MULTISCALE MODELING OF DIFFUSION PROCESSES IN DENDRITES AND DEDRITIC SPINES

by

Fredrik Eksaa Pettersen

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2014

Contents

List of Figures

List of Abbreviations

BE Backward Euler

FE Forward Euler

FLOPs Floating Point Operations (per second)

PDE Partial Differential Equation

PKCγ Protein Kinase Cγ

RW Random Walk

Chapter 1

Introduction

Diffusion processes are extremely important in modern science, and have so many applications that it is hard to list them all. Brownian motion of particles, momentum in liquids, and atomic diffusion of Helium through a balloon wall are just some examples of diffusion processes.
Both in general transport processes, which among other things include fluid flow, and particularly in diffusion processes, there are cases in which parts of the process cannot be described by a continuum model, but the rest of the process can. Examples of such processes are fluid flow in nanoporous media, diffusion in the extracellular space of the brain, and ...

These types of processes can be called multi scale processes in that two (or more) models, which are usually used on different length scales, are needed to achieve a complete description of the problem. Multi scale models are usually either solved by designing a meso scale, or intermediate scale model, or by a hybrid solver which seeks to combine the models that are used. An example of a meso scale model is dissipative particle dynamics []. Here, clusters of particles are modeled as individual particles that have different properties than the actual molecules or atoms that make up the substance. The other alternative is to use a hybrid solver. Some hybrid models exist today, but these are mostly aimed at specific problems, like dendritc solidification [7], or hybrid fluid flow models which combine molecular dynamics and Navier-Stokes solvers [6]. Other hybrid solvers for diffusion processes have also been developed [3], but they are closed in the sense that the computer code is not commonly available.

The aim of this masters thesis is to develop a simple, yet flexible hybrid diffusion solver from the ground and up where all parts of the theory and implementation are fully understood and transparent. In principle, any particle dynamics model could be used, but for the sake of verification a stochastic model has been implemented. This is no limitation, as the interface to the lower scale model is simple, and the lower scale model works as a standalone unit.

A large emphasis has been put on verification of all parts of the hybrid model. Both individually and for the combined, hybrid solver to verify the average behavior of the system.

Put shortly, the thesis in your hands has the following structure:
Chapter 2 contains a detailed look at the coupling between the two models, as well as a look at band diagonal linear systems. Chapter 3 defines the error estimate and shows a thorough analysis of all parts of the developed soft-

ware in order to verify that it functions properly. In chapter 4 the developed software is modified to describe a physical application in which a hybrid diffusion solver is demanded. The results of both the verification and the physical application are described and discussed in chapter 5, and finally, chapter 6 discusses the thesis as a whole and looks at possible improvements and extensions that can be done. The appendix is a general guide to debugging the methods that have been implemented during this project.

Chapter 2

Theory

2.1 Random walks

A random walker is a point object which after a certain amount of time jumps a fixed step length either right or left with equal probability for jumping either way. In d spatial dimensions the axis on which the jump happens is chosen at random before the direction of the jump is decided.
Say N random walkers are placed at the same position within a volume, V of arbitrary shape, to make a spatial distribution of walkers, $C = N\delta(x)$. V is contained by a surface S and the relation is shown in Figure 2.1.

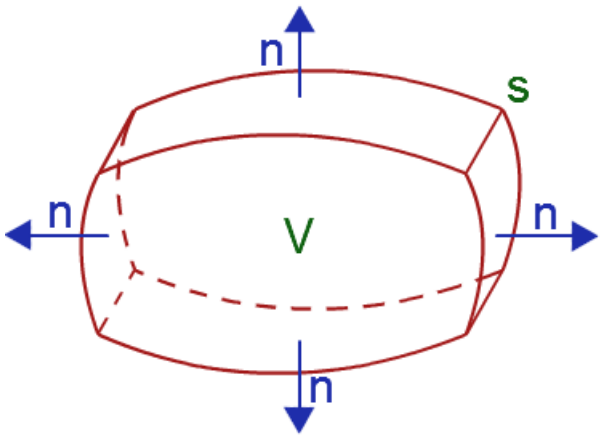


Figure 2.1

A flux of walkers is a vector size describing the number of walkers passing a surface element dS per unit time, per unit area.

As the random walkers start to move, some of them will leave V, which results in a change in C. The number of walkers leaving V is described by the change in C integrated over the entire volume.

$$\int_V \frac{\partial C}{\partial t}\, dV \tag{2.1}$$

As mentioned, the number of walkers leaving V can also be described by the flux of walkers, $\mathbf{J}$, out of S. Integrating the surface normal (illustrated in Figure 2.1) of $\mathbf{J}$ over S results in the number of walkers leaving V.

$$\int_S \mathbf{J} \cdot \mathbf{n}\, dS \tag{2.2}$$

Since the number of walkers is conserved, the change in C per time must be equal to the number of walkers leaving V plus any walkers added to V

$$\int_V \frac{\partial C}{\partial t}\, dV = \int_S \mathbf{J} \cdot \mathbf{n}\, dS + s(x,t) \tag{2.3}$$

where $s(x,t)$ denotes a source term. Through Greens theorem (2.3) can be reformulated

$$\int_V \frac{\partial C}{\partial t}\, dV = \int_V \nabla \cdot \mathbf{J}\, dV + s(x,t) \tag{2.4}$$

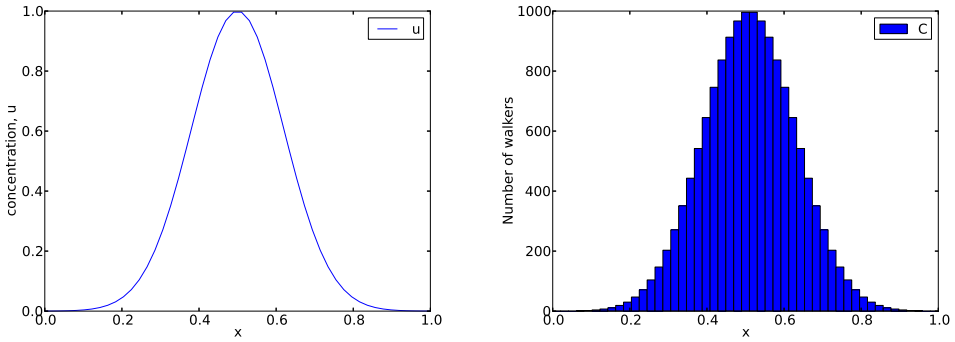
The flux $\mathbf{J}$ can be expressed by Fick's first law as the diffusive flux

$$\mathbf{J} = D\nabla C \tag{2.5}$$

where D is the diffusion constant. Since V was chosen as an arbitrary volume (2.4) is independent of the integration and the integrals can be dropped. The diffusive flux $\mathbf{J}$ is also inserted to yield the diffusion equation

$$\frac{\partial C}{\partial t} = D\nabla^2 C + s(x,t) \tag{2.6}$$

Throughout this thesis C will denote a spatial distribution of random walkers while u will denote the solution to the diffusion equation. The difference between u and C is best illustrated in Figure 2.2.



(a) (b)

Figure 2.2: Illustration of the difference between $u(x,t)$ in (a) which is the solution to the diffusion equation and $C(x,t)$ in (b) which is the number of random walkers within an area of $\pm\frac{\Delta x}{2}$ around each mesh point.

2.1.1 An error estimate for a diffusion RW solver

In general a Monte Carlo method will fluctuate around the correct (steady state) solution. The fluctuations have an amplitude related to the number of samples, N used by the relation

$$\epsilon \propto \frac{1}{\sqrt{N}} \tag{2.7}$$

(2.7) will be used as an error estimate for the solution of diffusion processes by a random walk (RW) model and is assumed to hold for all parts of the simulation, not only the steady state.

2.1.2 Random number generator

To produce a random walk one must have random numbers. The random numbers in this thesis are produced by a five seeded xor-shift algorithm copied from George Marsaglia [5]. This generator is chosen because of its very large period ($\sim 10^{48}$) compared to the computational cost.

2.2 Backward Euler schemes in 2 or more spatial dimensions

The reader is assumed to know the basics of solving partial differential equations (PDEs) by finite difference methods, but for the sake of clarity the two schemes used in this thesis are written out.

$$\frac{u^{n+1} - u^n}{\Delta t} = \theta D \frac{\partial^2 u^{n+1}}{\partial x^2} + (1-\theta) D \frac{\partial^2 u^n}{\partial x^2} \tag{2.8}$$

Having $\theta = 0$ gives the Forward Euler (FE) scheme, and $\theta = 1$ gives the Backward Euler (BE) scheme.
This section will concern a simple, but elegant way of efficiently solving the diffusion equation by the implicit BE scheme in more than one spatial dimension. First, let us look at the BE discretization of the diffusion equation in 1D.

$$u_i^{n+1} = \frac{\Delta t}{2\Delta x^2}\left((D_{i+1} + D_i)(u_{i+1}^{n+1} - u_i^{n+1}) - (D_i + D_{i-1})(u_i^{n+1} - u_{i-1}^{n+1})\right) + u_i^n \tag{2.9}$$

u is here the unknown function which solves the diffusion equation. Note that

$$u_i^n = u(t_0 + n \cdot \Delta t, x_0 + i \cdot \Delta x)$$

and not u to the n'th power.
Neumann boundary conditions will be used, these are described in (2.10).

$$\frac{\partial u}{\partial n} = 0|_{\text{on boundary}} \tag{2.10}$$

Solving the BE scheme by hand for a very small mesh consisting of 4 mesh points will reveal the structure of the BE scheme.

