MULTISCALE MODELING OF DIFFUSION PROCESSES IN THE BRAIN

by

Fredrik E Pettersen
f.e.pettersen@fys.uio.no

THESIS

for the degree of

MASTER OF SCIENCE

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2014

Contents

1 Introduction **3**
1.1 Background 4

2 Basic Theory **7**
2.1 Introduction to random walks 8
2.1.1 Further discussion and analysis of the introduction . . 8
2.1.2 More general Random Walks 11
2.1.3 Choosing random walk algorithm 13
2.1.4 Random walks and anisotropy 13
2.1.5 Random walks and drift 14
2.2 Some words about partial differential equations 17
2.2.1 Discretizing 17
2.2.2 Stability 20
2.2.3 Truncation error 21
2.2.4 Extension to 3 spatial dimensions 22
2.2.5 Tridiagonal linear systems 23
2.3 Combining the two solvers 25
2.3.1 The basic algorithm 25
2.3.2 Potential problems or pitfalls with combining solutions 26
2.3.3 Probability distribution and timesteps 28
2.4 Geometry 29

3 Analysis **31**
3.1 Some discussion 32
3.1.1 The error estimate 34
3.2 Manufactured Solutions 35
3.2.1 Exact numerical solution 39
3.2.2 Increasing the time step and the relative size of walk-area 42
3.2.3 The effects of adding drift to the walkers 44
3.2.4 Anisotropic diffusion 47
3.3 2D 47

3.3.1 Including random walks 50
3.4 Testing Random walks 52

4 Software **57**
4.1 About 58
4.2 Adaptivity 58
4.3 Computational cost 58
4.3.1 Memory 58
4.3.2 CPU time 58
4.3.3 Parallelizability 59

5 Results **61**
5.1 Validity of the model 62

A **63**
A.1 Various calculations 64
A.1.1 Backward Euler scheme in 2D 64
A.2 Debugging 65
A.2.1 Compiler/syntax errors 66
A.2.2 Segmentation faults 66
A.2.3 Finite difference methods 67

List of Figures

2.1 Initial condition for convection diffusion 16
2.2 Algorithm 26

3.1 Initial condition in 1d 33
3.2 Initial condition in 2d 34
3.3 Numerical error for 1D Forward Euler discretization 36
3.4 Effect of increasing number of walkers 37
3.5 Convergence test for the FE scheme. The x axis is $\ln(\Delta t)$. . . 38
3.6 Convergence test for the BE scheme. The x axis is $\ln(\Delta t)$, the y-axis (though a little hard to see) is zoomed in around r=1. . 39
3.7 Testing the relation between $(D\Delta t)^i$ and the binomial coefficients. 41
3.8 Verification for exact numerical solution 42
3.9 Numerical error for 1D Backward Euler discretization 43
3.10 The effect of increasing the size of the walk area for a fixed $\Delta t = 0.001$ using the BE discretization. 43
3.11 44
3.12 Errorplot, long simulation 44
3.13 Verification of Convection diffusion equation implementation . 46
3.14 Adding walkers influenced by drift 46
3.15 Verification of anisotropic diffusion equation implementation . 47
3.16 Numerical solution from the FE scheme versus the exact numerical solution of the FE scheme in 2d. we have used a Δt which is almost on the stability criterion, $\Delta t = \frac{\Delta x \Delta y}{5} = 8e - 05$. 49
3.17 Convergence test FE 2d 50
3.18 Verification of anisotropic diffusion equation implementation . 51
3.19 Convergence test for combined simulation 51
3.20 Convergence test for combined simulation 52
3.21 Convergence test RW 55
3.22 Error plot RW 56

Chapter 1

Introduction

This thesis is an attempt at modeling diffusion processes in which part of the process takes place on a length scale so small that the continuum approximation becomes invalid. In this part we will therefore try and introduce another model of the diffusion process in the hope that this will give us something extra. It is the hope of Hans Petter that this thesis will be an introduction to a new research project to understand mesoscale physics in collaboration with Gaute Einevoll at UMB.

The very first approach was to simply try the problem on a bit. That is to try and substitute some small part of the mesh in a Finite Difference Diffusion solver (Froward Euler scheme) with a stochastic diffusion solver. A random walk method was implemented on part of the mesh to take over the equation-solving. This was done in 1 and 2 spatial dimensions with the aim of finding potential difficulties so that we can further investigate them.
Upon switching length-scales a fundamental question arises almost immediately; what is the continuum limit? In our case this question takes a slightly different, and possibly more answerable form; what is the conversion rate between the continuum model and the microscopic model, and by extension, what does a walker correspond to? The first instinct of this candidate was to just try some conversion rate (say some value corresponds to some number of walkers), and this was implemented in both 1 and 2 dimensions.

1.1 Background

This is very much a first draft, and only my thoughts around what I understand as the basis of the thesis.
Though the scope of this thesis might seem a bit *sought*, it is in fact a real world concern from the computational neuroscience group at UMB. Their work contains network simulations of neurons, which they are trying to tweak with experimental data. The experimental data come from measurements of voltage levels in the Extra Cellular Space (ECS). The ECS is, essentially, the space which separates neurons and has an immensely complicated geometry as we can see from figure **??**. The width of the ECS varies, but is in general $\sim 1\mu$m. Parts of the ECS is, however, much narrower. In the ECS there are a number of diffusion governed transport mechanisms which are vital for the function of the neurons. In fact, there are examples of snake venom which have a shrinking effect on the ECS, and it is thought that this causes large scale neuronal death very quickly. For the researchers at UMB, however, the ECS is of importance because the diffusion tensor in the ECS is related to the conductivity tensor through equation 1.4, which in turn lets them teak parameters in their network models.

$$D = \frac{k_B T}{6\pi\eta r}, \tag{1.1}$$

Though there are several reasons to study diffusion processes in the ECS, this project has a specific goal in mind. The Einstein relation, eq. (1.1), relates the diffusion constant to the viscosity of the medium in which the diffusion is taking place. From the definition of viscosity, μ, we have

$$v_d = \mu F \tag{1.2}$$

where $F = qE$ is the standard electrical force acting on a charged particle. We can also define the current from the drift velocity of the particles as

$$J = cqv_d = \sigma E \tag{1.3}$$

where $\sigma = c\mu q^2$ is the electrical conductance, in this case, of the ECS. Inserting this in the Einstein relation, eq. (1.1), lets us express the conductivity in terms of the diffusion constant as

$$\sigma = \frac{cq}{k_B T} D \tag{1.4}$$

Equation (1.4) is trivially generalizable to tensor notation, where the conductance and diffusion "constant" are both 2. order tensors.

$$\langle r^2 \rangle = 2dDt \tag{1.5}$$

The fundamental problem here is that, while diffusion in its own is a truly multi-scale process, we have no way of knowing for sure that the continuum models, and all they bring with them, are correct for this type of geometry. In particular, the Einstein relation 1.1 has, to my knowledge, only been derived for diffusion in a homogeneous media. The definition of a homogeneous media includes a mean free path close to infinity, which we will have trouble arguing for the existence of in the ECS. On the other hand, I know that another Einstein relation 1.5 is widely used in other fields of physics where the media in question is hardly homogeneous. In molecular dynamics simulations, the two mentioned Einstein relations are used to measure the viscosity of fluids in nano-porous materials, and with success as far as I know.

Chapter 2

Basic Theory

In this chapter we will take a closer look at random walks, both in general and the transition from the statistical view to partial differential equations. We will take a look at different algorithms to produce random walks, and discuss their pros and cons in light of this project. Then we will take a quick look at partial differential equations and numerical solution of them.

2.1 Introduction to random walks

The most basic random walk 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) we draw a random number, r, between 0 and 1 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 walker will step up if $r \in (0.75, 1]$ and left if $r \in [0, 0.25)$, for example.

2.1.1 Further discussion and analysis of the introduction

The following derivation is borrowed from a compendium in statistical mechanics by Finn Ravndal.
If we do sufficiently many walks, 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 we will get a net displacement of $S = N - 2$ with the probability $P_N(R = N - 1) = Np^{N-1}q$. We can generalize this to finding the probability of a walk with a R steps to the right as

$$P_N(R) = \binom{N}{R} p^R q^{N-R} \tag{2.1}$$

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 2.1 is the Bernoulli probability distribution, which is normalized.

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

We can use this distribution to calculate various average properties of a walk consisting of N steps. For example, the average number of steps to the right is

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

From this we can also find the average value of the net displacement 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 we will be more interested in the mean square displacement 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

$$\langle S^2 \rangle = 4\langle R^2 \rangle - 4N\langle R \rangle + N^2 = 4((Np)^2 + Npq) - 4N^2 p + N^2$$
$$= N^2(4p^2 - 4p + 1) + 4Npq = N^2(2p-1)^2 + 4Npq = N^2(p-q)^2 + 4Npq.$$

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

$$\langle \Delta S^2 \rangle = \langle N^2 (p-q)^2 + 4Npq - (N(p-q))^2 \rangle = 4Npq \tag{2.2}$$

When the number of steps gets very large we can approximate the Bernoulli distribution (eq. 2.1) 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 steplengths to have a finite variance (*find something to refer to*). The Bernoulli distribution then simplifies to

$$P(S,N) = \left(\frac{1}{2}\right)^N \frac{N!}{R!L!} \tag{2.3}$$

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 $R+L = N$. We now insert for $\frac{2R}{N} = 1+\frac{S}{N}$ and $\frac{2L}{N} = 1-\frac{S}{N}$ and expand the logarithms to first order, $RL = \frac{N^2-S^2}{4}$ in the prefactor, and approximate $1 - \frac{S^2}{N^2} \simeq 1$. This gives

$$P(S,N) = \sqrt{\frac{2}{\pi N}} \exp\left(\frac{-S^2}{2N}\right) \tag{2.4}$$

which is an ordinary, discrete Gaussian distribution with $\langle S \rangle = 0$ and $\langle S^2 \rangle = N$. If we keep assuming that the walker is on the x-axis, and let the step length, a, get small the final position will be $x = Sa$ which we can assume is a continuous variable. Similarly, we let the time interval between each step, τ, be small and let the walk run for a continuous time $t = N\tau$. This changes the distribution 2.4 to

$$P(x,t) = \frac{1}{2a}\sqrt{\frac{2\tau}{\pi t}} \exp\left(-\frac{x^2\tau}{2a^2 t}\right). \tag{2.5}$$

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 $\Delta x = 2a$. We also introduce the diffusion constant

$$D = \frac{a^2}{2\tau} \tag{2.6}$$

making the distribution

$$P(x,t) = \sqrt{\frac{1}{4\pi Dt}} \exp\left(-\frac{x^2}{4Dt}\right) \tag{2.7}$$

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 by insertion we find the variance $\langle \Delta x^2 \rangle$

$$\langle \Delta x^2 \rangle = \langle \langle x^2 \rangle - \langle x \rangle^2 \rangle = \langle a^2 \langle S^2 \rangle - a^2 \langle S \rangle^2 \rangle = 4Npqa^2 \tag{2.8}$$

2.1.2 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

$$\mathbf{r}(t_i) = \sum_{j=0}^{i} \Delta \mathbf{x}(t_j) \tag{2.9}$$

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

$$P(x, M) \propto e^{-\frac{x^2}{2M\sigma^2}} \tag{2.10}$$

Remember that the actual Gaussian distribution is

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

We can introduce an Einstein relation $\sigma^2 = 2dD\Delta t$ and the obvious relation $t = M\Delta t$ to get a more desirable exponent. We see that $\langle \Delta x^2 \rangle = 2Dt$ in the one dimensional case where $d = 1$.

