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

Chapter 1

Introduction

The diffusion equation governs a large number of physical processes on many different length scales. Both heat diffusion through a wall and the Brownian motion of dust particles can be described by the diffusion equation. In some diffusion processes a large number of particles will diffuse from a large volume into a small volume in such a way that only very few particles enter the smaller volume. An example of this in the diffusion of the enzyme PKCγ from the body of a brain cell via a dendrite of large volume into dendritic spines. Dendritic spines are the receiving end of a connection between two brain cells, and PKCγ is associated with long term reinforcement of such a connection resulting in learning or associative memory storage. The increased concentration of PKCγ in a dendritic spine is measured to around $5\frac{\text{nMol}}{\text{L}}$ which translates to $1-2$ PKCγ enzymes in the spine. Seeing as the diffusion equation is derived from a continuous concentration distribution, using it on the enzymes in the spine seems questionable.

This thesis will look at hybrid diffusion processes where part of the process has very few particles and other parts have enough particles to be described by a continuous diffusion equation. The part with few particles will have a higher resolution in both space and time, and will be modeled by a stochastic process. 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.

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 a distribution of random walkers, C is chosen so that all the walkers are placed within an imagined volume, V of arbitrary shape. V is contained by a surface S and the relation is shown in Figure 2.1.

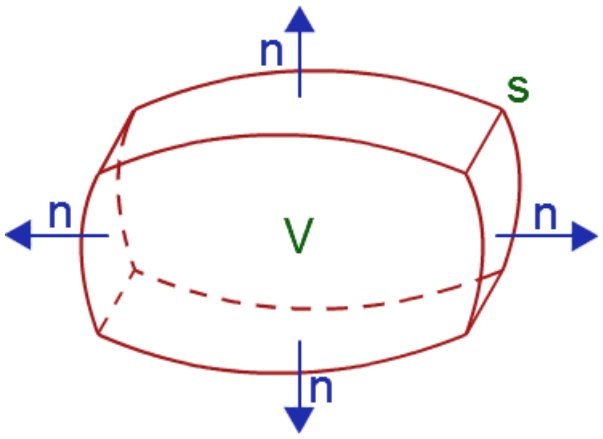


Figure 2.1

As the random walkers start to move some of them will leave the volume V which results in a change in C. The net movement of random walkers at a point is called the flux of walkers and is denoted by the flux vector $\mathbf{J}$. To find the number of walkers that leave V we integrate the normal component of $\mathbf{J}$ over S. Assuming that no walkers disappear or are created the flux of walkers leaving V must be balanced by the decrease in C, expressed in equation (2.1).

$$\int_V \frac{\partial C}{\partial t}\, dV = \int_S \mathbf{J} \cdot \mathbf{n}\, dS \tag{2.1}$$

Through Greens theorem equation (2.1) can be reformulated

$$\int_V \frac{\partial C}{\partial t}\, dV = \int_V \nabla \cdot \mathbf{J}\, dV \tag{2.2}$$

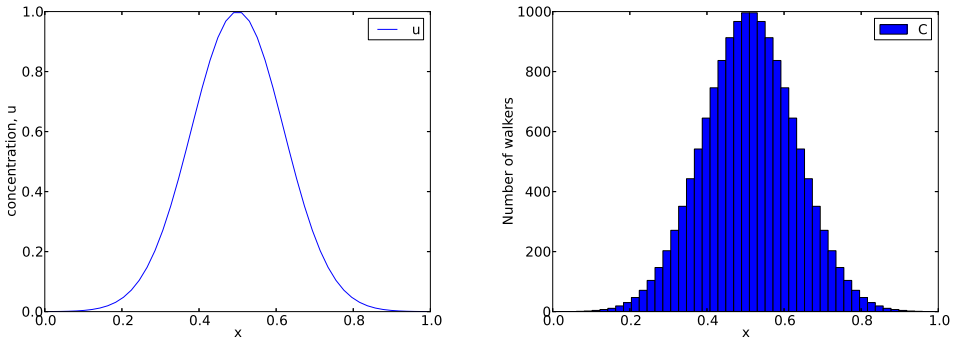
The flux $\mathbf{J}$ can be expressed by Fick's first law as the diffusive flux

