2.29 summarizes what has been done so far by writing out the FE discretization the way it will be implemented in 1d.

$$u_i^{n+1} = \frac{\Delta t}{2\Delta x^2}\left((D_{i+1}+D_i)(u_{i+1}^n - u_i^n) - (D_i + D_{i-1})(u_i^n - u_{i-1}^n)\right) + u_i^n \tag{2.29}$$

We will come back to the FE discretization when we discuss stability later.
The BE discretization has more unknowns to be found at each mesh-point, as equation 2.30 shows.

$$u_i^{n+1} = \frac{\Delta t}{2\Delta x^2}\left((D_{i+1}+D_i)(u_{i+1}^{n+1} - u_i^{n+1}) - (D_i + D_{i-1})(u_i^{n+1} - u_{i-1}^{n+1})\right) + u_i^n \tag{2.30}$$

Writing out the calculations for a small mesh reveals a pattern which can be exploited.

$$\begin{aligned}
u_0^{n+1} &= \frac{\Delta t}{2\Delta x^2}\left(2(D_0+D_1)(u_1^{n+1} - u_0^{n+1})\right) + u_0^n \\
u_1^{n+1} &= \frac{\Delta t}{2\Delta x^2}\left((D_2+D_1)(u_2^{n+1} - u_1^{n+1}) - (D_1 + D_0)(u_1^{n+1} - u_0^{n+1})\right) + u_1^n \\
u_2^{n+1} &= \frac{\Delta t}{2\Delta x^2}\left((D_3+D_2)(u_3^{n+1} - u_2^{n+1}) - (D_2 + D_1)(u_2^{n+1} - u_1^{n+1})\right) + u_2^n \\
u_3^{n+1} &= \frac{\Delta t}{2\Delta x^2}\left(2(D_2+D_3)(u_3^{n+1} - u_2^{n+1})\right) + u_3^n
\end{aligned}$$

Rearranging this and setting $a = \frac{\Delta t}{2\Delta x^2}$ results in a normal system of linear equations

$$\begin{aligned}
u_0^{n+1}\left(1 + 2a(D_0+D_1)\right) - 2au_1^{n+1}(D_1+D_0) &= u_0^n \\
u_1^{n+1}\left(1 + a(D_2+2D_1+D_0)\right) - au_2^{n+1}(D_2+D_1) - au_0^{n+1}(D_1+D_0) &= u_1^n \\
u_2^{n+1}\left(1 + a(D_3+2D_2+D_1)\right) - au_3^{n+1}(D_3+D_2) - au_1^{n+1}(D_2+D_1) &= u_2^n \\
u_3^{n+1}\left(1 + 2a(D_3+D_2)\right) - 2au_2^{n+1}(D_3+D_2) &= u_3^n
\end{aligned}$$

which is arranged as

$$\begin{pmatrix} 1+2a(D_0+D_1) & -2a(D_1+D_0) & 0 & 0 \\ -a(D_1+D_0) & 1+a(D_2+2D_1+D_0) & -a(D_2+D_1) & 0 \\ 0 & -a(D_2+D_1) & 1+a(D_3+2D_2+D_1) & -a(D_3+D_2) \\ 0 & 0 & 1+2a(D_3+D_2) & -2a(D_3+D_2) \end{pmatrix} \mathbf{u}^{n+1} = \mathbf{u}^n \tag{2.31}$$

$$\mathbf{M}\mathbf{u}^n = \mathbf{u}^{n-1} \tag{2.32}$$

If the system of equations is solved by the fool-proof Gaussian elimination, some $\mathcal{O}(n^3)$ FLOPs are required per time-step. This will get even worse in more spatial dimensions; $\mathcal{O}(n^6)$ in 2D and $\mathcal{O}(n^9)$ in 3D. As a comparison the explicit scheme will make due with $\mathcal{O}(n^d)$ FLOPs. There are, however ways to improve this. Seeing as the matrix $\mathbf{M}$ does not change as long as none of the parameters change a LU-decomposition can be used. This will demand a decomposition of $\mathcal{O}(n^3)$ FLOPs, but all the subsequent steps will be $\mathcal{O}(n^2)$ FLOPs ($\mathcal{O}(n^{2d})$ for higher dimensions). This is still not quite at the level of the explicit scheme, but it is a clear improvement.
Looking closer at $\mathbf{M}$ we notice that it is not only sparse, but tridiagonal. This calls for further optimization which brings the required number of FLOPs down to $\mathcal{O}(n)$ making it equally efficient to the explicit scheme. More on tridiagonal Gaussian elimination later.

Though the applications covered by this project do not cover advection diffusion (diffusion with a drift term), a simple version will be added to the implementation to match the derived expression for RW with a drift term (see section 2.3.5). Recall the advection diffusion equation derived in section 2.3.5,

$$\frac{\partial C}{\partial t} = D\nabla^2 C - \mathbf{v}\cdot\nabla C$$

This equation has a first order spatial derivative which will make the numerical scheme less accurate if discretized the wrong way. Seeing as the spatial resolution is (often) coarser than the resolution in time, a term with a residual of the order $\mathcal{O}(\Delta x)$ would dominate the error of the scheme. It will therefore be necessary to use a more accurate approximation to the first derivative. The approximation

$$\mathbf{v}\cdot\nabla C \approx v\frac{C_{i+1}-C_{i-1}}{2\Delta x}$$

will not only reduce the residual to $\mathcal{O}(\Delta x^2)$, it is also incredibly simple to implement both in the FE and BE scheme seeing as the Neumann boundary conditions forces this term to be zero at the boundary. The resulting extra

implementation therefore consists of adding a simple term on the diagonals of the assembled matrix in the BE discretization, and an equally simple term in the FE scheme.

2.4.3 Stability

In section 2.4.2 the Forward Euler was used as an approximation to the time derivative. Unfortunately the resulting scheme is potentially unstable, as this section will demonstrate. First, the solution to equation (2.15) ($u(x,t)$) is assumed to be on the form

$$u(x,t) = A^n e^{(ikp\Delta x)} \tag{2.33}$$

where $i^2 = -1$ is the imaginary unit and A^n is an amplification factor which, for the solution (eq. (2.33)) ideally should be $e^{-\pi^2 t}$, but will be something else in the numerical case. Notice the restriction $|A| \leq 1$ if u is to not blow up. Inserting eq. (2.33) in the simplified version of the variable coefficient scheme (where the coefficient is constant) gives the following

$$e^{(ikp\Delta x)}\left(A^{n+1} - A^n\right) = A^n \frac{D\Delta t}{\Delta x^2}\left(e^{(ik(p+1)\Delta x)} - 2e^{(ikp\Delta x)} + e^{(ik(p-1)\Delta x)}\right)$$

$$A^n e^{(ikp\Delta x)}\left(A - 1\right) = A^n e^{(ikp\Delta x)} \frac{D\Delta t}{\Delta x^2}\left(e^{(ik\Delta x)} - 2 + e^{(-ik\Delta x)}\right)$$

Using the well known identities

$$e^{(iax)} + e^{(-iax)} = \frac{1}{2}\cos^2\left(\frac{ax}{2}\right)$$

and

$$\cos^2(ax) - 1 = \sin^2(ax)$$

gives the intermediate expression eq. (2.34)

$$A - 1 = \frac{D\Delta t}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right) \tag{2.34}$$

The "worst case scenario" in eq. (2.34) is $max(\sin^2\left(\frac{k\Delta x}{2}\right)) = 1$. Inserting this extreme value helps find the worst possible error term

$$A = \frac{D\Delta t}{2\Delta x^2} + 1 \implies \Delta t \leq \frac{\Delta x^2}{2D} \tag{2.35}$$

In 2d this criterion is halved, and for the anisotropic case the maximum value of D must be considered which, again, will be the "worst case scenario". The same procedure is used to determine the stability of the BE scheme

$$e^{(ikp\Delta x)}\left(A^n - A^{n-1}\right) = A^n \frac{D\Delta t}{\Delta x^2}\left(e^{(ik(p+1)\Delta x)} - 2e^{(ikp\Delta x)} + e^{(ik(p-1)\Delta x)}\right)$$
$$A^n e^{(ikp\Delta x)}\left(1 - A^{-1}\right) = A^n e^{(ikp\Delta x)} \frac{D\Delta t}{\Delta x^2}\left(e^{(ik\Delta x)} - 2 + e^{(-ik\Delta x)}\right)$$

which leads to

$$A = \frac{1}{1 + \frac{D\Delta t}{\Delta x^2}} \tag{2.36}$$

Equation (2.36) is smaller than 1 for all $\Delta t > 0$ which means that the scheme is unconditionally stable.

2.4.4 Truncation error

The numerical derivative is not the analytical derivative, but an approximation. This approximation has a well defined residual, or truncation error which is found by Taylor expansion. The following section will derive the residuals for the approximations used in this project in order to verify the numerical implementation later. For the FE time-derivative scheme, the residual is defined as

$$R = \frac{u(t_{n+1}) - u(t_n)}{\Delta t} - u'(t_n)$$

Recall the Taylor expansion of $u(t+h) = \sum_{i=0}^{\infty} \frac{1}{i!}\frac{d^i}{dt^i}u(t)h^i$

$$\begin{aligned} R &= \frac{u(t_n) + u'(t_n)\Delta t + 0.5u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3) - u(t_n)}{\Delta t} - u'(t_n) \\ &= u''(t_n)\Delta t + \mathcal{O}(\Delta t^2) \\ R &\sim \mathcal{O}(\Delta t) \end{aligned}$$

Similarly, the BE scheme has the following residual Recall the Taylor expansion of $u(x-h) = \sum_{i=0}^{\infty} \frac{1}{i!}\frac{d^i}{dt^i}u(x)(-h)^i$

$$\begin{aligned} R &= \frac{u(t_n) - u(t_n - 1)}{\Delta t} - u'(t_n) \\ &= \frac{u(t_n) + \Delta t u'(t_n) + 0.5\Delta t^2 u''(t_n) + \mathcal{O}(\Delta t^3)}{\Delta t} - u'(t_n) \\ R &\sim \mathcal{O}(\Delta t) \end{aligned}$$

There are many discretization schemes with much smaller residuals than these, but in this project the PDE is not the only error source seeing as a random walk solver will be introduced, and so the FE/BE schemes are deemed accurate enough.

The spatial derivative also has a well defined residual defined by

$$R = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} - u''(x_i) \tag{2.37}$$

Doing the expansions and cleaning up a bit

$$\begin{aligned}
&R = \\
&\frac{u'(x_i)\Delta x + 0.5u''(x_i)\Delta x^2 + \frac{1}{6}u^{(3)}(x_i)\Delta x^3 + \frac{1}{24}u^{(4)}(x_i)\Delta x^4 + \mathcal{O}(\Delta x^5)}{\Delta x^2} + \\
&\frac{2u(x_i) - 2u(x_i)}{\Delta x^2} - u''(x_i) + \\
&\frac{-u'(x_i)\Delta x + 0.5u''(x_i)\Delta x^2 - \frac{1}{6}u^{(3)}(x_i)\Delta x^3 + \frac{1}{24}u^{(4)}(x_i)\Delta x^4 + \mathcal{O}(\Delta x^5)}{\Delta x^2} \\
&R = u''(x_i) + \frac{1}{12}u^{(4)}(x_i)\Delta x^2 + \frac{\mathcal{O}(\Delta x^5)}{\Delta x^2} - u''(x_i) \\
&R \sim \mathcal{O}(\Delta x^2)
\end{aligned}$$

There are discretizations that can reduce this residual even further (although a second order scheme is usually considered adequate), but this time the stability criterion on the time derivative (eq. (2.4.3)) will always be of the order $\mathcal{O}(\Delta x^2)$ and so we will never get a smaller error than this unless we change the time derivative.

Quantifying an error term for the random walk solver is not straightforward, but naturally it will be closely coupled to the number of walkers used. Statistical mechanics states that statistical fluctuations around a steady state is related to the number of samples, N, which in this case is the number of walkers, through eq. (2.38).

$$\langle \Delta u \rangle \propto \frac{1}{\sqrt{N}} \tag{2.38}$$

In the combined solver, we assume that equation 2.38 still holds for the RW-part of the solution even though we can only say for certain that it is correct for the first time-step. The number of walkers, N is now given by the defined conversion factor Hc as

$$N(x, y, t) = Hc \cdot U(x, y, t) \tag{2.39}$$

and the total number of walkers is the sum of the walkers on all the mesh-points. In each mesh-point the fluctuations are of the order $\sqrt{N(x_i, y_j, t_n)}^{-1}$, meaning that the convergence rate in each mesh-point is $\frac{1}{2}$.

A lot of time has gone into forcing this error to be negligible by introducing many walkers and fining a clever way to combine the RW solution with the PDE solution. Two problems turn out to make this a lot harder than it seems:

- The required number of walkers is very large.
- The walkers must be "reset" at each time-step.

At this point there does not seem to be any solution to this, and so only a similar example can be presented to show that the principle works, no actual evidence.

2.4.5 Tridiagonal linear systems

The implicit discretization results in a set of linear equations, or a linear system, to solve at each timestep. Neumann boundaries combined with a first derivative in time makes the linear system band diagonal, where the number of non-zero bands on the matrix is dependent on the number of spatial dimensions the problem is solved in. In one spatial dimension this reduces to a tridiagonal system, which can be solved extremely efficiently by the "tridiag" function listed below. In two spatial dimensions we are not quite as fortunate as in one dimension, and get a banded matrix with 2n bands and five non-zero bands, where n is the spatial resolution (which is equal in x and y direction). Rewriting the assembled matrix (see eq. A.2) to a block-matrix form makes the matrix tridiagonal again, but the entries are $n \times n$ matrices.

Assuming the system has a solution, the fool-proof way to solve a linear equation $\mathbf{Mx} = \mathbf{b}$ where $\mathbf{M}$ is not a sparse matrix, by Gaussian elimination.

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \mathbf{x} = \mathbf{b} \tag{2.40}$$

Is reduced to

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & (a_{22} - \frac{a_{21}a_{12}}{a_{11}}) & (a_{23} - \frac{a_{21}a_{13}}{a_{11}}) & (a_{24} - \frac{a_{21}a_{14}}{a_{11}}) \\ 0 & (a_{32} - \frac{a_{31}a_{12}}{a_{11}}) & (a_{33} - \frac{a_{31}a_{13}}{a_{11}}) & (a_{34} - \frac{a_{31}a_{14}}{a_{11}}) \\ 0 & (a_{42} - \frac{a_{41}a_{12}}{a_{11}}) & (a_{43} - \frac{a_{41}a_{13}}{a_{11}}) & (a_{44} - \frac{a_{41}a_{14}}{a_{11}}) \end{pmatrix} \mathbf{x} = \tilde{\mathbf{b}} \tag{2.41}$$

and further to

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & (a_{22} - \frac{a_{21}a_{12}}{a_{11}}) & (a_{23} - \frac{a_{21}a_{13}}{a_{11}}) & (a_{24} - \frac{a_{21}a_{14}}{a_{11}}) \\ 0 & 0 & (\tilde{a}_{33} - \frac{\tilde{a}_{32}a_{23}}{\tilde{a}_{22}}) & (\tilde{a}_{34} - \frac{\tilde{a}_{32}a_{24}}{\tilde{a}_{22}}) \\ 0 & 0 & (\tilde{a}_{43} - \frac{\tilde{a}_{42}a_{23}}{\tilde{a}_{22}}) & (\tilde{a}_{44} - \frac{\tilde{a}_{42}a_{34}}{\tilde{a}_{22}}) \end{pmatrix} \mathbf{x} = \tilde{\mathbf{b}} \quad (2.42)$$

finally resulting in an upper triangular matrix. A backwards sweep is then performed to solve for one element of the unknown vector, $\mathbf{x}$ at a time.

Since most entries are zero we can easily get away with only doing one forward sweep down the matrix, eliminating all the sub-diagonal matrix-entries, and then one backward sweep, which calculates the unknown vector $\mathbf{x}$. The algorithm is listed below as a function implemented in C++.

```
void tridiag(double *u, double *f, int N, double *a, double
    *b, double *c){
    double *temp = new double[N];
    for(int i=0;i<N;i++){
      temp[i] = 0;
    }
    double btemp = b[0];
    u[0] = f[0]/btemp;
    for(int i=1; i<N; i++){
        //forward substitution
        temp[i] = c[i-1]/btemp;
        btemp = b[i]-a[i]*temp[i];
        u[i] = (f[i] -a[i]*u[i-1])/btemp;
    }
    for(int i=(N-2);i>=0;i--){
        //Backward substitution
        u[i] -= temp[i+1]*u[i+1];
    }
    delete[] temp;
}
int main()
{
    return 0;
}
```

The tridiagonal solver from the one-dimensional case can be modified so it can be used on block-tridiagonal systems. The modified algorithm for the block-tridiagonal matrix (2.43) is listed in (2.44), and is in fact only the linear algebra version of the "tridiag" function, replacing the $1.0/btmp$ -terms with $(B_i + A_i H_{i-1})^{-1}$. The result of rewriting the $2n$-band diagonal matrix is the block matrix in equation (2.43).

$$\begin{pmatrix} B_0 & C_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & A_i & B_i & C_i & 0 & & 0 & 0 \\ 0 & \ddots & 0 & \ddots & \ddots & \ddots & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{n-1} & B_{n-1} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0^{n+1} \\ \mathbf{u}_1^{n+1} \\ \vdots \\ \mathbf{u}_i^{n+1} \\ \vdots \\ \mathbf{u}_n^{n+1} \end{pmatrix} = \mathbf{u}^n \quad (2.43)$$

which can also be expressed as $\mathbf{Mx} = \mathbf{k}$. Block-matrices named B_i are tridiagonal, and the ones named A_i or C_i are strictly diagonal.