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

$$P(x,t) = \sqrt{\frac{1}{4Dt}} \exp\left(-\frac{x^2}{4Dt}\right) \tag{2.11}$$

If we have a large number, N, of walkers, their concentration will be $C(x,t) = NP(x,t)$. The concentration is conserved, so any amount that flows out of an area must reflect as a decrease in concentration. We can express this by the flow of concentration

$$\frac{\partial C}{\partial t} - \nabla \cdot \mathbf{J} = S \tag{2.12}$$

where $\mathbf{J}$ is the flow vector and S is a source term which in our case will be zero. From Fick's first law we know that $\mathbf{J} = -D\nabla C$. Inserting this gives us

$$\frac{\partial C}{\partial t} = \nabla \cdot (D \cdot \nabla C) \tag{2.13}$$

which is the diffusion equation. By insertion we can check that this version (2.11) of the Gaussian distribution fulfills the diffusion equation. Starting with only the time derivative gives us

$$\frac{\partial P}{\partial t} = -\frac{4\pi D \exp\left(-\frac{x^2}{4Dt}\right)}{2\sqrt{(4\pi Dt)^3}} + \frac{x^2 \exp\left(-\frac{x^2}{4Dt}\right)}{4Dt^2\sqrt{4\pi Dt}}$$

$$= \exp\left(-\frac{x^2}{4Dt}\right)\left(\frac{8Dx^2}{2\sqrt{\pi}(4Dt)^{5/2}} - \frac{(4D)^2 t}{2\sqrt{\pi}(4Dt)^{5/2}}\right) = \frac{4D\exp\left(-\frac{x^2}{4Dt}\right)(x^2 - 2Dt)}{\sqrt{\pi}(4Dt)^{5/2}}$$

We then finish by doing the spatial derivative

$$D\frac{\partial^2 P}{\partial x^2} = \frac{D}{\sqrt{4\pi Dt}}\frac{\partial}{\partial x}\left[-\exp\left(\frac{-x^2}{4Dt}\right)\left(\frac{-2x}{4Dt}\right)\right]$$

$$= \frac{2D}{4Dt\sqrt{4\pi Dt}}\exp\left(\frac{-x^2}{4Dt}\right)\left[1 - x\left(\frac{2x}{4Dt}\right)\right] = \frac{4D\exp\left(-\frac{x^2}{4Dt}\right)(x^2 - 2Dt)}{\sqrt{\pi}(4Dt)^{5/2}}$$

and see that they are equal, meaning that the diffusion equation is satisfied.

2.1.3 Choosing random walk algorithm

As this article points out [1] 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.

However, this model will struggle with reproducing anisotropic diffusion, that is $D = D(x)$. Farnell and Gibson also suggest a method for improving the results by adjusting the step length and probability according to position, thus effectively adjusting the diffusion constant of the walk as well. We see then that the simplest model is rather robust, and well tested. However, the aim of this project is to combine two realistic models for diffusion on different length scales, and the simplest random walk model has one fundamental flaw in that view; it is not a realistic model for diffusing particles. Brownian motion is a more realistic physical model for diffusing particles, and can (I think) quite easily be modified to model anisotropic diffusion as well. We can model Brownian motion simply by using equation 2.9, and we can with a bit of work expand it to model collisions between walkers as well.
That being said, by the central limit theorem both models will after some timesteps 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.

2.1.4 Random walks and anisotropy

Any real problem where parts of the diffusion process cannot be modelled by the continuum approximation is bound to be anisotropic. 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 do we model this. Equation 2.37 shows the step length as a function of the diffusion constant. If we simply replace the diffusion constant by a function $D = D(\mathbf{x})$ we are at least started, but this will not quite be sufficient as Farnell and Gibson point out [1]. Through their experiments they found that only adjusting the steplength 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 steplength and an adjusted step probability, a solution they landed on after trial and error. The expressions they proposed are listed in equations 2.14 to 2.18.

$$\Delta_p(x) = \frac{1}{2}\left(L(x) + L(x + \Delta_p(x))\right) \to L(x) + \frac{1}{2}L(x)L'(x) \tag{2.14}$$

$$\Delta_m(x) = \frac{1}{2}\left(L(x) + L(x - \Delta_m(x))\right) \to L(x) + \frac{1}{2}L(x)L'(x) \tag{2.15}$$

where $L(x)$ is defined in equation 2.16 and $\Delta_p(x)$ and $\Delta_m(x)$ are the adjusted steplengths to the right and left, respectively.

$$L(x) = \sqrt{2D(x)\Delta t} \tag{2.16}$$

We also have the adjusted jump probabilities $T_r(x)$ and $T_l(x)$ which are the probabilities for a walker in position x to jump right or left, respectively. These are defined in equations 2.17 and 2.18

$$T_r(x) = \frac{1}{2} + \frac{1}{4}L'(x) \tag{2.17}$$

$$T_l(x) = \frac{1}{2} - \frac{1}{4}L'(x) \tag{2.18}$$

We notice that the adjusted steplength we proposed to start with is still a part of the final expressions.

2.1.5 Random walks and drift

Another point we have yet to say something about is diffusion that has a drift term, $\frac{\partial u}{\partial x}$.

Initially one thought that diffusion in the Extra Cellular Space of the brain was governed by a drift term, but the modern perception is that this drift term is in the very least negligible [2]. The drift term might be of importance in other applications, however, and something we should look into it.

How do we model random walks with drift?

A first instinct is to simply add 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 $\mathbf{d}$ 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.

$$\frac{\partial C}{\partial t} + \nabla \cdot \mathbf{j} = S \tag{2.19}$$

Where $\mathbf{j}$ 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 (2.20).

$$\frac{\partial C}{\partial t} = \nabla \cdot (D\nabla C) - \nabla \cdot (\mathbf{v}C) + S \tag{2.20}$$

Which in many cases will simplify to

$$\frac{\partial C}{\partial t} = D\nabla^2 C - \mathbf{v} \cdot \nabla C \tag{2.21}$$

In order to determine all the parameters of the convection diffusion equation 2.20 we will need to go through some of the calculations from chapter 2.1. 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 timestep. 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 2.22.

$$C(x,t) = \frac{N}{\sqrt{4\pi Dt}} \exp\left(-\frac{(x-vt)^2}{4Dt}\right) \tag{2.22}$$

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 $\sigma^2 = 2D\Delta t$.
The only problem with the solution 2.22 is that it is invalid for $t = 0$. In order to use it we will need an initial condition which fits the rest of the solution. We try and find this by first checking by simulation if it converges towards some initial solution as $t \to 0$ and if so, extrapolate this to $t = 0$

as the initial condition. Figure 2.1 shows the result of this little experiment, and as we clearly see it does diverge as $t \to 0$. However, it also converges to a Dirac delta function, $\delta(x)$ which is defined by its properties

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases}$$

and

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1$$

The Dirac delta function often sought as an initial condition in experimental setups for measurements on diffusion processes because of its compatibility with a variety of diffusion equations, and should do a very good job in our numerical setup as well.

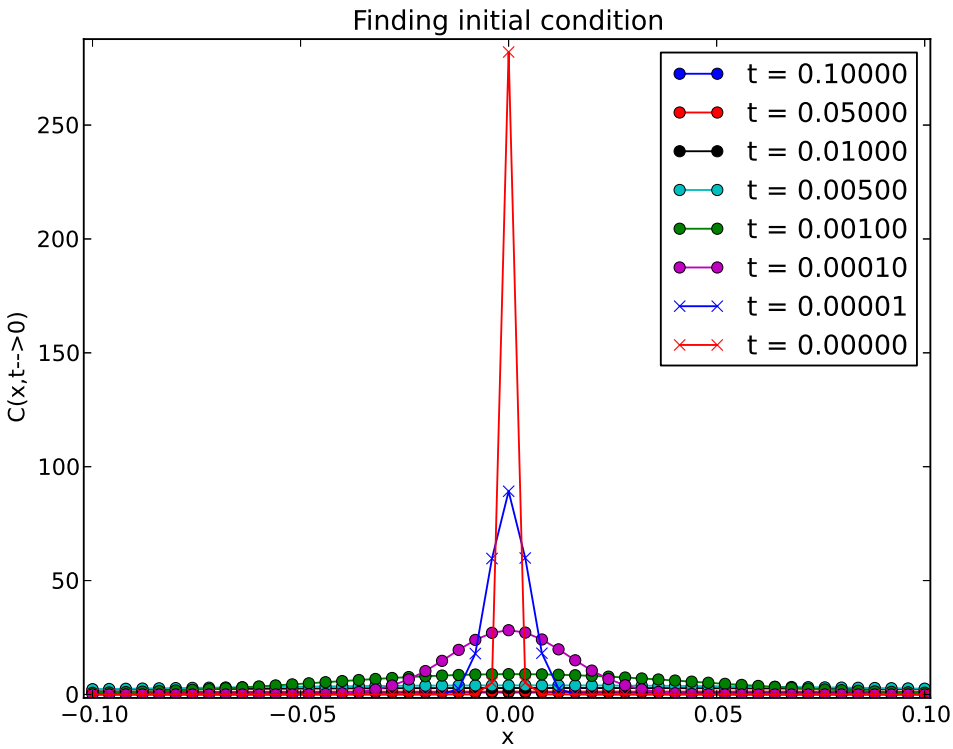


Figure 2.1: Experiment to determine the initial condition of equation 2.22

2.2 Some words about partial differential equations

2.2.1 Discretizing

To maintain a bit of generality we will look at the (potentially) anisotropic diffusion equation in 2d. The extension to 3d is trivial, as is the 1d version.

$$\frac{\partial u}{\partial t} = \nabla D \nabla u + f \tag{2.23}$$

where f is some source term. The final expression and scheme will depend on how we chose to approximate the time derivative, but the spatial derivative will mostly have the same approximation.
We start off by doing the innermost derivative in one dimension. The 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 we have made the approximate derivative around the point x_i. We then set $\phi(x) = D\frac{du}{dx}$ and do the second derivative

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

and insert for ϕ

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

Since we can only evaluate the diffusion constant at the mesh points (or strictly speaking since it is a lot simpler to do so) we must approximate $D_{i\pm 1/2} \approx 0.5(D_{i\pm 1} + D_i)$. Inserting this gives us

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

The Forward Euler discretization has one drawback which is its instability for "large" Δt. A common tradeoff for explicit numerical schemes. We could sacrifice ease of implementation for unconditional stability in our scheme. At

least in 1d the implementation is not that much worse either if we chose the Backward Euler discretization for an isotropic (homogeneous) equation. The discretization itself is found in equation 2.24.

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{2.24}$$

The advantage of this discretization is that it results in a tridiagonal linear system, which we can easily find if we just insert the first couple of steps.

$$\begin{aligned}
\frac{u_i^n - u_i^{n-1}}{\Delta t} &= D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \\
u_i^n - \frac{D\Delta t}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) &= u_i^{n-1} \\
u_i^n\left(1 + 2\frac{D\Delta t}{\Delta x^2}\right) - u_{i-1}^n\frac{D\Delta t}{\Delta x^2} - u_{n+1}^n\frac{D\Delta t}{\Delta x^2} &= u_i^{n-1}
\end{aligned}$$

If we insert for a few steps we see that this takes the form of

$$\begin{pmatrix}
\left(1+2\alpha\right) & -\alpha & 0 & \dots & & 0 & 0 \\
-\alpha & \left(1+2\alpha\right) & -\alpha & 0 & \dots & & 0 \\
0 & \ddots & \ddots & \ddots & 0 & \dots & 0 \\
0 & \dots & & & 0 & -\alpha & \left(1+2\alpha\right) & -\alpha \\
0 & \dots & & & & 0 & -\alpha & \left(1+2\alpha\right)
\end{pmatrix}\mathbf{u}^n = \mathbf{u}^{n-1}$$

where we have set $\alpha = \frac{\Delta t}{D\Delta x^2}$.

$$\mathbf{A}\mathbf{u}^n = \mathbf{u}^{n-1}$$

This can be solved very efficiently by a sparse Gaussian elimination, which is quite simple to implement once you take the boundary conditions into account. We implement the Backward Euler discretization in order to efficiently simulate for longer times, and by extension to test the effects the longer time steps will have on the random walk model. More on the tridiagonal Gaussian elimination in the appendix.

