Will Frierson

Research Methods Skill Module: Simulating Diffusion in a Cell With Finite Difference Methods

[**Getting Started**: This skill module requires you to use *Mathematica software*, which you can find on any of the computers in the *math computer lab* (7.122) or the *physics computer lab* (RLM 7.306). You also need to download two Mathematica files called "*Diffusion Module - FTCS.nb*." and "*Diffusion Module - Crank-Nicolson.nb*," which are on the Research Methods website along with this PDF.]

1 Purpose

In this skill module, you will model the diffusion of chemical X in a biological cell using two simulation techniques via the *finite difference method*. In practice, you might do this if running the experiment is too costly, or too difficult. In this example, we know that two sides of the cell are in contact with a resovoir of chemical X, and so they always have a constant chemical concentration. On the other hand, a third side is in contact with an organelle that consumes chemical X. Through our model, we wish to see if any chemical X remains after exposure to the organelle, and if so, to what extent.

This module is like a mini-lab in numerical analysis and simulation. The point of it is to expose you to these ideas, so don't get bogged down in the details! As you go through this assignment, you will answer some questions along the way.

2 Background on the Finite Difference Method

2.1 Discretization

The finite difference method is a simulation technique which *discretizes* the region on which you would examine the dynamics of a certain variable. In our case, we are looking at the dynamics of the chemical X concentration within the region of a biological cell as time varies. For simplicity, we assume the cell is two dimensional, and approximate it as a square.

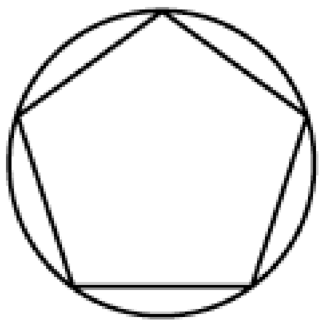
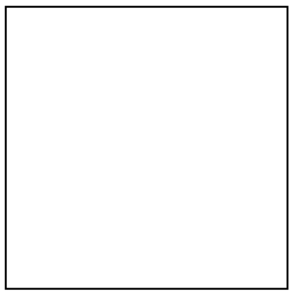


Figure 1: Discretization Example – A circle can be discretized by the five vertices of a pentagon placed at the circle's center.

We discretize our biological cell with tiny squares of size, Δx, and call this layout our *grid*. See Fig. 2. In addition, we discretize time, and define a fundamental time unit, Δt. Since we are trying to model diffusion, we must impose some *rule set* on our grid, which relays information about chemical X concentrations between time step n and $n+1$.

(a) Actual Cell


(b) Square Approximation of Cell

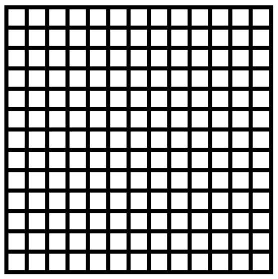

(c) Discretization of Square Approximation

Figure 2: Steps in Modeling

2.2 Diffusion Equation

We can find this rule set from the *diffusion equation*:

$$\frac{\partial u}{\partial t} = D\Delta u, \tag{1}$$

where D is the *diffusion constant*, Δ is the *Laplacian* defined by,

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \tag{2}$$

and u is the chemical X concentration. The diffusion equation is a well known and understood *partial differential equation* (PDE). The equations that describe light, Maxwell's Equations, are also a set of PDEs, as is the wave equation, which describes the behavior of a vibrating string. As this module will show you, PDEs describe a **huge** amount of mathematical and physical information in a few short equations. How do we go about extracting our discrete rules from the diffusion equation to apply to our grid? The answer lies with *Taylor polynomial* approximations of functions.