There is a forward substitution

$$\begin{aligned} H_1 &= -B_1^{-1}C_1 \\ H_i &= -\left(B_i + A_i H_{i-1}\right)^{-1} C_i \\ \mathbf{g}_1 &= B_1^{-1}\mathbf{k}_1 \\ \mathbf{g}_1 &= \left(B_i + A_i H_{i-1}\right)^{-1}\left(\mathbf{k}_i - A_i \mathbf{g}_{i-1}\right) \end{aligned} \quad (2.44)$$

Followed by a backward substitution

$$\begin{aligned} \mathbf{x}_{n-1} &= \mathbf{g}_{n-1} \\ \mathbf{x}_i &= \mathbf{g}_i + H_i \mathbf{x}_{i+1} \end{aligned}$$

The algorithm requires inverting approximately 3n $n \times n$ matrices, which might be expensive. However, the inversion only needs to be done once as long as the mass-matrix, $\mathbf{M}$ is unchanged, and so the expense is reduced. This should result in a computational intensity of around $\mathcal{O}(n^2)$ seeing as we only need to do one matrix-matrix multiplication where one matrix is diagonal, and two matrix-vector multiplications. All of which demand $\mathcal{O}(n^2)$ operations. This reduction in computational cost makes the implicit scheme as effective as the implicit FE scheme.

In three dimensions we are even more unfortunate and get an $2n^2$-banded matrix and seven non-zero bands. As the appendix shows (eq. A.5) this can be written so it looks like the block tridiagonal linear system from the 2d case, and that could be solved by the block-tridiagonal solver. The difference is that the entries i n the block-matrices A_i, B_i and C_i are block-matrices themselves meaning that the block tridiagonal solver must work with $n^2 \times n^2$ matrices rather than $n \times n$. All in all the performance should be around $\mathcal{O}(n^3)$ if the inverted matrices from the forward substitution are saved as in the 2d case. This is about the same performance as the explicit scheme gives, but without the stability issue.

2.5 Combining the two solvers

This section will deal with the actual combination of the two models.

2.5.1 Changing between length scales

As was mentioned in the introduction to this thesis, the combination of two length scales immediately raises the question of exactly where the limit between the two scales goes, and why exactly there. This question is usually answered with "models are switched when effects from the smaller scale becomes negligible" when moving from small to larger scales, and similarly "models are switched when effects from the smaller scale become dominant" in the opposite case. In effect this is what is done for the applications of this project, but an attempt to combine the two models in one simulation will also be done. An are of the computational mesh will be solved by a RW model first, representing a small length scale, and this solution will be used as input for the PDE solver. This approach requires a way to quickly convert from a concentration to a distribution of walkers, and back. The conversion is done by a conversion factor which will be named Hc (as it was named by Plapp and Karma [10], though they used a conversion field) which is introduced in equation (2.45).

$$C_{ij} = \frac{a}{\Delta t} U_{ij} \tag{2.45}$$

For the most part, equation (2.45) will be rewritten to just one conversion factor times the PDE solution, giving us some flexibility should we want to add more dependencies in the conversion. As of now, the conversion factor, Hc, is defined in equation (2.46). One "unit" of U_{ij} will directly correspond to Hc random walkers.

$$Hc = \frac{a}{\Delta t} \implies C_{ij} = Hc \cdot U_{ij} \tag{2.46}$$

Methods of combination

Effectively the problem is solved twice as the software stands now, both by the PDE and by the RW model. These two solutions must be coupled, and there are a few possibilities as to how this can be done.

- Regression
  Some kind of regression can be used, where the solution from the RW

model is used as data points. This results in an expression that approximates the solution from the RW model. Alternatively, the average of the two solutions can be done before the regression. The resulting expression is used to calculate the solution over the area in question.

- Average
  A simple average can be used. This will to some extent follow the "correct" solution from the PDE method, ensuring some smoothness, and have the fluctuating properties from the RW solution.

- Interpolation
  This approach can ensure that the endpoints follow the solution outside the relevant area, but requires choosing some points in stead of other points from the RW solution.

- Replacing
  Alternatively none of the above can be chosen, and the PDE solution can simply be replaced by the RW solution. This might well be the best approach.

2.5.2 The basic algorithm

The basic structure of the program is rather similar to the physical problem. There is one dendrite-object which contains the PDE-solver for the normal diffusion equation, with the possibility to use a random walk solver instead. On the dendrite object spines can be placed, which in the physical world are the receiving end of a synapse. Depending on what is being modeled, synaptic input is modeled by randomly added spikes of some random number of molecules which spawn at the far end of the spine. In the overlapping points where the spine is located on the dendrite mesh, the coupling is done as follows: If a random walker in the spine comes in contact with the position labeled as the "end" of the spine it is moved from the list of active walkers to a list of walkers which have moved out of the spine. Similarly, at each time-step a part of the PDE-solution corresponding to one walker will diffuse into the spine with a certain probability. It might be desirable for a walker to only be able to diffuse out of the spine with some probability as well, or for the walkers which diffuse into the spine to have some drift term, but these are minor updates and might be added later if needed.

There is also the possibility of modeling parts of the dendrite-mesh as random walk (this can be done in 2d as well as 1d). This is done by choosing some points on the mesh and sending the to the "AddWalkArea" method

which will map them to an index and set the initial condition for the walk. Although anisotropy will follow into the random walk solver, by the method provided by Farnell and Gibson [3]. At each time-step the solve-method of the combined solver is called, which in turn calls the solve method for the PDE-solver. The solution from the PDE-solver is used to calculate the number of walkers by eq. 2.46 in each mesh-point on the PDE-mesh, and then give each walker a random position in a square around its mesh-point ($\pm\frac{\Delta x}{2}$). Because the sum of the PDE-solution over the random walk area of the mesh might be different from one time-step to the next (eq. 2.47) the conversion from PDE-solution to random walker distribution must be done at every time-step. The alternatives are to remove or add the difference at each time-step, but this will require checking that each mesh-point has the "correct" number of walkers and updating the number to correspond with the solution from the PDE-solver. *which is what we are doing already?* Or the conversion factor could be adjusted at each time-step. The latter is largely a bad solution because it ruins transparency and might introduce even more fluctuations in the solution.

$$\sum u_{i,j}^{n} \neq \sum u_{i,j}^{n+1} \tag{2.47}$$

After the random walk integration the two solutions are combined by a simple average. A few other methods have been tested (see chapter 2.5.1) but discarded. The average of the two solutions is then set as the new "initial condition" for the next time-step, and the process repeats itself.

The results of these are inserted in the solution from the PDE using some routine (e.g. the average of the two) and the time-step is done. A schematic of the algorithm is provided in figure 2.1.

Figure 2.1: Schematic diagram of the algorithm.

2.5.3 Convergence rate

In chapter 2.4.4 the error that arises as a result of adding random walkers on parts of the mesh was discussed. The amplitude of the fluctuations per mesh-point was found proportionate to $\frac{1}{\sqrt{N}}$ where N is the number of walkers related to the mesh-point. Combining the two models means adding fluctuations to the approximation to the exact solution. Seeing as this combined solution is sent to the PDE-solver as an "initial condition" for the next time-step we have made a compromise in accuracy. The error-estimate, which will be defined later, is still dependent of Δt, but the dependency is now of the

order $\mathcal{O}(\sqrt{\Delta t})$. This also further supports the claim that there is no need to find a very precise scheme to solve the PDE.
We will test this by doing a convergence test in time keeping the number of walkers constant.

2.5.4 Potential problems or pitfalls with combining solutions

This section will identify and discuss a few obvious difficulties which might arise in this project. As far as possible solutions or workarounds will be presented, but some problems might not be solvable.

Different timescales

The PDE-solver will be operating with some time-step Δt which will, depending on the discretization of the PDE, have some constraints and will definitely have an impact on the error. The walkers will, as we have just seen, solve the diffusion equation as well, but with some different $\Delta\tilde{t}$ which is smaller than the time-step on the PDE level. Depending on the coupling chosen between the two models this difference will have some effect or a catastrophic effect on the error. Running some number of steps, N, on the random-walk level should eventually sum up to the time-step on the PDE level, $\sum_{i=0}^{N} \Delta\tilde{t} = \Delta t$. Section 2.5.5 shows that restricting the step length of the walkers will improve the coupling between the two solvers as far as possible.

Boundary conditions

To combine the two models, some restricting boundary conditions must be put on the random walkers. This is not usually done (as far as I have seen), but not very difficult. Finding a boundary condition that accurately models the actual system turns out to be quite straightforward. The assumption that the number of walkers in the walk-domain is conserved for each PDE time-step can be made, and thus no walkers can escape the domain. Implementing perfectly reflecting boundaries solves this quite well. This means that the flux of walkers out of a boundary is zero, which is the same as Neumann boundary conditions on the PDE level.
Dirichlet boundaries can (probably) be implemented by adding or removing walkers on the boundaries (or in a buffer-zone around them) until the desired concentration of walkers is reached.

Negative concentration of walkers

The concentration of walkers is calculated as $NP(x,t)$ where $P(x,t)$ is really only an estimate of the actual probability distribution, calculated by dividing the number of walkers in one area $x \pm \frac{\Delta x}{2}$ by the total number of walkers. Seeing as negative probabilities does not make sense, and neither does a negative number of walkers, we will eventually run into some problems if the solution of the PDE takes negative values (which it most likely will not do). No good solutions have been found to this problem, but a workaround consists of storing the sign of the solution over each time-step, converting the absolute value to a distribution of random walkers and multiplying back with the sign after the RW solution is done. This workaround has a problem in that a transition from positive to negative value will lead to a "valley" in the absolute-value solution. A normal PDE solution of this kind of initial condition will very rapidly even out the "valley", and so a value which should have been zero (a node-point) will get some other value (say some fraction of the conversion factor). This again leads to a larger discontinuity when the solution from the RW model is multiplied by the sign again.

Smooth solutions

A diffusion process is very effective when it comes to dampening fast fluctuations, and so any solution of the diffusion equation will be smooth. When a stochastic process is introduced, rapid fluctuations from one time-step to the next might follow. In this case a dilemma arises; on the one hand there is the smoothness of the solution to consider, on the other hand the stochastic term was introduced believing that it adds detail to the model. The approach we tried to use to this was to do some curve-fitting using both of the solutions. A polynomial regression model was implemented in 1d, but regardless of degree and what points were used, the result was a lot worse than just the average of the two solutions. Another idea is to implement a cubic spline interpolation over the area, but this too has its problems. An interpolation forces the solution to have a value at the interpolating points, and seeing as we cannot say for certain which value is correct how shall we pick the interpolating points?

Number of time-steps on the random walk level

As the time-step on the PDE level is increased above the stability criterion of the FE scheme towards more efficient sizes we are faced with the problem of whether or not to increase the number of time-steps on the RW level. Strictly speaking we do not have to do this, seeing as we adjust the step-length of the

walkers with respect to the time-step (see eq 2.51). As an initial value we put the number of time-steps to 100, but this was more a guess of how many are necessary for the central limit theorem to have effect than anything else. The question really boils down to how we define our model, which we have yet to do in an accurate way.

Random walks in 3D

Both 1 and 2 dimensional space are spanned completely by a random walk, but space of 3 or more dimensions is not. This does not have to be a problem, seeing as we have proved that the random walk fulfills the diffusion equation (chapter 2.3.2) and we are not trying to span the complete 3d space, but we could potentially meet some difficulties as a result of this property of the random walk.

2.5.5 Probability distribution and time-steps

As section 2.3.2 shows, the probability of finding a walker at a position x_i after some N time-steps (on the walk-scale) is (in the limit of large N) given as the Gaussian distribution. In this application, however, finding the walker at an exact position is not of interest, but finding it in an interval around the mesh-points sent to the walk-solver is. This interval is (for obvious reasons) $x_i \pm \frac{\Delta x}{2}$ where Δx is the mesh resolution on the PDE level. The walk solver will also run for some N time-steps on the random-walk scale (where N steps on the random walk scale is the same as one step on the PDE scale). This slightly modifies the distribution into

$$P(x_i \pm \Delta x, t_{n+1}) = \frac{1}{\sqrt{4\pi DN\Delta\tilde{t}}} \exp\left(\frac{(x \pm \Delta x)^2}{4DN\Delta\tilde{t}}\right) \tag{2.48}$$

Making the concentration of walkers $C(x,t) = MP(x,t)$

$$C(x_i \pm \Delta x, t_{n+1}) = \frac{M}{\sqrt{4\pi DN\Delta\tilde{t}}} \exp\left(\frac{(x \pm \Delta x)^2}{4DN\Delta\tilde{t}}\right) \tag{2.49}$$

For each PDE time-step the walkers are reset to have some new initial condition. This is done because the concentration over the "walk-area" will change with each PDE time-step. The point is that $C(x_i \pm \Delta x, t_{n+1})$ will be dependent on the initial condition $C(x_i \pm \Delta x, t_n)$.

Equation (2.6) shows that the step-size on the random walk scale is dependent on the variance in the actual steps (This is in principle the Einstein relation).

$$\sigma^2 = \langle \Delta x^2 \rangle = 2DN\Delta\tilde{t} \implies \Delta\tilde{t} = \frac{\langle \Delta x^2 \rangle}{2DN} \tag{2.50}$$

Equating this with 2.10 gives a first order approximation to the step-length, l

$$\begin{aligned} \langle \Delta x^2 \rangle &= 4pqNl^2 = 2DN\Delta\tilde{t} \\ l &= \sqrt{2D\Delta\tilde{t}}. \end{aligned} \tag{2.51}$$

Of course this is assuming that a random walk algorithm of fixed step-length is used.
Equation 2.51 is proportional to the square-root of the adjusted time-step. We have already suggested that the error term from the RW simulation depends on the number of walkers we use (or the conversion rate). This equation suggests that the error term also depends on the time step. Though this might seem a bit frustrating at first glance, it answers a question asked earlier; how many time-steps are needed at the RW-level. We have the intuitive relation between the RW time-step, $\Delta\tilde{t}$ and the PDE time-step through the number of steps at the RW level, T:

$$\Delta\tilde{t} = \frac{\Delta t}{T}$$

Further, the error from the RW simulation was suggested to be proportional to the square root of the time-step $\epsilon \propto \sqrt{\Delta\tilde{t}} = \sqrt{\frac{\Delta t}{T}}$. Forcing the error to behave as $\epsilon \propto \mathcal{O}(\Delta t)$ might be achieved by adjusting the number of time-steps taken at the RW level (but actually not).

$$\mathcal{O}(\Delta t) > \sqrt{\frac{\Delta t}{T}} \implies T > \frac{1}{\Delta t} \tag{2.52}$$

Combined with the demand to the number of walkers this will quickly result in an extreme computational demand in order to force our model to have first order convergence (not to mention second order convergence). Fortunately, this demand will be ignored outside of verification because there is little physical meaning left if the demanded number of walkers is used.