$$\mathbf{J} = D\nabla C \tag{2.3}$$

where D is the diffusion constant. Since V was chosen as an arbitrary volume equation (2.2) is independent of the integration and the integrals can be dropped. The diffusive flux $\mathbf{J}$ is also inserted to yield the diffusion equation

$$\frac{\partial C}{\partial t} = D\nabla^2 C \tag{2.4}$$

Throughout this thesis C will denote a spatial distribution of random walkers. 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 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 [1]. This generator is chosen because of its very large period 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. This section will concern the special case of how to solve PDEs by the implicit Backward Euler (BE) scheme, especially in more than one spatial dimension. First, let us look at the BE discretization of the diffusion equation in 1D.

$$u_i^{n+1} = \frac{\Delta t}{2\Delta x^2}\left((D_{i+1}+D_i)(u_{i+1}^{n+1}-u_i^{n+1}) - (D_i+D_{i-1})(u_i^{n+1}-u_{i-1}^{n+1})\right) + u_i^n \tag{2.5}$$

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 equation (2.6).

$$\frac{\partial u}{\partial n} = 0|_{\text{on boundary}} \tag{2.6}$$

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

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

which is arranged as

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 \\ 0 & a_2 & b_2 & c_2 \\ 0 & 0 & a_3 & b_3 \end{pmatrix} \mathbf{u}^{n+1} = \mathbf{u}^n \tag{2.7}$$

$$\mathbf{M}\mathbf{u}^n = \mathbf{u}^{n-1} \tag{2.8}$$

If a linear system like the one in equation (2.8) 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 (2.8) is by exploiting the fact that it is tridiagonal. In order to be more consistent with the algorithm, eq. (2.8) will be rewritten as

$$\mathbf{M}\mathbf{u} = \mathbf{up} \tag{2.9}$$

where **up** denotes the solution **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 in equation (2.10)

$$u_{i,j}^n = \underbrace{\frac{-D\Delta t}{\Delta x^2}}_{\alpha}\left(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}\right) + \underbrace{\left(1 + \frac{2D\Delta t}{\Delta x^2} + \frac{2D\Delta t}{\Delta y^2}\right)}_{\gamma} u_{i,j}^{n+1} \underbrace{-\frac{2D\Delta t}{\Delta y^2}}_{\beta}\left(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right) \tag{2.10}$$

Equation (2.10) assembles to the linear system in equation (2.11) when solved on a 3×3 mesh.

$$\left(\begin{array}{ccc:ccc:ccc} \gamma & -2\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 & 0 \\ -\beta & \gamma & -\beta & 0 & -2\alpha & 0 & 0 & 0 & 0 \\ 0 & -2\beta & \gamma & 0 & 0 & -2\alpha & 0 & 0 & 0 \\ \hdashline -\alpha & 0 & 0 & \gamma & -2\beta & 0 & -\alpha & 0 & 0 \\ 0 & -\alpha & 0 & -\beta & \gamma & -\beta & 0 & -\alpha & 0 \\ 0 & 0 & -\alpha & 0 & -2\beta & \gamma & 0 & 0 & -\alpha \\ \hdashline 0 & 0 & 0 & -2\alpha & 0 & 0 & \gamma & -2\beta & 0 \\ 0 & 0 & 0 & 0 & -2\alpha & 0 & -\beta & \gamma & -\beta \\ 0 & 0 & 0 & 0 & 0 & -2\alpha & 0 & -2\beta & \gamma \end{array}\right) \mathbf{u}^{n+1} = \mathbf{u}^n \quad (2.11)$$

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.

$$\left(\begin{array}{c:c:c} B_0 & C_0 & 0 \\ \hdashline A_1 & B_1 & C_1 \\ \hdashline 0 & A_2 & B_2 \end{array}\right) \mathbf{u}^{n+1} = \mathbf{u}^n \quad (2.12)$$

All entries in eq. (2.12) 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 (2.13)

There is a forward substitution

$$\begin{aligned} H_0 &= -B_0^{-1} C_0 \\ \mathbf{g}_0 &= B_0^{-1} \mathbf{up}_0 \\ H_i &= -\left(B_i + A_i H_{i-1}\right)^{-1} C_i \\ \mathbf{g}_i &= \left(B_i + A_i H_{i-1}\right)^{-1} \left(\mathbf{up}_i - A_i \mathbf{g}_{i-1}\right) \end{aligned} \quad (2.13)$$