We would, of course, also like an implicit solver in 2 and 3d as well to make sure that we can take longer time steps here as well. Doing the BE discretization in 2d gives us a slightly more difficult linear problem (eq A.2), which is still sparse, but we do not have any simple, efficient solver for five-band matrices. Further difficulties arise when we attempt to implement Neumann boundary conditions of girds larger than 3×3 since the off-diagonal elements do not "hit" their intended matrix-elements. In order to use the 2D

BE scheme we must either invert a cubic matrix and multiply it by a normal matrix, or we will have to start rearranging the elements in the vectors (converted matrices) and the entries in the linear-problem matrix correspondingly. All things considered, we will therefore try and find another implicit scheme for our 2D problem. There is another discretization which lets us utilize our tridiagonal solver, and is implicit and unconditionally stable in 2d, the Alternate Direction Implicit (ADI) scheme. The ADI scheme divides the time step in two and only propagates one of the spatial derivatives at a time, making it very similar to the Crank Nicholson scheme. In fact, the truncation error in the ADI scheme is of second order, just as in the CN scheme, but this is (as discussed earlier) of small importance in this thesis. Discretizing the simple diffusion equation by the ADI scheme is done as follows.

$$\frac{u^{n+1/2} - u^n}{\Delta t/2} = D\left(D_x D_x u^{n+1/2} + D_y D_y u^n\right)$$
$$\frac{u^{n+1} - u^{n+1/2}}{\Delta t/2} = D\left(D_x D_x u^{n+1/2} + D_y D_y u^{n+1}\right)$$

We can write this as two standard equations on the form

$$\mathbf{A}_{1/2}\mathbf{u}^{n+1/2} = \mathbf{u}^n$$
$$\mathbf{A}_1\mathbf{u}^{n+1} = \mathbf{u}^{n+1/2}$$

Note that the indices are only intended as labels. For a 3×3 mesh the equations are shown in the appendix.
After more careful calculations, we find that the ADI discretization does not, in fact, produce a tridiagonal linear problem when we introduce the Neumann boundary conditions. While the scheme is still a viable alternative for other applications, considering its second order convergence, there is no benefit in using it for our application and we can concentrate on the normal Backward Euler discretization using a pre-exisiting solver for the linear problem as a first approach.
As an improvement we note that the Matrix in the linear problem stays the same as long as Δt is unchanged. This means that we can get away with assembling the matrix once, and it lets us do an LU decomposition which will reduce the number of FLOPS needed per time-step from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$ where N is the product of the number of mesh-points in x and y direction. In other words, since we have chosen the same resolution in x and y direction (and the time dependence can be related to this resolution as well for the explicit schemes) if we chose a resolution of n we will need n^4 FLOPS per time-step as opposed to the n^2 for the explicit scheme.

2.2.2 Stability

In section 2.2.1 we used the Forward Euler approximation to the time derivative. Unfortunately the resulting scheme is potentially unstable, as we shall now see. We start out by assuming that the solution $u(x,t)$ is on the form

$$u(x,t) = A^n \exp(ikp\Delta x) \tag{2.25}$$

where $i^2 = -1$ is the imaginary unit and A^n is an amplification factor which, for the solution 2.25 ideally should be $\exp(-\pi^2 t)$, but will be something else in the numerical case. We notice that we must have $|A| \leq 1$ if u is to not blow up. Inserting 2.25 in the simplified version of the variable coefficient scheme (where the coefficient is constant) gives us the following

$$\begin{aligned}
&\exp(ikp\Delta x)\left(A^{n+1} - A^n\right) = \\
&\qquad A^n \frac{D\Delta t}{\Delta x^2}\left(\exp(ik(p+1)\Delta x) - 2\exp(ikp\Delta x) + \exp(ik(p-1)\Delta x)\right) \\
&A^n \exp(ikp\Delta x)\,(A-1) = \\
&\quad A^n \exp(ikp\Delta x)\frac{D\Delta t}{\Delta x^2}\left(\exp(ik\Delta x) - 2 + \exp(-ik\Delta x)\right)
\end{aligned}$$

Using the well known identities $\exp(iax) + \exp(-iax) = \frac{1}{2}\cos^2\left(\frac{ax}{2}\right)$ and $\cos^2(ax) - 1 = \sin^2(ax)$ gives us

$$A - 1 = \frac{D\Delta t}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right) \tag{2.26}$$

We now insert for the "worst case scenario" $max(\sin^2\left(\frac{k\Delta x}{2}\right)) = 1$

$$A = \frac{D\Delta t}{2\Delta x^2} + 1 \implies \Delta t \leq \frac{\Delta x^2}{2D} \tag{2.27}$$

In 2d this criterion is halved, and for the anisotropic case we must insert for the maximum value of D which, again, will be the "worst case scenario".
If we insert the same solution (eq. 2.25) in the BE scheme we get

$$\begin{aligned}
&\exp(ikp\Delta x)\left(A^n - A^{n-1}\right) = \\
&\qquad A^n \frac{D\Delta t}{\Delta x^2}\left(\exp(ik(p+1)\Delta x) - 2\exp(ikp\Delta x) + \exp(ik(p-1)\Delta x)\right) \\
&A^n \exp(ikp\Delta x)\left(1 - A^{-1}\right) = A^n \exp(ikp\Delta x)\frac{D\Delta t}{\Delta x^2}\left(\exp(ik\Delta x) - 2 + \exp(-ik\Delta x)\right)
\end{aligned}$$

which leads to

$$A = \frac{1}{1 + \frac{D\Delta t}{\Delta x^2}} \tag{2.28}$$

Equation 2.28 is smaller than 1 for all $\Delta t > 0$ which means that the scheme is unconditionally stable.

2.2.3 Truncation error

As we know the numerical derivative is not the analytical derivative, but an approximation. This approximation has a well defined residual, or truncation error which we can find by Taylor expansion.

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

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

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

We can do better than this by using another discretization scheme for the PDE, but in our case the PDE is not the only error source seeing as we will combine it with a random walk solver. 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. So far the error seems to behave as expected, meaning that introducing very many walkers might reduce the error to $\mathcal{O}(\Delta t^2)$ if the number of walkers, N is proportionate to $N \propto \frac{1}{\Delta t^2}$. Since $\Delta t \leq \frac{D\Delta x^2}{2}$ by the stability constraint (in 1D), we will already for small meshes of some 20 points need to introduce ~ 600000 walkers per unit "concentration" per meshpoint in the walk-area. This will be such a costly operation that it will not necessarily be worth it.
The spatial derivative also has a well defined residual which is found by Taylor expansion.

$$R = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} - u''(x_i) \tag{2.29}$$

Remember Taylor expansion of $u(x-h) = \sum_{i=0}^{\infty} \frac{1}{i!}\frac{d^i}{dt^i}u(x)(-h)^i$

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

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 2.2.2 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.

2.2.4 Extension to 3 spatial dimensions

As we see in the appendix the assembly of the linear problem that arises from the BE discretization in 2d with a variable diffusion constant is a rather messy thing. While a 2d simulation might tell us a great deal, and be sufficient for many applications such as modeling of experimental setups of diffusion in the ECS (in vitro experiments on very thin slices), we should at least look into an extension of our model to 3 dimensions. The BE discretization is very similar in 3d to the 2d case. In fact we must only add a term for the z-direction. If we translate the resulting expression to a linear problem we immediately come across the problem that our solution, U, is a cubic matrix and so we must have a 4-dimensional matrix to describe to system. The problem is solved by first of all describing the solution as a vector of matrices, and in turn as a vector of vectors (which is a matrix). We now have the same situation as we had in the 2d case, and by following the same procedure as before we arrive at exactly the same situation which is a normal linear problem. In fact, using block-matrix notation we can even write the problem in an identical way, as we have done in the appendix (eq. **??**). The matrix is now 7-band diagonal, but the LU-decomposition still does the trick.
Just as for the 2d case, the computiational cost per time step is of the order of $\mathcal{O}(N^2)$ where N is now the spatial resolution cubed ($N = n^3$). In other words the computiational cost is now quite high ($\mathcal{O}(n^6)$). For the actual decomposition the cost is $\mathcal{O}(N^3) = \mathcal{O}(n^9)$

2.2.5 Tridiagonal linear systems

We can solve a normal linear equation $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is not a sparse matrix, by Gaussian elimination.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \mathbf{x} = \mathbf{b} \tag{2.30}$$

Is reduced to

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & (a_{22} - \frac{a_{21}a_{12}}{a_{11}}) & (a_{23} - \frac{a_{21}a_{13}}{a_{11}}) & (a_{24} - \frac{a_{21}a_{14}}{a_{11}}) \\ 0 & (a_{32} - \frac{a_{31}a_{12}}{a_{11}}) & (a_{33} - \frac{a_{31}a_{13}}{a_{11}}) & (a_{34} - \frac{a_{31}a_{14}}{a_{11}}) \\ 0 & (a_{42} - \frac{a_{41}a_{12}}{a_{11}}) & (a_{43} - \frac{a_{41}a_{13}}{a_{11}}) & (a_{44} - \frac{a_{41}a_{14}}{a_{11}}) \end{pmatrix} \mathbf{x} = \tilde{\mathbf{b}} \tag{2.31}$$

and further to

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & (a_{22} - \frac{a_{21}a_{12}}{a_{11}}) & (a_{23} - \frac{a_{21}a_{13}}{a_{11}}) & (a_{24} - \frac{a_{21}a_{14}}{a_{11}}) \\ 0 & 0 & (\tilde{a}_{33} - \frac{\tilde{a}_{32}a_{23}}{\tilde{a}_{22}}) & (\tilde{a}_{34} - \frac{\tilde{a}_{32}a_{24}}{\tilde{a}_{22}}) \\ 0 & 0 & (\tilde{a}_{43} - \frac{\tilde{a}_{42}a_{23}}{\tilde{a}_{22}}) & (\tilde{a}_{44} - \frac{\tilde{a}_{42}a_{34}}{\tilde{a}_{22}}) \end{pmatrix} \mathbf{x} = \tilde{\mathbf{b}} \tag{2.32}$$

through the well known Gaussian elimination scheme until we have an upper triangular matrix. We can then do a backwards sweep to solve for one element of the unknown vector, $\mathbf{x}$ at a time. If the matrix $\mathbf{A}$ is tridiagonal, most of the entries will be zero, and so we will not only be calculating a lot of zeros, we will also be saving a lot of them in the matrix (should we be so stupid as to save the matrix). 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 implicit discretization gives us a set of linear equations, or a linear system, to solve at each timestep. The physics of the system gives us a special form of linear system, namely a band diagonal system, where the number of non-zero bands on the matrix is dependent on the number of spatial dimensions we are in. The one dimensional case gives us a tridiagonal system, which can be solved extremely efficiently by the "tridiag" function listed above. In two spatial dimensions we are not quite as fortunate as in one dimension. We 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 gives us a tridiagonal matrix, where the entries are $n \times n$ matrices. We can actually modify the tridiagonal solver from the one-dimensional case so we can use it on the block-tridiagonal system. The modified algorithm for the block-tridiagonal matrix 2.33 is listed in **??**, and is in fact only the linear algebra version of the "tridiag" function, replacing the $1.0/btmp$ -terms with $(B_i + A_i H_{i-1})^{-1}$.

$$\begin{pmatrix} B_0 & C_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & A_i & B_i & C_i & 0 & & 0 & 0 \\ 0 & \ddots & 0 & \ddots & \ddots & \ddots & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{n-1} & B_{n-1} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0^{n+1} \\ \mathbf{u}_1^{n+1} \\ \vdots \\ \mathbf{u}_i^{n+1} \\ \vdots \\ \mathbf{u}_n^{n+1} \end{pmatrix} = \mathbf{u}^n \quad (2.33)$$

which we can also express as $M\mathbf{x} = \mathbf{k}$. Block-matrices named B_i are tridiagonal, and the ones named A_i or C_i are strictly diagonal. The block-tridiagonal solver is taken from [], and is listed below.

There is a forward substitution

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

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, we only need to to the inversion once as long as the mass-matrix, M is unchanged so the expense is reduced. This should give us 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 better than 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 we can see in the appendix (eq. A.5) we can write this so it looks like the block tridiagonal linear system we got in 2d, and that could be solved by the block-tridiagonal solver. The difference is that the entries in the block-matrices A_i, B_i and C_i are block-matrices themselves meaning that we must work with $n^2 \times n^2$ matrices. All in all we should get a performance of around $\mathcal{O}(n^3)$ if we take care to save the inverted matrices from the forward substitution. This is about the same performance as the explicit scheme gives us, but again without the stability issue.

2.3 Combining the two solvers

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

2.3.1 The basic algorithm

The basic structure of the program is to have one solver-object which contains one PDE-solver for the normal diffusion equation, and a linked list of random walk-solvers and their relevant areas. Before we start we must add an initial

condition along with some parameters such as the diffusion constant (/tensor) and Δt, and we have the opportunity to mark areas on the mesh where we want random walk solvers. The method for adding walk-areas will map them to an index and set the initial condition for the walk. In the future we plan to add the possibility of setting boundary conditions and having anisotropy follow into the random walk solvers as well. At each timestep we call the solve-method of the combined solver, which in turn calls the solve method for the PDE-solver. We then loop over the random walk solvers and call their solve-methods. The results of these are inserted in the solution from the PDE using some routine (e.g. the average of the two) and the timestep is done. A schematic of the algorithm is provided in figure 2.2.

Figure 2.2: Schematic diagram of the algorithm.

section

2.3.2 Potential problems or pitfalls with combining solutions

There are a few obvious difficulties we can expect to run into in our planned project. Future ones will be added here as well.

- Different timescales
  The PDE-solver will be operating with some timestep 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 timestep 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 timestep on the PDE level, $\sum_{i=0}^{N} \Delta\tilde{t} = \Delta t$. It turns out, as we will see in section 2.3.3 that we can make sure the coupling is as good as it gets by restricting the step length of the walkers.