2.6 Geometry

Any finite difference method is problematic to solve on anything else than a rectangular grid. Using an implicit FD method will add an additional "demand" of having a square grid as well. Fortunately, using an implicit solver will eliminate the stability issue.
The purpose of this project is to investigate the actual coupling of two models

for the same problem.
If we want to model diffusion on a general geometry by a FD method we could transform the grid to a unit-square through a general transform.

$$\mathbf{r} = \mathbf{T}(\mathbf{q})$$

Here, $\mathbf{r} = (x, y)$ is the position in physical space, $\mathbf{q} = (\xi, \eta)$ is the position on the unit-square that is the computational space and $\mathbf{T}$ is the transformation. The transformation is achieved by the functions $x(\xi, \eta)$ and $y(\xi, \eta)$. After a lot of math, including some differential geometry the diffusion equation in computational space is found to still have the form

$$\frac{\partial C}{\partial t} - \nabla \cdot \mathbf{J} = 0$$

but the total flux vector $\mathbf{J} = \mathbf{f} + \phi$ now has the properties

$$\nabla \cdot \mathbf{f} = \frac{1}{g} \cdot \left(\mathbf{f} \cdot \frac{\partial}{\partial \xi} \left(\frac{\partial y}{\partial \eta}, \frac{-\partial x}{\partial \eta} \right) + \mathbf{f} \cdot \frac{\partial}{\partial \eta} \left(\frac{-\partial y}{\partial \xi}, \frac{\partial x}{\partial \xi} \right) \right)$$
$$\nabla \phi = \frac{1}{g} \left(\phi \cdot \frac{\partial}{\partial \xi} \left(\frac{\partial y}{\partial \eta}, \frac{-\partial x}{\partial \eta} \right) + \phi \cdot \frac{\partial}{\partial \eta} \left(\frac{-\partial y}{\partial \xi}, \frac{\partial x}{\partial \xi} \right) \right)$$

where g is the Jacobian of the transformation $\mathbf{T}$. In other words, a whole new and much more complicated equation must be discretized in order to solve for a general geometry. Some idea of what the functions $x(\xi, \eta)$ and $y(\xi, \eta)$ are is also required, and this information is generally not available. Furthermore, there already exists Finite Element software which can take any geometry as a mesh, and so a simpler way of using a more interesting geometry would be to change discretization from FD to a finite element method, for example in the open-source finite element software FEniCS. This is considered an extension, and will not be covered here.

Chapter 3

Analysis

This chapter will contain most of the numerical error analysis and some of the discussion of this analysis as well as a recap of the methods used for error analysis in general, and how they are adapted to this particular problem.

In the numerical setup chosen for this project some new error sources might be introduced in addition to the normal errors introduced by numerical solution of any equation (see section 2.4). When a part of the solution acquired is replaced by the solution from a stochastic model, the initial condition to the next iteration in time will be changed. This might have a number of effects on the final solution. Figure 3.1 shows the typical effects of solving an equation numerically. When a stochastic solution is imposed on top of this again, it will lead to fluctuations around the approximations to the solution at the new time-step which most likely worsens the approximation. The aim of this chapter is to verify the implementation and investigate the effects of adding a stochastic solution to a normal PDE solution.

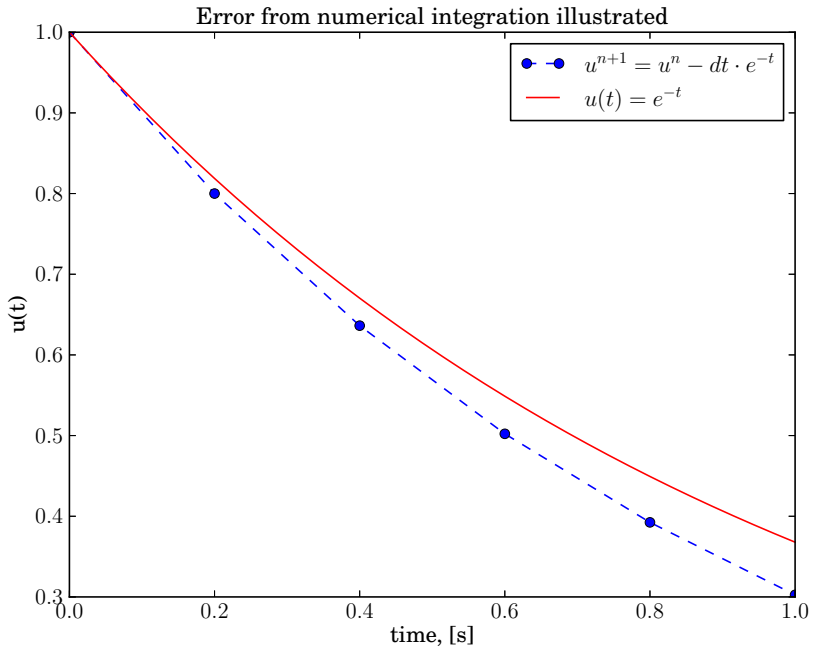


Figure 3.1: Illustration of how numerical errors appear. The dashed line shows the numerical solution of the equation $\frac{du}{dt} = -e^{-t}$ calculated with 6 points using a FE scheme, while the full line is the exact solution $u(t) = e^{-t}$. The numerical scheme approximates the derivative as a constant between two points, an approximation which will become increasingly good as the resolution improves.

3.1 Introduction

3.1.1 The error estimate

First and foremost the error measure, which will be denoted ϵ will be specified. Throughout this thesis the term error is used quite lazily, but unless something else is specified we refer to the expression

$$\epsilon(t_i) = ||u_e(t_i) - u(t_i)||_2 \tag{3.1}$$

where u_e is an exact (manufactured) solution to the equation, and u is the result from the numerical simulation. This error-estimate is used because it is time-dependent, thus explicitly showing how the error evolves over time. The error is calculated over the entire mesh, which will clearly show if the error from the random-walk areas are dominating, or (otherwise) how the PDE-scheme is holding up. Another error estimate that will be used later is the integrated norm of the previously mentioned error estimate. This will be used for the convergence tests to make sure that no effects of simulating for a long time are overlooked, and is defined as

$$\epsilon = \sqrt{\sum_i h^2 \epsilon(t_i)} \tag{3.2}$$

where h is the parameter in question (usually Δt).

3.1.2 Verification techniques

There are quite a few ways to verify an implementation. This thesis will focus on three which are considered adequate for no particular reason. Unfortunately some of the tests require isolating one error-term, meaning that the test works better if the stability criterion for the FE scheme is broken. The result of this is that some of the tests are only done on the BE scheme. Fortunately, the BE scheme will be used for the actual simulations and will see more thorough verification.

- Method of manufactured solutions
  A normal way of checking that our scheme of choice is implemented correctly is by constructing an exact solution to the equation and checking that the error is of the expected order. Since the software contains both an explicit FE scheme and an implicit BE scheme they will both be verified alongside one another. We will start with the simplest diffusion equation (eq. (2.15)) and add complexity until the expression verifies

all parts of the implementation. Both schemes are expected to have an error-term of the order of Δt, which in the FE case is limited by a stability criterion. Next, the initial condition for a simulation is set to the exact solution at $t = 0$, and the simulation can be run with different parameter values. This approach makes it trivial to calculate the error, seeing as the solution is known, and to check that it takes the expected values.
There are some variations of this method which should be mentioned. Setting the manufactured solution to a constant (or a linear polynomial if the boundary conditions allow it) should eliminate the error completely since the derivative is zero which eliminates the numerical error. This is also a nice way of verifying that the scheme conserves energy (or matter for that matter).

- Exact numerical solutions
  For the explicit schemes, exact solutions to the scheme itself can be found seeing as the scheme is a difference equation. The calculations are shown step-by-step in section 3.2.3.

- Convergence tests
  This must be combined with the manufactured solution, but takes a slightly different approach to quantifying the error estimate. We start by calculating some form of error estimate, and chose a value to represent the error of the entire simulation. This could be the maximum error for the entire simulation, or an integrated measure. Multiple simulations will be done while improving the parameter of interest, for example how the size of the time-step influences the error. Finally, a number indicating the improvement in the error estimate by improving the parameter in question is calculated. This number indicates the order of convergence which is one for FE and BE since the error goes as $\mathcal{O}(\Delta t)$, and two for our approximation to the second derivative in space since the error goes as $\mathcal{O}(\Delta x^2)$. A convergence rate of 1 means that halving the parameter will (roughly) halve the error, while the same reduction for a second order convergence will reduce the error by 4.

3.2 Verification of PDE solvers

To verify the implementation of the PDE-solvers, the steps outlined in the previous section will now be performed. The approach will be to begin the testing on the simplest diffusion equation using the full implementation, and

add complexity along the way. This is another way of verifying that the implementation is correct since the average of the anisotropic diffusion coefficient at two mesh-points is the same as the diffusion constant if there is no change over the mesh-points. Effectively this is the same as having only a constant. The same applies to setting the drift-velocity equal to zero.

Since both the FE and BE discretization have been implemented both of them will be tested, but for the most part the BE discretization will be used in simulations because of its unconditional stability. For most of the tests the exact (manufactured) solution will be equation (3.3) which satisfies the diffusion equation if the diffusion constant is 1 and $x \in [0, 1]$. In 1D the solution will be correct since $y = 0$ over the entire mesh.

$$e^{-\pi^2 t}\cos(\pi x)\cos(\pi y) + 1 \tag{3.3}$$

All of the verification on the PDEs will be done without adding random walkers. The implementation of these will be done individually, along with some tests of the combined solver in the end.

3.2.1 Manufactured Solutions

As mentioned the solution

$$u(x,t) = e^{-t\pi^2}\cos(\pi x) + 1 \tag{3.4}$$

is chosen because it fulfills the chosen boundary conditions. Equations (3.5) and (3.6) prove that the manufactured solution does indeed fulfill the diffusion equation (eq. 2.15).

$$\frac{\partial}{\partial t}e^{-t\pi^2}\cos(\pi x) + 1 = D\frac{\partial^2}{\partial x^2}e^{-t\pi^2}\cos(\pi x) + 1 \tag{3.5}$$

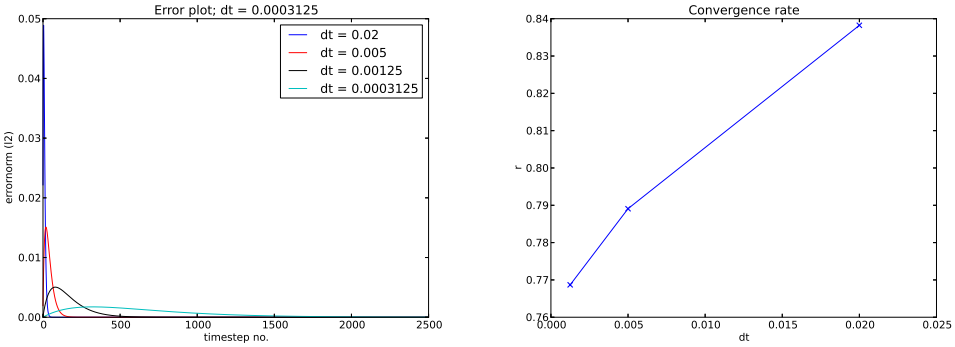
$$-\pi^2 e^{-t\pi^2}\cos(\pi x) = -\pi^2 e^{-t\pi^2}\cos(\pi x) + 1 \tag{3.6}$$

$$\implies 1 = 1$$

The error in space is determined by two factors, the actual error caused by the approximation to the second derivative, which is of the order of Δx^2 and, in the FE case, the error term coming from the time derivative due to the stability criterion (eq. 2.4.3), which is also of the order Δx^2.
Figure 3.2 shows error and convergence plots for the FE scheme in 1D. For longer simulations, the analytic solution is expected to reach a steady state which is found in the limit of large t,

$$u(x, t \to \infty) \to e^{-\infty}\cos(\pi x) + 1 \to 1 \tag{3.7}$$

The numerical scheme should be able to represent this to machine precision (10^{-16}), meaning that the numerical solution should start converging to zero after some number of times steps, but this might depend on how the derivatives as estimated so we say that it should in the very least stabilize. The error plots in Figure 3.2a clearly show that the error tends to zero as the steady state is reached. As for the convergence, it is not perfect, but it could have been worse. There might be some effects from the spatial error which influences the error, but due to the stability criterion the time step cannot be increased beyond $\frac{\Delta x^2}{2}$ which prevents isolating the error from the time derivative. Considering the tests which will be done later with respect to the numerical exact, the scheme seems to be performing fairly good, if not perfect.



(a) (b)

Figure 3.2: Numerical error for 1D Forward Euler discretization of the PDE. Nothing else is done to the simulation.

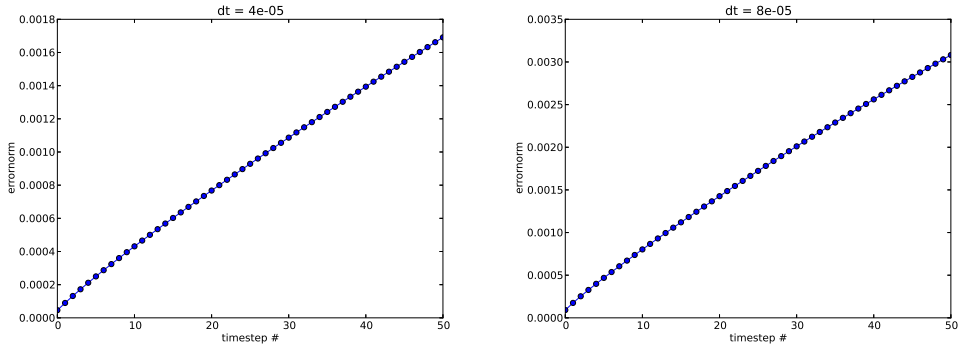
Testing of more advanced implementations requires some additional calculations in order to still have the same manufactured solution as before. The variable diffusion coefficient

$$D(x) = \pi x$$

is introduced and the drift term is still zero. The new source term required is

$$\begin{aligned} -\pi^2 \exp\left(-\pi^2 t\right) \cos\left(\pi x\right) &= -\pi \exp\left(-\pi^2 t\right) \frac{\partial}{\partial x} \pi x \sin(\pi x) + f(x,t) \\ -\pi^2 \cos\left(\pi x\right) &= -\pi^2 \left(\sin(\pi x) + \pi x \cos(\pi x)\right) + \tilde{f}(x) \\ \tilde{f}(x) &= \pi^2 \left(\sin(\pi x) + \cos(\pi x)(\pi x - 1)\right) \end{aligned}$$

where $f(x,t) = \exp(-\pi^2 t)\,\tilde{f}(x)$. Figure 3.3 shows the error norm of the result of simulations of this equation with different values of Δt.



(a) (b)

Figure 3.3: Verification of anisotropic diffusion equation implementation

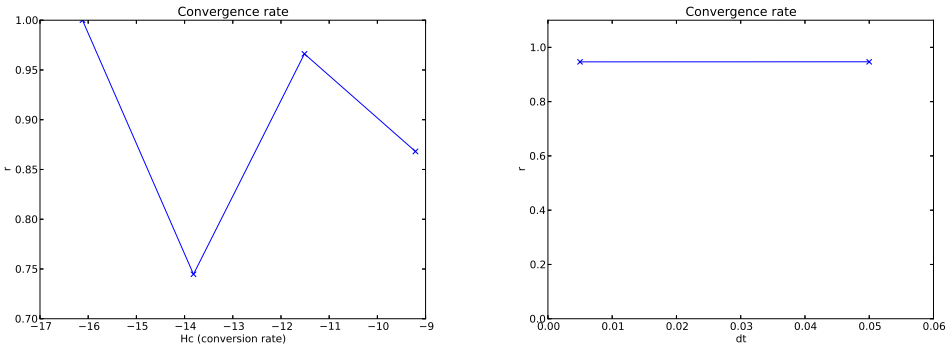
Again, the error is of the order of Δt and is roughly halved by halving Δt.

3.2.2 Convergence Tests

The next step in testing the implementation will be to perform a convergence test in time. The spatial convergence test is toned down because an incorrect implementation of the spatial derivative would be clearly visible both in the visualization of the simulation (the solution blows up), and in the error plot seeing as the spatial error would dominate. As mentioned, the convergence tests are carried out by doing several simulations with different values for Δt and comparing the errors by equation 3.8.

$$r = \frac{\ln(E_{i+1}/E_i)}{\ln(\Delta t_{i+1}/\Delta t_i)} \tag{3.8}$$

A result of such an experiment for the FE scheme using the Δt values listed below is found in Figure 3.4a. The expected value of r is approximately 1. The result is not perfect, but still close to 1. For the BE scheme we get the convergence rate shown in Figure 3.4b. Again, the expected order of convergence is 1 and this time the result is almost perfect.



(a) Convergence test for the FE scheme. (b) Convergence test for the BE scheme. The x axis is $\ln(\Delta t)$.

Figure 3.4: Convergence tests for explicit and implicit schemes solving the simple diffusion equation (eq. (2.15)).

We can also do a convergence test, equal to the one we did in 1D, to check that the scheme converges to 1 (by equation (3.8)) for smaller Δt in 2D as well. The results of this test are shown in Figure 3.5 and it does converge nicely to one.