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

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

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

which is arranged as a coefficient matrix multiplied by the unknown next time step

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 \\ 0 & a_2 & b_2 & c_2 \\ 0 & 0 & a_3 & b_3 \end{pmatrix} \mathbf{u}^{n+1} = \mathbf{u}^n \tag{2.11}$$

The coefficient matrix in (2.11) will be labeled $\mathbf{M}$ throughout the thesis.

$$\mathbf{M}\mathbf{u}^{n+1} = \mathbf{u}^n \tag{2.12}$$

If a linear system like the one in (2.12) has a solution it can always be found by Gaussian elimination check this. However, for a sparse linear system a

lot of the calculations in a Gaussian elimination will be done with zeros. A dramatically more efficient way to solve the linear system in (2.12) is by exploiting the fact that it is tridiagonal. In order to be more consistent with the algorithm, (2.12) will be rewritten as

$$\mathbf{Mu} = \mathbf{up} \tag{2.13}$$

where $\mathbf{up} = \mathbf{u}^n$ denotes the solution $\mathbf{u}$ at the previous time step. Tridiagonal systems can be solved extremely efficiently by the "tridiag" algorithm listed below.

```
void tridiag(double *u, double *up, int n, double *a,
    double *b, double *c){
     double *H = new double[n];
     double *g = new double[n];
     for(int i=0;i<n;i++){
       H[i] = 0;
          g[i] = 0;
     }
     g[0] = up[0]/b[0];
     H[0] = c[0]/b[0];
     for(int i=1; i<n; i++){
          //forward substitution
          H[i]  = -c[i]/(b[i] + a[i]*H[i-1]);
          g[i] = (up[i] - a[i]*g[i-1])/(b[i] + a[i]*H[i-1]);
     }
     u[n-1] = g[n-1];
     for(int i=(n-2);i>=0;i--){
          //Backward substitution
          u[i] = g[i] - H[i]*u[i+1];
     }
}
```

In two spatial dimensions the BE discretization gives the scheme shown below

$$u_{i,j}^n = \underbrace{\frac{-D\Delta t}{\Delta x^2}}_{\alpha}\left(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}\right) + \underbrace{\left(1 + \frac{2D\Delta t}{\Delta x^2} + \frac{2D\Delta t}{\Delta y^2}\right)}_{\gamma} u_{i,j}^{n+1} \underbrace{-\frac{2D\Delta t}{\Delta y^2}}_{\beta}\left(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right) \tag{2.14}$$

If (2.14) is solved by hand on a 3×3 mesh by the same steps as in 1D it

gives the following linear system

$$\left(\begin{array}{ccc:ccc:ccc} \gamma & -2\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 & 0 \\ -\beta & \gamma & -\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 \\ 0 & -2\beta & \gamma & 0 & 0 & -2\alpha & 0 & 0 & 0 \\ \hdashline -\alpha & 0 & 0 & \gamma & -2\beta & 0 & -\alpha & 0 & 0 \\ 0 & -\alpha & 0 & -\beta & \gamma & -\beta & 0 & -\alpha & 0 \\ 0 & 0 & -\alpha & 0 & -2\beta & \gamma & 0 & 0 & -\alpha \\ \hdashline 0 & 0 & 0 & -2\alpha & 0 & 0 & \gamma & -2\beta & 0 \\ 0 & 0 & 0 & 0 & -2\alpha & 0 & -\beta & \gamma & -\beta \\ 0 & 0 & 0 & 0 & 0 & -2\alpha & 0 & -2\beta & \gamma \end{array}\right) \mathbf{u}^{n+1} = \mathbf{u}^n \quad (2.15)$$

The dashed lines confine a total of nine 3×3 matrices which can be used to reformulate the $2n + 1 = 7$ band diagonal system as a block tridiagonal system as shown below.

$$\left(\begin{array}{c:c:c} B_0 & C_0 & 0 \\ \hdashline A_1 & B_1 & C_1 \\ \hdashline 0 & A_2 & B_2 \end{array}\right) \mathbf{u}^{n+1} = \mathbf{u}^n \quad (2.16)$$

All entries in eq. (2.16) are 3×3 matrices. In the general case where the diffusion equation is discretized and solved on a mesh of size $n \times n$ the resulting block tridiagonal system will consist of n^2 matrix entries which all are $n \times n$ matrices.

One should note that the $n \times n$ matrix $\mathbf{u}$ which solves the 2D diffusion equation has be rewritten as a vector $\mathbf{u}$ of size n^2.

In order to solve this block tridiagonal linear system the "tridiag" solver from before must be generalized to support matrices as entries in $\mathbf{M}$. Luckily this generalization is trivial and all that is needed is to replace divisions by matrix inverses. An updated *pseudo coded* scheme is listed in (2.17)
There is a forward substitution

$$\begin{aligned} H_0 &= -B_0^{-1} C_0 \\ \mathbf{g}_0 &= B_0^{-1} \mathbf{up}_0 \\ H_i &= -\left(B_i + A_i H_{i-1}\right)^{-1} C_i \\ \mathbf{g}_i &= \left(B_i + A_i H_{i-1}\right)^{-1} \left(\mathbf{up}_i - A_i \mathbf{g}_{i-1}\right) \end{aligned} \quad (2.17)$$

Followed by a backward substitution

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

It is possible to solve the BE scheme in 3D by the block tridiagonal solver as well. The linear system will be $2n^2 + 1$ band diagonal and must first be reduced to a block matrix which is $2n + 1$ band diagonal like the one in eq. (2.15) before it is further reduced to a block tridiagonal form. Matrix entries will be $n^2 \times n^2$ matrices.

Efficiency of the block tridiagonal algorithm

In general the FE scheme will solve a discrete PDE in d spatial dimensions with n mesh points in each dimension using

$$\mathcal{O}(n^d)$$

floating point operations (FLOPs). This is the maximum efficiency possible for a PDE in d dimensions (without using some form of symmetry argument) and will be the benchmark for the BE scheme as well.

Implicit schemes like BE result in linear systems which, as mentioned, can always be solved by Gaussian elimination by some $\mathcal{O}(n^3)$ FLOPs for a $n \times n$ matrix. In d spatial dimensions the matrices are $n^d \times n^d$, meaning that Gaussian elimination will require $\mathcal{O}(n^{3d})$ FLOPs which is extremely inefficient.

The block Tridiagonal solver requires inverting n matrices of size $n^{d-1} \times n^{d-1}$ as well as some matrix-matrix multiplications. All of these steps require $\mathcal{O}(n^{3(d-1)})$ FLOPs, but as long as the coefficient matrix is constant they need only be done once. In other words, the H_i terms from (2.17) can be stored. The remaining calculations include three matrix vector multiplications, one of which demands n FLOPs because the matrix is diagonal. The other two matrices are dense, so (for now) the remaining two matix - vector multiplications require n^2 FLOPs. In summary, the block tridiagonal solver requires $\mathcal{O}(n^{2n-1})$ FLOPs, which is an order more efficient than a general LU decomposition*, and in fact any other method I have been able to find*. In 2D the block tridiagonal solver is one order slower than the FE scheme, which means it is still very much usable for small meshes.

Another point which should be mentioned is that the block tridiagonal solver is is extremely memory efficient compared to other linear system solvers. Since the algorithm only loops through the non zero entries, there is no need to assemble the entire matrix. For a 2D simulation on a 100×100 mesh, the full assembled matrix will have $100^2 \times 100^2 = 10^8$ matrix entries, all of which are double precision floats. The total size of the matrix in RAM will be $8 \cdot 10^8$ bytes, which is getting close to the available RAM of a normal

computer at $\sim 8 \cdot 10^9$ bytes. In comparison, storing only the non zero entries requires three vectors of size 100 which contain 100×100 matrices. A total of $24 \cdot 10^6$ bytes, or two orders of magnitude less. Effectively, the memory impact is also reduced from $8 \cdot n^{2d}$ bytes to $8 \cdot n^{2d-1}$ bytes by switching to the block tridiagonal solver. *There might still room for improvement since all the stored matrices are sparse.*

2.3 Combining micro and macro scale models

2.3.1 The algorithm

After setting an initial condition and diffusion constant the diffusion problem is solved on both the microscopic and macroscopic meshes and combined into a common solution by the following steps.

- The result from previous PDE time step, **up**, is converted to a distribution of random walkers and sent to the RW solver
- The RW solver does a predefined number of micro scale time steps which correspond to one PDE time step
- The result from the RW solver is converted back to a concentration and this replaces the PDE solution, **up**
- **up** is then used as input to calculate the next time step

2.3.2 The conversion between length scales

As the previous section states the result from the last PDE time step is converted to a distribution of random walkers. This is done by specifying a conversion rate denoted Hc (as was done by Plapp and Karma [7]) which is a single, real integer defined by

$$C_{ij} = Hc \cdot u_{ij} \tag{2.18}$$

As before u_{ij} is the solution to the 2D diffusion equation evaluated at $x_0 + i\Delta x$ and $y_0 + j\Delta y$, and C_{ij} is the number of random walkers located in the rectangle defined by $x_i \pm \frac{\Delta x}{2}$ and $y_j \pm \frac{\Delta y}{2}$. Walkers are given a random position within this rectangle at the beginning of each PDE time step.

Equation (2.18) effectively states that one "unit" of the solution u will directly correspond to Hc random walkers. Depending on the application, $Hc \approx 20 - 50$.