Followed by a backward substitution

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

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. (2.11) 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 Forward Euler (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 can be solved by multiplying both sides of the equation with the inverse of the matrix in question.

$$\mathbf{M}^{-1}\mathbf{M}\mathbf{u} = \mathbf{M}^{-1}\mathbf{u}\mathbf{p}$$

Inverting a dense matrix demands $\mathcal{O}(n^3)$ FLOPs for an $n \times n$ matrix. By rewriting the previously mentioned $n^2 \times n^2$ band diagonal matrix as a block tridiagonal matrix and using the modified block tridiagonal solver we have reduced the demanded number of FLOPs from $\mathcal{O}(n^6)$ to $n \cdot \mathcal{O}(n^3) = \mathcal{O}(n^4)$. Though an improvement of 2 orders is very good, compared to the FE scheme this is still one order larger. there is still room for improvement, however. As long as the mass matrix is unchanged, which it will be as long as the time step and diffusion constant are constant, the n matrix inversions need only be done once. In other words, the H_i terms from eq. (2.13) can be stored. The remaining calculations include three matrix vector multiplications, all of which demand n^2 FLOPs, thus further reducing the computational cost to $\mathcal{O}(n^2)$ FLOPs which is the same as the FE scheme.

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 last 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** 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 [2]) which is defined in equation (2.14)

$$C_{ij} = Hc \cdot u_{ij} \tag{2.14}$$

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 $y_j \pm \frac{\Delta y}{2}$.

Equation (2.14) effectively states that one "unit" of the solution u will directly correspond to Hc random walkers.

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.

$$\langle \tilde{\Delta x}^2 \rangle = 2Dd\tilde{\Delta}t \tag{2.15}$$

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 (2.15) gives the final expression for the step length

$$l = \sqrt{2dD\frac{\Delta t}{\tau}} \tag{2.16}$$

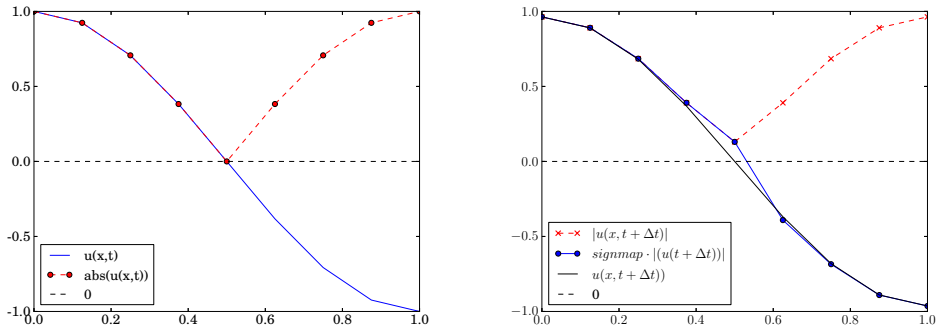
2.3.4 Boundary conditions for the random walk

2.4 Potential problems or pitfalls

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

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 well it works (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

Number of time-steps on the random walk level

Chapter 3

Analysis

3.1 The error estimate

Solving PDEs numerically will result in errors because derivatives are approximated by finite differences. For the schemes used in this thesis the error, ϵ will follow equation (3.1)

$$\epsilon = C_x \Delta x^2 + C_t \Delta t \tag{3.1}$$

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.

The error 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 equation (3.2)

$$\begin{aligned} \epsilon(t^n) &= ||u(t^n) - u_e(t^n)||_2 \\ &= \sqrt{\Delta x \Delta y \sum_{i=0}^{n} \sum_{i=0}^{n} \left(u(t^n, x_i, y_j) - u_e(t^n, x_i, y_j)\right)^2} \end{aligned} \tag{3.2}$$

ϵ is time dependent 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 $\epsilon(t)$, defined in eq. (3.3) is used.

$$\epsilon = \sqrt{\Delta t \sum_{n=0}^{T} \epsilon(t^n)^2} \tag{3.3}$$

3.2 Verification techniques

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 error term follows equation (3.1). 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. 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 omitted for this scheme.