2.3 Taylor Polynomials and Taylor Series

2.3.1 Taylor Series

Before we begin talking about Taylor polynomials, we will talk about the *Taylor Series* representation of a function. In semi-rigorous mathematical terms, if a function $f(x)$ is *infinitely differentiable* in a *neighborhood* about a fixed point x_0 in the domain of f, then f can be represented by an infinite sum of polynomials, which are weighted in proportion to the derivatives of f. Although *neighborhood* has a very precise meaning, for our purposes we shall take it to mean "very close": if x is in a neighborhood of x_0, then x is "very close" to x_0. In symbols, if x is within a neighborhood of x_0, then,

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n, \tag{3}$$

where $f^{(n)}(x_0)$ means the n^{th} derivative of f evaluated at x_0. **In short, a Taylor series representation says that a function looks like a polynomial if you "zoom in" about a neighborhood of a point.**

To simplify our notation, we recall that our grid has a fundamental spacial step size, Δx. Let's rewrite Eq. 3 in terms of Δx. We fix x such that $x = x_0 + \Delta x$, so that $\Delta x = x - x_0$. Hence, Eq. 3 is now,

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(\Delta x)^n. \tag{4}$$

Lastly, since we assume x is in a neighborhood of x_0, then x is very close to x_0, and so Δx is very small. For example, Δx is on the order of $\frac{1}{20}$th of the length of the biological cell, which is 0.1-10 μm.

2.3.2 Taylor Polynomial

The fundamental technique for finite difference calculations comes from Taylor Polynomials. A Taylor Polynomial is a *truncation* of the Taylor Series representation of a function, *i.e.*, instead of letting n reach ∞, we have a finite sum in Eq. 4. Practically, this means that we approximate a function f by a finite sum of polynomials, weighted by its derivatives. We say a Taylor Polynomial has m-degrees, if the highest power of Δx is m, e.g.,

$$\begin{aligned} f(x) &\approx \sum_{n=0}^{m} \frac{f^{(n)}(x_0)}{n!}(\Delta x)^n \\ &\approx f(x_0) + \frac{\Delta x}{1!}\frac{df}{dx}(x_0) + \cdots + \frac{(\Delta x)^m}{m!}\frac{d^m f}{dx^m}(x_0). \end{aligned} \tag{5}$$

The higher the value of m, the more accurate our approximation of f. In this module, we will only consider Taylor Polynomials of degrees 1 and 2. An $m = 1$ Taylor polynomial is called a *linearization* of f, and $m = 2$ a *quadratic approximation* of f.

2.3.3 Truncation Error

Since a Taylor Polynomial is only a truncation of a Taylor series, if we add the left over terms from the truncation, we have our Taylor Series again. These left over terms are called the *truncation error*, R_n:

$$f(x) = \sum_{n=0}^{m} \frac{f^{(n)}(x_0)}{n!}(\Delta x)^n + R_n \tag{6}$$

Note that we now have equality in Eq. 6 because of R_n.

It turns out that R_n is very useful for our simulations. *Taylor's Theorem* tells us the following about R_n:

$$R_n = \frac{(\Delta x)^{m+1}}{(m+1)!} f^{(m+1)}(\xi_{m+1}), \tag{7}$$

where ξ_{m+1} is an *unknown number* such that, $x_0 < \xi_{m+1} < x$.

Example

Let's do a quick example. One trick that you will use frequently in this module is the following,

$$x = x_0 + (x - x_0) = x_0 + \Delta x. \tag{8}$$

So,

$$f(x) = f(x_0 + \Delta x) = f(x_0) + \Delta x \frac{df}{dx}(x_0) + \frac{(\Delta x)^2}{2!}\frac{d^2 f}{dx^2}(\xi_2), \tag{9}$$

where the last term is the truncation error, R_1. Since we are assuming Δx is small, we know ξ_2 is bounded in a small interval. **Hence, we can approximate R_1 with the already known x_0:**

$$R_1 \approx \frac{(\Delta x)^2}{2!}\frac{d^2 f}{dx^2}(x_0). \tag{10}$$

Lastly, we say that the truncation error is $O(\Delta x)^2$, which is read as "order delta-x squared," meaning,

$$|R_1| \leq C(\Delta x)^2, \tag{11}$$

where C is some constant.