2.3.3 Coupling the models through step length

To ensure that the τ time steps performed by the RW solver sums up to one time step for the PDE solver a limitation must be imposed on the RW solver. This limitation can only be imposed on the step length of the random walkers, so the task at hand is to relate the step length to the PDE time step and τ.
Through an Einstein relation the variance in position is coupled to the diffusion constant.

$$\langle \tilde{\Delta x}^2 \rangle = 2Dd\tilde{\Delta}t \tag{2.19}$$

Where $\tilde{\Delta}t$ is the time step for the random walkers which is exactly $\tilde{\Delta}t = \frac{\Delta t}{\tau}$. Similarly $\tilde{\Delta}x$ is the spatial resolution on the micro scale which is also the step length, l, for the random walkers. Rewriting equation (2.19) gives the final expression for the step length

$$l = \sqrt{2dD\frac{\Delta t}{\tau}} \tag{2.20}$$

2.3.4 Boundary conditions for the random walk

Because the RW area must correspond to an area on the PDE mesh which has a finite, constant size the RW model must have boundary conditions imposed on it. In order to make absolutely sure that no walkers disappear and by extension that the amount of concentration is conserved, perfectly reflecting boundaries are chosen for the RW model. Reflecting boundaries correspond to zero flux Neumann boundaries

$$\frac{\partial C}{\partial n} = 0 \tag{2.21}$$

Neumann boundaries might seem unphysical, but the redistribution of walkers at each PDE time step must also to some extent be considered as a boundary condition in which fluxes are exchanged between the two models. In hindsight, a possibly better choice of boundary conditions for the RW model would have been perfect exchange of flux between the two solvers.

$$\frac{\partial u}{\partial n} = \frac{\partial C}{\partial n} \tag{2.22}$$

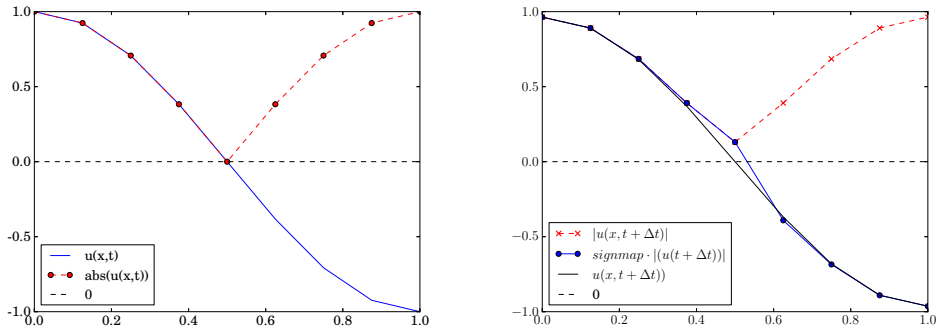
Since time does not allow for this to be implemented and tested it will be left as a reference to possible future work.

2.4 Potential problems or pitfalls

This section will list and discuss some of the problems which were encountered while working on the thesis and present the solutions or workarounds.

Negative concentration of walkers

Physically a negative concentration does not make sense, but if an initial condition which takes negative values is imposed on the system the software will try to allocate a negative number of walkers. Trying to handle this is more of an oddity than anything else, but a workaround has been found. If the absolute value of the negative concentration is taken as input to the RW solver and the sign at each mesh point is stored while the RW solver advances the system, the resulting solution can be multiplied by the stored sign to give back a negative concentration. The workaround will at least keep the simulation from crashing, but has a fundamental problem which is illustrated in Figure 2.3.



(a) (b)

Figure 2.3: An illustration of the proposed workaround for negative concentrations and an illustration of how it performs (b). The fundamental problem is that a diffusion process will even out discontinuities such as the one in (a). The result of this evening out is shown in (b) as an increase in concentration in the middle mesh point which should remain equal to zero.

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. Random walks are stochastic and though a RW will describe a diffusion process it will fluctuate around the solution. In this case a dilemma arises; on the one hand

there is the smoothness of the solution to consider, on the other hand the stochastic term was introduced believing that it adds detail to the model. This is partly the reason for solving the RW model before the PDE model. The fluctuations will be reduced by the PDE solver and the solution, though still fluctuating, will be smoother.

Section 2.5 describes some other methods which were attempted in order to reduce the fluctuations from the RW model.

Number of time steps on the random walk level

The reason for introducing a random walk model in the diffusion solver was to include an area with more advanced dynamics than regular diffusion on a smaller time-, and length scale. This means that the RW solver must take τ time steps for each PDE time step. τ could in principle be any integer, but should be chosen large enough so the central limit theorem is satisfied. This usually means $\tau \geq 50$.
Alternatively τ can be calculated from the assumption that the error term wrt. time arising from the RW model is proportional to the square root of the time step. This assumption comes from the fact that the step length is proportional to the square root of the time step.

$$\epsilon \propto \sqrt{\tilde{\Delta} t} \tag{2.23}$$

In order to make the error of the same size as the error from the PDE model we use

$$\begin{aligned} \tilde{\Delta} t &= \frac{\Delta t}{\tau} \\ \implies \epsilon &\propto \sqrt{\frac{\Delta t}{\tau}} \\ \mathcal{O}(\Delta t) &\geq \sqrt{\frac{\Delta t}{\tau}} \\ \implies \tau &\geq \frac{1}{\Delta t} \end{aligned}$$

Note that the error associated with the number of walkers introduced is of much higher significance than the error from the time step.

2.5 Other possible coupling methods

Running the solution from the PDE model through a RW model gives two solutions for a part of the mesh. As it turns out it is sufficient to simply

replace the PDE solution with the RW solution in this area, which is nice seeing as it is the simplest possible method. There are however some other ways to do this which have been tested, and found inadequate or simply been disregarded. Three of these are mentioned below.

- Averaging the two solutions
  Doing the arithmetic mean of the two solutions will reduce the magnitude of the fluctuations, as shown in Figure 2.4. This method is simple and reduces fluctuations by approximately one half (approximately because fluctuations will be present after the first time step).

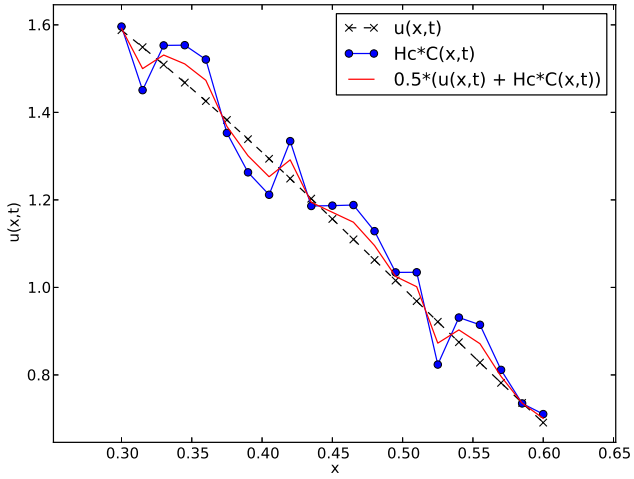


Figure 2.4: This plot shows the PDE solution before and after the stochastic solver has been called, and the arithmetic mean of the two solutions which has fluctuations of smaller amplitude.

  The reason an average was scraped is simply that results just as good were achieved by only replacing the solution.

- Polynomial regression
  In order to smoothen out the solution from the RW model a polynomial regression function was tested. The principle was to look at the solution from the RW model as a set of data points which have some dependency on their position on the PDE mesh. Figure 2.5 shows how this method worked for a slightly exaggerated case. This approach turns out to have two problems; first and foremost the amount of concentration is no longer conserved, secondly the endpoints are usually way off.

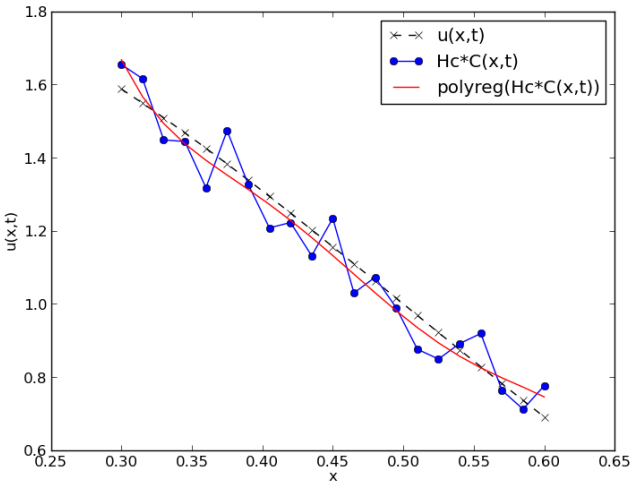


Figure 2.5: This plot shows the PDE solution before and after the stochastic solver has been called, and a polynomial regression of degree 6 using $C(x,t)$ as data points.

- Cubic spline interpolation
  Cubic spline interpolation has the same problem as polynomial regression when it comes to conserving energy.

Chapter 3

Analysis

3.1 Computation of the error

Solving PDEs numerically will result in errors because derivatives are approximated by finite differences and the PDE is required to be fulfilled only on the mesh points.

The error, ϵ, is measured by comparing the result from a numerical simulation to an exact solution, u_e and taking the norm of the difference. Specifically ϵ is measured by the L2 norm which is defined in (3.1)

$$\begin{aligned}\epsilon(t^n) &= ||u(t^n) - u_e(t^n)||_2 \\ &= \iint \sqrt{\left(u(t^n, x, y) - u_e(t^n, x, y)\right)^2}\, dx\, dy \\ &\approx \sqrt{\Delta x \Delta y \sum_{i=0}^{n}\sum_{i=0}^{n} \left(u(t^n, x_i, y_j) - u_e(t^n, x_i, y_j)\right)^2}\end{aligned} \tag{3.1}$$

A time dependent error estimate has been chosen because it allows for investigation of the evolution of the error over the course of a simulation. Some of the error tests will require a single number as an error measure. In these cases the norm of $\epsilon(t)$, shown below, is used.

$$\epsilon = \sqrt{\Delta t \sum_{n=0}^{T} \epsilon(t^n)^2} \tag{3.2}$$

3.2 Verification techniques

All parts of the hybrid diffusion solver are subject to testing. First, the PDE and RW solver will be tested individually, then the hybrid solver is tested. This thesis will focus on three verification techniques which are described below. The aim for all of these techniques is to make sure that the implementation is correct by comparing simulations to an exact solution.

For the PDE solver, there are errors arising from both the spatial derivative and from the time derivative. Since an incorrect implementation of the spatial derivative will cause the solution to be unstable, which is easily noticed through visual inspection, the tests will focus on verifying the time derivative. In some cases, isolating the contribution to $\epsilon(t)$ from the time derivative will be necessary, and is ensured by setting $\Delta t \gg \Delta x^2$. A time step of this size violates the stability criterion for the FE scheme, and so some of the tests are slightly inaccurate for this scheme.