The verification techniques are

- Manufactured solutions
  By choosing an adequate initial condition the exact solution to the diffusion equation can be found with relative ease. The chosen solution is
  $$u(x, y, t) = e^{-\pi^2 t} \cos(\pi x) \cos(\pi y) + 1 \tag{3.4}$$
  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 proportional to the time step to some power r when the error from the time derivative is dominant.
  $$\epsilon \simeq C_t \Delta t^r \tag{3.5}$$
  By comparing the error from two simulations with different time steps the exponent r, called the convergence rate, can be found
  $$\begin{aligned} \epsilon_1 &\simeq C_t \Delta t_1^r \\ \epsilon_2 &\simeq C_t \Delta t_2^r \end{aligned}$$
  the two expressions are divided
  $$\begin{aligned} \frac{\epsilon_1}{\epsilon_2} &\simeq \frac{C_t \Delta t_1^r}{C_t \Delta t_2^r} \\ \log\left(\frac{\epsilon_1}{\epsilon_2}\right) &\simeq r \log\left(\frac{\Delta t_1}{\Delta t_2}\right) \\ r &\simeq \frac{\log\left(\epsilon_1/\epsilon_2\right)}{\log\left(\Delta t_1/\Delta t_2\right)} \end{aligned}$$
  The FE and BE schemes have errors proportional to Δt so the tests are expected to measure $r = 1$.

- Exact numerical solutions
  The numerical schemes are actually reformulations of the PDE we are trying to solve 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

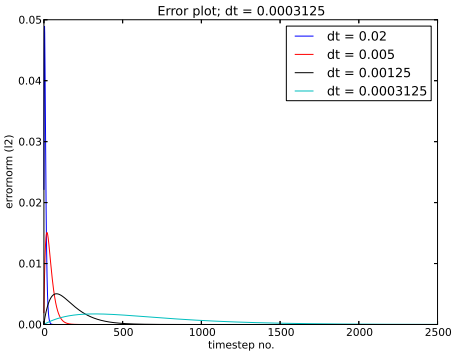
3.3.1 Verification by manufactured solutions

Equation (3.4) 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 the discretization scheme is that the difference between these solutions is of the same order as the time step.

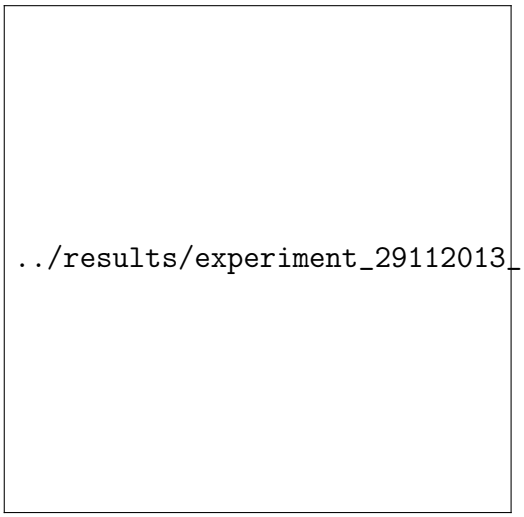
$$\epsilon(t) \sim \mathcal{O}(\Delta t)$$

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. 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 equation (3.4) 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)

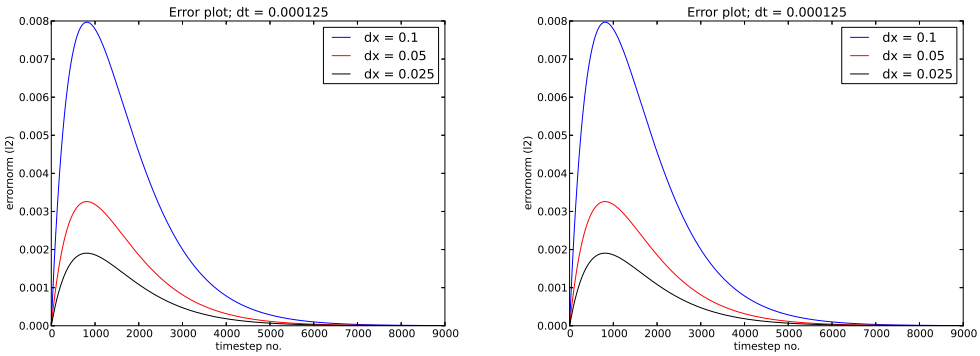


(b)