- Boundary conditions
  To combine the two models we will need to put restricting boundary conditions on the random walks. 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, so long as the walk-domain is not on the actual boundary of the whole system. We can assume that the number of walkers in the walk-domain is conserved for each PDE-timestep, 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 we have the desired concentration of walkers.

- 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 when the solution of the PDE takes negative values (which it might do). The solution to this is simply to store the signs of the solution to the PDE in an array, and send only positive values to the random walk solver. When we convert the number of walkers back to a PDE solution we still have the sign from before and can multiply the concentration by the sign it had in the last timestep.

- 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 we introduce a stochastic process, we will potentially also introduce fast fluctuations from one timestep to the next. In this case we are faced with a dilemma; on the one hand there is the smoothness of the solution to consider, on the other hand we have introduced the stochastic term believing that it adds detail to our model. The approach we use to this is to do some curve-fitting using both of the solutions. This will give us some difference between the two models and some smoothness.

- Number of timesteps on the random walk level
  As the timestep 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 timesteps on the RW level. Strictly speaking we do not have to do this, seeing as we

adjust the steplength of the walkers with respect to the timestep (see eq 2.37). As an initial value we put the number of timesteps 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 2.1.2) 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.

2.3.3 Probability distribution and timesteps

As we saw in section 2.1.2 the probability of finding a walker at a position x_i after some N timesteps (on the walk-scale) is (in the limit of large N) given as the Gaussian distribution. In our application, however, we are not interested in finding the walker at an exact position, but in an interval around the mesh-points sent to the walk-solver. This interval is (for obvious reasons) $x_i \pm \frac{\Delta x}{2}$ where Δx is the mesh resolution on the PDE level. We will also run the walk solver for some N timesteps 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 our distribution into

$$P(x_i \pm \Delta x, t_{n+1}) = \frac{1}{\sqrt{4\pi DN\Delta\tilde{t}}} \exp\left(\frac{(x \pm \Delta x)^2}{4DN\Delta\tilde{t}}\right) \tag{2.34}$$

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

$$C(x_i \pm \Delta x, t_{n+1}) = \frac{M}{\sqrt{4\pi DN\Delta\tilde{t}}} \exp\left(\frac{(x \pm \Delta x)^2}{4DN\Delta\tilde{t}}\right) \tag{2.35}$$

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

Looking at the difference in timestep size between the two length scales we see from equation 2.4 that the stepsize on the random walk scale is dependent on the variance in the actual steps (This is in principle the Einstein relation).

$$\sigma^2 = \langle \Delta x^2 \rangle = 2DN\Delta\tilde{t} \implies \Delta\tilde{t} = \frac{\langle \Delta x^2 \rangle}{2DN} \tag{2.36}$$

Equating this with 2.8 gives us a first order approximation to the steplength, l

$$\begin{aligned} \langle \Delta x^2 \rangle &= 4pqNl^2 = 2DN\Delta\tilde{t} \\ l &= \sqrt{2D\Delta\tilde{t}}. \end{aligned} \tag{2.37}$$

Of course this is assuming that we use a random walk algorithm of fixed steplength.

2.4 Geometry

Any finite difference method is problematic to solve on anything else than a rectangular grid. When we additionally use an implicit FD method we will add a "demand" of having a square grid as well. Fortunately the implicit solvers we use are stable.
In the physical scope of this thesis we do not find any rectangular shapes to apply our system to. Actually we do not have any well defined geometry to apply our system on, but that is of less importance. The developed software lets us specify an initial condition

Chapter 3

Analysis

3.1 Some discussion

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

In this numerical setup we will potentially introduce several new error sources in addition to the normal errors introduced by numerical solution of any equation (see section 2.2). When a part of the solution acquired is replaced by the solution from another model, which in this case is stochastic, we will change the initial condition to the next iteration in time. This might have a number of effects on our final solution. When we solve a differential equation numerically we only get an approximation to the actual solution because we are using approximate derivatives (see figure **??**). To investigate the error we are introducing we will first need to test that the truncation error of the numerical PDE solver behaves as expected.

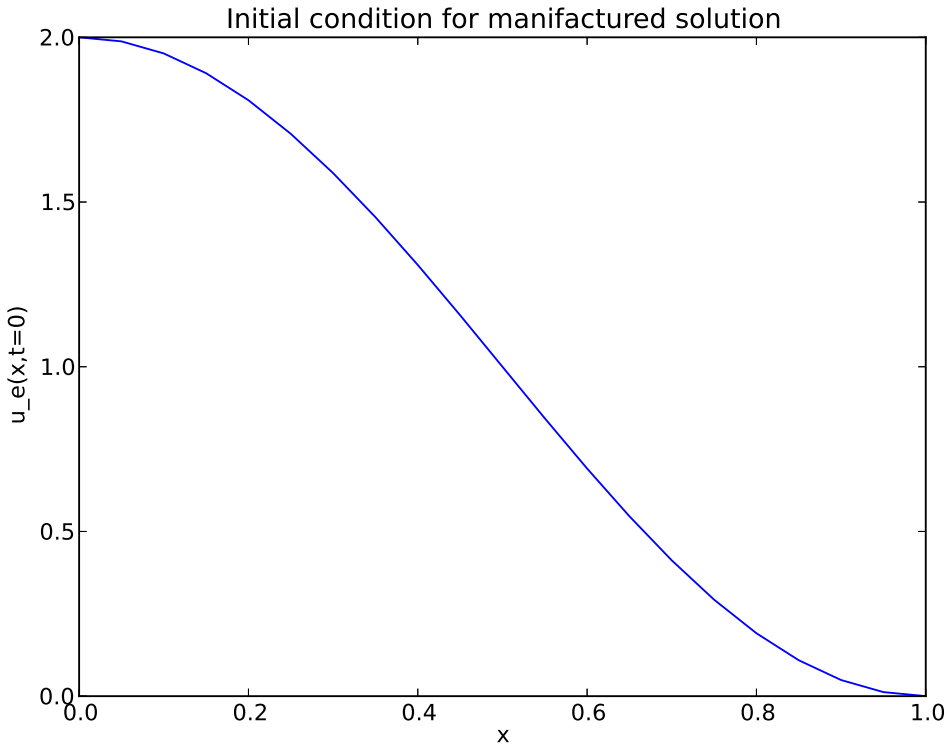


Figure 3.1: Initial condition of manufactured solution in 1d and the simulation.

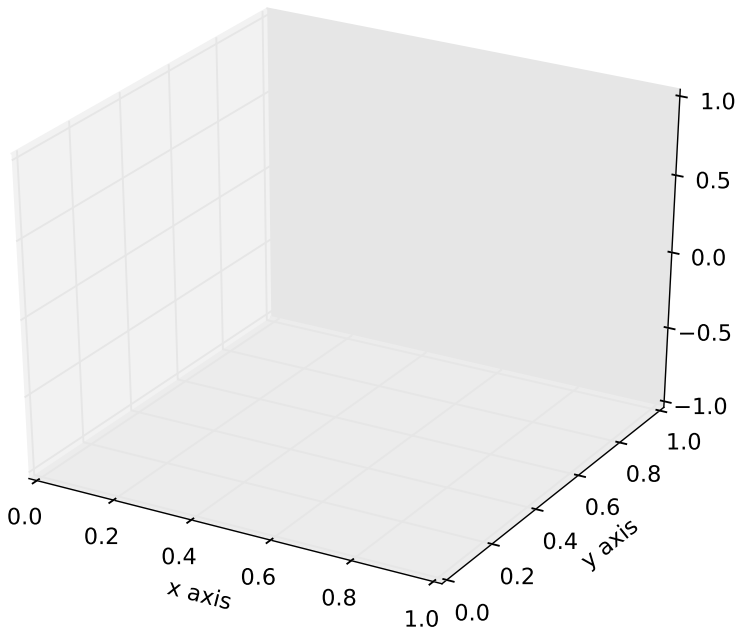


Figure 3.2: Initial condition of manufactured solution in 2d and the simulation.

3.1.1 The error estimate

Before we do anything we should specify what we use to measure the error. Throughout this thesis the term error is used quite lazily, but unless something else is specified we refer to the expression

$$err = max\left(||u_e - u||_2\right) \tag{3.1}$$

where u_e is the exact (manufactured) solution to the equation, and u is the result from the numerical simulation. We use this error-estimate because it is time-dependent, thus letting us explicitly see how the error evolves over time. The error is calculated over the entire mesh, letting us see clearly if the error from the random-walk areas are dominating, or (otherwise) how the PDE-scheme is holding up. Other error measures that might come in handy later will be the integrated spatial error as a function of time

$$err = ||\sum u_e - u||_2 \tag{3.2}$$

and the same error integrated in time as well

$$err = \sum || \sum u_e - u||_2 \tag{3.3}$$

3.2 Manufactured Solutions

A normal way of checking that our scheme of choice is implemented correctly is by making an exact solution to the equation and checking that the error is of the expected order. As a first, simple implementation we have worked with the explicit Forward Euler discretization of the simplest form of the diffusion equation (eq. 2.13). This discretization is expected to have an error-term of the order of Δt, which again is limited by a stability criterion. We can now decide that the solution to equation (eq. 2.13) should be

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

which satisfies our equation if we set the diffusion constant to 1.

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

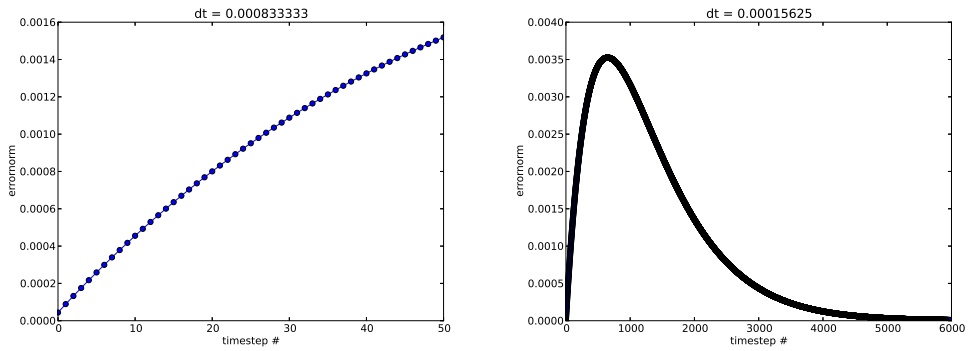
$$-\pi^2 e^{-t\pi^2}\cos(\pi x) = -\pi^2 e^{-t\pi^2}\cos(\pi x) + 1 \implies 1 = 1 \tag{3.6}$$

As we saw in section 2.2.3 the Forward Euler scheme is expected to have an error of the order of Δt in time. 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 the error term coming from the time derivative due to the stability criterion (eq. 2.2.2), which is also of the order Δx^2. Testing only the scheme first, in 1D we get the following plot 3.3a of the maximum of the absolute value of the difference between the exact solution and the numerical solution to the equation, showing us that the error is linear in the first part of a simulation. For longer simulations, however, we expect the analytic solution to take a steady state which we find 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. Figure 3.3b shows a longer simulation of the equation which includes the converging part as well. Looking closely at figure 3.3b, which has a smaller Δt by a factor of ~ 6 with respect to figure 3.3a, we can note that the error norm 0.0015 is reached after some $100 - 200$

time steps, which is later than in figure 3.3a. Although this is not an exact measurement, it is another indication that the implementation is correct.



(a) (b)

Figure 3.3: Numerical error for 1D Forward Euler discretization of the PDE. Nothing else is done to the simulation.

We then introduce an area on the domain where we switch models from the normal PDE to an average of the PDE solution and the result of a random walk simulation where the initial condition is the last time step from the PDE converted to walkers by the conversion rate given in equation 3.8. In this case we have used the parameters $a = 3$, $\Delta t = \frac{\Delta x^2}{3.0}$, $\Delta x = \frac{1}{20}$. These parameters makes one unit of $u(x,t)$ equal to some 1000 walkers.

$$C_{ij} = \frac{a}{\Delta t} U_{ij} \tag{3.8}$$

Mostly we will rewrite equation 3.8 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 3.9. One "unit" of U_{ij} will directly correspond to Hc random walkers.

$$Hc = \frac{a}{\Delta t} \implies C_{ij} = Hc \cdot U_{ij} \tag{3.9}$$

The area where the model has been replaced is between $x = 0.6$ and $x = 0.7$, which is three mesh points. In the same way as for only the simple 1D PDE case we compare the combined numerical solution from the two models to the exact solution. Figure 3.4 shows that the error is still of the order of Δt, and the difference between the two models are negligible.