The verification techniques that will be carried out are:

- Manufactured solutions
  By adjusting the source term, $s(x,t)$, one can, in principle, choose any solution to the diffusion equation. Should we, for example, want the solution to be $u(x,t) = \frac{1}{x+1}$, all that is needed is to set

$$s(x,t) = \frac{2D}{(x+1)^3}$$

  and the problem is solved (as long as the boundary conditions are correct). For the verifiation that follows, the chosen solution is

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

  which requires no source term and fulfills the zero flux boundary conditions naturally. The point of these tests is to verify that $\epsilon \sim \Delta t$ and the tests can be done for a time step which fulfills the stability criterion.

- Convergence tests
  In the general case the error term is described by

$$\epsilon = C_x \Delta x^2 + C_t \Delta t \tag{3.4}$$

  where the coefficients C_x and C_t are unknown. Notice that there is one term arising from the time derivative and one from the spatial derivative and that they are of different order.
  Where possible, convergence will be tested by isolating one of the error terms. For example, setting $\Delta t \gg \Delta x^2$, implies $C_x \Delta x^2 \approx 0$ and

$$\epsilon \propto \Delta t^r$$

  where $r = 1$. As Δt tends to zero, r is expected to converge to one, and that is the purpose of these tests. By comparing the error from two simulations with different time steps, r, called the convergence rate, can be measured from

$$r \simeq \frac{\log(\epsilon_1/\epsilon_2)}{\log(\Delta t_1/\Delta t_2)}$$

  Notice that a single number must be used to describe the error, as shown in (3.2).

- Exact numerical solutions
  The numerical schemes are actually reformulations of the PDE as difference equations, which have their own exact solutions. These will be called the numerical exact solutions, and they are slightly different from the exact solutions to the PDE. The reason for finding the numerical exact solutions is that the scheme theoretically will represent this solution with no error. In practice there will always be round off errors and other factors, but an error term close to machine precision is expected.

3.3 Testing the PDE solver

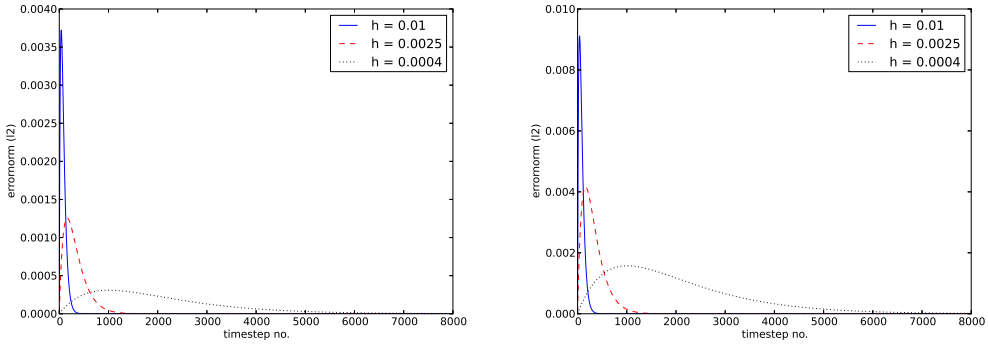
3.3.1 Verification by manufactured solutions

Equation (3.3) solves the diffusion equation, so using $u(x, y, t = 0)$ as the initial condition for a simulation will give us both the numerical solution and the exact solution. An important property of both the FE and BE discretization schemes is that the difference between these solutions is of the same order as the time step.

$$\epsilon(t) \sim \mathcal{O}(\Delta t)$$

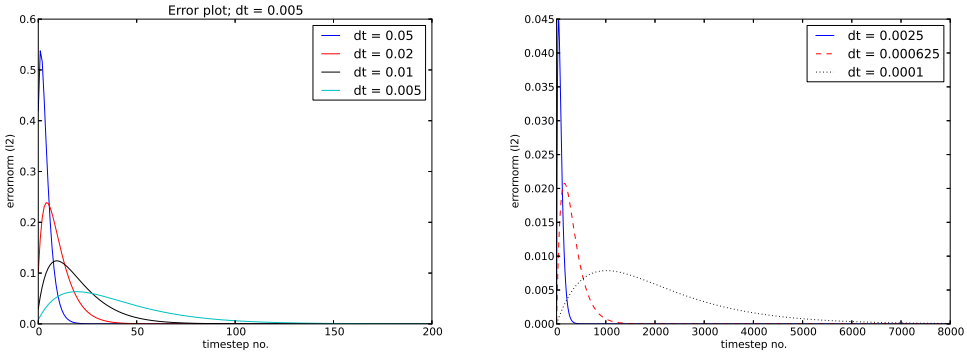
Figures 3.1 and 3.2 show that for both the FE and BE scheme in both 1D and 2D the error is of the expected magnitude, which is roughly Δt. Another interesting property of the error plots is that the error tends to zero after a large number of time steps. By inserting the limit $t \to \infty$ in (3.3) we observe that the error is expected to tend to zero because the limit value of one can be exactly recreated by both schemes.

$$\lim_{t\to\infty} e^{-\pi^2 t}\cos(\pi x)\cos(\pi y) + 1 = 1$$



(a) (b)

Figure 3.1: Error plot for the FE scheme in 1D (a) and 2D (b). Note that there is now only one discretization parameter, called h, which is related to the time step by $h = k\Delta t$, where k is a constant.



(a) (b)

Figure 3.2: Error plot for the BE scheme in 1D (a) and 2D (b). Notice that the error is roughly of the same order as the time step. Figures 3.1b and (b) show the same simulation using FE and BE schemes in 2D. The error is slightly larger in the BE discretization by a factor of 4.

3.3.2 Verification by convergence tests

The convergence tests are done by isolating the error term from either the time derivative or the spatial derivative, and refining the relevant discretization parameter over several simulations. However, for the FE scheme it is impossible to isolate the error from the time derivative without violating the

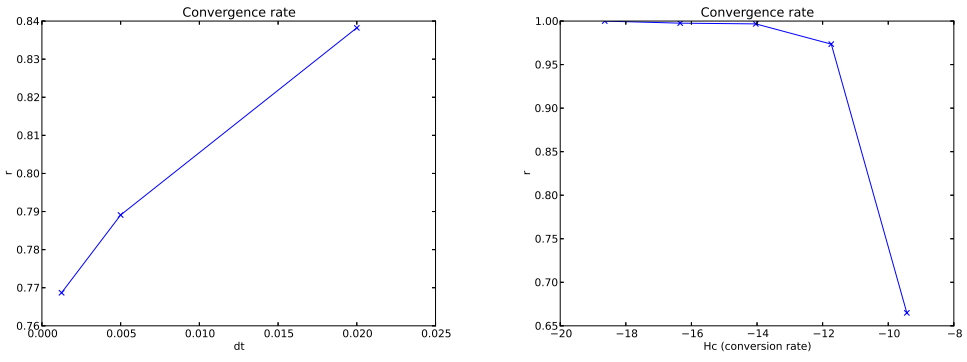
stability criterion. To cope with this problem a single discretization parameter, h, is introduced by ensuring

$$\frac{\Delta t}{\Delta x^2} = k$$

where k is a constant. $\epsilon(t)$ can now be rewritten by setting $\Delta t = h$

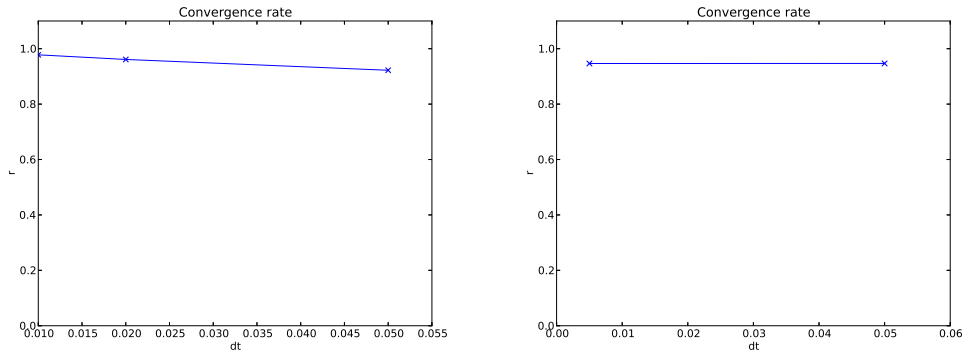
$$\begin{aligned}\epsilon(t) &= C_t h + C_x \frac{h}{k} \\ &= (C_t + C_x/k)h = C_k h^r\end{aligned}$$

r is expected to converge to one. Figures 3.3 and 3.4 verify that the error associated with the time derivative is of the expected order. In Figure 3.5 the error from the spatial derivative has been isolated, and tested for both schemes.



