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 each particle will be modeled as a random walker which hops to the left or to the right at every time step with equal probability. A large emphasis has been put on verification of the average behavior of the system in the limit where many particles are on the microscopic scale.

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.

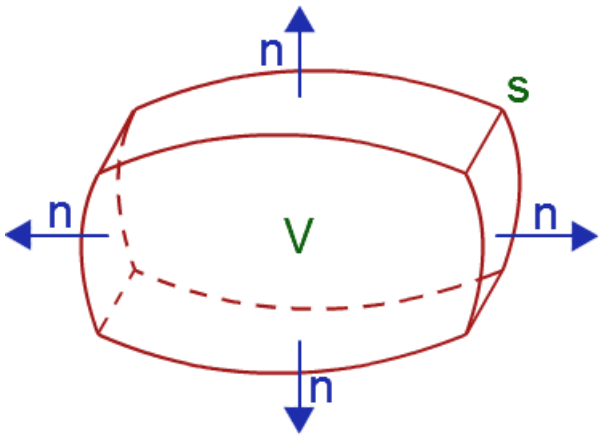


Figure 2.1

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

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

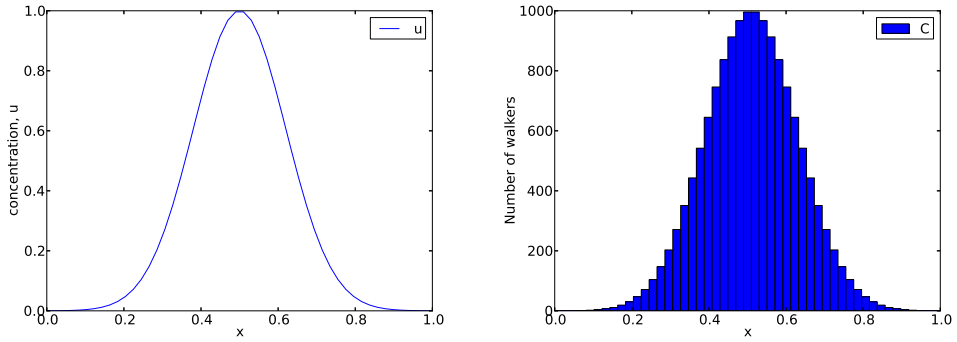
The flux $\mathbf{J}$ can be expressed by Fick's first law as the diffusive flux

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

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 Pseudo 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 Numerical solution of partial differential equations

The basics of solving partial differential equations (PDEs) by finite difference methods is considered to be known. 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

$$\begin{pmatrix}
1+2a(D_0+D_1) & -2a(D_1+D_0) & 0 & 0 \\
-a(D_1+D_0) & 1+a(D_2+2D_1+D_0) & -a(D_2+D_1) & 0 \\
0 & -a(D_2+D_1) & 1+a(D_3+2D_2+D_1) & -a(D_3+D_2) \\
0 & 0 & 1+2a(D_3+D_2) & -2a(D_3+D_2)
\end{pmatrix}\mathbf{u}^{n+1} = \mathbf{u}^n \tag{2.7}$$

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

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 more efficient way to solve the linear system in eq. (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

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

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 will result in a a $2n$ band diagonal matrix of size $n^2 \times n^2$ with 5 nonzero bands. This band diagonal matrix can be rewritten as a block tridiagonal matrix with $n \times n$ matrices as entries. A simplified block tridiagonal linear system is shown in eq. (2.10)

$$\begin{pmatrix} B_0 & C_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & A_i & B_i & C_i & 0 & & 0 & 0 \\ 0 & \ddots & 0 & \ddots & \ddots & \ddots & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{n-1} & B_{n-1} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0^{n+1} \\ \mathbf{u}_1^{n+1} \\ \vdots \\ \mathbf{u}_i^{n+1} \\ \vdots \\ \mathbf{u}_n^{n+1} \end{pmatrix} = \mathbf{up} \quad (2.10)$$

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. The calculations for this are done in the appendix if the reader should be interested.

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 (2.11)

There is a forward substitution

$$\begin{aligned} H_1 &= -B_1^{-1} C_1 \\ \mathbf{g}_1 &= B_1^{-1} \mathbf{up}_1 \\ H_i &= -\left(B_i + A_i H_{i-1}\right)^{-1} C_i \\ \mathbf{g}_i &= \left(B_i + A_i H_{i-1}\right)^{-1} \left(\mathbf{up}_i - A_i \mathbf{g}_{i-1}\right) \end{aligned} \quad (2.11)$$

Followed by a backward substitution

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

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).

In general, a linear system can be solved by multiplying both sides of the equation with the inverse of the matrix in question.

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

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. (2.11) 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

2.4 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 (2.12)

$$C_{ij} = Hc \cdot u_{ij} \tag{2.12}$$

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 $y_j \pm \frac{\Delta y}{2}$. This means that a "unit" of the solution u will directly correspond to Hc random walkers.

2.5 Other possible coupling methods

2.6 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.

Different timescales

The time scales are coupled through the step length of the random walkers which is shown in equation (2.13).

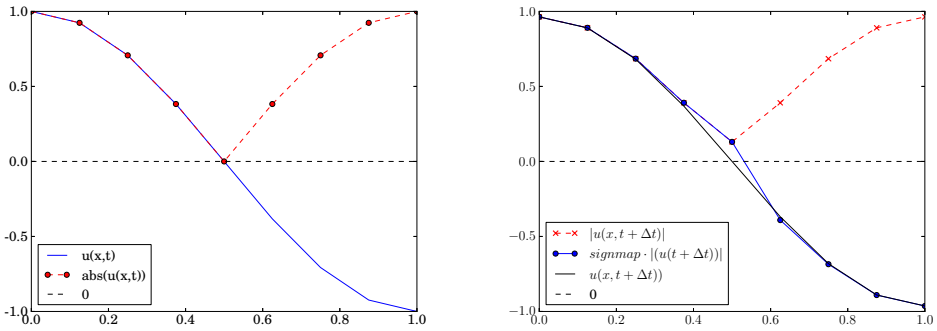
$$l = \sqrt{2Dd\frac{\Delta t}{\tau}} \tag{2.13}$$

Here τ is the number of time steps taken on the microscopic scale for each time step on the PDE scale, and Δt is the time step on the PDE scale.

Boundary conditions for the random walk

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 is shown in (b) as an increase in concentration in the middle mesh point in (a) 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

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.