Figure 3.1

3.3.2 Verification by convergence tests

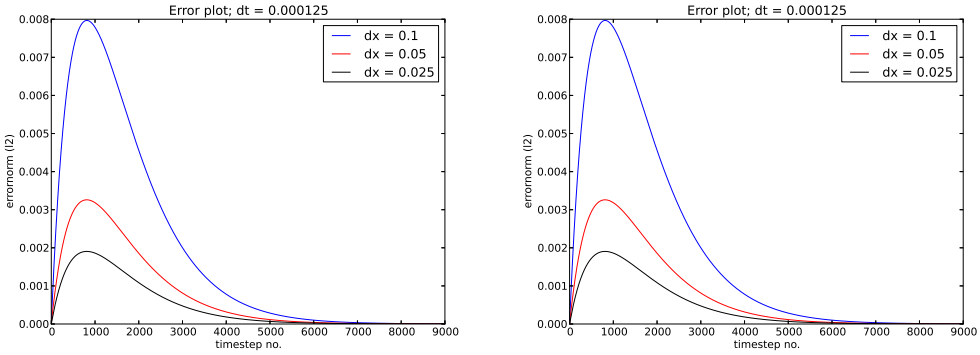
The convergence tests are done by isolating the error term from either the time derivative or the spatial derivative, and refining the relevant discretiza-



(a) (b) *This is not the correct plot!*

Figure 3.2

tion parameter over several simulations. Figures 3.3 and 3.4 verify that the error associated with the time derivative is of the expected order while figure **??** verifies the error from the spatial derivative for the BE scheme.



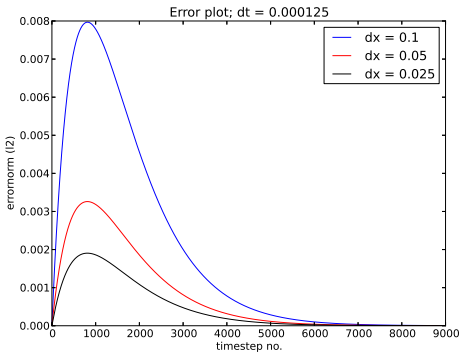
(a) (b) *This is not the correct plot!*

Figure 3.3

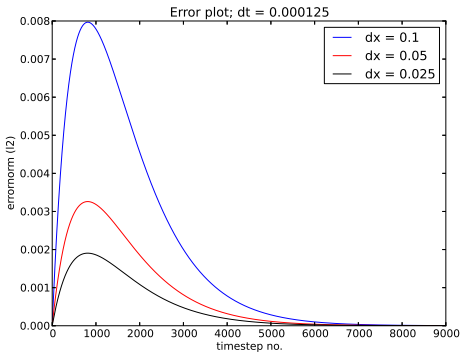
3.3.3 Verification of FE scheme by exact numerical solution

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

$$u^{n+1} = D\Delta t u^n_{xx} + u^n \tag{3.6}$$

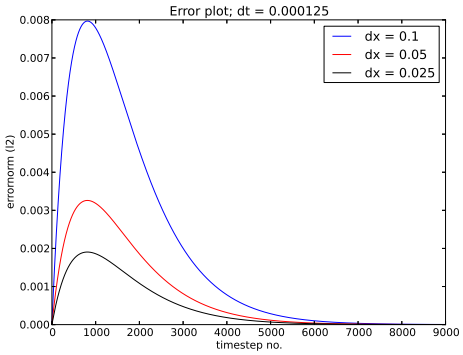


(a)

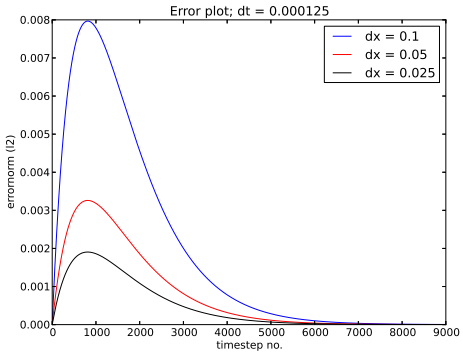


(b) *This is not the correct plot!*

Figure 3.4



(a)