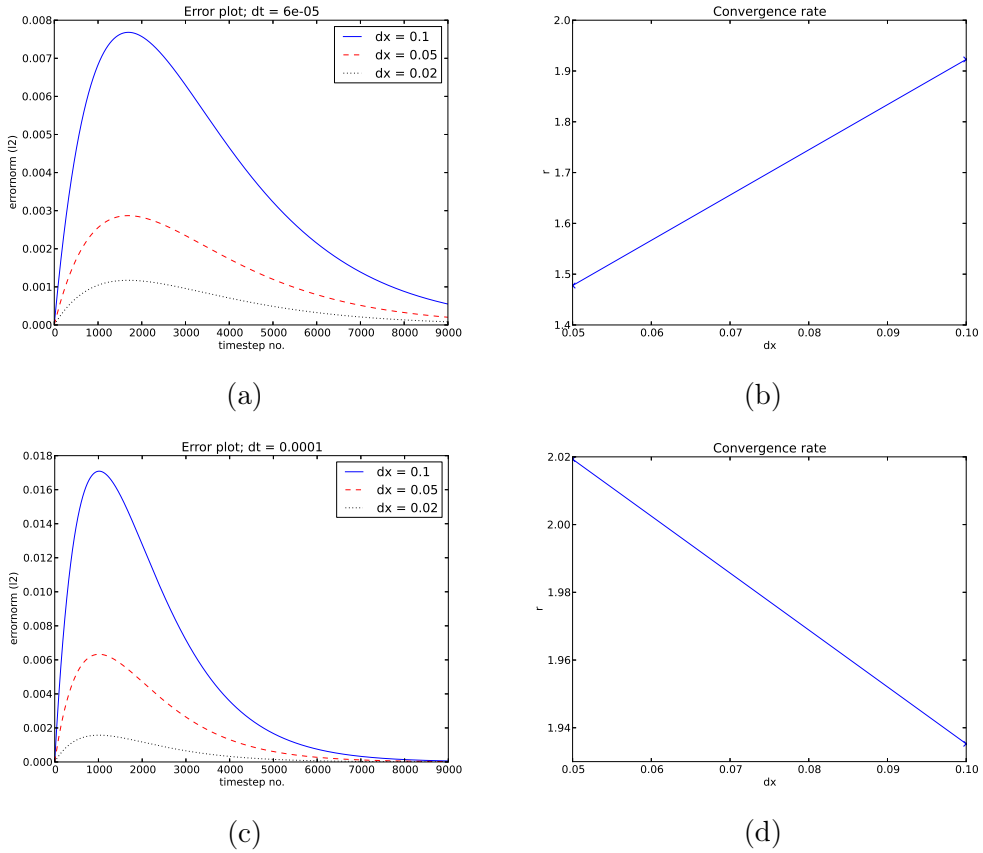
(a) (b)

Figure 3.3: Convergence tests with respect to the time derivative for the FE scheme in 1D (a) and 2D (b).



(a) (b)

Figure 3.4: Convergence tests with respect to the time derivative for the BE scheme in 1D (a) and 2D (b).



(a) (b)

(c) (d)

Figure 3.5: These plots show the error and convergence from the spatial derivative using BE (a & b) and FE (c & d) discretization schemes. The expected convergence rate for both discretizations is $r = 2$, which is recreated to a decent accuracy. Testing has been done using 1D simulations for the BE scheme, and 2D simulations for the FE scheme. This should not make any difference for the results.

3.3.3 Verification of FE scheme by exact numerical solution

Discretization of the diffusion equation by the FE scheme yields the following numerical scheme in 1D

$$u^{n+1} = D\Delta t u^n_{xx} + u^n \tag{3.5}$$

where u_{xx} denotes the double derivative of u with respect to x. (3.5) is a difference equation, which is most easily solved by writing out the first four iterations

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

From the iterations above a pattern can be recognized for iteration $n+1$

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i u^0_{2ix} \tag{3.6}$$

where u^0 is the initial condition

$$u^0 = \cos(\pi x) \tag{3.7}$$

The spatial derivatives are found by inserting the initial condition into the centered difference approximation to the second derivative

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

The pattern continues allowing the final numerical exact solution to be expressed below.

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \frac{2^i}{\Delta x^{2i}} (\cos(\pi\Delta x) - 1)^i \cos(\pi x) \tag{3.8}$$

By construction the Neumann boundary conditions are fulfilled since $\frac{\partial \cos(\pi x)}{\partial x} = -\pi \sin(\pi x)$ and $\sin(0) = \sin(\pi) = 0$.

Although the FE scheme is expected to reproduce (3.8) to machine precision ($\epsilon \approx 10^{-16}$) there are two problems with the solution which will have an effect on the error:

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

As a side note, (3.8) illustrates how the stability criterion for the FE scheme comes into place. In the numerical exact solution, the exponential which is found in the exact solution to the PDE (eq. 3.3) is replaced by an amplification factor A^n. This amplification factor can be found in equation (3.8) as

$$A^n = \left(\frac{2D\Delta t}{\Delta x^2}\right)^i \tag{3.9}$$

Inserting a time step larger than $\frac{\Delta x^2}{2D}$ will make the amplification factor, A, larger than one, which in turn makes the solution blow up.

The stability criterion also illustrates why the terms where

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

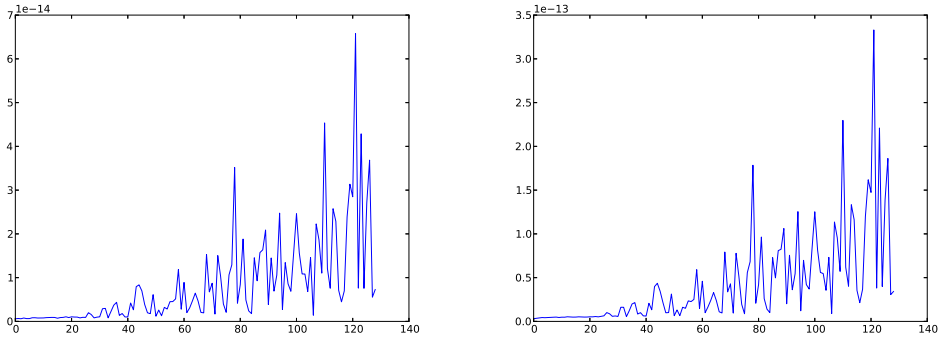
can be dropped. By the stability criterion, the time step will cancel out Δx^2, and the result will be a number smaller than 1 raised to a rather large power, resulting in a number comparable to zero.

The results from comparing a 1D simulation to the numerical exact is shown in Figure 3.6a. As expected the error is larger than machine precision by at most two orders of magnitude because of accumulating error terms from the dropped terms in (3.8).

Using the same method as in the 1D case, a numerical exact solution can be found to the 2D FE scheme.

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \left[2^{i-1} \cos(\pi x) \cos(\pi y) \left(\frac{(\cos(\pi \Delta x))^i}{\Delta x^{2i}} + \frac{(\cos(\pi \Delta y))^i}{\Delta y^{2i}} \right) \right] \tag{3.10}$$

The same problems as in the 1D case will apply to (3.10) with the same solutions. Figure 3.6b shows how the 2D simulation compares to the numerical exact solution. As was the case in 1D, the error is larger than machine precision, but much smaller than Δt.



(a) (b)

Figure 3.6: Numerical solution from the FE scheme in 1D (a) and 2D (b) versus the exact numerical solution of the FE scheme. Some of the terms in the numerical exact solutions are ignored to prevent overflow and this is responsible for the increasing error which is slightly larger than machine precision.

3.3.4 Verification of BE scheme by exact numerical solution

The exact numerical solution to the BE scheme is found by solving the linear system which arises from the discretization (see Section 2.2). Given the n'th time step, the next time step is found by

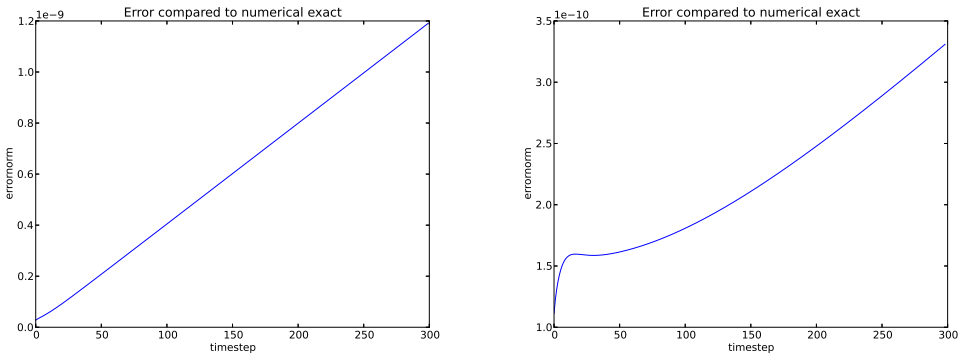
$$\begin{aligned}\mathbf{M}\mathbf{u}^{n+1} &= \mathbf{u}^n \\ \mathbf{u}^{n+1} &= \mathbf{M}^{-1}\mathbf{u}^n \\ &= \mathbf{M}^{-1}\left(\mathbf{M}^{-1}\mathbf{u}^{n-1}\right)\end{aligned}$$

Doing the separation to the end relates the n'th time step to the initial condition

$$\mathbf{u}^n = \left(\mathbf{M}^{-1}\right)^n \mathbf{u}^0 \tag{3.11}$$

Where $(\mathbf{M}^{-1})^n$ is the inverse of $\mathbf{M}$ to the n'th power.

Taking the inverse of $\mathbf{M}$ will result in a dense matrix where a lot of the entries are close to zero (e.g. 10^{-20}). Doing calculations with such a matrix gives a lot of round off errors which will reduce the accuracy of the numerical exact. The error should theoretically be machine precision, but is expected, to at least, be much smaller than Δt. Errors from both 1D and 2D simulations are shown in Figure 3.7. Though computationally intensive, the numerical exact solution for the BE scheme has none of the limitations that were found for the FE scheme.



(a) (b)

Figure 3.7: Plots showing the error for the BE scheme in 1D (a) and 2D (b) compared to the numerical exact solution. The error is not machine precision, but significantly smaller than Δt which for these simulations is $\Delta t = 0.01$. This increased error originates in the many roundoff errors in the inverted matrix where a lot of terms 10^{-16} and smaller.