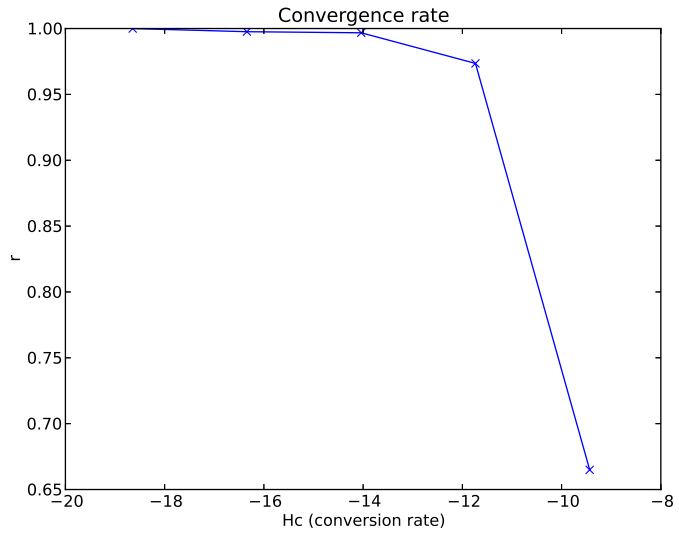
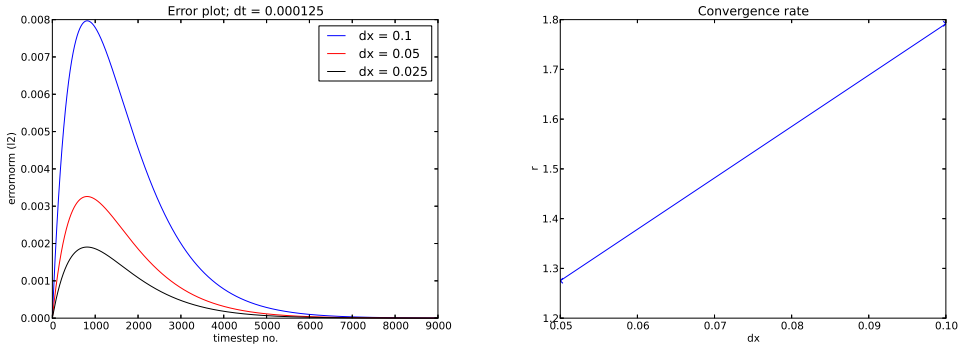


Figure 3.5: Convergence test for the FE scheme in 2D using Δt ranging from the stability criterion $\frac{\Delta x \Delta y}{5}$ to the same ratio divided by 100000 in increments of 10^{-1}.

Spatial convergence

Figure 3.6 shows the converge for the spatial error. This test is done only using the implicit scheme since it is an advantage for the spatial error, which is of second order, to be larger than the error from the time discretization. Because of the stability criterion for the FE scheme it will break down for these tests, and not give the desired results. Another complicating factor is having the time step sufficiently smaller than the spatial resolution. As Figure 3.6 shows, the test was not perfect, however it starts out with a convergence of almost 2 done on a coarser mesh. The normal error plot is also included in the figure to illustrate that the improvement is significant.



(a) Error plot (b) Convergence test

Figure 3.6: Error plot (a) and convergence test (b) focusing on the spatial error term. This test requires the time step to be much smaller than Δx^2 in order for the spatial error to dominate. As is apparent from figure b, the convergence test is difficult to get right. Notice, however that the first value in the convergence test is close to 2, and that the second one is clearly larger than 1 suggesting that other effects might be at play. The test was done in 1D using the BE discretization with $\Delta t = \frac{1}{8000}$ for all values of Δx. This value is problematic because it is only a factor $\frac{1}{5}$ smaller than Δx^2 for $\Delta x = \frac{1}{40}$ which is the last simulation.

3.2.3 Exact numerical solution

Both schemes can be reformulated as difference equations, and will therefore have some form of exact solution. Comparing a simulation to the expected exact numerical solution is a good way to verify that the implementation does what it is expected to do. The numerical exacts are expected to be represented to more or less machine precision, though some deviances might

occur.

As was done in section 2.4, the schemes must first be reformulated as a theta-rule discretization

$$\frac{u^{n+1}-u^n}{\Delta t} = \theta D\Delta t u_{xx}^{n+1} + (1-\theta) D\Delta t u_{xx}^n \tag{3.9}$$

where u_{xx}^n denotes the double derivative of u at time step t^n.

The FE scheme

Inserting $\theta = 0$ in equation (3.9) yields the FE scheme. The first few iterations are written out as

$$\begin{aligned}
u^1 &= D\Delta t u_{xx}^0 + u^0 \\
u^2 &= D\Delta t u_{xx}^1 + u^1 = D\Delta t \left[D\Delta t u_{4x}^0 + u_{2x}^0\right] + u^0 \\
&= (D\Delta t)^2 u_{4x}^0 + 2D\Delta t u_{2x}^0 + u^0 \\
u^3 &= D\Delta t u_{xx}^2 + u^2 = D\Delta t \left[(D\Delta t)^2 u_{6x}^0 + 2D\Delta t u_{4x}^0 + u_{2x}^0\right] + (D\Delta t)^2 u_{4x}^0 + 2D\Delta t u_{2x}^0 + u^0 \\
&= (D\Delta t)^3 u_{6x}^0 + 3(D\Delta t)^2 u_{4x}^0 + 3D\Delta t u_{2x}^0 + u^0 \\
u^4 &= D\Delta t u_{xx}^3 + u^3 = \ldots \\
&= (D\Delta t)^4 u_{8x}^0 + 4(D\Delta t)^3 u_{6x}^0 + 6(D\Delta t)^2 u_{4x}^0 + 4D\Delta t u_{2x}^0 + u^0
\end{aligned}$$

This can be generalized to

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i u_{2ix}^0 \tag{3.10}$$

where the initial condition is known from equation (3.4)

$$u_{2ix}^0 = (-1)^i \pi^{2i} \cos(\pi x)$$

Inserting the initial condition results in the exact solution to the FE scheme

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} \pi^{2i} (-1)^i (D\Delta t)^i \cos(\pi x) \tag{3.11}$$

Notice that the analytical spatial derivative has been used rather than the approximation to the spatial derivative. In the same way as for the time

derivative, the approximation to the spatial derivative must be inserted.

$$\begin{aligned}
u^0_{xx} &= \frac{1}{\Delta x^2}\left(\cos(\pi(x+\Delta x)) - 2\cos(\pi x) + \cos(\pi(x-\Delta x))\right) \\
&= \frac{2}{\Delta x^2}\left(\cos(\pi\Delta x) - 1\right)\cos(\pi x) \\
u^0_{4x} &= [u^0_{xx}]_{xx}\frac{1}{\Delta x^2}\left[\frac{u^0_{xx}}{\cos(\pi x)}\left(\cos(\pi(x+\Delta x)) - 2\cos(\pi x) + \cos(\pi(x-\Delta x))\right)\right] \\
&= \frac{4}{\Delta x^2}\left(\cos(\pi\Delta x) - 1\right)^2\cos(\pi x) \\
&\ldots
\end{aligned}$$

This pattern continues, and the exact numerical solution can be expressed by equation (3.12) in 1D for the previously mentioned initial condition.

$$u^{n+1} = \sum_{i=0}^{n}\binom{n}{i}\left(D\Delta t\right)^i\frac{2^i}{\Delta x^{2i}}\left(\cos(\pi\Delta x) - 1\right)^i\cos(\pi x) \tag{3.12}$$

The FE scheme is expected to represent this solution more or less to machine precision, at least to 15 digits. There are, however two issues with the solution (3.12):

- Δx^{2i} will quickly tend to zero, and the computer will interpret it as zero. This will cause division by zero, which again results in "Not a number" (nan) and ruins the simulation. This can be fixed rather simply by testing if $\Delta x^{2i} > 0$ and returning zero if the test returns false.
- $\binom{n}{i}$ goes to infinity for large n and i. The computer can only represent numbers up to $\sim 10^{308}$, which limits the number of steps to 170 since $n! > 10^{308}$ for $n > 170$. The argumentation for dropping the troublesome terms is given below.

As a side note, equation (3.12) illustrates how the stability criterion (eq. 2.4.3) comes into place. The solution used in the derivation of the stability criterion assumes an amplification factor A^n to replace the exponential amplification in the actual solution. This amplification factor can be found in equation (3.12) as

$$\left(\frac{2D\Delta t}{\Delta x^2}\right)^i$$

Inserting a time step larger than the stability criterion ($\Delta t \leq \frac{\Delta x^2}{2D}$) will make the amplification factor A larger than one which in turn will make the solution blow up. This also illustrates why the terms where

$$\frac{1}{\Delta x^{2i}} \to \infty$$

can be dropped. By the stability criterion, the time step will cancel out Δx^2, and the result will be some number smaller than 1 raised to a rather large power, i, resulting in something comparable to zero.

The results from testing the FE scheme are found in Figure 3.7. We see that the error in the worst case is about an order of magnitude worse than expected. This is most likely due to the fact that we are cutting part of the solution, and over several time steps the error we do might accumulate. This is most likely a

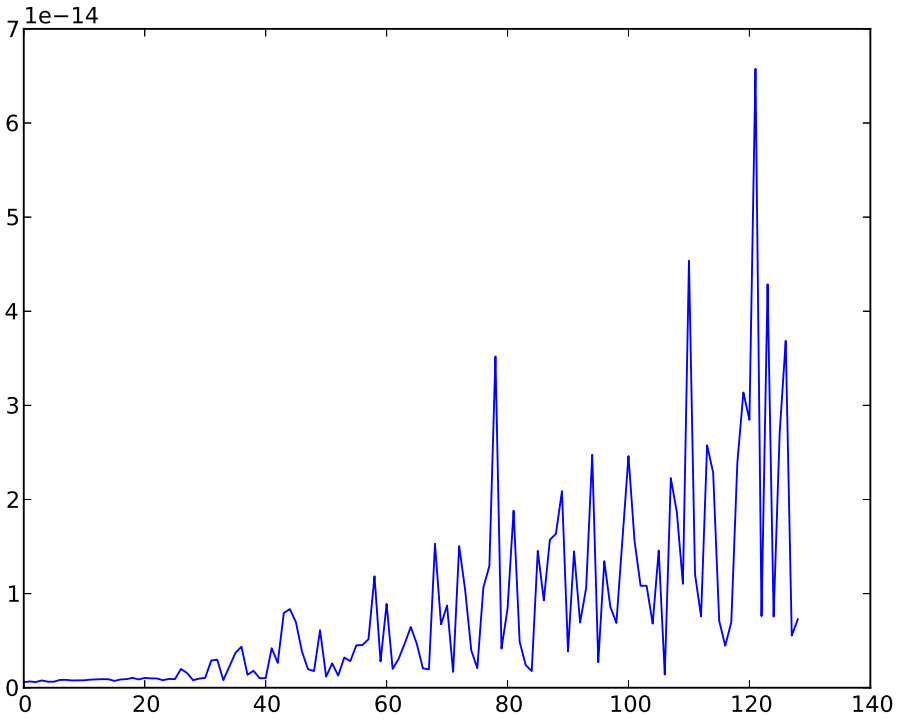


Figure 3.7: Error plot for 1d FE scheme compared to the exact numerical solution 3.12 with a few modifications, like ignoring terms where Δx is truncated to zero, added.

The exact numerical solution to the 2d diffusion equaiton with eq 3.13 as initial condition is found by using the same method as for the 1D case. It is listed in equation (3.14), and the FE scheme is expected to reproduce this

to more or less machine precision as well.

$$u(x, y, t = 0) = \cos(\pi x)\cos(\pi y) \tag{3.13}$$

The same problems with truncation of small values apply to a larger degree than in the 1D case, however and as a result some accumulation of error terms might be expected.

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \left[2^{i-1} \cos(\pi x)\cos(\pi y) \left(\frac{(\cos(\pi \Delta x))^i}{\Delta x^{2i}} + \frac{(\cos(\pi \Delta y))^i}{\Delta y^{2i}} \right) \right] \tag{3.14}$$

The result of a test simulation of this is shown in Figure 3.8. Again, as was the case in 1D, the error is very small and starts out at machine precision. It does, however increase with time as the truncated terms begin to accumulate, and even more so than in the 1D case. We should, in other words, be pleased that the error starts out with machine precision, and stays small for the amount of time steps we can simulate and still have something to compare it with.

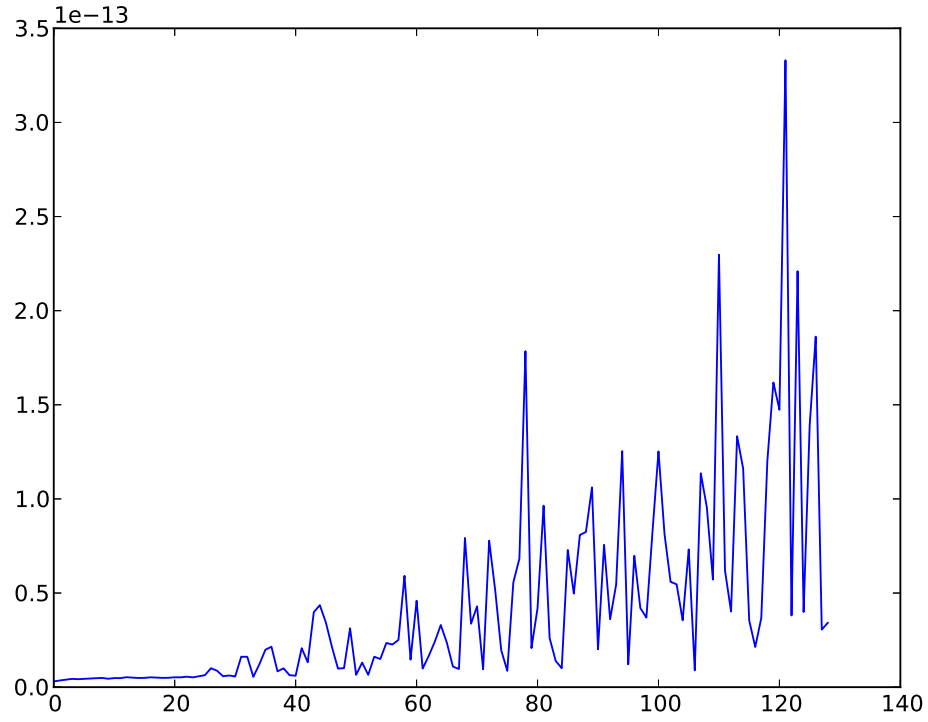


Figure 3.8: Numerical solution from the FE scheme versus the exact numerical solution of the FE scheme in 2D. Parameters of importance are Δt which is almost on the stability criterion, $\Delta t = \frac{\Delta x \Delta y}{5} = 8 \cdot 10^{-5}$ and the diffusion coefficient D which must be $D = \frac{1}{2}$ in order to fulfill the diffusion equation.

The BE scheme

For the BE scheme the exact numerical solution is more implicit than for the FE scheme. As for the FE scheme, the numerical exact solution is simply the solution to the difference equation that the PDE is rewritten as. However, the BE scheme is implicit and results in a system of linear equations. This was derived in section 2.4.2 and the resulting linear system is

$$\mathbf{M}\mathbf{u}^{n+1} = \mathbf{u}^n$$

The solution to this linear system is trivial

$$\mathbf{u}^{n+1} = \mathbf{M}^{-1}\mathbf{u}^n$$

though it is an inefficient way of solving the system in this case. As indicated in section 2.4.2 all the information about the system lies in the

mass matrix $\mathbf{M}$, and this matrix must be carefully assembled. It is then preconditioned and the system is solved by the "tridiag" solver mentioned earlier. In principle this is the same as doing and storing the inverse of $\mathbf{M}$ and then multiplying this inverse by the solution at the previous time-step. Another way of doing this is by raising $\mathbf{M}$ to the required power and then doing the multiplication with the initial condition as stated in equation (3.15).

$$\mathbf{u}^{n+1} = \left(\mathbf{M}^{-1}\right)^{n+1} \mathbf{u}^0 \tag{3.15}$$

Equation (3.15) is the numerical exact solution to the BE scheme, but it has a few problems compared to the exact solution to the FE scheme. Testing is done by storing the inverse of the assembled matrix, solving eq. (3.15) for the desired number of steps and comparing the solution with the result of a simulation. The problem with this approach is that the inverse of $\mathbf{M}$ gives a lot of round off errors, especially with entries of 10^{-20} and smaller. These entries would be zero if the analytical inverse was taken, but are clearly not zero in the matrix. Terms of this magnitude give rise to a lot of potential uncertainty, and the result is that the desired accuracy must be reduced. Figure 3.9 shows the accuracy of the simulation compared to the result of a simulation with the BE discretization. Although the accuracy is some 5 orders of magnitude worse than the numerical exact for the FE scheme, it is still not too bad.