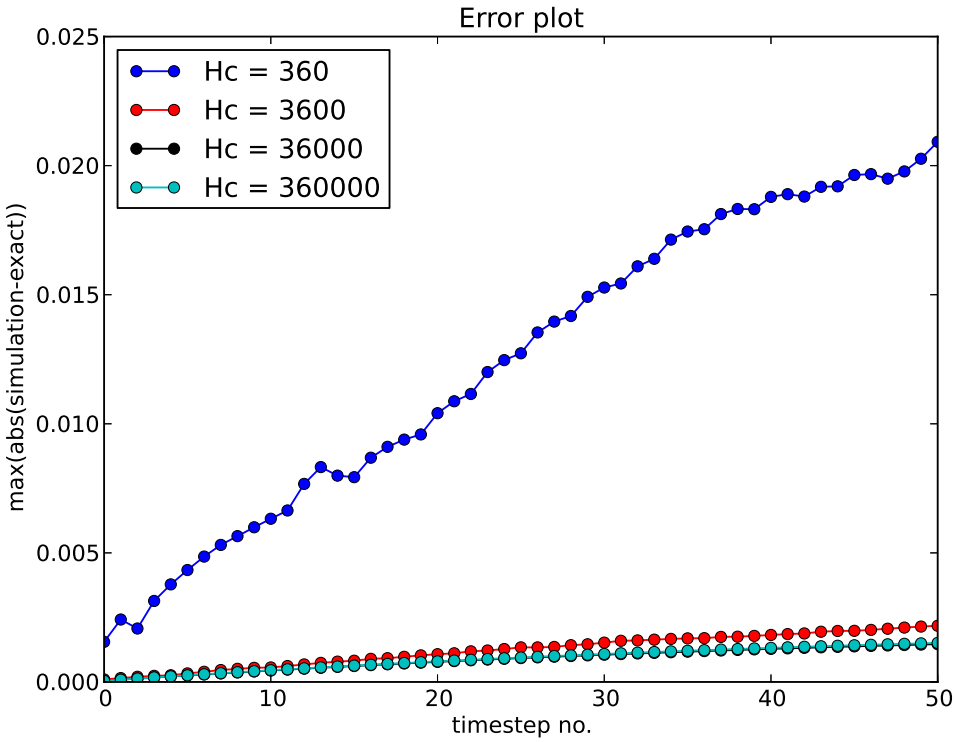


Figure 3.4: Numerical error for 1D Forward Euler discretization combined with random walk model between $x = 0.6$ and $x = 0.7$. Other parameters of importance are $\Delta t \approx 0.00083333$, $\Delta x = 0.05$.

As we can see from figure 3.4 increasing the number of walkers that each "unit concentration" corresponds to has a positive effect on the error norm up to a certain point. After we reach $Hc \sim 10^4$ the error is dominated by the truncation error of the Forward Euler scheme. This tells us that the error associated with introducing a random walk model on parts of the mesh behaves roughly as we hoped; it tends to zero for large numbers of walkers. Meaning of course that the calculations in section **??** are not hopeless.

We can also do a convergence test to see that the error is approximately of the expected order. This is done by doing several simulations with different values for Δt and comparing the errors by equation 3.20. A result of such an experiment for the FE scheme using the Δt values listed below is found in figure 3.6. The expected value of r is approximately 1. The result is not perfect, but still close to 1.

```
dt = [1e-4,1e-5,1e-6,1e-7,1e-8]
```

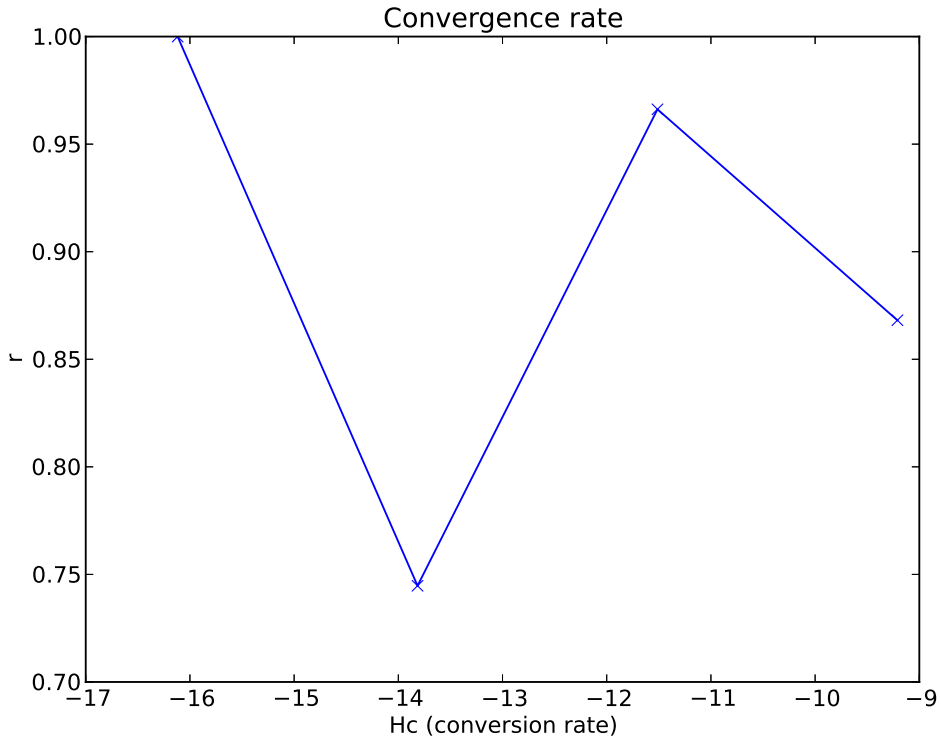


Figure 3.5: Convergence test for the FE scheme. The x axis is $\ln(\Delta t)$.

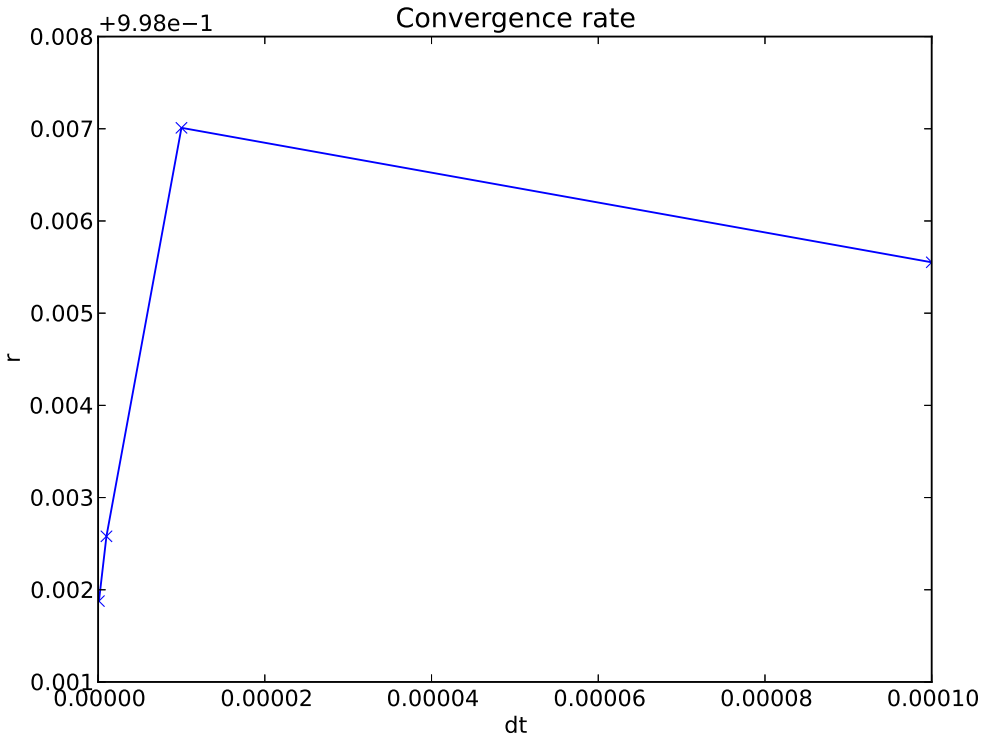


Figure 3.6: Convergence test for the BE scheme. The x axis is $\ln(\Delta t)$, the y-axis (though a little hard to see) is zoomed in around r=1.

3.2.1 Exact numerical solution

The FE scheme also has an exact solution seeing as it is in fact a difference equation. We can find this (for this exact initial condition 3.1) if we formulate the scheme as

$$u^{n+1} = D\Delta t u^n_{xx} + u^n \tag{3.10}$$

and insert for the first few iterations.

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

Which we can generalize to

$$
u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i u^0_{2ix} \tag{3.11}
$$

where we have

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

from the initial condition. This finally gives us the exact numerical solution of the FE scheme in time.

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

Notice that we have not used the approximation to the spatial derivative, but rather the analytical derivative. We can find the approximation that the computer uses to the spatial derivative in the same way as for the time derivative.

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

We can immediately see that this pattern continues, and we get the general formula in equation 3.13 for the exact numerical solution.

$$
u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \frac{2^i}{\Delta x^{2i}} \left(\cos(\pi\Delta x) - 1\right)^i \cos(\pi x) \tag{3.13}
$$

We expect the FE scheme to represent this solution more or less to machine precision, at least to 15 digits. There are, however two issues with the solution 3.13:

- Δx^{2i} will quickly tend to zero, and the computer will interpret it as zero. This will cause division by zero, which again 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. We will eventually (for $n >\sim 170$) meet overflow. Figure 3.7 shows that at least for relatively few time steps we can drop the troublesome terms.

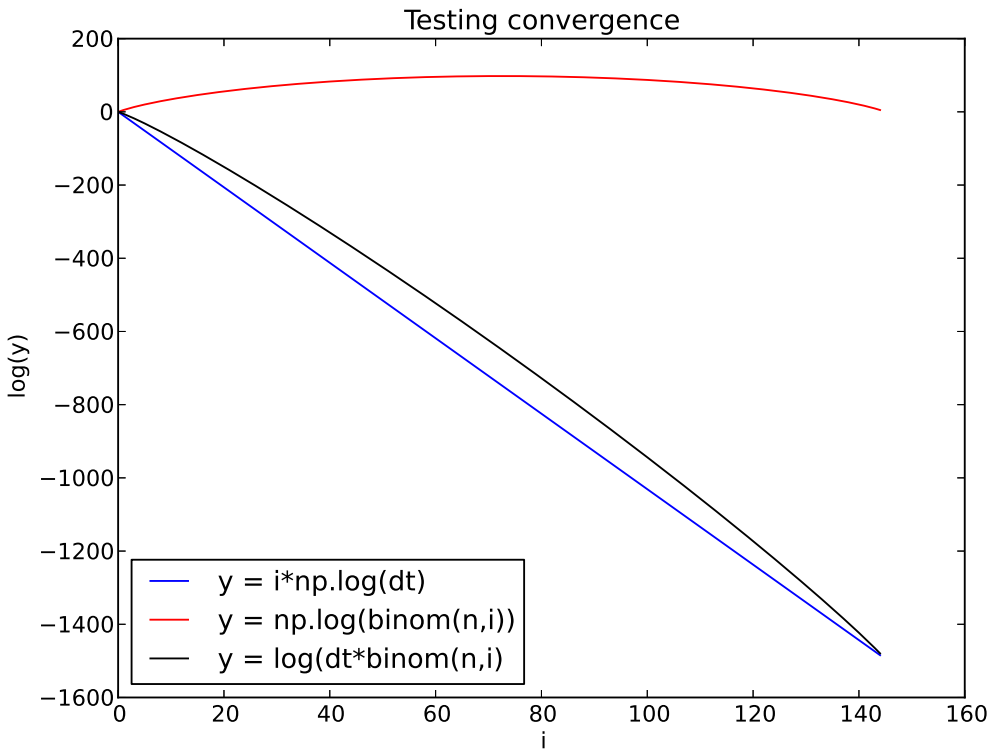


Figure 3.7: Testing the relation between $(D\Delta t)^i$ and the binomial coefficients.

The results from testing the FE scheme are found in figure 3.8. We see that the error in the worst case is about an order of magnitude worse than we 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.