3.4 Verification of the RW solver

The aim of this section is to verify that the implemented RW model will solve the diffusion equation on the correct time scale, and to verify that the error term from the RW solver is dependent on the conversion factor, Hc. In order to carry out the tests, a new initial condition that can easily be recreated by random walkers will be introduced.

$$u(t=0,x) = H\left(x - \frac{1}{2}\right) \tag{3.12}$$

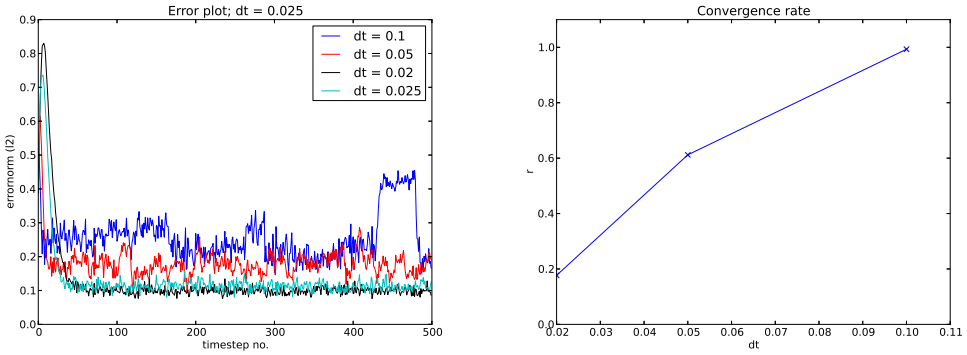
Where $H(x)$ denotes the Heaviside step function, which is defined by its properties

$$H(x-a) = \begin{cases} 1 & x > a \\ \frac{1}{2} & x = a \\ 0 & x < a \end{cases} \tag{3.13}$$

The new initial condition gives a new exact solution which is found by separation of variables and Fourier series as

$$u(x,t) = \frac{1}{2} + 2\sum_{n=\text{odd}}^{\infty} \frac{(-1)^{\frac{n-1}{2}}}{n\pi} e^{-(n\pi)^2 t}\cos(n\pi x) \tag{3.14}$$

Figure 3.9 verifies that the error term improves by adding more walkers. And Figure 3.8 shows that the RW model will fulfill the chosen time step on the PDE level. The convergence rate is not very good, but this is expected. Stochastic effects causes the error to fluctuate around a certain value rather than converge to zero, making it very hard to get a good error measure.



(a) (b)

Figure 3.8: Error plot (a) and convergence test (b) for 1D RW solver using a Heaviside step function as initial condition. In these tests both the time step and the conversion factor are changed for each simulation, and the conversion factor follows the previously proposed limit $Hc \geq \frac{1}{\Delta t^2}$. For each Δt the RW simulation does 250 steps with a step length calculated from (2.20). The expected convergence rate is 0.5, and to some extent this is achieved here. Note, however, that due to fluctuations in the solution, getting a good error measure is difficult and beyond our control.

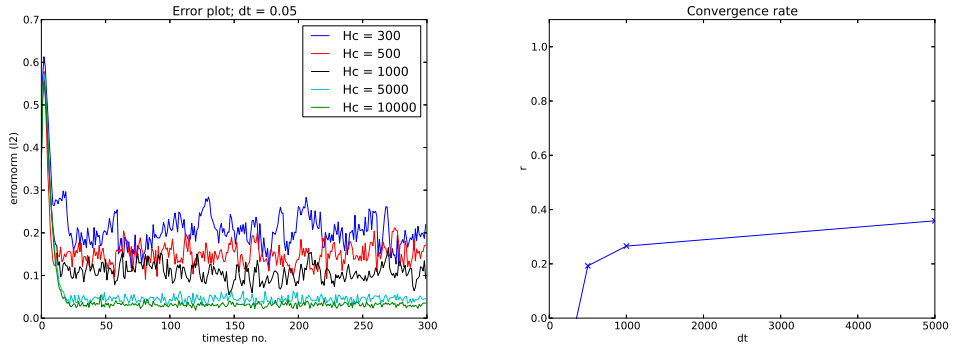


Figure 3.9: As a comparison to Figure 3.8, this test has been done for the same 1D Heaviside step function as initial condition, but keeps the time step fixed at $\Delta t = 0.05$ and increases the conversion factor. The convergence rate (fig. b) is worse than for the time convergence test, and seems to reach a limit where increasing the number of walkers has little effect on the error.

3.5 Verification of the hybrid solver

The hybrid diffusion solver is a combination of the PDE and RW solvers, which means that the error term for the hybrid model is described by

$$\epsilon(t) = C_t \Delta t + C_x \Delta x^2 + \frac{C_{RW}}{\sqrt{Hc}} \tag{3.15}$$

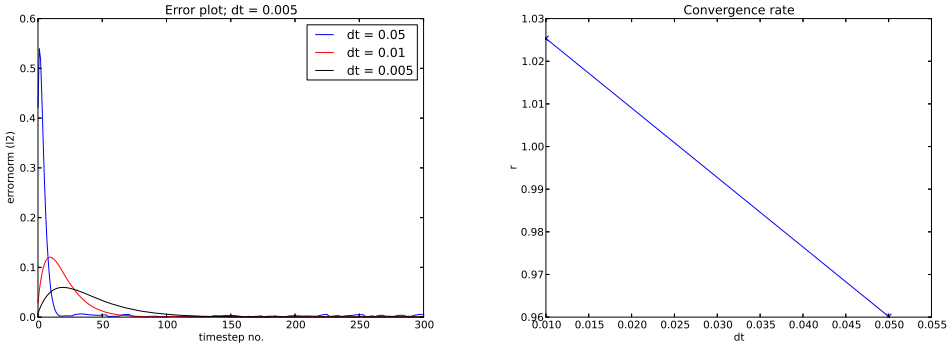
where C_{RW} is an unknown stochastic quantity.

This section aims to achieve first order convergence in time for the hybrid solver by introducing a sufficient number of walkers. Effects of varying the number of mesh points affected by the RW solver will also be illustrated.

The number of walkers needed to achieve first order convergence in time for the hybrid solver is governed by

$$Hc \geq \frac{1}{\Delta t^2} \tag{3.16}$$

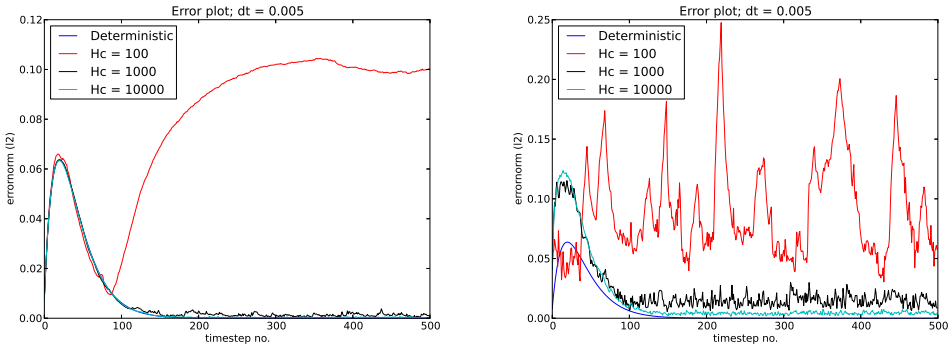
The stability criterion for the FE scheme limits the size of the time step, and by (3.16) vastly increases the conversion rate. All tests will be done with the BE scheme in order to reduce the demand for walkers and get a reasonable performance.
Figure 3.10 shows first order convergence for the hybrid model.



(a) (b)

Figure 3.10: 3.10a shows the error plot for a test where Δx was fixed at $\Delta x = \frac{1}{100}$ and Δt was reduced from 0.05 to 0.01 and finally to 0.005. The conversion rate, Hc was updated for each simulation to have the value $Hc = \frac{1}{\Delta t^2}$, meaning the error from the walkers should be smaller than the error from the time derivative. Walkers are placed on 10% of the mesh from $x = 0.4$ to $x = 0.5$. 3.10b shows the convergence rate in time for the same test.

Decreasing or increasing the relative size of the microscopic model has serious effects on the error. This is illustrated in Figure 3.11



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

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

Figure 3.11: The effect of increasing the size of the walk area for a fixed $\Delta t = 0.05$ and $\Delta x = 0.01$ using the BE discretization.

Chapter 4

Physical application

The following chapter is included to show that the developed hybrid diffusion solver can be applied to real life problems with only minor modifications.

4.1 Physical scope

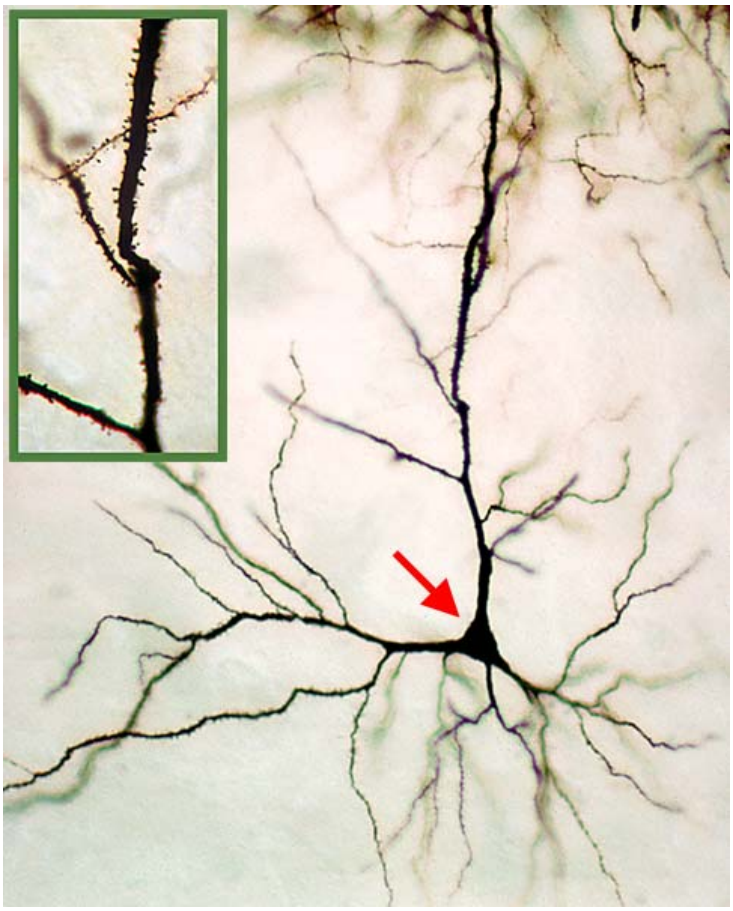


Figure 4.1: A pyramidal neuron with an apical dendrite (staring by the arrow). The inset shows an amplification of the apical dendrite and illustrates the size of dendritic spines (small outgrowths). The arrow points to the cell body, called the Soma. Note that this image does not show how neurons are located within the brain. Image from `www.neurolex.org`.