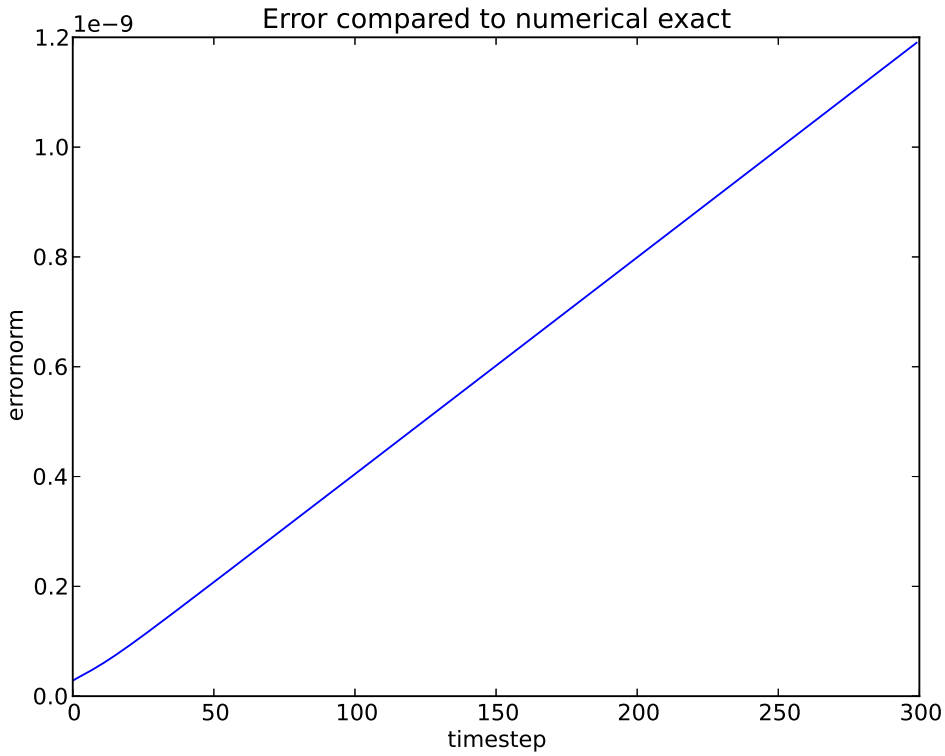


Figure 3.9: Error plot showing the norm of the difference between the numerical exact solution for the BE scheme in 1D (eq. 3.15) and a simulation. The error is not machine precision, but significantly smaller than Δt which for this simulation is $\Delta t = 0.01$. This increased error most likely originates in the many roundoff errors in the inverted matrix where a lot of terms which analytically would be zero are represented as 10^{-16} and smaller.

A nice property of the numerical exact solution to the BE scheme is that it generalizes to the 2D scheme. Figure 3.10 shows the accuracy of the BE scheme when compared to its numerical exact solution. As for the 1D case, there will be some round-off errors which will have an effect on the accuracy.

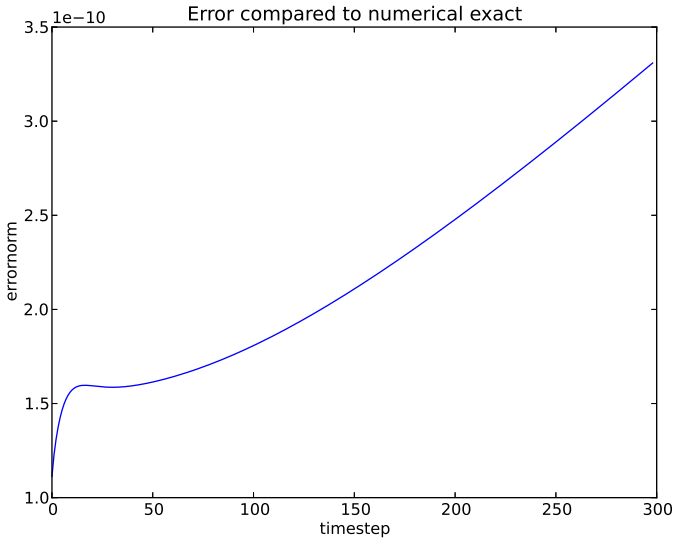


Figure 3.10: Error plot showing the norm of the difference between the numerical exact solution for the BE scheme in 2D (eq. 3.15) and a simulation. The error is not machine precision, but significantly smaller than Δt which for this simulation is $\Delta t = 0.01$. This increased error most likely originates in the many roundoff errors in the inverted matrix where a lot of terms which analytically would be zero are represented as 10^{-16} and smaller.

3.3 Testing the Random walk implementation

The RW implementation will be verified using the same techniques that were used to verify the PDE solvers with the exception of a numerical exact solution since there is none. However, the initial condition used for the PDE solvers cannot be represented to a satisfying accuracy by a distribution of walkers. A Heaviside step function on the other hand can be perfectly represented by said distribution and is defined in equation (3.16).

$$H(x-a) = \begin{cases} 1 & x \geq a \\ 0 & x < a \end{cases} \tag{3.16}$$

In order to verify the RW implementation an initial distribution of walkers which follows the Heaviside step function is given to the program, and a simulation is run for some number of time steps. An exact solution must also be found to the diffusion equation (eq. 2.15) for error calculations and this is done by separation of variables. We have

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{3.17}$$

$$\frac{\partial u(0,t)}{\partial x} = \frac{\partial u(1,t)}{\partial x} = 0 \tag{3.18}$$

$$u(x,0) = H\left(x - \frac{1}{2}\right) \tag{3.19}$$

$$D = 1 \tag{3.20}$$

and

$$u(x,t) = F(x)T(t) \implies \frac{T'(t)}{T(t)} = \frac{F''(x)}{F(x)}$$

where the primes denotes the respective derivatives. We separate the equation using a separation constant λ

$$\begin{aligned} T'(t) - \lambda T(t) &= 0 \\ \implies T(t) &= C\exp(\lambda t) \\ F''(x) - \lambda F(x) &= 0 \\ \implies F(x) &= C_1\exp(\sqrt{\lambda}x) + C_2\exp(-\sqrt{\lambda}x) \end{aligned}$$

where C, C_1 and C_2 are arbitrary constants. Choosing $\lambda = -\mu^2$ lets us rewrite the spatial solution in terms of sines and cosines. There are really three choices here; $\lambda = -\mu^2$, $\lambda = \mu^2$ and $\lambda = 0$, but the first is chosen because the results of the other choices are unphysical or uninteresting. Inserting $\lambda = -\mu^2$ into the expression found for $F(x)$ gives

$$F(x) = a\cos(\mu x) + b\sin(\mu x)$$

The boundary conditions result in

$$F'(0)T(t) = F'(1)T(t) = 0$$

Since the time dependent solution cannot be exactly zero and is independent of position by construction ($C\exp(\lambda t)|_{x=0} = C\exp(\lambda t) \neq 0$), the first derivative of the spatial solution must be zero at the boundaries

$$\begin{aligned} F'(x) &= -a\mu\sin(\mu x) + b\mu\cos(\mu x) \\ F'(0) &= -a\mu\sin(0) + b\mu\cos(\mu x) \\ &= b\mu\cos(\mu x) \implies b = 0 \\ F(1) &= a\cos(\mu) \implies \mu = n\pi \end{aligned}$$

Which suggests that a Fourier series in cosines is the solution to the equation, and it will look like this.

$$u(x,t) = a_0 + \sum_{n=1}^{\infty} a_n \exp\left(-(n\pi)^2 t\right) \cos(n\pi x) \tag{3.21}$$

The coefficients are found by approximating the initial condition

$$\begin{aligned}
a_0 &= \int_0^1 H(x-0.5)dx = \frac{1}{2} \\
a_n &= 2\int_0^1 H(x-0.5)\cos(n\pi x)dx \\
&= 2\int_{0.5}^1 \cos(n\pi x)dx \\
&= \frac{2}{n\pi}\left[sin(n\pi x)\right]_{0.5}^1 \\
&= \frac{2}{n\pi}\sin(n\pi) - \sin(\frac{n\pi}{2}) \\
&= \frac{2\sin(\frac{n\pi}{2})}{n\pi} \\
a_n &= \frac{2}{\pi m}(-1)^m \;\; ; \;\; m = 2n+1
\end{aligned}$$

which gives us the final solution

$$u(x,t) = \frac{1}{2} + \sum_{m=0}^{\infty} \frac{2(-1)^m}{m\pi} e^{-(m\pi)^2 t} \cos(m\pi x) \tag{3.22}$$

This is the manufactured solution the simulations will be tested against for the verification of the RW implementation.

The convergence test shown in Figure 3.12 suggests that the convergence rate for random walks follows the proportionality in equation (3.23).

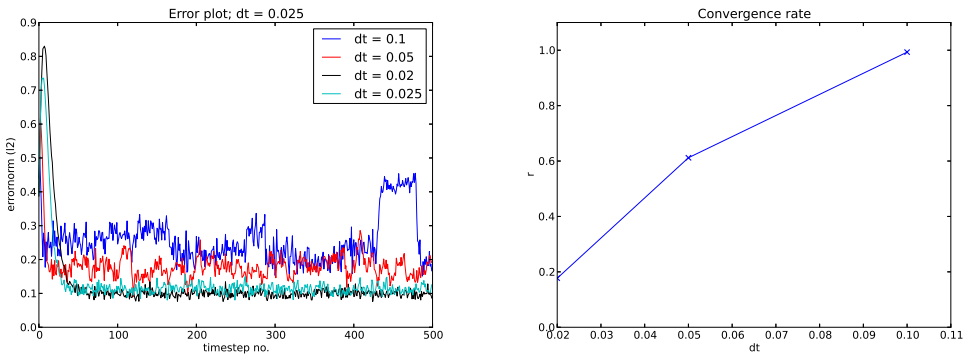
$$\epsilon \propto Hc^{\frac{-1}{2}} \tag{3.23}$$

This relation says that while increasing the number of walkers will in fact reduce the error, the convergence is very slow. Should we wish to do so, we can force the error to $\mathcal{O}(\Delta t^2)$, but this will be extremely inefficient. In fact we can find the relation as $Hc \sim \Delta t^{-2}$ for $\epsilon \sim \mathcal{O}(\Delta t)$, and $Hc \sim \Delta t^{-4}$ for

$\epsilon \sim \mathcal{O}(\Delta t^2)$.
Clearly there will be enough trouble getting the scheme to recreate first order convergence where a relatively "kind" simulation which fulfills the stability criterion for the FE scheme using a spatial resolution of $\Delta x = \frac{1}{10}$ demands

$$Hc = \frac{1}{\Delta t^2} \geq \frac{2D}{\Delta x^4} = 2D \cdot 10^4 \tag{3.24}$$

walkers per unit per mesh point. Simply improving the spatial resolution by a factor of 10 will increase the demand for walkers by a factor of 10^4



(a) (b)

Figure 3.11: Error plot (a) and convergence test (b) for 1D RW solver using a Heaviside step function as initial condition. In these tests both the time step and the conversion factor are changed for each simulation, and the conversion factor follows the previously proposed limit $Hc \geq \frac{1}{\Delta t^2}$. For each Δt the RW simulation does 250 steps with a step length calculated from equation (2.51). The expected convergence rate is 0.5, and to some extent this is achieved here, note however that due to fluctuations in the solution getting a good error measure is difficult and beyond our control.

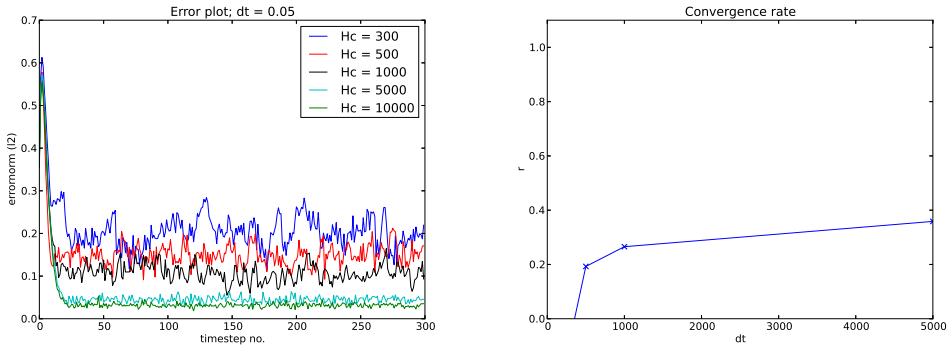


Figure 3.12: As a comparison to Figures 3.11 this test has been done for the same 1D Heaviside step function as initial condition, but keeps the time step fixed at $\Delta t = 0.05$ and increases the conversion factor. The convergence rate (fig. b) is worse than for the time convergence test, and seems to reach a limit where increasing the number of walkers has little effect on the error.

3.4 Testing the combined solution

This section will test how combining the RW model with the PDE model affects the error, and whether it is possible to make the error term have first order convergence. As will be discussed in chapter 4.1 there are many candidates as to the combination of the solutions, but it turns out that simply replacing the solution in the relevant area with the solution from a RW simulation before the PDE integration is done is sufficient.

> A scientific theory should be as simple as possible, but no simpler

It will therefore be unnecessary to add any fancy curve fitting when it is not needed.

Before we start with the verification, however, a simplified problem which in principle is the same as the diffusion - RW problem, will be presented.

3.4.1 A simplified version of the algorithm

Monte Carlo (MC) methods are immensely important in modern computational science (*reference*), and can be used to solve integrals as well as random walks. As a simplified analogy to our method for solving the diffusion equation we ca look to the Ordinary Differential Equation (ODE) in equation

(3.25).

$$\frac{\partial f}{\partial x} = g(x) \ , \ x \in [a, c] \tag{3.25}$$

Equation (3.25) is easily solvable (see eq. (3.26)). For the sake of illustration we also specify that $g(x) = \frac{1}{x}$ and divide the integral in two parts, introducing $b \in (a, c)$.

$$f(x) = \int_a^b g(x)\, dx + \int_b^c g(x)\, dx \tag{3.26}$$

This is a case where we have complete control over all parts of the solution which is $f(x) = f(b) - f(a) + f(c) - f(b)$, and we can solve the two parts of the integral in two different ways; by the midpoint method and by MC integration, respectively. The convergence rates of these methods are 2 and 0.5 respectively. By the relation found in chapter 2.5.1 (eq. (2.46)) the number of MC samples should be proportionate to the resolution used by the midpoint-rule to the power of four.

$$\frac{1}{\sqrt{N}} \simeq \Delta x^2$$
$$N \simeq \frac{1}{\Delta x^4} = N_x^4$$

The following output is from a program (donated) by Hans Petter Langtangen which does the required integration and calculates the convergence rates. It uses the relation described, but multiplies with a constant, giving us

$$N = 2000 \times N_x^4 \tag{3.27}$$

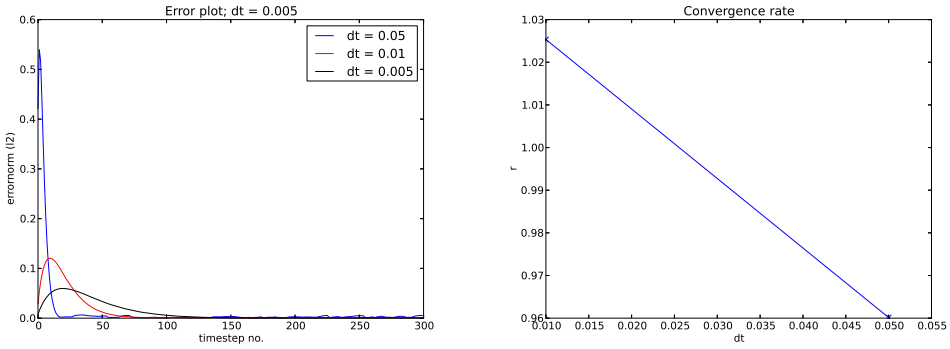
```
  N_x       N_MC            error        MC_error    MP_error
   1        2000 (    1)  2.650E-02   2.648E-02   2.391E-05
   2       32000 (    1)  7.392E-03   7.433E-03  -4.136E-05
   4      512000 (    1)  1.918E-03   1.927E-03  -8.954E-06
   8     8192000 (    8)  4.683E-04   4.866E-04  -1.832E-05
  16   131072000 (  131)  1.176E-04   1.220E-04  -4.320E-06

  Convergence rates
total    MP    MC
-1.84  -1.83  0.20
-1.95  -1.95 -0.55
-2.03  -1.99  0.26
-1.99  -2.00 -0.52
```

The convergence rate for the whole integral is roughly 2 which is what we expect. This suggests that the idea behind the algorithm is sound.
Another thing to notice is the convergence rate of the MC method which is sort off all over the place, this illustrates how difficult it is to verify the simulations. As the listed output and equation (3.27) shows, the number of walkers or MC samples grows very fast making it computationally very demanding to do the calculations.

3.4.2 Introducing walkers

First of all, using random walkers on parts of the mesh will have a considerable, negative impact on the error estimate. As we have discussed before, the solution from the random walkers will fluctuate around the "correct" (it is in fact correct while verifying) solution with amplitude proportional to $\frac{1}{\sqrt{N_{ij}}}$ which will depend on the PDE-solution in the mesh-point. It will also, as demonstrated in chapter 3.4.1, be possible to force the combined solution to have the desired properties in terms of error-estimates but at considerable computational cost. The various error tests will therefore be carried out using only the implicit scheme since the time-step can be chosen more freely, thus reducing the required number of walkers.
Keeping in mind that the spatial error goes like Δx^2 the time-step should be chosen so that $\Delta t > \Delta x^2$ to make sure the error from the time derivative is dominant. Figure 3.13 shows the results of a test where Δx was fixed at $\frac{1}{100}$ and both the time step and the conversion rate for walkers were improved over three simulations. The figure shows that most of the fluctuations are irrelevant in the beginning of the simulation, but the become increasingly more important as steady state is reached. A convergence rate of 1 is also

reached, suggesting that the walkers are converging to the PDE solution.