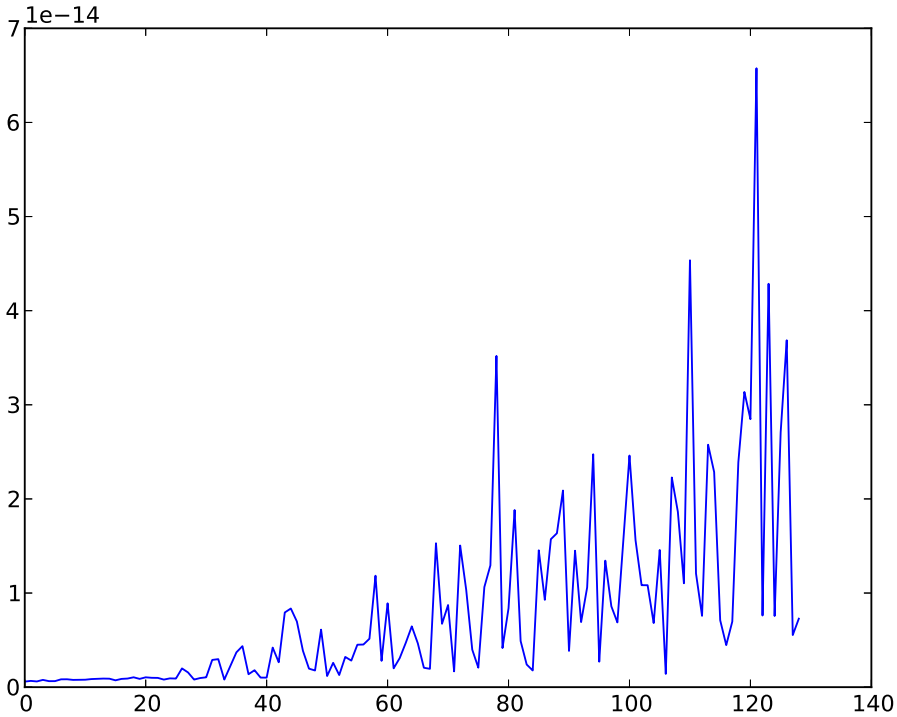
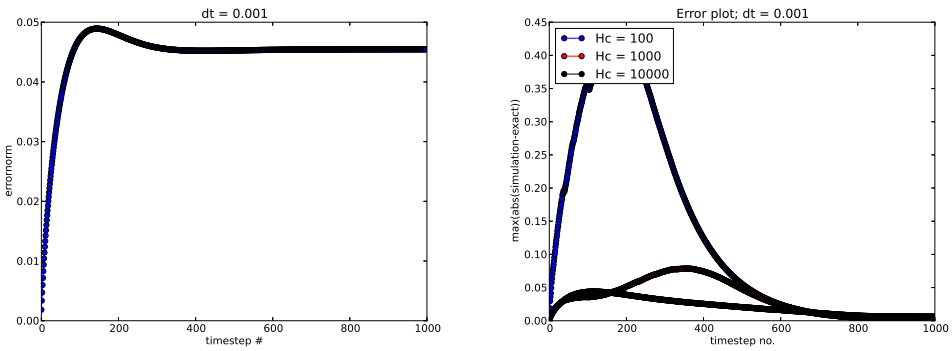


Figure 3.8: Error plot for 1d FE scheme compared to the exact numerical solution 3.13 with the modifications suggested earlier.

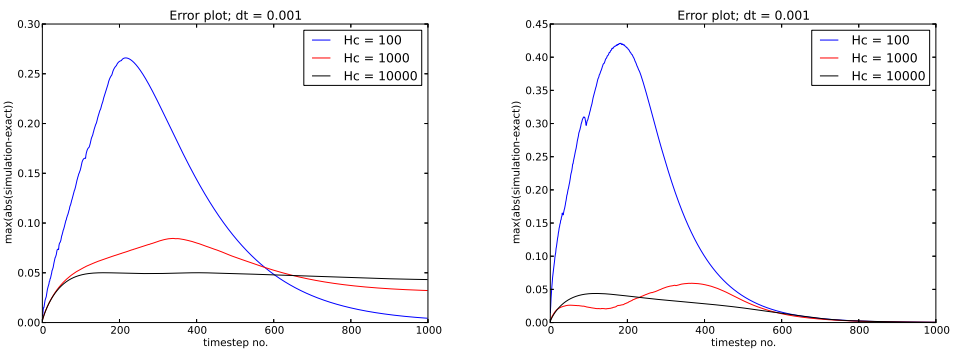
3.2.2 Increasing the time step and the relative size of walk-area

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 3.9a shows the error norm of a simulation of the simplest diffusion equation 2.13 discretized by the Backward Euler scheme 2.24 using a time step which would make the Forward Euler discretization unstable (There is something strange about its convergence). Figure 3.9b 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 we can turn to figures 3.10a and 3.10b which have 5% and 35% of the mesh points affected by walkers.



(a) (b)

Figure 3.9: Numerical error for 1D Backward Euler discretization of the PDE. In figure b there has been added walkers to the solution in the area $x \in [0.5, 0.7]$.



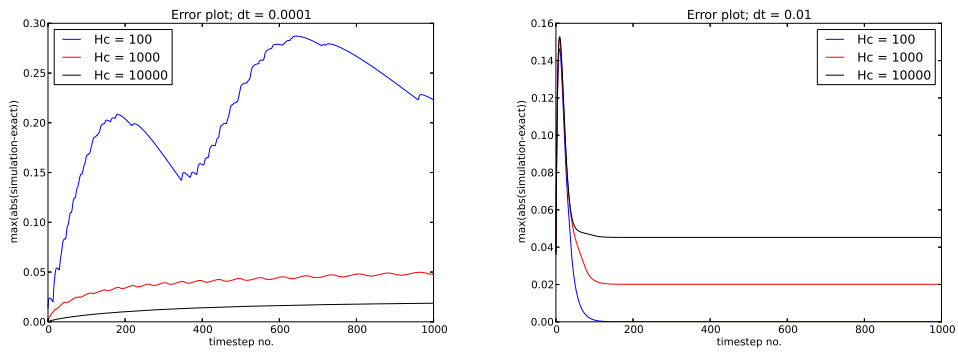
(a) Having walkers on 5% of the mesh points.

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

Figure 3.10: The effect of increasing the size of the walk area for a fixed $\Delta t = 0.001$ using the BE discretization.

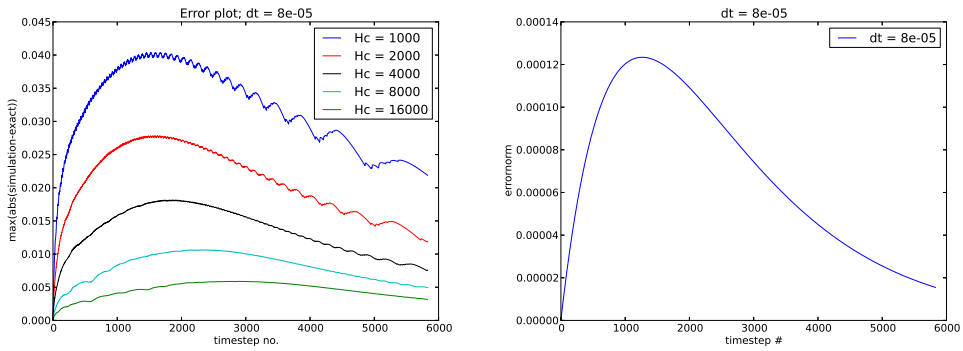
These experiments have been done using the Backward Euler discretization so that we can simulate for a longer time and still see the effects to their full extent. We have also investigated the effects of changing the time step (also using the BE discretization to avoid instabilities and be able to do longer simulations). The results are summarized in figure 3.11. We notice something a bit unexpected in figure 3.11b. Unlike almost all the other comparable plots, it seems that using the least amount of walkers gives the best result here. This might be because the system quickly reaches its steady

state, and will then be very well described by the continuum model. Having a small conversion factor, Hc, will mean that very quickly there will be no walkers which sort of ruins the point. This particular equation has steady state $u(t \to \infty, x) = 0$, and so not having any walkers will be perfect. What we should read from this figure is rather that the simulation with the most walkers converges to an acceptable error, and that this is achieved just as fast as for the other two simulations.



(a) Using a normal $\Delta t = 0.0001$. (b) Using a large $\Delta t = 0.01$.

Figure 3.11



(a) (b)

Figure 3.12: The deterministic error and the error from simulations with walkers for long simulations.

3.2.3 The effects of adding drift to the walkers

As shown in section 2.1.5 adding drift to the walkers will modify our model to represent the convection diffusion equation (2.20) rather than the simple

diffusion equation. In our analysis so far we have completely ignored the spatial truncation error because it is of second order, and the truncation error in time is of first order. When discretizing the convection diffusion equation however, we must take care to use an approximation to the first order spatial derivative that has a truncation error of second order.

Otherwise the truncation error in this term will completely dominate seeing as $\Delta t < \Delta x$. We must also find a new stability criterion for the scheme.
As for now, the Leap-frog discretization will do (though it is not by far a perfect choice seeing as it is unstable). As a side note we can also note that the Neumann boundary condition will be very clear in this scheme. $\frac{\partial C}{\partial n} = 0 \implies \frac{\partial C}{\partial x} = 0$ on the boundary, leading to $C_{-1} = C_1$ on the boundary and canceling the drift term on the boundary.

$$C^{n+1} = \frac{D\Delta t}{\Delta x^2}\left(C_{i+1}^n - 2C_i^n + C_{i-1}^n\right) - \frac{v\Delta t}{2\Delta x}\left(C_{i+1}^n - C_{i-1}^n\right) + C^n \tag{3.14}$$

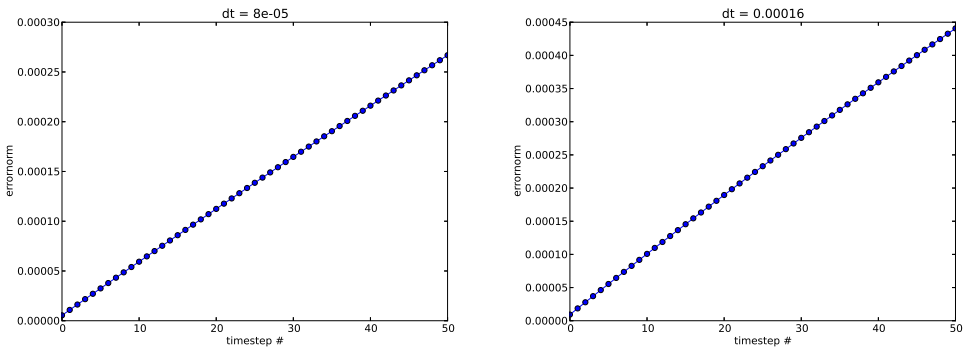
The truncation error for the first order derivative using the Leap-frog scheme is obtained as follows

$$\begin{aligned} u(t, x+\Delta x) &= u(t,x) + \frac{\partial u(t,x)}{\partial x}\Delta x + \frac{\partial^2 u(t,x)}{2\partial x^2}\Delta x^2 + \mathcal{O}(\Delta x^3) \\ u(t, x-\Delta x) &= u(t,x) - \frac{\partial u(t,x)}{\partial x}\Delta x + \frac{\partial^2 u(t,x)}{2\partial x^2}\Delta x^2 + \mathcal{O}(\Delta x^3) \end{aligned}$$

which we recognize as the same error we got from the second order spatial derivatives.
If we now do the same analysis as we have already done, by finding a manufactured solution to the convection diffusion equation, implementing a numerical scheme like 3.14 to solve it after and taking the error norm.
The error norm does not have to go as Δt, but it must be halved (approximately) if we halve Δt. Figure **??** shows two simulations of equation 2.20 for $D = 1$ and $v = 1$ compared to the manufactured solution 3.4 without adding walkers. Before the simulation we must find a source term so the manufactured solution will solve the equation.

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

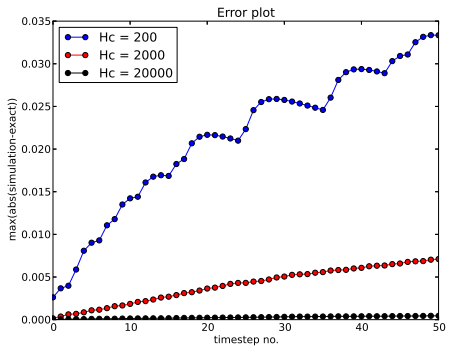
Where $f(x,t) = \exp\left(-\pi^2 t\right)\tilde{f}(x)$.