Figure 4.1 shows a pyramidal neuron with an apical dendrite, and an inset with dendritic spines. Pyramidal neurons take their name from the triangular shape of the Soma, and an apical dendrite is a dendrite springing from the apex of the Soma. Spines are (often) the receiving end of a chemical

synapse, which is a junction between two neurons.

This application will look at the diffusion of PKCγ in the apical dendrite and into dendritic spines. PKCγ is a protein found in neurons which is associated with memory storage and associative learning [8]. Upon activation it will diffuse out of the Soma, through a dendrite, and into a dendritic spine to reinforce or reduce the absorption of neurotransmitters. The results will be compared to results by Craske et.al [2], who in 2005 studied a similar problem in rodent pyramidal neurons harvested from the hippocampus.

Effectively we will investigate the diffusion time for random walkers through spine necks which are very narrow ($\leq 0.5\mu$m).

4.2 Implementation

There is a difference between the approach of the developed hybrid solver and the dendrite - spine system with respect to geometry which is best summarized in figure 4.2.

The dendrite is approximated as a cylinder with radius $\sim 10\mu$m and length $\sim 50\mu$m. Furthermore, little of interest is assumed to happen in the radial direction. All things considered, diffusion in the dendrite will be modeled as a 1D process.

Spines have a wide variety of geometries, with various properties. For this application, however, only the neck length and width of a spine is of interest. The spines will therefore be modeled as two dimensional objects with a funnel shape.

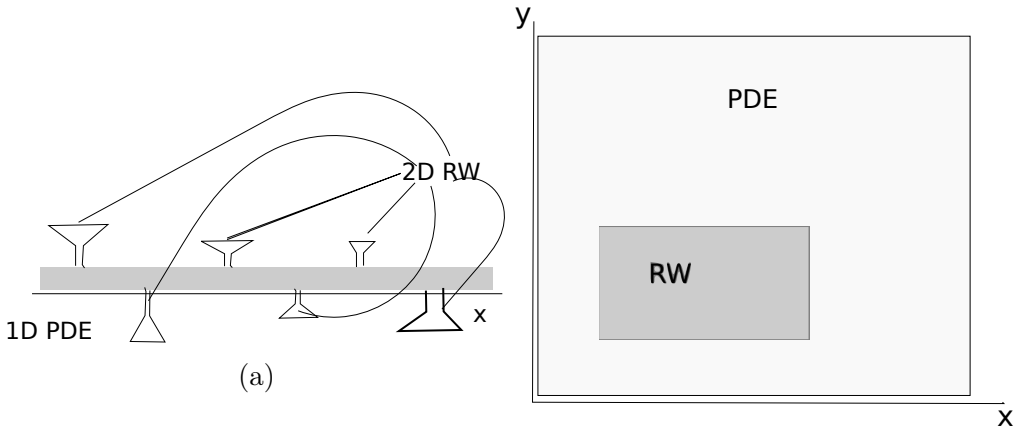


Figure 4.2: The geometric difference between the original hybrid diffusion solver (b) and the numerical setup to model PKCγ diffusing into dendritic spines (a). Spines are attached by a few mesh points on the PDE mesh, which also determines the neck width of the spine.

The hybrid diffusion solver has been slightly modified in order to recreate the new geometry. Mathematically the largest difference is that the random walkers are left alone, meaning that the diffusion equation is not solved in the spines. There are also some differences in the coupling; at each PDE time step there is a probability for an enzyme to diffuse into a spine if the concentration at the beginning of the spine neck is large enough. This probability decreases with the number of enzymes which have already diffused into the spine. The width of a spine neck is determined by the number of PDE mesh points it connects to, which is chosen at random. All other length parameters are also determined randomly, but required to fit with measurements from Arrelano et.al [1].

4.2.1 Initial Condition

In the presence of glutamate the diffusion constant for PKCγ has been measured to $0.33\frac{\mu m^2}{s}$, but this does not reproduce the diffusion times in dendrites which are illustrated by Craske et.al. There are two possible fixes to this; *either I have read the paper wrong, and the diffusion constant is really* 5.72 *in the dendrite, in which case everything is ok.* The other possibility is to modify the initial condition. The latter is not as far fetched as is sounds because there is considerable absorption of PKCγ in the dendrite walls during the first few seconds after release from soma. The modified initial condition can

be viewed as the concentration distribution in the dendrite some 2 seconds after release from soma.

4.3 Parameters

The new problem requires the introduction of some additional parameters as well as a reformulation of limiting factors for some known parameters, all of which are described in the following table.

Parameter	**Explanation**	**Expression/ typical value**	**Origin**
Δt	time step	Δx^2	Stability criterion FE scheme
Δx	spatial resolution	$\frac{1}{2}$min(spine neck width)	Estimated; in order for a spine to be 2D, it must connect to at least two PDE mesh points.
Hc	Conversion factor	5-24	Estimated by calculations of concentration levels taken from [4] and spine/dendrite volume ratios. See later for discussion.
$u(t=0)$	Initial condition value	$5\frac{\text{nMol}}{\text{L}}$	Estimated from values found in [4]
p_{ds}	Probability to diffuse into a spine	$0.1 \cdot \Delta x \cdot \Delta t$	Estimated. An important ability of this parameter should be that wide necked spines have larger probability and that a certain flux is maintained (on average), meaning that the flux should be independent of Δt
p_{ab}	Probability for PKCγ to be absorbed, and removed from simulation, per time-step taken in spine head	100%	Estimated

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

Chapter 5

Results

This chapter will summarize and discuss the results from chapters 3 and 4.

5.1 Results of verification

5.1.1 The FE scheme

Although not all of the results from verification of the FE scheme are perfect, the results from testing the exact numerical solution are very good. The numerical exact solution test is a verification of the implementation of the FE scheme. A successful test indicates that the scheme is correct and will solve the diffusion equation to the expected accuracy. Both the error plots from section 3.3.1 and the spatial convergence test are very close to perfect, further suggesting a correct implementation. When the general convergence tests for the FE scheme could be better (figures 3.3), it is most likely due to other effects.
Considering that two out of three tests are close to perfect, the FE scheme is deemed correctly implemented within the limits of the applied tests.

5.1.2 The BE scheme and the block tridiagonal solver

Two of the tests done on the BE scheme are slightly off target. These are the spatial convergence test and the numerical exact solution.
Though the numerical exact solution test is expected to give an error close to machine precision the results are five orders of magnitude larger than machine precision. On the other hand, the error term is also eight orders of magnitude smaller than the time step and there are a lot of possible round off errors in the inverted coefficient matrix which can explain the lack of accuracy.
The spatial convergence test demands that the error associated with the spatial derivative is much larger than the error from the time derivative. As the spatial resolution is increased, the spatial error becomes less dominant because the time step is held constant. Because the spatial convergence is still clearly larger than one, and starts out at two (which is the value it should converge to), this is considered to be adequate.

Effectively, the manufactured solution test, and convergence tests verifies the assembly of the coefficient matrix, since an incorrect assembly would produce a larger error term. The numerical exact solution test verifies the implementation and accuracy of the block tridiagonal solver, since an incorrectly implemented solver would give results comparable to the numerical

exact solution.

In summary, the tests done on the BE scheme verifies that it, and the block tridiagonal solver, is implemented correctly.

5.1.3 The RW solver

Due to fluctuations in the solution it is hard to get a good convergence rate for the RW solver. However it is measured to be close to 0.5 both with respect to the time step and with respect to the number of walkers introduced. The result of verifying the RW solver is that it is deemed correctly implemented within the limits of the applied tests.

5.1.4 The hybrid diffusion solver

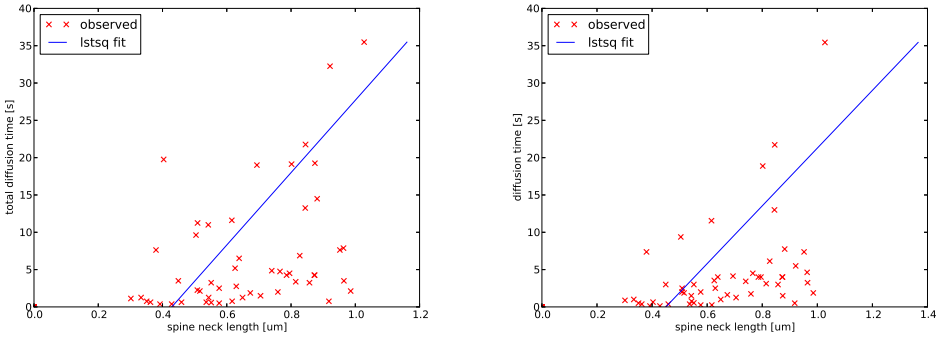
For this thesis it is considered very important that the hybrid diffusion solver can have first order convergence in time given a high enough conversion rate. As Figure 3.10b shows this was achieved, and the hybrid solver is therefore considered adequately implemented for the time being.

Effects of varying the relative size of the RW area were also illustrated. The result of increasing the portion of the PDE mesh affected by an RW model is clearly a larger error term overall, while decreasing the size of the RW area significantly improves the error term. All of this is expected since more fluctuations are introduced.

5.2 Results for physical application

Craske et.al. suggest that the neck of spines act as diffusion barriers which slow down, but don't completely stop the diffusion of PKCγ into spines. The function of this barrier is a bit unclear, but the presence of it is undisputed. In their measurements they found a delay of around $5 - 10$ seconds from elevated concentration levels in the dendrite until a similarly elevated concentration level occurred in spines with necks longer than 0.5μm. Using parameter values which resemble the values found in actual (rodent) neurons and neurites in the developed software, the observed delay-times have been recreated. Figure 5.1 shows plots of the observed diffusion times into spines. This figure shows a clear trend for longer diffusion times as the neck length of the spine increases. Figure 5.2 further support this claim and implies the average diffusion time for PKCγ into long necked spines to be of accordance

to the results from Craske et.al. Seeing as there are no additional complexities added to the random walk model we can assume that the spine neck does in fact function as a diffusion barrier.