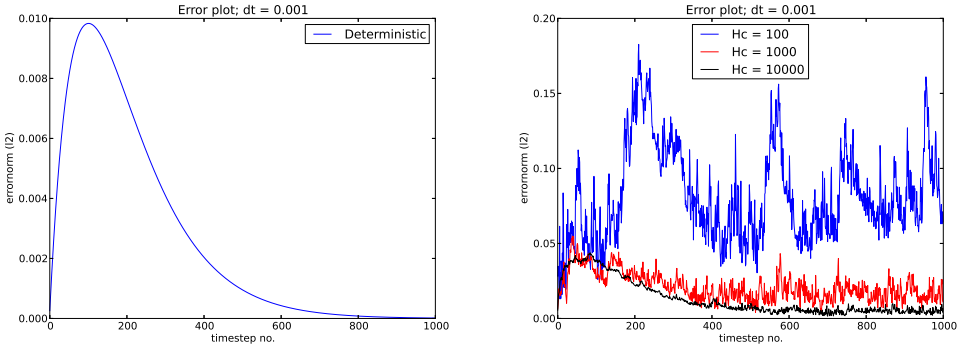
(a) (b)

Figure 3.13: 3.13a shows the error plot for a test where Δx was fixed at $\Delta x = \frac{1}{100}$ and Δt was reduced from 0.05 to 0.01 and finally to 0.005. The conversion rate, Hc was updated for each simulation to have the value $Hc = \frac{1}{\Delta t^2}$, meaning the error from the walkers should be smaller than the error from the time derivative. Walkers are placed on 10% of the mesh from $x = 0.4$ to $x = 0.5$. 3.13b shows the convergence rate in time for the same test.

3.4.3 Increasing the time step and the relative size of walk-area

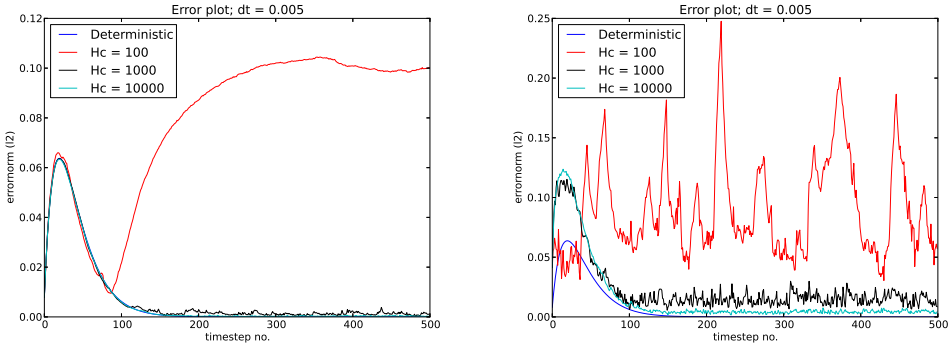
Now that we have an estimate of how to adjust the step length of the walkers in order to adjust for the time step, Δt, on the PDE level we would like to investigate the actual effects of running the simulation with a larger time step to verify our calculations. First off all, Figure 3.14a shows the error norm of a simulation of the simplest diffusion equation (2.15) discretized by the BE scheme using a time step which would make the FE discretization unstable (There is something strange about its convergence). Figure 3.14b shows the same simulation for various conversion parameters for the random walk. These simulations have input from the random walk model on some 20% of the mesh points. As a comparison Figures 3.15a and 3.15b have 5% and 35% of the mesh points affected by walkers. An interesting property of both these figures is the instability in the error for very low conversion factors. *There seems to be a limit as to how large of an inaccuracy the scheme can handle and still produce meaningful results.* Another thing to notice from these figures is that a larger relative area of walkers implies more walkers are required in order to make the scheme as exact as it will get. Apparently, the

introduction of walkers will give an increased error no matter how many are used.



(a) (b)

Figure 3.14: Error for 1D BE scheme combined with RW solver using increasing number of walkers, but at most 1% of the required number. In figure b there has been added walkers to the solution in the area $x \in [0.5, 0.7]$ with $\Delta x = \frac{1}{50}$, which adds up to 20% of the mesh. Compared to only the deterministic error in a, the 1% simulation is not that bad.



(a) Having walkers on 5% of the mesh points.

(b) Having walkers on 35% of the mesh points.

Figure 3.15: The effect of increasing the size of the walk area for a fixed $\Delta t = 0.05$ and $\Delta x = 0.01$ using the BE discretization.

The effects of changing the time step have also been investigated, and the results are shown in Figure 3.16. This test illustrates that the step length tweaking derived in section 2.5.5 works since the error stabilizes around the

size of Δt and the fluctuations seem to be around $\sqrt{\frac{1}{N}}$ as predicted.

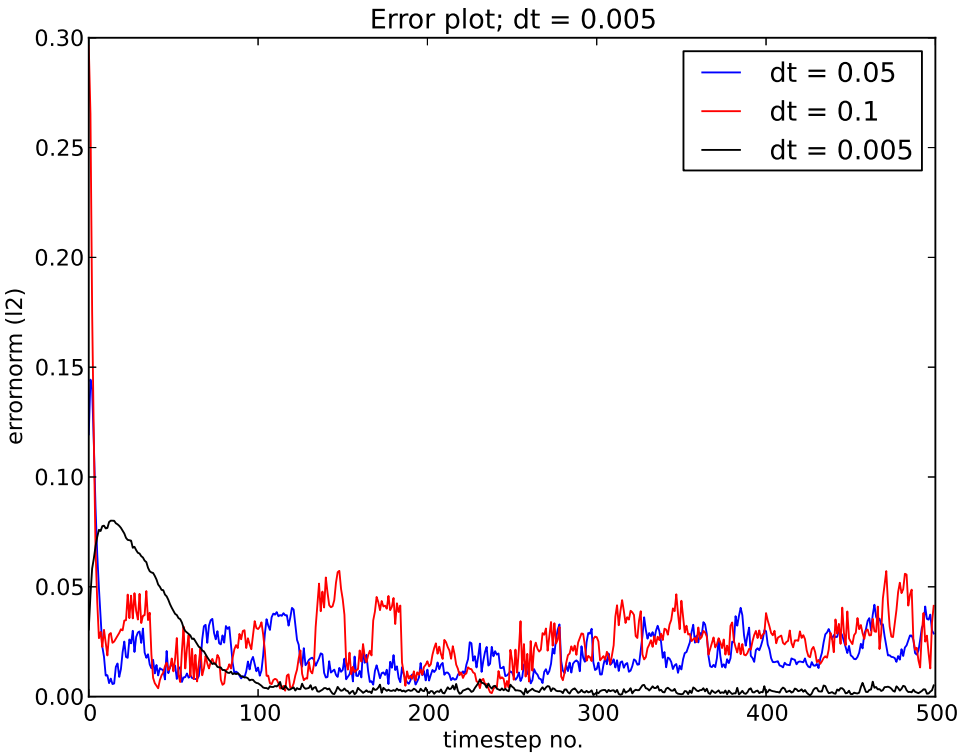


Figure 3.16: Error norm of three tests where the spatial resolution was fixed at $\Delta x = \frac{1}{10}$ and the time step and conversion factor were changed. Δt started at the stability criterion for the FE scheme $\Delta t = 0.5 \cdot \Delta x^2$ and was increased up to $\Delta t = \Delta x$, maintaining the demand of $Hc = \frac{1}{\Delta t^2}$ for all the tests.

Chapter 4

Software

4.1 About

The current version of the software (14.02.14) works in the following way. Various parameters are specified in a python-script which calls the program. After initialization, where anisotropic diffusion constant, initial condition and areas of combined solution are set up we start solving the equation (using the implicit BE scheme). At each time-step we call the Solve-method of the combine-class which in turn calls the solve-method of the PDE-solver, calculates the number of walkers by the conversion-rate, gives them a random position in proximity to the coarse-grained PDE-mesh-point we are looking at, and writes this position to a file. The random walk program is then called with some parameters. This program reads the file with walker-positions, advances some specified number of time-steps and writes the final positions to the same file it read. The main program now reads this file, maps the walker-positions back to the coarse-grained PDE-mesh and converts the spatial distribution of walkers to a concentration distribution. Here we are faced with a few choices with respect to the combination of the two solutions we have for the same area:

- Some form of least squares fitting could be done. A polynomial regression was developed, but tests indicate that it is not a good solution.
- Cubic spline interpolation might be slightly better seeing as we force the derivatives to be equal at the end-points. Interpolation forces the fitted curve to take the measured values at the interpolation-points, and so we must chose only some of the points to be used by the interpolation in order for there to be a difference. Immediately we are faced with the problem of which points to use and which to throw away. If we consistently chose the points which are closest to the PDE-solution (and the end-points which must be included for smoothness) we might as well not include a random walk model, and vice versa.
- We could use only the result from the random walk model
- Some sort of average might also help us. There are many to chose from, both arithmetic and geometric and with different kinds of weighting.

Intuitively, I find it better to use either an average or only the random walk result. At present (14.02.14) the simplest arithmetic mean is implemented at each mesh-point.
After each time-step the program also writes the combined solution on the coarse-grained mesh to a unique file.

The python script also does some other more or less clever stuff at each run. This is described in the appendix as a form of debugging A.2.

4.1.1 Limitations

As with any software there are limitations. The limitations discussed in this chapter will be with regard to physical problems, and not memory or CPU limitations which are described later.
Assuming the dendritic spines stuff will be the final application:
First of all, to specify a source function, one must program it in the Diffusion class (Diffusion::f()). This might be fixed in a later version using inheritance.
The geometry is, currently, limited to a square. Of course we can argue that we have scaled the size to a unit square, but as discussed in chapter 2.6 the actual geometry remains a unit square. Furthermore the same issue arises on the random walks.
As of now, it is unclear if 3D-modeling is supported. This is actually a smaller issue since there is usually little more to learn from switching from 2D to 3D. In any case, the extension to 3D is most likely not very hard, provided we use an array of slices from the cubic matrix describing the solution.

4.2 Adaptivity

There are two adaptive parts of the software. First of all, the number of walkers which depend on the concentration, or the solution the the PDE in the relevant area. This must change in order to keep physical meaning and give results. Without this adaptivity, the results would either be wrong, or the model would not make physical sense.
Since a diffusion process in general has rapid changes in the beginning where for example high frequency variations are dampened and very slow convergence to a steady state later, we have introduced a test of the amount of change between two subsequent time-steps. If this amount is smaller than some limit, we will increase the time-step.
This increase should be done in a more elegant manner(linearly?)

4.3 Computational cost

This chapter will consider the expensive parts of the code and look at possible improvements.

4.3.1 Memory

The memory-expensive parts of the code include storing the decomposed matrices, and storing the random walkers. None of these pose any problems.

4.3.2 CPU time

The program as it stands now (v 1.∞) uses the BE discretization and a highly specialized tridiagonal solver. The random walk-part of the software has been excreted as a stand-alone program which communicates with the main-program through a binary .xyz file containing the positions of all walkers in 3d space. This makes it easier to change solver, and implement more advanced solvers like the Direct Simulation Monte Carlo (DSMC) Molecular Dynamics (MD) simulator written by Anders Hafreager (see chapter 7.3.2). There are three expensive operations in the algorithm as it stands now with the BE discretization using standard LU-decomposition.

- Random walks are expensive if there are many walkers. The number of calls to the random number generator follows eq. (4.1) and for the verification process, which required a lot of walkers, this represented a considerable cost.

$$N_{\text{calls}} \propto Hc\frac{(x_1 - x_0)}{\Delta x}\frac{(y_1 - y_0)}{\Delta y}\tilde{T} \tag{4.1}$$

  where $\tilde{T}$ denotes the number of time steps on the PDE level times the number of time-steps one PDE-step corresponds to on the RW level. Note that this expression will NOT be zero in 1D, and that it is dependent on the PDE-solution. As an additional problem we will encounter some overhead upon calling the program, initializing variables and instances and so on. The computationally most demanding function seems to be the round-off function, which is used to place the walkers on the coarse-grained PDE-mesh.

- Communicating the positions of all the walkers between the two programs each time-step is very costly.

- Translating the positions of the walkers from the unit-square they are walking on to the coarse-grained PDE grid requires calling the "round" function from the math library in C++. This function is rather slow, and the program suffers from it.

4.3.3 Parallelizability

In the final algorithm there are the following stages

- Initialization
  Read parameters from command-line, initial condition and diffusion "tensor" from file. Setup instances of solvers etc. Practically no point in parallelizing this.

- Forward sweep
  As is explained in section 2.4.5 the solution is done by a forward and a backward sweep of the mass matrix. Since the forward sweep is both the most computationally intensive and independent of time it can be done just once and stored for future use. The forward sweep consists of two matrix-matrix multiplications and one matrix inversion, all of which require some $\mathcal{O}(N^3)$ FLOPs depending on what algorithm is used. N is here the size of the block matrix in the mass matrix, typically n^{d-1} where n is the spatial resolution and d is the spatial dimension.

- Solving
  This step includes a backward sweep of the mass matrix, only consisting of matrix-vector multiplications and additions which are not computationally intensive. It also includes the random walk part which is both expensive (depending on the number of walkers left) and highly parallelizable. We also write stuff to file which is quite costly. This is probably not possible to parallelize.

Parallelization of the random walk solver should scale linearly because the only form of communication required is shared memory. Solving the linear system might be parallelizable, but will not scale linearly due to communication. This step is not the bottleneck at this point.

4.3.4 Some fancy title about changing stuff

It should be rather simple to replace parts of the code as long as certain conditions are met. Perhaps the simplest part to replace is the random walk part. Requirements for this part are:

- Locating executable main-file of the program in the folder "stochastic" and naming it "walk_solver".

- This executable should read the filename of the ini-file containing the positions of all walkers , and the local diffusion "tensor" in the relevant area.

- Upon completion, all positions must be written to the same file.

The PDE-solver should also be rather simple to replace, but some more programming will be required. First of all your solver must be included in the header-file. Next, the "Combine" class has an instance of the PDE-solver which it calls the advance-method of at each time-step. Seeing as this method is most regularly named "solve" you will either have to rename the method or the call. There are really very few dependencies on the PDE-solver, seeing as it is mostly left alone, but in addition to being able to respond to function calls it must:

- Have its own Δt attribute named "dt".
- Work on "double**" data types for all spatial dimensions (or implement some form of workaround)
- Be able to respond to increasing the time-step. In practice this means that the solver should be implicit.

As discussed, the implementation of random walks on parts of the mesh will reduce the convergence-rate to 0.5, and so there are really only two reasons to implement a new PDE solver. The current one only implements Neumann boundaries, and consequently must either be modified or replaced in order to work with other boundary conditions. It also only works on a square mesh. As discussed in section 2.6 it will be immensely complicated to implement a grid transformation, and this is already implemented in most finite element software.

4.4 Numerical model

The numerical model of the PKCγ diffusion problem is implemented as follows. In the same way as Craske et.al [2] a section of dendrite will be modeled by one-dimensional diffusion. This dendrite section is thought to be in contact with the cell soma, and to have some spines on it. The soma is modeled as a source at the one end of the dendrite section, whereas the other end of the dendrite section is thought to continue branching into narrower and narrower dendrite branches. The branching and the rest of the dendrite will not be modeled in the beginning at least, because the situation in thin branches is thought to be the same as in thick branches with a time-delay and some effects due to a higher surface to volume ratio, all of which are described by Craske et.al. Note that although the diffusion coefficient of PKCγ is known, the actual diffusion from the cell soma and into the dendrite is much faster

because of membrane binding. This effect becomes obvious when studying the videos referred to by Craske et.al. in which a concentration increase some 40μm from the cell soma takes only a few seconds bu would have taken several hours in a normal diffusion process with the respective diffusion coefficient. The simulations take this into account by constructing a random, small concentration over the entire dendrite and a larger peak close to soma.

Dendritic spines are modeled as a two-dimensional funnel attached to the dendrite at some number of mesh-point which must correspond to the typical range of spine-neck diameters. This limits the spatial resolution by demanding the narrowest spine necks to connect at two mesh-points (in order for the spine to be two-dimensional). The other parameters on a spine are set randomly, but chosen so they correspond to values reported by Arellano [1].