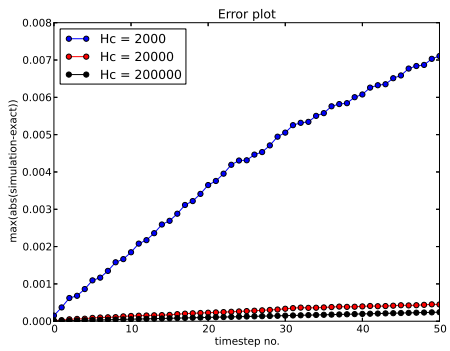
(a) (b)

Figure 3.13: Verification of Convection diffusion equation implementation

As we see from figure 3.13 the error norm is halved when Δt is roughly halved, just as we expected. It is also of the order of Δt, which is nice. We can then advance to testing the effect of adding an area of walkers for different values of Hc as before. Note that we can no longer use the manufactured solution now if we wish to add drift to the walkers because we have forced the solution to fit the equation by tampering with the source term. The effect of



(a)



(b) Adding more walkers still has the desired effect on the error, suggesting that the central limit theorem is at work behind the scenes.

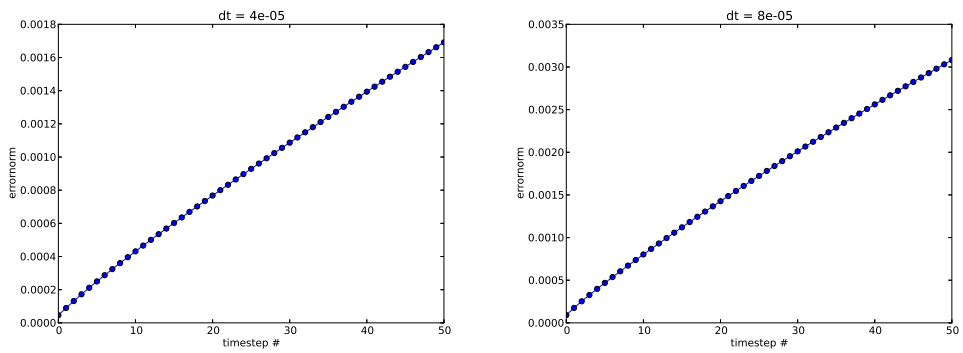
Figure 3.14: The figure shows the effect adding walkers influenced by drift has on the error. The drift term is the step length, which is quite large. The strangest thing is, however, that changing the direction of the drift has little effect on the error, suggesting that the drift term is completely wrong in relation to the steplength. $\Delta t = 8e - 5$

3.2.4 Anisotropic diffusion

As we discussed in section 2.1.4 we should also be able to model diffusion where the diffusion constant is not a constant. The scheme we derived in section 2.2.1 already takes this into account, and so all we need to do is to add the drift term which was discussed in section 3.2.3 to the scheme. Like before we should verify that the scheme solves the equation to the expected accuracy by using a manufactured solution 3.4 and tweaking the source term so this function solved the equation 2.20. When $v = 0$ and $D(x) = \pi x$ the source term becomes

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

Once again $f(x,t) = \exp\left(-\pi^2 t\right) \tilde{f}(x)$. Figure 3.15 shows the error norm of the result of simulations of this equation with different values of Δt.



(a) (b)

Figure 3.15: Verification of anisotropic diffusion equation implementation

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

3.3 2D

Doing the same tests is 2D gives slightly different results; adding a 2D walk-domain has an influence on the error, but a rather small one. This can, however be tweaked by increasing the conversion parameters.

$$u(x, y, t = 0) = \cos(\pi x)\cos(\pi y) \tag{3.15}$$

The exact numerical solution of the FE scheme can be found in equation 3.16, and again we expect the scheme to be able to reproduce this to more or less machine precision. The result of a test simulation of this, using the initial condition in equation 3.15, is shown in figure 3.16. Again, as in we did in 1d, we see that although the error is very small, and start out with machine precision, it does increase and even more than in the 1d case. This is *probably* because of the terms we have do drop in the exact numerical solution due to overflow and so on, which accumulate in the numerical solution from the scheme. 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.

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \left[2^{i-1} \cos(\pi x)\cos(\pi y) \left(\frac{(\cos(\pi\Delta x))^i}{\Delta x^{2i}} + \frac{(\cos(\pi\Delta y))^i}{\Delta y^{2i}} \right) \right] \tag{3.16}$$

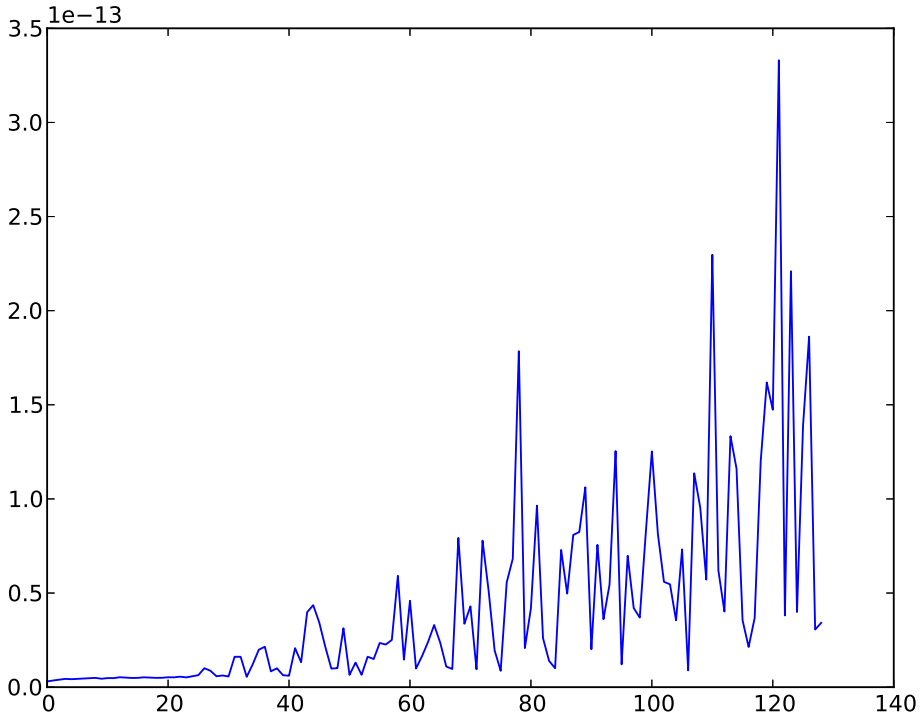


Figure 3.16: Numerical solution from the FE scheme versus the exact numerical solution of the FE scheme in 2d. we have used a Δt which is almost on the stability criterion, $\Delta t = \frac{\Delta x \Delta y}{5} = 8e - 05$.

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 3.20) for smaller Δt. The results of this test are shown in figure 3.17 and it does converge nicely to one.

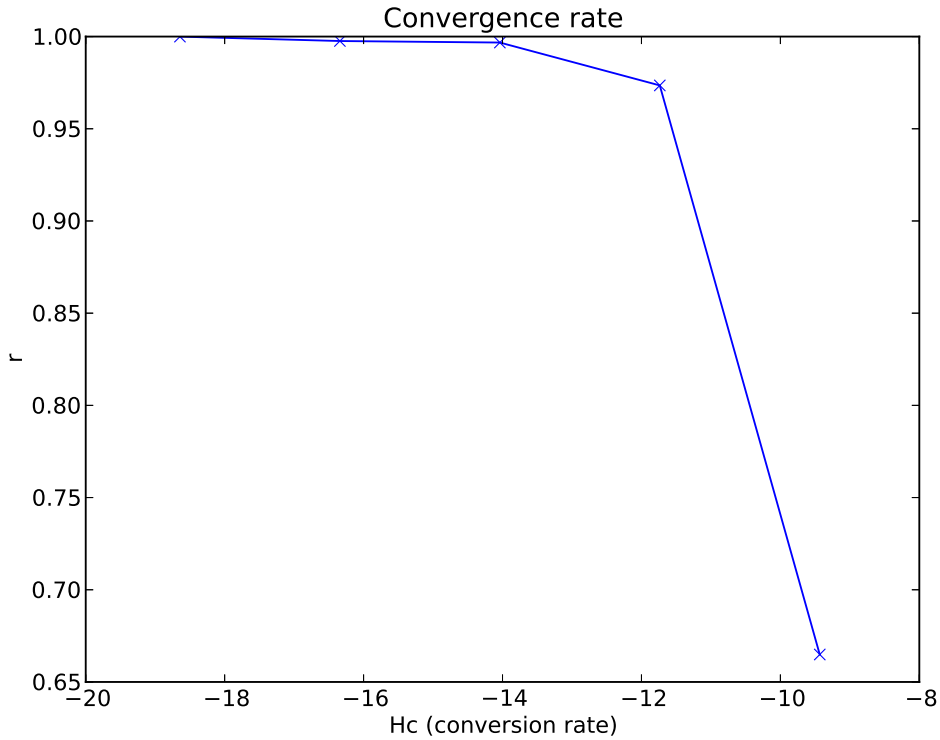
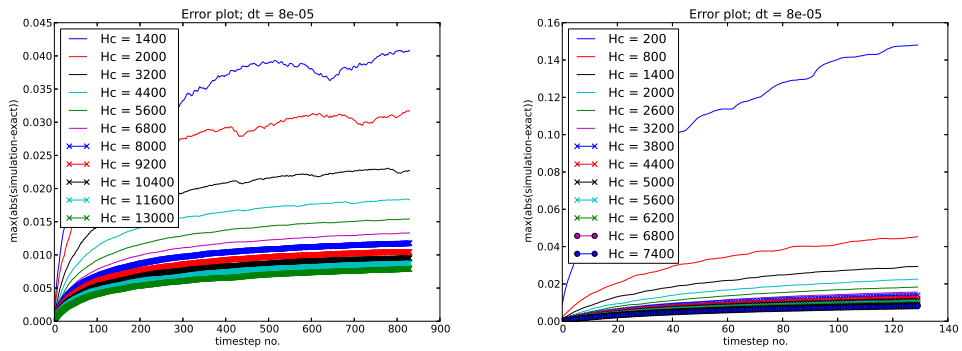


Figure 3.17: Convergence test for the FE scheme in 2d using Δt ranging from the stability criterion $\frac{\Delta x \Delta y}{5}$ to the same ratio divided by 100000 in increments of 10^{-1}.

3.3.1 Including random walks

In the same way as in figure 3.4 we can test the effect of introducing walkers in part of the mesh. In figure 3.18 we have more or less recreated the plot from the 1d case, only simulating for more time steps and with more walkers. First of all we note that we in 2d will need quite a lot more walkers than in 1d in order to get a decent error. Secondly, and more importantly, because we have done longer simulations, we notice that the error is completely dominated by the walkers. In the simulation in question the rectangle x0=0.4, x1=0.7 and y0=0.6 y1=0.7 has walkers on it. The error does in fact seem to behave like a square root function. This is rather unnerving, and to check if this was the case in 1d as well a long simulation was run for a similar case. Figure 3.12 shows that the error in the long term behaves similarly to the error from the deterministic simulations, but with a *much* larger amplitude. For the "best" case the maximum amplitude seems to be about a factor 100 times

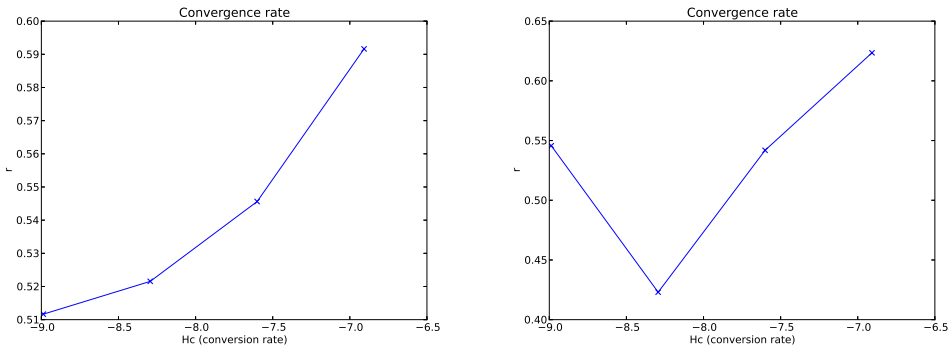
the deterministic one.