What does all this mean? Well, this means that *our error in linearizing f is proportional to* $(\Delta x)^2$. **So, for small Δx, linearization has very little error, and so is very accurate!** As you will show in the following problem set, a linearization of f *does not* have the same truncation error as a non-centered difference approximation of a derivative.

2.3.4 Problems

Using the definition of a derivative,

$$\frac{df}{dx}(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}, \tag{12}$$

1) Derive the truncation error of the *forward difference* approximation of $\frac{df}{dx}$:

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}. \tag{13}$$

2) Derive the truncation error of the *backward difference* approximation of $\frac{df}{dx}$:

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0 - \Delta x) - f(x_0)}{-\Delta x}. \tag{14}$$

3) Derive the truncation error of the *centered differenc*e approximation of $\frac{df}{dx}$:

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}. \tag{15}$$

4) Derive the the *centered difference* approximation for the *second derivative*, $\frac{d^2 f}{dx^2}$:

$$\frac{d^2 f}{dx^2}(x_0) \approx \frac{f(x_0 + \Delta x) - 2f(x_0) + f(x_0 - \Delta x)}{(\Delta x)^2}. \tag{16}$$

5) Derive the truncation error for the *centered difference second derivative*, $\frac{d^2 f}{dx^2}$.

2.4 Finite Difference Approximations for Partial Derivatives

If you recall Eq. 1, we are dealing with partial derivatives. Lucky for us, we can use the results and your answers from the latter section. Instead of just x_0, we use a fixed vector, (x_0, y_0), and we apply, *e.g.*, an x-derivative only with the x-components :

$$\frac{\partial u}{\partial x}(x_0, y_0, t_0) \approx \frac{f(x_0 + \Delta x, y_0, t_0) - f(x_0, y_0, t_0)}{\Delta x}, \tag{17}$$

$$\frac{\partial u}{\partial y}(x_0, y_0, t_0) \approx \frac{f(x_0, y_0 + \Delta y, t_0) - f(x_0, y_0, t_0)}{\Delta y}, \tag{18}$$

etc.

Finally, we treat time derivatives the small way as spacial derivatives:

$$\frac{\partial u}{\partial t}(x_0, y_0, t_0) \approx \frac{f(x_0, y_0, t_0 + \Delta t) - f(x_0, y_0, t_0)}{\Delta t}, \tag{19}$$

etc.

2.4.1 Problems

6) Derive the difference approximation for the *Laplacian* from Eq. 2 using a 2D, spatial-temporal function $f = f(x, y, t)$.

3 Applying the Finite Difference Method for the Diffusion Equation

Since we are dealing with many derivatives, it seems like it would be easy to lose track of what is going on. So to prevent confusion, we use *grid notation.* In grid notation, we view a function evaulation on the grid, such that $x = \Delta x \cdot i$, $y = \Delta y \cdot j$, and $t = \Delta t \cdot n$. We shorten notation by removing all the Δ's:

$$u(x, y, t) = u^n_{i,j}. \tag{20}$$

Hence, the forward difference approximation for the time derivative, *e.g.*, is represented as,

$$\frac{\partial u}{\partial t} \approx \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t}, \tag{21}$$

and the other derivatives are written in the same manner.

3.0.2 Problems

Take a look again at the diffusion equation, Eq. 1. From your answers in previous problems and using grid notation,

7) Use a forward difference in time, and the Laplacian difference approximation to discretize the diffusion equation. Assume $\Delta x = \Delta y$, and solve for $u_{i,j}^{n+1}$.

We call this method a *Forward-Time Central-Space* (FTCS) scheme. It is also known as an *explicit method*, since we explicitly solve for the next time step, $u_{i,j}^{n+1}$.

From your result in problem (7), you should realize that the boundary of the grid cannot be calculated. So, we also have to take into account *boundary conditions.* Recall that we are simulating two sides of the cell as having a constant chemical X concentration, and one side of the cell exposed to an organelle which depletes chemical X. To achieve this, we set the desired constant concentrations at the two boundaries,

$$u \mid_{\text{boundary 1}} = 2 \text{ concentration units}, \tag{22}$$

and

$$u \mid_{\text{boundary 2}} = 0.5 \text{ concentration units}. \tag{23}$$

To properly model the organelle, we would have to include some sort of chemical reaction equation, known as a *reaction-diffusion equation.* These can be tricky to setup, and so I have made the third boundary negative, which effectively "eats" away at the chemical X concentration.