(b) *This is not the correct plot!*

Figure 3.5

where u_{xx} denotes the double derivative of u with respect to x. To illustrate how the equation is solved by the computer, the first four iterations are

written out

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

From the iterations above a pattern emerges for iteration $n+1$

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i u^0_{2ix} \tag{3.7}$$

Where u^0 is the initial condition

$$u^0 = \cos(\pi x) \tag{3.8}$$

The spatial derivatives are found as

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

The pattern continues allowing the final numerical exact solution to be expressed in equation (3.9).

$$u^{n+1} = \sum_{i=0}^{n} \binom{n}{i} (D\Delta t)^i \frac{2^i}{\Delta x^{2i}} \left(\cos(\pi \Delta x) - 1 \right)^i \cos(\pi x) \tag{3.9}$$

Although the FE scheme is expected to represent equation (3.9) 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, equation (3.9) 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. 3.4) is replaced by an amplification factor A^n. This amplification factor can be found in equation (3.9) as

$$A^n = \left(\frac{2D\Delta t}{\Delta x^2}\right)^i \tag{3.10}$$

Inserting a time step larger than the stability criterion ($\Delta t \leq \frac{\Delta x^2}{2D}$) will make the amplification factor A larger than one which in turn will make the solution blow up.
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, i, resulting in a number comparable to zero.

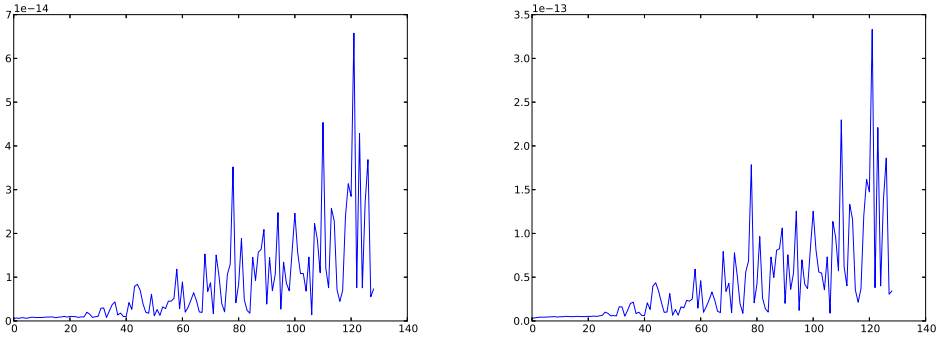
The results from comparing a 1D simulation to the numerical exact is shown in Figure 3.6a. As expected the error is larger than machine precision by at most two orders of magnitude because of accumulating error terms from the dropped terms in eq. (3.9).

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

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

The same problems as in the 1D case will apply to equation (3.11) 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 suggesting that the scheme is implemented correctly.



(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.

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

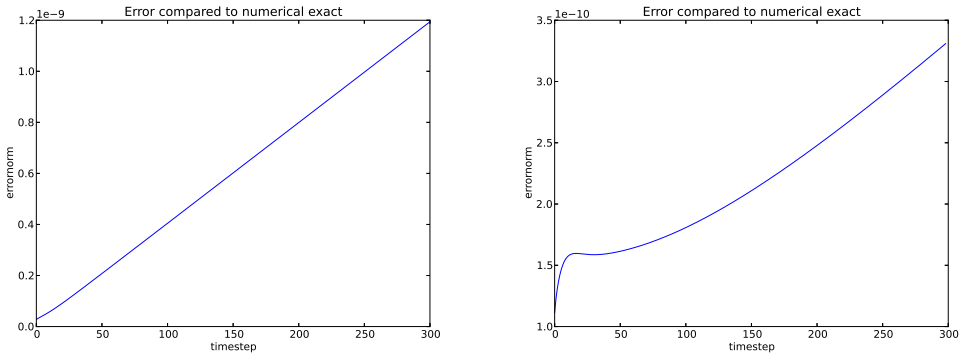
Doing the separation to the end relates the n'th time step to the initial condition

$$\mathbf{u}^{n+1} = \left(\mathbf{M}^{-1}\right)^{n+1}\mathbf{u}^0 \tag{3.12}$$

Note that $(\mathbf{M}^{-1})^{n+1}$ is the inverse of $\mathbf{M}$ to the $n+1$'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



(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.

Bibliography

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

[2] 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.