The model can without very much effort be extended to model an entire neuron with a dendrite tree (some modification to the dendrite class is required in order to add a dendrite to an existing dendrite) by making a neuron class containing a dendrite tree linked list and possibly a soma object should one wish to model the soma individually. Other possible extensions include adding and removing spines according to the rules proposed by (*artikkelen Gaute snakket om*) in order to investigate how this affects various diffusion processes in dendrites or dendrite trees. Recall that the possibility to add spikes in spines is already built-in.

4.4.1 Parameters

As is the general problem in computational neuroscience this project requires setting a lot of parameters whose values are not known. There are of course a lot of parameters with known values as well, like the diffusion coefficient for PKCγ. A summary of the parameters of particular interest is shown in Table 4.1, with an indication of where the values are taken from, if they are calculated or if they are simply given a value to ensure reasonable behaviour.

Parameter	**Explanation**	**Expression/ typical value**	**Origin**
Δt	time-step	Δx^2	stability criterion (sect. 2.4.3)
Δx	spatial resolution	$\frac{1}{2}$min(spine neck diameter)	estimated
Hc	conversion factor	5-24	estimated by calculations of concentration levels taken from [7] and spine/dendrite volume ratios. See later for discussion.
$u(t=0)$	initial condition value	$5\frac{\text{nMol}}{\text{L}}$	estimated from values found in [7]
p_{ds}	probability to diffuse into a spine	$0.1 \cdot \Delta x \cdot \Delta t$	estimated. An important ability of this parameter should be that wide necked spines have larger probability and that a certain flux is maintained (on average), meaning that the flux should be independent of Δt
p_{ab}	probability for PKCγ to be absorbed, and removed from simulation, per time-step taken in spine head	100%	estimated

Table 4.1: Parameters which play an important role in the simulation of PKCγ diffusion into dendritic spines with explanations, expressions/typical values and an indication as to where the value/expression has its origin.

The conversion factor Hc which was defined in equation (2.46) must also take reasonable values in the simulation, and this might mean changing it to another expression. Light et.al. [7] state that the concentration of conventional PKC which PKCγ is a part of, is typically $20\frac{\text{nMol}}{\text{L}}$ in cardiac cells. There is reason to believe that these values are typical for all cells. Since there are four conventional PKC types (α, β_I, β_{II} and γ), the assumption

that PKCγ makes up for a quarter of this concentration ($5\frac{\text{nMol}}{\text{L}}$) is made. The dendrite is already being modeled as a one-dimensional object, meaning that it is assumed to be much wider than a spine. Wide dendrites can have a diameter of around 10μm and around 50μm long before they start branching [**wikipedia??**], making the volume some $3900\mu\text{m}^3$.

The unit $\frac{\text{nMol}}{\text{L}}$ is converted to more manageable units below

$$\begin{aligned} 1\frac{\text{nMol}}{\text{L}} &= 10^{-15}\frac{\text{nMol}}{\mu\text{m}^3} \\ &= 10^{-24}\frac{\text{Mol}}{\mu\text{m}^3} \\ &\simeq 0.6\frac{\text{particles}}{\mu\text{m}^3} \end{aligned}$$

Craske et.al. measured an increase of $5\frac{\text{nMol}}{\text{L}}$ in the spine heads they were investigating [2] which, using numbers from [1] for average spine volumes, is roughly one PKCγ particle. In the simulations, this means that the total available concentration at the contact-point between the dendrite and a spine must exceed one particle. Assuming that the dendrite is cylindrical gives a cylinder volume segment with height equal to the neck width of the spine in question in which a concentration equal to at lest one particle must be present before any diffusion into the spine can take place. This makes $Hc \in (5, 24)$ since that is the way we have defined Hc. In other words, the integrated concentration must exceed some value defined either directly by the spine neck width or some average neck width depending on how much control we want to have over Hc.

The initial condition is open to discussion, but at the moment a skewed Gaussian function i used with a peak value between 1 and 5. Release of PKCγ into dendrites is followed by binding of PKCγ on the dendrite-wall, but in any case this resembles simply removing some portion from the simulation. This step can just as well be dropped, seeing as it does not provide any extra information, and so only some initial pulse is required. The Gaussian distribution is chosen for simplicity.

All values in the simulations are scaled to reduce round-off errors. Positions are measured in μm, concentrations in nMol/L or as particle-numbers when in spines. Only the timescale has normal units of seconds. Craske et.al. did experiments on the scale of $<$ 3min, but most of the simulations go for even shorter times.

Chapter 5

Results

5.1 Validity of the model

Section 2.3 shows that a random walk can be described by the Gaussian distribution and that it satisfies the diffusion equation as well as deriving the diffusion equation from a random walk. Mathematically these models are considered equivalent in the limit of sufficiently many walkers. This limit is defined by the time step as

$$N \geq \frac{1}{\Delta t^2} \tag{5.1}$$

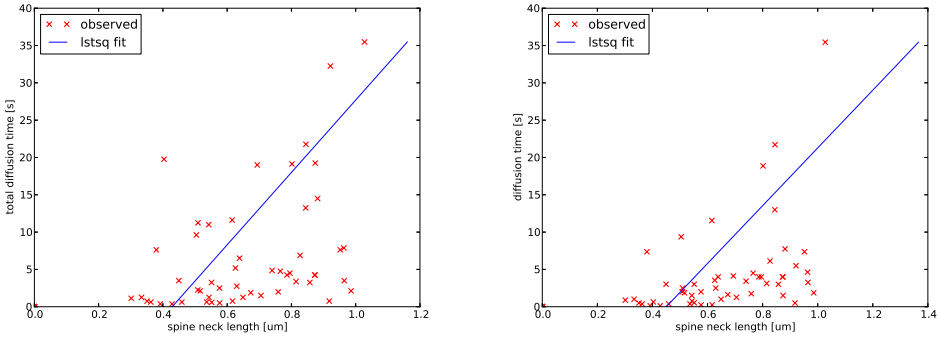
Figure 3.13b verifies that given a sufficient amount of walkers, the convergence in time is unaffected by the walkers. Thus supporting the claim that the models are equivalent, and that they can be combined in a meaningful way.

For all practical purposes, however, the demand for walkers is too high, and will generally only be met for verification purposes. This is no problem, since the models were combined to study effects from both length scales, which would not be meaningful to study if the demand for walkers is met.

In other words, the model converges to the continuum model in the limit of sufficient walkers, but this limit is seldom met.

5.2 Diffusion times into spines

Craske et.al. suggest that the neck of spines act as diffusion barriers which slow down, but don't completely stop the diffusion of PKCγ into spines. The function of this barrier is a bit unclear, but the presence of it is undisputed. In their measurements they found a delay of around $5 - 10$ seconds from elevated concentration levels in the dendrite until a similarly elevated concentration level occurred in spines with necks longer than 0.5μm. Using parameter values which resemble the values found in actual (rodent) neurons and neurites in the developed software, the observed delay-times have been recreated. Figure 5.1 shows plots of the observed diffusion times into spines. This figure shows a clear trend for longer diffusion times as the neck length of the spine increases. Figure 5.2 further support this claim and implies the average diffusion time for PKCγ into long necked spines to be of accordance to the results from Craske et.al. Seeing as there are no additional complexities added to the random walk model we can assume that the spine neck does in fact function as a diffusion barrier.



(a) Absolute diffusion times. (b) Relative diffusion times.

Figure 5.1: Absolute (a) and relative (b) diffusion times into spines. The lines represent a least squares fit of the results. This least squares fit should have been done after removing outliers and should have an expression written out somewhere.

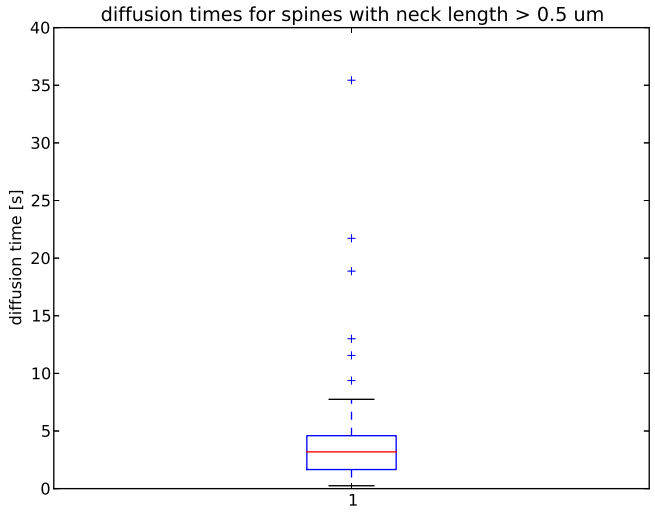


Figure 5.2: Boxplot of the relative diffusion times (time between elevated concentration in dendrite and elevated concentration in spine head) into spines with necks longer than 0.5μm. Similar studies were done by Crase et.al. and found diffusion time (unclear whether relative or not) to be somewhere around 5-10 seconds.

Through the simulations it became apparent that there must be some sort of limiting factor which limits the number of PKCγ particles that are let into

the spine. In real life this is achieved by a concentration gradient which tends to zero (or negative values) meaning that no particles will diffuse into the spine after it is "filled" up. A random walker will not feel this concentration gradient unless it is explicitly told so. The alternative solution then, is to reduce the probability for particles to diffuse into a spine for each particle that get caught in the spine head by some reasonable factor.

Chapter 6

Discussion

6.1 Properties of the model

In many ways this model has its strength in its simplicity and ease of added complexity. By separating the two solvers, and having the possibility to run them both individually, changing one of them will not affect the other. Of course, added complexity often means added overhead by having to initialize more variables and parameters. This must be solved by adding functionality to the class in charge of running the two solvers and combining their results, which is considered a simple task.

The weakness of the model is mainly the geometry question. As has been mentioned before, the current version only supports quadratic meshes which of course limits the possible applications of the model. One possible workaround which has been suggested consists of changing the PDE solver to a finite element solver. However, this is associated with a large workload, and for some FEM software it will be nontrivial to map mesh points from the PDE solver to the RW solver.

Installation of the Armadillo linear algebra library is required for the BE scheme to work. This scheme is highly recommended to use both because it is the most well-tested solver, and because there is no stability criterion. Compiling the code without Armadillo installed requires some changing and exclusion of the code.

6.2 The results

The model gives fairly good results for the application on PKCγ into spines which largely are in agreement with the results by Craske et.al. However, the mean diffusion times found from simulations seem to consistently lie towards the lower limits of the experimental results. One possible extension to the model which might fix this is to introduce an absorption probability in the spine head which is fairly large (say 80% per second), or even increases with the amount of time spent in the spine head. A setup like this should increase the average diffusion time by a few seconds. It does not, however reflect the physical process to a better accuracy seeing as a concentration increase in a spine head will be measured quickly.

Chapter 7

Conclusion

This (last) chapter contains the conclusion and sums up what has been done, what went wrong and finally suggests some future improvements and extensions that would be interesting to implement.

7.1 Workflow

In summary this thesis has two parts; the implementation and testing of both the PDE and RW models and the software which combines them, and the implementation and simulations of the diffusion of PKCγ into spines - problem. The former has without doubt been the most time consuming, mostly because of a bug which made the results appear correct without them being so, but also because of several other minor bugs which resulted in redoing most of the verifications several times.

Quite a lot of time went into assembly of the mass-matrix for the BE scheme in 2D. This was also because of a small bug causing the boundary conditions on one boundary to rely on the wrong parts of the previous time step. Assembling a mass matrix for a 3D BE scheme will probably involve some messy calculations, but is not considered difficult (since this has been done on paper already).

7.2 The model

As was mentioned in section 5.1 the analysis suggests that the developed model is stable and gives good results. The important detail which makes the model work is to do the RW "integration" first, pass the result as input to the PDE model and then solve the PDE by a method of choice. It is unclear whether a finite element method will give as good results, but highly probable seeing as it is the fundamental property of a diffusion process which ensures this. Namely that a diffusion process will dampen fast fluctuations more efficiently than slow fluctuations.

7.3 Future work

The developed model shows clear signs of being a first approach to the problem, and is in some ways a bit simple. Several improvements can be suggested to create a more realistic model both with respect to the diffusion processes on both length scales, and when it comes to the combination of the two

models. This section will mention some of the improvements that can be done.

7.3.1 PDE solver

Although there is a well expressed limitation to the accuracy of the model determined by the stochastic term, there are a few possible extensions which can be made to the PDE solver. Not all will necessarily improve the error term, but might introduce other properties which are of interest.

Finite Element Methods

Finite Element PDE solvers are mathematically and implementation-wise much more advanced than finite difference schemes. Depending on which solver is implemented, this step will ensure that more complex (and realistic) properties of a diffusion process like nonlinearity can be introduced with relative simplicity. Another question is how this can be implemented in the lower-scale model..
A finite element solver will also, as discussed in section 2.6, vastly simplify the implementation of more realistic mesh geometries in more than 1D.

More accurate time derivatives

Experimenting with time derivative approximations that are better than first order could also be interesting seeing as a second order convergence was achieved by the similar model described in section 3.4.1. One example of a more accurate approximation to the first derivative is listed in equation (7.1) (no name has been found for this scheme).

$$\frac{\partial u}{\partial t} \approx \frac{3u(t^n) - 4u(t^{n-1}) + u(t^{n-2})}{2\Delta t} \tag{7.1}$$

Note that equation (7.1) does not include the right-hand side of the actual equation and that this side will be evaluated at t^n making the scheme implicit and (hopefully) stable. The residual for the scheme proposed in equation (7.1) is calculated below

$$
\begin{aligned}
R &= \frac{3u(t) - 4u(t-\Delta t) + u(t-2\Delta t)}{2\Delta t} - u'(t) \\
&= \frac{1}{2\Delta t}\left[3u(t) - 4\left(u(t) - \Delta t u'(t) + \frac{\Delta t^2}{2}u''(t) - \frac{\Delta t^3}{6}u'''(t)\right)\right] + \dots \\
&\frac{1}{2\Delta t}\left[u(t) - 2\Delta t u'(t) + 2\Delta t^2 u''(t) - \frac{8\Delta t^3}{6}u'''(t)\right] - u'(t) \\
&= \frac{2\Delta t u'(t) - \frac{4\Delta t^3}{6}u'''(t)}{2\Delta t} - u'(t) \\
&= -\frac{\Delta t^2}{3}u'''(t) \\
R &\sim \mathcal{O}(\Delta t^2)
\end{aligned}
$$

A nice feature of this scheme is that it will result in the following linear system

$$\mathbf{M}\mathbf{u}^n = 4\mathbf{u}^{n-1} - \mathbf{u}^{n-2}$$

which is very similar to the system already being assembled and solved by the implemented BE scheme. Only a minor modification in the assembly of the diagonal of $\mathbf{M}$ seems to be necessary, and this is a trivial change.

The actual benefit of introducing a better approximation to the time derivative must of course be tested.

7.3.2 Lower scale models

In section 2.3.3 the argument that the Brownian motion model converges to the Gaussian distribution was given for choosing the simple RW model. The same argument can be made for most of the other possible models if they do not possess any special capabilities like drift or anisotropy. However, the argument is only valid in the verification phase when the number of walkers (or whatever) is large. For the actual simulations the number of walkers (or whatever) will typically be very small, and the central limit theorem does not apply. This opens the possibility of adding a variety of lower scale models, some of which will be mentioned below.

Variations of Random Walk

Although the current RW implementation supports some added complexity like anisotropy and drift, there is always the possibility to make the algorithm more complex. Of course, there is not much reason to do this without an actual physical problem which results in some more complex RW algorithm,

but finding these applications should not be to hard. For example, some attraction/drift term could be added to simulate Coulomb-attraction.
As was mentioned in section 6.2, the results from simulating PKCγ diffusion are good, but not perfect. An extension where walkers in spine heads do not immediately get registered as a concentration increase might improve the results. Another possibility is to model "wall-collisions" slightly differently by introducing a delay time where the PKCγ particle is stationary for some number of time steps after colliding with the spine neck. This approach will most likely better reflect the actual physics of the process.

Alternatives to Random walks

This section will mention some alternatives to the random walk model used in this project and discuss how realistic they are to combine with a diffusion PDE as one goal in this project has been. Both applications towards computational neuroscience and more general applications will be discussed.
These models are pretty complex with many details, and this project does not in any way try to do more than introduce them. Further reading is cited in the end of each section.

Molecular Dynamics

Molecular dynamics is the simulation of the dynamics of atoms and molecules using classical, Newtonian mechanics in the sense that the molecules are affected by a potential, and that the sum of forces describes the dynamics. Their dynamics are then integrated forward in time, and used to describe for example flow in nanoporous media. This means that the system is fully described by the position and velocities of all the atoms. Of course, there is a vast variety in the level of complexity here and we will only look at the simplest example, namely the Lennard-Jones potential, eq. (7.2). This potential consists of an r^{-12} term which denotes the Pauli repulsion at short ranges, and an r^{-6} long range, attractive Van Der Waals term. The relative distance between two atoms is denoted by r. The Lennard-Jones potential is derived to simulate Argon in the Van Der Waals equation of state.

$$U = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \tag{7.2}$$

It is possible to do simulations of flow in nanoporous materials using the Lennard-Jones potential, although it is a far from perfect model, by only allowing some of the atoms to move. This will result in a matrix of stationary atoms, simulating a wall (note that there will still be forces acting from these

atoms), and a liquid inside this matrix.