(a) (b)

Figure 3.18: Verification of anisotropic diffusion equation implementation

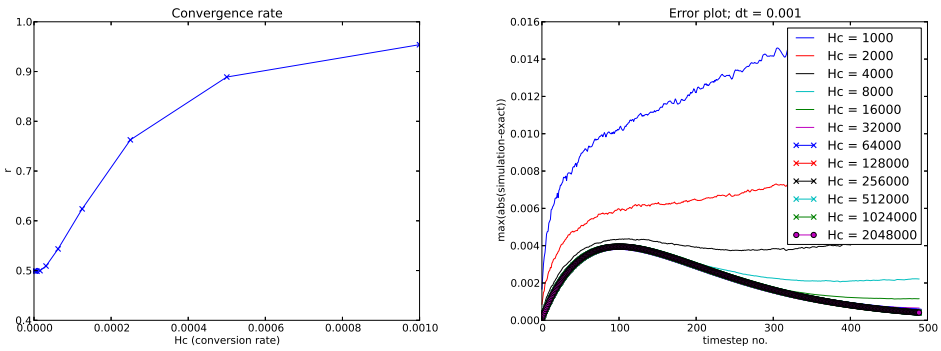
In fact, we can test the convergence rate for a simulation where we hold Δt, Δx and the number of time steps constant, and then increase the conversion rate for the walkers. The result of this is shown in figure 3.19 and shows that introducing random walks reduces the convergence rate from 1 to $\frac{1}{2}$. A result we actually expect since the convergence test, and the error measure in general, are sensitive to the largest error in the solution. In chapter 3.4 we will see that the random walk algorithm has a convergence which goes as 0.5, and this makes the error larger than for the FE scheme alone.



(a) 2D (b) 1D

Figure 3.19: Convergence rate for simulation where only the conversion faction, Hc, in varied (from 1000 to 16000) over 170 timesteps. Initial condition is given by $u(\mathbf{x}, t = 0) = H(x - 0.5)$ in both 1d and 2d, and the time step is the same as well, $\Delta t = 8 \cdot 10^{-7}$ which by figure 3.17 should give us perfect convergence.

We argued earlier that for the error from the random walk part of the mesh not to dominate we will need to have some Δt^{-2} walkers per unit concentration per mesh-point. Using an explicit scheme this will be an unreasonable demand, but if we use an implicit solver, we can at least check that this is the case for a larger Δt. Figure 3.20 shows the same convergence test as we see in figure 3.17, but using the ADI scheme and a rather large time-step, $\Delta t = 0.001$. We see that as the conversion factor, Hc, increases beyond 10^6 the convergence rate goes to 1. This proves the concept that we can include random walks as a viable part of the solver and still reproduce the exact solution to any desired error should we wish to do so.



(a) Convergence rate (b) Error plot

Figure 3.20: Convergence rate for simulation where only the conversion faction, Hc, is varied over the values shown in the legend in figure b over 490 timesteps. Initial condition is given by $u(x, y, t = 0) = \cos(\pi x)\cos(\pi y)$ and $\Delta t = 0.001$. Note that the x-axis on figure a is wrong, and should be interpreted simply as Hc.

3.4 Testing Random walks

To verify our implementation, and perhaps gain some new knowledge we will do some testing on the random walks implementation as well. This means that we have to find a solution of the diffusion equation 2.13 for an initial condition we can recreate with random walkers to the best possible precision. The absolute simplest initial condition to recreate is the Heaviside step function, defined in equation 3.17.

$$H(x-a) = \begin{cases} 1 & x \geq a \\ 0 & x < a \end{cases} \tag{3.17}$$

We then solve the equation by separation of variables. We have

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}; \quad \frac{\partial u(0,t)}{\partial x} = \frac{\partial u(1,t)}{\partial x} = 0$$
$$u(x,0) = H\left(x - \frac{1}{2}\right); \quad D = 1$$

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

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

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

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

The boundary conditions gives us

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

Since the time dependent solution cannot be exactly zero, the first derivative of the spatial solution must be zero

$$F'(x) = -a\mu\sin(\mu x) + b\mu\cos(\mu x)$$
$$F'(0) = -a\mu\sin(0) + b\mu\cos(\mu x) = b\mu\cos(\mu x) \implies b = 0$$
$$F(1) = a\cos(\mu) \implies \mu = n\pi$$

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

$$u(x,t) = a_0 + \sum_{n=1}^{\infty} a_n \exp\left(-(n\pi)^2 t\right)\cos(n\pi x) \tag{3.18}$$

The coefficients are found by approximating the initial condition

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

which gives us the final solution

$$
u(x,t) = \frac{1}{2} + \sum_{n=1}^{\infty} \frac{2\sin(\frac{n\pi}{2})}{n\pi} \exp\left(-(n\pi)^2 t\right)\cos(n\pi x) \tag{3.19}
$$

We can now perform a convergence test to find the convergence rate for the random walkers, this is the order the error of the scheme goes as and it is defined in equation 3.20. We will modify it slightly by testing for the number of walkers rather than the time step. We must also use another error estimate E_i which gives us one number for each simulation. The candidates are either the maximum of the error we are already using, or an integrated error.

$$
r = \frac{\ln(E_{i+1}/E_i)}{\ln(\Delta t_{i+1}/\Delta t_i)} \tag{3.20}
$$

Using the maximum of the error measure already in use, we have tested the convergence rate measure for the following measures of Hc:

```
Hc = [100,500,1000,5000,10000,50000]
```

Meaning that the largest value of Hc we will get an estimate for is $Hc = 10000$. The convergence test suggests that the convergence rate for random walks follows the proportionality in equation 3.21. This relation tells us just what we have been expecting the whole time; while increasing the number of walkers will reduce the error, the convergence is 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 $err \sim \mathcal{O}(\Delta t)$, and $Hc \sim \Delta t^{-4}$ for $err \sim \mathcal{O}(\Delta t^2)$. Clearly we will have enough trouble for the simpler cases.

$$err \propto Hc^{\frac{-1}{2}} \tag{3.21}$$

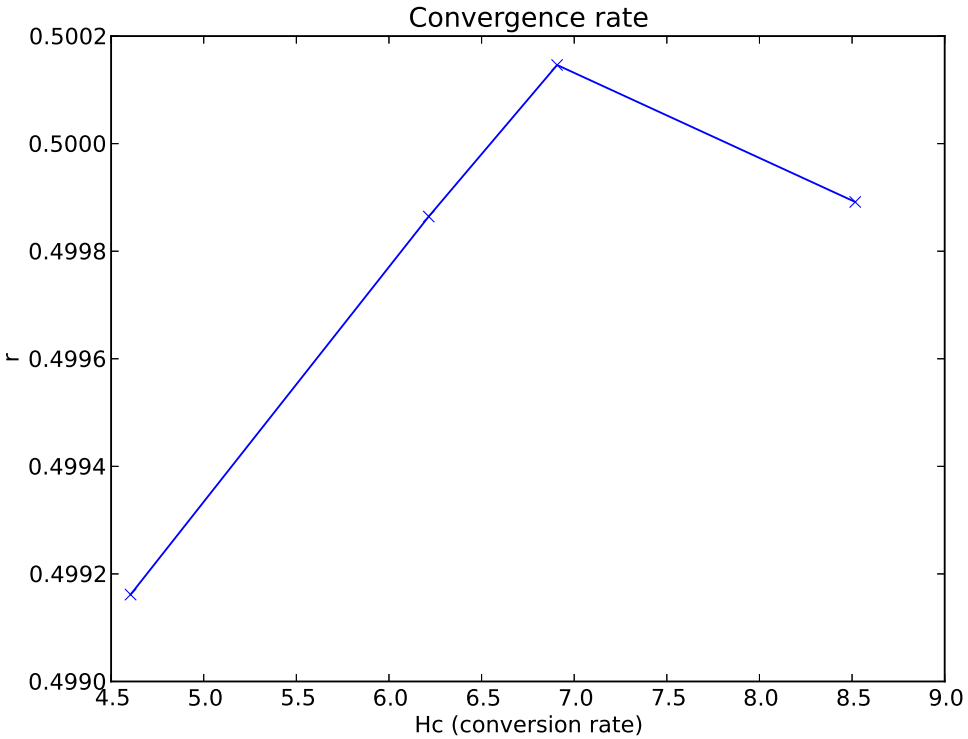


Figure 3.21: A convergence test for the isotropic random walk implementation using different conversion factors, Hc. The x axis (conversion rate) is log transformed and ranges from 100 to 50000 in real numbers.

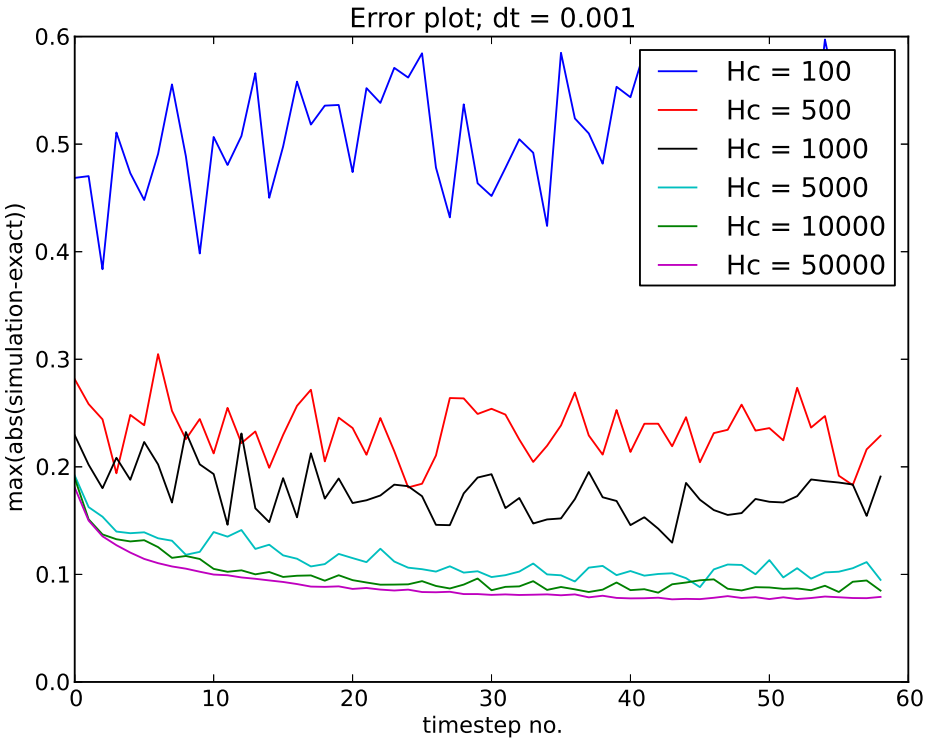


Figure 3.22: A "normal" error plot for the same simulation as in figure 3.21. The Δt in question is used to couple the RW simulation and the exact solution. For each Δt the RW simulation does 100 steps with a step length calculated from equation 2.37

Chapter 4

Software

4.1 About

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.

4.3.2 CPU time

There are four expensive operations in the algorithm as it stands now with the BE discretization using standard LU-decomposition.

- Performing the LU decomposition is very expensive. There might be a way to make this step more efficient by utilizing the sparsity of the linear system.

- The cost of solving the PDE increases drastically with the spatial dimension, at least as long as we use an implicit scheme. The FE scheme will actually save us d orders of operations in d dimensions, at least with the BE discretization using the standard LU decomposition. If we could utilize some specialized Gaussian elimination like the one we have for tridiagonal matrices we might be able to improve this.

- 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.

- De- and re-allocation of random walkers each time-step is also an immensely costly procedure. An alternative might be to keep the number of walkers constant and rather change the conversion-factor, Hc, although this approach sacrifices physical meaning.

- Writing the results to file will require $\mathcal{O}(n^d)$ operations where the operations are quite slow. We can get some speed-up by writing in binary, but as of now it has proved problematic to make python interpret the result-files.

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

where $\tilde{T}$ denotes the number of timesteps 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.

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.

- LU-decomposition
  The actual LU-decomposition is a sort of Gaussian Elimination which is costly ($\mathcal{O}(N^3)$). Although the decomposition is pre-implemented at this point, the plan is to implement my own version of this. This step should be possible to parallelize.

- Solving
  This step includes a back-transform of the decomposed matrix which is expensive for $d > 1$. This step should be parallelizable. 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. The LU decomposition and back-substitution require some communication and will not scale linearly, but will still benefit from parallelizing.

Chapter 5

Results

5.1 Validity of the model

The results of the testing done in the Analysis-chapter, particularly the convergence tests based on varying the conversion-factor, Hc, to exceed the expected numerical error arising from the scheme itself based on equation **??**, suggest that the implementation of our model is correct. The mathematics overlap, as we have seen in chapter **??**, and if we introduce enough walkers there seems to be no difference between adding an area on the mesh where random walks solve the equation, and not doing so. In other words, our model converges to the continuum model in the limit of sufficient walkers.

Appendix A

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.

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

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 simly 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 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 constant, 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 3.2.1. 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 3.2.1).

- Convergence test

Bibliography

[1] 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.

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