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

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

Through the simulations it became apparent that there must be some sort of limiting factor which limits the number of PKCγ particles that are let into the spine. In real life this is achieved by a concentration gradient which tends to zero (or negative values) meaning that no particles will diffuse into the spine after it is "filled" up. A random walker will not feel this concentration gradient unless it is explicitly told so. The alternative solution then, is to reduce the probability for particles to diffuse into a spine for each particle that get caught in the spine head by a factor $\frac{1}{10}$.

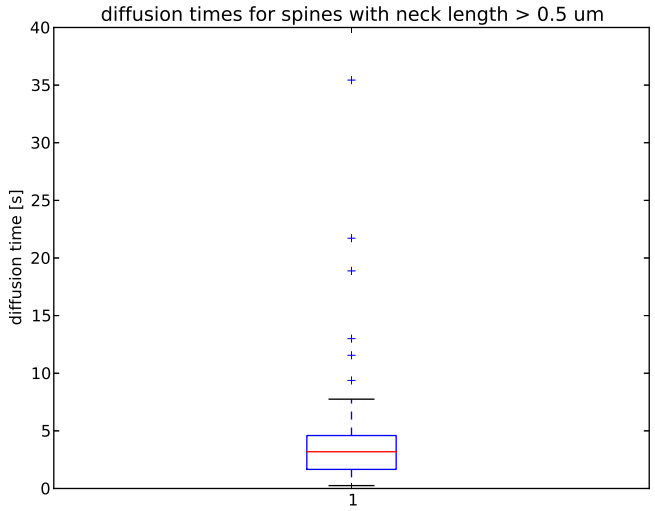


Figure 5.2: Boxplot of the relative diffusion times (time between elevated concentration in dendrite and elevated concentration in spine head) into spines with necks longer than 0.5μm. Similar studies were done by Crase et.al. and found diffusion time (unclear whether relative or not) to be somewhere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

Chapter 6

Discussion and conclusion

6.1 Discussion

A large portion of the work that has been put into this thesis has been on code implementation. Because it is really demanding to read computer code, and it does not necessarily provide extra clarity, the code is not included in this text. Should it be of interest, the complete computer code is published at `github.com` under the following address

`https://github.com/fepettersen/thesis`

In summary, this project contains

- Implementation of both explicit and implicit solvers for the (anisotropic) diffusion equation using finite difference methods in one and two spatial dimensions. The implicit solver can also handle three dimensions, but the discretization is incomplete.
- Implementation of a direct solver for banded linear systems which rewrites the system as block tridiagonal. As far as I'm aware, this block tridiagonal solver is one order more efficient than all other direct solvers.
- Implementation of a random walk model for (anisotropic) diffusion in one and two spatial dimensions, with the grounds for three dimensions laid.
- Combination of the PDE and RW models into a hybrid diffusion model.
- Thorough testing and verification of all the implemented solvers.
- An application of a slightly modified version of the hybrid diffusion model on diffusion of PKCγ from thick apical dendrites into very narrow dendritic spines.

6.1.1 Application

The model gives fairly good results for the application on PKCγ into spines which largely are in agreement with the results by Craske et.al. However, the mean diffusion times found from simulations seem to consistently lie towards the lower limits of the experimental results. One possible extension to the model which might fix this is to introduce an absorption probability in the spine head which is fairly large (say 80% per second), or even increases with the amount of time spent in the spine head. A setup like this should increase

the average diffusion time by a few seconds. It does not, however reflect the physical process to a better accuracy seeing as a concentration increase in a spine head will be measured quickly.

6.1.2 Possible extensions

As the project stands now it is mostly a proof of concept for hybrid diffusion solvers as well as a first attempt at a flexible framework for similar problems. This means that there are quite a few possible extensions that can be done to the project, perhaps in conjunction with another masters thesis.

A simple RW was chosen for the lower scale model because it fulfills the diffusion equation and is therefore easy to work with. The idea, however, was always to create a software in which the lower scale model can easily be substituted for a better one. By letting the lower scale model work as a standalone unit which communicates with the rest of the software through a file containing the positions of all the walkers (or particles), this is ensured. All that is needed to switch lower scale model is to put the new solver in the correct place with respect to the rest of the software, and make sure it can communicate in the described manner. As a test, the DSMC code developed by Anders Hafreager was used as a lower scale model for one simulation. Naturally, a few problems arise, but from a strictly programming point of view it worked.

As mentioned in section 2.3.4, perfect flux exchange between the higher and lower scale model might have been a better boundary condition for the RW model than zero flux boundary conditions. In principle, changing boundary conditions on the RW solver is simple, seeing as it is completely separate from the rest of the software. The reason the boundary conditions have not been changed is that it requires a complete workover of the coupling between the two models as well, which is a much larger job.

Perhaps the biggest weakness of the software, as it stands, is the limitation in mesh geometry. Implementing a mesh geometry which is not square in a finite difference method turns out to be very complicated, and requires transforming the PDE to a new set of coordinates. Ultimately one must solve an entirely different equation. Alternatively, a finite element method can be used. Finite element software will already have support for new mesh geometries built in, making the suggested transformations unnecessary. Implementing a finite element PDE solver will also make it a lot easier to use a higher order approximation to the time derivative, hence getting a more accurate PDE solution.

There are, of course, a lot of physical problems which the developed software can be applied to. One possibility is modeling for example sodium ions that diffuse through ion channels in the cell membrane of neurons into the extracellular space of the brain. Ion channels are very narrow, often only allowing one ion through at a time, and they are (often) specific to one ion. Two 1D PDEs can describe the intra-, and extra-cellular concentrations of ions, and a lower scale model can describe the ion channel. This problem would also need to consider drift terms arising from Coulomb forces.

In principle, any diffusion process a small portion of a large number of particles diffuse into narrow passages could be modeled by the developed hybrid diffusion solver.

6.1.3 Other work on the subject

As the project was being finished, I came across an article by Flekkøy et.al.[3] describing the same problems that are addressed in this thesis. In this article, the authors try to combine a simple diffusion solver with a simple random walk solver and end up concluding that this is possible. This thesis has been done completely independently of said article, and takes a slightly different approach to the problem as well.

6.2 Concluding remarks

In this thesis a hybrid diffusion solver in which parts of the process can be modeled by a particle dynamics description has been developed. All parts of the solver have been verified to work properly, including the hybrid model.

The developed software mainly relies on the implicit BE scheme to solve the diffusion equation, both in 1D and in 2D. Like any implicit scheme, the BE scheme results in a system of linear equations of size $n^d \times n^d$ which must be solved at each time step. In order to do so, a block tridiagonal solver has been implemented, with an efficiency of $\mathcal{O}(n^{2d-1})$. To my knowledge, this is the most effective direct solver available, with alternatives like LU decomposition and Levinson recursion using $\mathcal{O}(n^{2d})$ FLOPs. The limiting factors of the block tridiagonal solver are two matrix-vector multiplications which will use $\mathcal{O}(n^{2(d-1)})$ FLOPs. If a faster matrix-vector multiplication scheme exists it will reduce the computational work to $\mathcal{O}(n^d)$.

Appendix A

Appendix

A.1 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.1.1 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.1.2 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.1.3 Finite difference methods

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

- Manufactured solution
  Find some function which fulfills the equation you are working with.

Remember that you have a source term which can be whatever you want it to be at this point, meaning that you can more or less decide what solution you want to your equation.

- Stationary solution
  This boils down to energy-conservation. If the initial condition is a 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 3.3.3. Your scheme should reproduce this solution to more or less machine precision. Note that you might run into round-off errors and overflow here in some cases (again, see chapter 3.3.3).

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

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

A.1.4 Random walk and Monte Carlo methods

The main difference between debugging a MC based solver and a deterministic solver is the fact that you often do not have a clear idea of what the results of the intermediate steps should be. What you might know (or should know during development), however is the result of the complete MC integration, and some statistical properties of your random numbers. Using uniformly

distributed random numbers will give you a certain mean and standard deviation, and a Gaussian distribution will give you another. You should check that the random number generator (RNG) you chose actually reproduces these properties to a reasonable precision. If you are working with random walkers it also helps to look at the behavior of a small number of walkers, to check that they behave more or less as you expect. One thing to look out for is the fact that a random walker in both one and two spatial dimensions will fill all space given enough steps. Of course enough steps is infinitely many, but if you also implement reflecting boundaries and use some 4-5 walkers you will see a tendency after approximately 10^4 steps.
As we have discussed earlier the fluctuations in a MC-model are usually of a magnitude $\frac{1}{\sqrt{N}}$ this is also smart to verify.
Finally, you should absolutely have the possibility to set the random seed and check that two runs with the same random seed produces the exact same result and makes sure you are using a RNG with a large enough period. The xor-shift algorithm by Geroge Marsagla [] has a period of around 10^{48} which usually is more than adequate.

A.1.5 The developed software

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

Another point to be made is that it will probably be helpful to construct the different parts of your code in such a way that they can be run as independently of each other as possible. As an example, both the PDE-solver, its tridiagonal linear system solver and the spine object can with relatively

small changes to the main-file be run independently. This allows for easier testing of the various parts of the code, and makes it more likely that the code will be reused in other projects.

A.1.6 When you cannot find the bug

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

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

Bibliography

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

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

[3] EG Flekkøy, J Feder, and G Wagner. "Coupling particles and fields in a diffusive hybrid model". In: *Physical Review E* 64.6 (2001), p. 066302.

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

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

[6] Sean T O'Connell and Peter A Thompson. "Molecular dynamics–continuum hybrid computations: a tool for studying complex fluid flows". In: *Physical Review E* 52.6 (1995), R5792.

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

[8] Naoaki Saito and Yasuhito Shirai. "Protein kinase Cγ (PKCγ): function of neuron specific isotype". In: *Journal of biochemistry* 132.5 (2002), pp. 683–687.