Derivation of CS403 Model Problem

February 19, 2002

Our model problem is based on the 1D-Advection-Reaction-Diffusion Equation:

$$\begin{aligned} \frac{\partial C}{\partial t} &= u\frac{\partial C}{\partial x} + \frac{\partial}{\partial x}\left(k\frac{\partial C}{\partial x}\right) + r(C,x,t) \\ \text{total change} = \text{advection} &+ \text{diffusion} + \text{reaction} \end{aligned}$$

In this equation, $C(x,t)$ represents the concentration of something at point x and time t. Three functions $u(x,t)$, $k(x,t)$, and $r(C,x,t)$ specify the velocity field, diffusivity field, and the change due to local processes (birth, death, chemistry, etc.), respectively. We want to compute the distribution of C at some point in the future, starting from an initial distribution $C(x,0)$.

To derive our numerical scheme we need to first carry the derivative opertor through the diffusion term:

$$\begin{aligned} \frac{\partial C}{\partial t} &= u\frac{\partial C}{\partial x} + \frac{\partial k}{\partial x}\frac{\partial C}{\partial x} + k\frac{\partial^2 C}{\partial x^2} + r(C,x,t) \\ \frac{\partial C}{\partial t} &= \left(u + \frac{\partial k}{\partial x}\right)\frac{\partial C}{\partial x} + k\frac{\partial^2 C}{\partial x^2} + r(C,x,t) \end{aligned}$$

Thus, if k is a function of x, then a diffusion gradient will produce an additional advective component.

To solve this problem on a computer, we place m evenly spaced points on our domain. If our domain runs from $x = 0$ to $x = L$, then the distance between the points is $dx = L/(m-1)$, and the points will be located at 0, dx, 2*dx, ...,L. We can divide the time span we want to simulate into intervals separated by dt in a similar manner. To (hopefully) simplify the notation, we will represent the time level as a superscript and the spatial position as a subscript, e.g. $C(x,t) = C_x^t$. Since we only have information about C at m discrete points, we will have to approximate the $\frac{\partial}{\partial x}$ terms by taking differences at adjacent points and dividing by dx. Doing this, and treating the time derivative in the same way yields:

$$\begin{aligned}
\frac{C_x^{t+dt} - C_x^t}{dt} &= \left(u_x + \frac{k_{x+dx} - k_{x-dx}}{2}\right)\frac{C_{x+dx}^t - C_{x-dx}^t}{2dx} \\
&\quad + k_x\frac{C_{x+dx}^{t+dt} - 2C_x^{t+dt} + C_{x-dx}^{t+dt}}{dx^2} + r(C_x^t, x, t) \\
&= \widetilde{u}\frac{C_{x+dx}^t - C_{x-dx}^t}{2dx} \\
&\quad + k_x\frac{C_{x+dx}^{t+dt} - 2C_x^{t+dt} + C_{x-dx}^{t+dt}}{dx^2} + r(C_x^t, x, t)
\end{aligned}$$

where $\widetilde{u}$ replaces the advection term. For a single $\frac{\partial}{\partial x}$ operation, we must choose whether to take the difference on the left or right. Unfortunately, the sign of u must agree: if $u > 0$ then we must look to the right (upwind). One way around this is to compute both the left and right differences and average them. This will work with any u and was used to compute both $\frac{\partial C}{\partial x}$ and $\frac{\partial k}{\partial x}$. Note that we can choose the time level at which we want to take the spatial derivatives. If we choose time level t, then our method is *explicit*, if we choose level $t+dt$ then our method is *implicit*. This distinction has important consequences on the numerical properties and computational efficiency of a PDE solver. Explicit methods can be computed quickly, but dt must be small. Implicit methods require more work (we must solve a system of linear equations), but we can use a larger time step. For several reasons beyond the scope of this course, I chose to take the derivative in the advection term at time t and the derivative in the diffusion term at time $t + dt$. Thus, the scheme we will use is *semi-explicit*.

Finishing our derivation requires some simple algebra. First, multiply throught by dt:

$$\begin{aligned}
C_x^{t+dt} &= C_x^t + \widetilde{u}\frac{dt}{2dx}(C_{x+dx}^t - C_{x-dx}^t) \\
&\quad + k_x(C_{x+dx}^{t+dt} - 2C_x^{t+dt} + C_{x-dx}^{t+dt})\frac{dt}{dx^2} + r(C_x^t, x, t)dt.
\end{aligned}$$

Then, we introduce two variables $\lambda = \widetilde{u}dt/dx$ and $\sigma = k_x dt/dx^2$:

$$\begin{aligned} C_x^{t+dt} &= C_x^t + 0.5 * \lambda(C_{x+dx}^t - C_{x-dx}^t) \\ &\quad +\sigma(C_{x+dx}^{t+dt} - 2C_x^{t+dt} + C_{x-dx}^{t+dt}) + r(C_x^t, x, t)dt. \end{aligned}$$