There are two main problems with using molecular dynamics more or less regardless of the application: It requires a potential which describes the forces working on all molecules in the simulation. This part may be particularly difficult when it comes to modeling macromolecules like proteins.
It might be really difficult to create the desired geometry for a simulation.

For diffusion purposes this model is extremely temperature dependent, and not directly transferable to the diffusion processes described in this project. Especially seeing as the diffusion coefficient is a derived quantity, not a parameter to be specified.

Direct simulation Monte Carlo

Direct simulation Monte Carlo (DSMC) is a numerical method first developed by G.A.Bird to model low density gas flow. With some extensions it can also model continuum flow and give results comparable to the Navier Stokes equations. The DSMC method works by modeling molecules which represent a large number of fluid molecules (or atoms) in a probabilistic manner.
DSMC has a lot of applications today varying from supersonic fluid flow modeling to micro electromechanical systems to micro- and nano- porous flow.

Compared to the molecular dynamics this method has the advantage of adding general geometries with relative ease. There aren't necessarily any new problems with DSMC compared to RW other than complexity of implementation, but it is primarily designed to model fluid flow.

For diffusion purposes this model is extremely temperature dependent, and not directly transferable to the diffusion processes described in this project. Especially seeing as the diffusion coefficient is a derived quantity, not a parameter to be specified.

Appendix A

Appendix

A.1 Various calculations

In this appendix some more tedious and rather boring, but no less important calculations can be found. We will also list some algorithms that are important, but not quite in the scope of this thesis.

A.1.1 Backward Euler scheme in 2D

Using the BE discretization on the simple 2D diffusion equation will yield the general scheme in equation A.1.

$$u_{i,j}^n = \underbrace{\frac{-D\Delta t}{\Delta x^2}}_{\alpha}\left(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}\right) + \underbrace{\left(1 + \frac{2D\Delta t}{\Delta x^2} + \frac{2D\Delta t}{\Delta y^2}\right)}_{\gamma} u_{i,j}^{n+1} \underbrace{-\frac{2D\Delta t}{\Delta y^2}}_{\beta}\left(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right) \tag{A.1}$$

This can, again, be written as a linear problem where the vectors are simply the matrices u^n and u^{n+1} written as column vectors. The matrix is written out for a 3×3 grid with no-flux Neumann boundary conditions in equation A.2. We see that it is a five-band diagonal matrix, and so the tridiagonal solver cannot be used in this case. It is fully possible to use for example a Gaussian elimination in order to solve this equation, but it will require $\frac{2}{3}\mathcal{O}(n^3)$ operations per time step, where n is the size of the matrix (in this case $n = 9$). Another way to solve this equation, and by extension use the BE scheme, is to use some form of sparse LU decomposition.

$$\begin{pmatrix} \gamma & -2\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 & 0 \\ -\beta & \gamma & -\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 \\ 0 & -2\beta & \gamma & 0 & 0 & -2\alpha & 0 & 0 & 0 \\ -\alpha & 0 & 0 & \gamma & -2\beta & 0 & -\alpha & 0 & 0 \\ 0 & -\alpha & 0 & -\beta & \gamma & -\beta & 0 & -\alpha & 0 \\ 0 & 0 & -\alpha & 0 & -2\beta & \gamma & 0 & 0 & -\alpha \\ 0 & 0 & 0 & -2\alpha & 0 & 0 & \gamma & -2\beta & 0 \\ 0 & 0 & 0 & 0 & -2\alpha & 0 & -\beta & \gamma & -\beta \\ 0 & 0 & 0 & 0 & 0 & -2\alpha & 0 & -2\beta & \gamma \end{pmatrix} \mathbf{u}^n = \mathbf{u}^{n+1} \tag{A.2}$$

When we try to implement Neumann boundary conditions for grids that are larger than 3×3 we come across a problem. Doing the matrix-vector multiplication in equation A.2 reproduces the BE scheme with boundary conditions perfectly. However, if we extend to a 4×4 grid using a matrix on the same form we will start producing equations which will not arise from the scheme. This is illustrated in eqs. A.3 and A.4. Moving the off-diagonal entries with α one more column to the right and left will solve the problem,

but this will force us to use some more general solver of a sparse linear system. All in all we will probably be better off using another scheme (at least in 2D). The first equation that arises from the BE scheme in 2D (where $i = j = 0$) is

$$u_{0,0}^n = \gamma u_{0,0}^{n+1} - 2\alpha u_{1,0}^{n+1} - 2\beta u_{0,1}^{n+1} \tag{A.3}$$

while the first equation produced by the linear system in the 4×4 case is

$$u_{0,0}^n = \gamma u_{0,0}^{n+1} - 2\alpha u_{0,3}^{n+1} - 2\beta u_{0,1}^{n+1} \tag{A.4}$$

which is an equation that will never be produced by the BE scheme. In the 3×3 grid-case the off-diagonal matrix entries with α are on the third column before and after the diagonal.

Moving the corresponding entries to the fourth column in the 4×4 case, and similarly to the n'th column in the $n \times n$ case will fix the problem, but also increase the complexity of the matrix seeing as it will be $n + 2$ band diagonal.
Extending the model to three spatial dimensions gives a very similar matrix to the 2d-case.

$$\begin{pmatrix} D_{00} & -2\beta I & 0 & -2\alpha I & 0 & 0 & 0 & 0 & 0 \\ -\beta I & D_{01} & -\beta I & 0 & -2\alpha I & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 \\ -\alpha I & 0 & 0 & D_{10} & -2\beta I & 0 & -\alpha I & 0 & 0 \\ 0 & \ddots & 0 & \ddots & \ddots & \ddots & 0 & \ddots & 0 \\ 0 & 0 & 0 & -2\alpha I & 0 & 0 & D_{n0} & -2\beta I & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & -2\alpha I & 0 & -2\beta I & D_{nn} \end{pmatrix} \begin{pmatrix} u_{00k}^{n+1} \\ u_{01k}^{n+1} \\ \dots \\ u_{10k}^{n+1} \\ \dots \\ u_{n0k}^{n+1} \\ \dots \\ u_{nnk}^{n+1} \end{pmatrix} = \mathbf{u}^n \tag{A.5}$$

In equation A.5 I denotes the $n \times n$ identity, D_{ij} denotes the tridiagonal $n \times n$ matrix with entries similar to the ones in eq. A.1, and off-diagonal entries similar to the ones in eq. A.1. All 0's denote the $n \times n$ zero-matrix. The values α and β are the relevant coefficient matrices for the calculations in question. These will be diagonal as well (or simply numbers in the isotropic case). Note also that the vector entries u_{ijk}^{n+1} are column vectors, making the vector $\mathbf{u}^{n+1}$ have the shape $1 \times n^3$.

A.2 Debugging

In any project which involves programming one is bound to do some debugging. This project is no exception. Debugging can be extremely frustrating

because no-one sees all the hours that go into finding the bugs, only the ones that do not (when the bug is not fixed). This section will deal with some general techniques for debugging finite difference solvers and random-walk implementations and some special words on how to debug the software developed to combine the two solvers.

A.2.1 Compiler/syntax errors

If you are programming in a compiled language like fortran or C/C++ you will forget some syntax, or misspell it, use a compiler-flag that outputs as much info as possible to terminal (-Wall for the gnu compiler), and start with the errors you understand. If you are building a larger project which requires linking, remember that packages must be linked in the correct order. For example; the armadillo linear algebra library is backened by LAPACK and BLAS. Both these libraries must be linked as well and they must be linked in the following order:

```
g++ *.cpp -o myprog -O2 -larmadillo -llapack -lblas
```

Anything else will give very cryptic compiler errors.
If you are using an interpreted language like python or MatLab the interpreter usually gives extensive information about the errors you have done, read them thoroughly!

A.2.2 Segmentation faults

In an interpreted language you will be told exactly where and what is wrong, in compiled languages you will not unless you are using extensions that give you some more information like armadillo. Segmentation faults are often quite simple to find, and most compilers have some sort of debugger which can help you find them. The gnu-compiler has an environment called gdb in which you can run your program which will catch seg.-faults and tell you where they are. If you are using some advanced editor like qt creator you can also easily place breakpoints in your code where you can get information about the various variables, instances, attributes etc. of your code at the exact time of the break. You can also step through the code. thoroughly Some times though the thing that works best is to print things at various places. I like having the possibility that every function in my code can print its name when it is called. There are even some python modules which tells you where it was called from. This will make it very easy to find out when the code went wrong, and what function is the problem.

A.2.3 Finite difference methods

First and foremost: Have a correct discretization. There are (probably) webpages which can discretize your equation(s) for you, but it is almost always useful to do this by hand. It will help you in your further debugging. There is one very important rule in programming in general: "First make it work, then make right, then make it fast". For implementing FDMs this means that you should start coding as soon as you have a clear image of what to be implemented, and what dependencies are needed. You will need a well defined initial condition (preferably one where you have the exact solution of the equation) and boundary conditions before you start coding. Personally, I like starting with the simplest Dirichlet boundary conditions $u|_{\partial\Omega} = 0$ and make them work before I go any further. You should note, however, that implicit schemes will be greatly influenced by the choice of boundary conditions.
Visualization is invaluable during debugging, seeing as a plot will let you see when and where the error occurs. *Show some example* Most likely you will now have something wrong with your solution (if not, cudos). This is where you look over your discretization again to make sure that it is correct, and then look over your implementation to check that it actually does what you think it does. At this point I would like to introduce rubber-duck debugging which was invented by the C-developer Dennis Ritchie. The story goes that he would keep a rubber duck at his desk and whenever he was stuck, would describe the code in detail (what each statement did and was supposed to do) to the rubber duck. Asking questions often reveals a lot of information. Personally I like my rubber duck to challenge me, so I prefer to involve a friend, but the concept is the same.
When your code seems to reproduce the intended results it is time to start the verification. This is where we make an error estimate and do some numerical analysis (you should of course have checked for the numerical stability of your chosen scheme when you discretized it). Making sure your implementation is correct is a lot harder than it sounds, but there are a few points that should be fulfilled:

- Manufactured solution
  Find some function which fulfills the equation you are working with. Remember that you have a source term which can be whatever you want it to be at this point, meaning that you can more or less decide what solution you want to your equation.

- Stationary solution
  This boils down to energy-conservation. If the initial condition is a con-

stant, there should be no time-dependencies (assuming your boundary conditions match; an initial condition $u = 1$ with Dirichlet boundaries $u|_{\partial\Omega} = 0$ will not work), and the solution will be constant.

- Exact numerical solution
  For a fitting initial condition (and discretization) you will be able to find an exact solution to the discretized equation you are implementing. An example of this is found in chapter 3.2.3. Your scheme should reproduce this solution to more or less machine precision. Note that you might run into round-off errors and overflow here in some cases (again, see chapter 3.2.3).

- Convergence test
  The discretization that is implemented will have some error term dependent on a discretization parameter (usually Δt, Δx or some parameter h used to determine the other discretization parameters) to some power. This power will determine the convergence rate of the numerical scheme, and you should verify that your implementation has the expected convergence. A convergence test is another way of saying that reducing the discretization parameter should reduce the error by the expected ammount. For a first order scheme the error should be halved by halving the time-step where as a second order scheme will get a reduction of $\frac{1}{4}$ for the same havlving of the discretization parameter.

There are probably more ways to make sure that a finite difference scheme is working properly, but the ones listed will usually give a good implication.

A.2.4 Random walk and Monte Carlo methods

The main difference between debugging a MC based solver and a deterministic solver is the fact that you often do not have a clear idea of what the results of the intermediate steps should be. What you might know (or should know during development), however is the result of the complete MC integration, and some statistical properties of your random numbers. Using uniformly distributed random numbers will give you a certain mean and standard deviation, and a Gaussian distribution will give you another. You should check that the random number generator (RNG) you chose actually reproduces these properties to a reasonable precision. If you are working with random walkers it also helps to look at the behavior of a small number of walkers, to check that they behave more or less as you expect. One thing to look out for is the fact that a random walker in both one and two spatial dimensions will

fill all space given enough steps. Of course enough steps is infinitely many, but if you also implement reflecting boundaries and use some 4-5 walkers you will see a tendency after approximately 10^4 steps.
As we have discussed earlier the fluctuations in a MC-model are usually of a magnitude $\frac{1}{\sqrt{N}}$ this is also smart to verify.
Finally, you should absolutely have the possibility to set the random seed and check that two runs with the same random seed produces the exact same result and makes sure you are using a RNG with a large enough period. The xor-shift algorithm by Geroge Marsagla [] has a period of around 10^{48} which usually is more than adequate.

A.2.5 The developed software

For some 2 months while working with this project I got really good results which seemed to verify all the important parts of the theory. Unfortunately it turned out that, while individually both parts of the program did exactly what they were supposed to do (verified by various tests), the combination of the two parts was implemented wrong. What actually happened was a finer and finer round-off rather than taking some number of steps with random walkers and combining the two models. It turned out that I sent an empty array to the random walk class as a new initial condition for the current time-step.
The moral behind this little story is that you should make 100% sure that every part of every function you write does exactly what you think it does, and nothing else. Furthermore, if you rewrite your code, you should remove the old parts as soon as possible. If you use some kind of version control software, which you definitely should, you will have older versions saved in the version control anyways. Do not be nostalgic and simply comment out the old parts just in case something, this makes your code very messy, and leaves the possibility of something slipping past you.

Another point to be made is that it will probably be helpful to construct the different parts of your code in such a way that they can be run as independently of each other as possible. As an example, both the PDE-solver, its tridiagonal linear system solver and the spine object can with relatively small changes to the main-file be run independently. This allows for easier testing of the various parts of the code, and makes it more likely that the code will be reused in other projects.

A.2.6 When you cannot find the bug

While debugging (or any other repetitive task involving your own work) it is remarkably easy to become blind to your mistakes. The psychology behind this is (probably) that you have a clear idea of what should happen in each statement, and so you read that in stead of what the statement actually says. When it comes to proof-reading you can supposedly read backwards word by word, but can you do something similar when reading code? While I have never tried reading my code backwards because a statement usually depends on the previous statement, I have tried doing hand-calculations for almost every statement. Although hand calculations do not always show where things go wrong, they point out what variable or array entry etc. is wrong, and so the previous calculations can be checked. For finite difference schemes one can reduce the number of spatial mesh points to something manageable like three or four, and then do the same calculations that you think the computer does. If the solution is a matrix you can pinpoint the invalid matrix-entries with this method.

Another very important point if you are stuck is to never use "nice" values. If a parameter is set to zero or one just because it needs to be something, the probability that a potential problem disappears because it cancels out increases dramatically. Similarly, never do matrix calculations for 3×3 matrices. Use 4×4 matrices instead. The reasoning behind this is that banded matrices might fool you on 3×3 matrices, making you think your problem is tridiagonal when it in fact is n-band diagonal for example.

Bibliography

[1] Jon I Arellano et al. "Ultrastructure of dendritic spines: correlation between synaptic and spine morphologies". In: *Frontiers in neuroscience* 1.1 (2007), p. 131.

[2] Madeleine L Craske et al. "Spines and neurite branches function as geometric attractors that enhance protein kinase C action". In: ().

[3] L Farnell and WG Gibson. "Monte Carlo simulation of diffusion in a spatially nonhomogeneous medium: A biased random walk on an asymmetrical lattice". In: *Journal of Computational Physics* 208.1 (2005), pp. 253–265.

[4] Bruce Graham, Andrew Gillies, and David Willshaw.

[5] Morten Hjorth-Jensen. "Computational physics". In: *Lecture notes* (2011).

[6] Frederick James. "A review of pseudorandom number generators". In: *Computer Physics Communications* 60.3 (1990), pp. 329–344.

[7] Peter E Light et al. "Protein Kinase C–Induced Changes in the Stoichiometry of ATP Binding Activate Cardiac ATP-Sensitive K+ Channels A Possible Mechanistic Link to Ischemic Preconditioning". In: ().

[8] George Marsaglia. "Xorshift rngs". In: *Journal of Statistical Software* 8.14 (2003), pp. 1–6.

[9] Charles Nicholson. "Diffusion and related transport mechanisms in brain tissue". In: *Reports on progress in Physics* 64.7 (2001), p. 815.

[10] Mathis Plapp and Alain Karma. "Multiscale finite-difference-diffusion-Monte-Carlo method for simulating dendritic solidification". In: *Journal of Computational Physics* 165.2 (2000), pp. 592–619.