4 FTCS Diffusion Simulation

Open up the attached Mathematica file called "*Diffusion Module - FTCS.nb.*" There are three main code areas, which are separated by the blue brackets on the far right. Execute the first two cells, and you should see the following 3D plot as the output for the second cell:

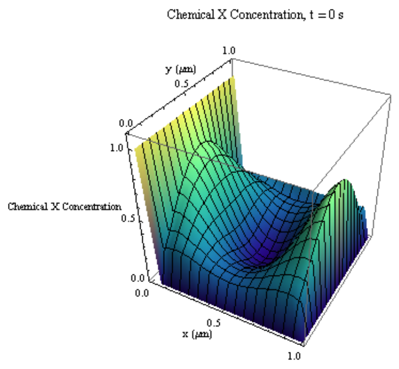


Figure 3: Distribution of Chemical X throughout Square Cell at Time = 0 s

This plot shows the distribution of chemical X in the square cell. Once you are ready, execute the third cell, and you should see the simulated diffusion and destruction of chemical X. To stop an execution, press *Alt* and *the period key.*

4.0.3 Problems

With the default values, Δt is set at 0.045 s. Note that the following questions require you to change the values of Δt.

8) Describe what you see when you run the simulation (*i.e.*, execute the third cell) with the following Δt values: 0.045 s, 0.05 s, and 0.055 s. Do the simulations appear to make physical sense, *i.e.*, do they seem reasonable?

One of these Δt values should have a very "interesting" simulation! A certain technique called *Von Neumann Analysis* shows that this interesting simulation comes from a special inequality called the *stability condition*:

$$\sigma = \frac{D\Delta t}{(\Delta x)^2} \leq \frac{1}{4}, \tag{24}$$

9) The FTCS simulation uses $D = \frac{1}{80}$ and $\Delta x = \frac{1}{20}$. Calculate the stability number, σ, for the Δt values in problem (8). How do these values relate to the stability condition?

5 Crank-Nicolson Diffusion Simulation

Open up the attached Mathematica file called "*Diffusion Module - Crank-Nicolson.nb.*" With this code, you will examine the output of the Crank-Nicolson method, which is described later. There are again three code areas. Execute the first two code areas, and wait for the 3D plot to show up as before. Note that this method takes about a minute to initialize. Now pick any time step, Δt, and execute the third code area. Pick a few more time steps, and rerun the simulation by executing the third area.

5.0.4 Problems

10) Describe the simulation for any time step you choose.

11) Take a screenshot of the final distribution of chemical X.

In problem (7), you derived the FTCS finite difference scheme for the diffusion equation. You also should have found that all spatial derivatives occurred on the n^{th} time step. Two researchers, Crank and Nicolson, discovered that if you average each spatial derivative with adjacent time steps, *e.g.*,

$$\frac{d^2 f}{dx^2} \approx \frac{1}{2}\left(\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{(\Delta x)^2} + \frac{u^{n+1}_{i+1,j} - 2u^{n+1}_{i,j} + u^{n+1}_{i-1,j}}{(\Delta x)^2}\right), \tag{25}$$

then the stabiltiy condition is always satisfied, *i.e.*, **the simulation is stable for all time steps**. In addition, the truncation error is $O(\Delta x)^2 + O(\Delta t)^2$, which is still very small. This method is called the Crank-Nicolson scheme. Also, this is known as an *implicit method*, because finding the next solution in time requires solving a linear system (*i.e.*, solving a matrix).

12) In a few paragraphs, summarize all that you have done in this module, and compare and contrast the FTCS and Crank-Nicolson methods.

6 References

1. Haberman, Richard. Applied Partial Differential Equations. 4th. Upper Saddle River: Pearson Education Inc., 2004. Print.