Note that both λ and σ may depend on x and/or time. We complete the derivation by moving all of the $t + dt$ terms to the left:

$$-\sigma C_{x+dx}^{t+dt} + (1+2\sigma)C_x^{t+dt} - \sigma C_{x-dx}^{t+dt} = C_x^t + 0.5*\lambda(C_{x+dx}^t - C_{x-dx}^t) + r(C_x^t, x, t)dt.$$

This scheme would work but it is rather simplistic. I would like to make one change involving the advection operator. Rather than our simple treatment (known as the "forward Euler method"), I would like to use a scheme known as "Lax-Wendroff." Our original method approximated the derivative by fitting a line between the intervals to the left and right. Lax-Wendroff adds a small correction that is equivalent to fitting a parabola to the data at x and at the points to the left and right. Replacing our original advective scheme with Lax-Wendroff yields:

$$-\sigma C_{x+dx}^{t+dt} + (1 + 2\sigma)C_x^{t+dt} - \sigma C_{x-dx}^{t+dt} = \tag{1}$$

$$C_x^t + 0.5 * \lambda(C_{x+dx}^t - C_{x-dx}^t) + 0.5 * \lambda^2(C_{x+dx}^t - 2 * C_x^t + C_{x-dx}^t) + r(C_x^t, x, t)dt.$$

Equation 1 describes how the value of C_x^{t+dt} depends on previous values of C. Unfortunately, because we treated the diffusion term implicitly, C_x^{t+dt} depends on unkown information, namely, the neighboring values at time $t+dt$. We have an identical set of equations for the other $m-1$ points in our domain. If you count up the number of unknowns (all of the C^{t+dt} terms), you'll find that we have m equations (one for each point in the domain) but $m+2$ unkowns. The two extra unknowns, C_{0-dx}^{t+dt} and C_{L+dx}^{t+dt}, occur next to the first and last points. In order to close the system, we need to eliminate these unkowns by providing *boundary conditions*. There are a number of ways we could do this. In my opinion, the easiest and most interesting boundary conditions are called "periodic boundary conditions." Periodic boundary conditions assume the information flowing out of the left side of the domain returns on the right, and vice-versa (rather than modeling on a fixed segment, we're modeling on a circle). Using these conditions, our two extra unkowns, C_{0-dx}^{t+dt} and C_{L+dx}^{t+dt}, become C_L^{t+dt} and C_0^{t+dt}, respectively.

Now that we have m equations and m unkowns we need a way to solve for the unkowns. Because the equations are linear, we can pose our system of equations as a matrix problem, and use a variety of algorithms from numerical linear algebra to solve our system. The matrix problem looks like

$$Ac = b$$

where c is a length-m vector containing each of the C_x^{t+dt} terms, b is a length-m vector containing the value of advective and reactive equations at each point (the right-hand side in 1), and A is an m-by-m matrix. Each row of A represents a linear equation, and the jth row looks like

$$a_{j,1}C_0^{t+dt} + a_{j,2}C_{dx}^{t+dt} + a_{j,3}C_{2*dx}^{t+dt} + ... + a_{j,m-1}C_{(m-2)*dx}^{t+dt} + a_{j,m}C_L^{t+dt} = f(C(t))$$

Equation 1 says that C_x^{t+dt} depends only on neigboring points. Thus, only $a_{j,j-1}$, $a_{j,j}$, and $a_{j,j+1}$ will be non-zero (assuming j is not 0 or m). This implies that most of A is *tridiagonal*: only values lying above, below, and on the main diagonal are non-zero. However, the periodic BC's ensure the A is not perfectly tri-diagonal. Because C_0^{t+dt} depends on C_L^{t+dt}, a_{1m} is non zero. Similarly, C_L^{t+dt} depends on C_0^{t+dt}, so a_{m1} is non zero. We now have the structure of A, but what are its values? From equation 1, we get

$$\begin{aligned} A(1,:) &= \begin{bmatrix} (1+2\sigma) & -\sigma & ... & -\sigma \end{bmatrix} \\ A(j,:) &= \begin{bmatrix} ... & -\sigma & (1+2\sigma) & -\sigma & ... \end{bmatrix} \\ A(m,:) &= \begin{bmatrix} -\sigma & ... & -\sigma & (1+2\sigma) \end{bmatrix} \end{aligned}$$

We can now describe, in vague terms, how to solve our PDE. Assume that we are given an array containing the values of C at the grid points at time 0 and that u and k are also described at these points. Using this data and dt and dx, we can construct A and the right-hand side vector b. We then select from several algorithms for solving linear systems to get the values of C at time dt. We use update b using the new values of C (and A if k changes with time) and repeat. We iterate forward in this manner until $t >$ some specified time.