CME 104 numerical methods review

Enze Chen (enze)

June 13, 2018

1 Overview

Over the last three weeks we've looked at numerical methods of solving partial differential equations, and more generally, systems of equations. Now I will attempt to highlight some of the key points that we discussed during lectures 13-17.

2 Finite difference

The *finite difference* (FD) method is one of the simplest and still most successful numerical methods used to solve PDEs. As the name implies, instead of solving a PDE exactly in continuous time and space, we will solve an approximate analog of the problem by discretizing our domain into a finite number of grid points and use difference formulas to approximate the value of derivatives at each grid point.

2.1 Taylor series

How do we obtain these difference formula approximations? **Taylor series**! Recall that the Taylor series of $f(x+h)$ about x for *small* h (assuming f is continuously differentiable) is given by

$$\boxed{f(x+h) = f(x) + f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 + \frac{f'''(x)}{6}\cdot h^3 + O(h^4)}$$

where the big-O notation swallows *higher order terms* and represents the *accuracy* of the Taylor series (we'll call the above approximation "fourth-order accurate").[1]

Note: When we're developing and using numerical methods, we should understand how large this error is, but we usually omit it. However, when we're doing any sort of analysis, we must carry the big-O around with us.

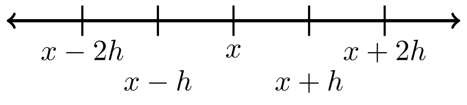

Figure 1: DRAW THIS ON THE BOARD.

We break up the domain into (typically uniformly sized) sections of length h. We then have **forward difference** schemes, that use the current point and points in front; **backward difference** schemes, that use the current point and points behind; and **central difference** schemes that use both points. In general, if we want higher order accuracy and/or higher order derivatives, we need to incorporate more neighboring points and the Taylor series about those points. Let's see this in action with a sample problem.

[1] In math, we say a Taylor series is of order k when the highest derivative is the k-th derivative; that's not in conflict with what we're saying, but just FYI.

Problem 1: Find a second-order accurate, backward difference scheme for the first derivative.

Solution: A single Taylor series for the first derivative is first-order accurate, so we write out the Taylor series for $f(x-h)$ and $f(x-2h)$ and manipulate both. Note that because we want second-order accuracy, and because eventually we have to divide by h in order to isolate $f'(x)$, we need a third-order accurate Taylor series expansion.

$$\begin{aligned}
f(x-h) &= f(x) - f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 + O(h^3) \\
f(x-2h) &= f(x) - 2f'(x)\cdot h + 2f''(x)\cdot h^2 + O(h^3) \\
4f(x-h) &= 4f(x) - 4f'(x)\cdot h + 2f''(x)\cdot h^2 + O(h^3) \\
f(x-2h) - 4f(x-h) &= -3f(x) + 2f'(x)\cdot h + O(h^3) \\
f'(x) &= \frac{f(x-2h) - 4f(x-h) + 3f(x)}{2h} + O(h^2) \qquad \square
\end{aligned}$$

This approximation for the first derivative is very useful when dealing with *Neumann boundary conditions*. Now let's use FD to approximate a PDE:

Problem 2: Using second-order central difference schemes and a uniform square grid of step size h, approximate the 2D Poisson equation given by $\nabla^2 u = f(x,y)$.

Solution: We have from Taylor series:

$$\begin{aligned}
f(x+h) &= f(x) + f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 + \frac{f'''(x)}{6}\cdot h^3 + O(h^4) \\
f(x-h) &= f(x) - f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 - \frac{f'''(x)}{6}\cdot h^3 + O(h^4) \\
f(x+h) + f(x-h) &= 2f(x) + f''(x)\cdot h^2 + O(h^4) \qquad \text{(add the two equations)} \\
f''(x) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)
\end{aligned}$$

This allows us to write:

$$\begin{aligned}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= f(x,y) \\
\frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} + \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} &= f(x,y) \\
\frac{1}{h^2}\left[u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)\right] &= f(x,y) \qquad \square
\end{aligned}$$

The above FD scheme applies to every interior grid point of our domain (notice that in applying the FD approximation we drop the big-O). At this step we can setup a matrix of coefficients, a vector of non-homogeneous terms, and another vector of solutions that we solve for. If we're careful about how we order the nodes and arrange the coefficient matrix, then it's possible to write the above equation in matrix-vector form, although this is outside the scope of this class. Instead, we will turn to iterative schemes to solve such problems.

3 Iterative schemes

To make steady-state FD approximations more accurate, we hope to decrease the step size h. Doing so, however, will lead to huge matrices that are very sparse, which makes normal methods for solving such systems inefficient.

With iterative schemes, instead of solving $A\vec{x} = \vec{b}$, we introduce a simpler matrix M and iteratively find solutions using the recurrence relation

$$\boxed{\vec{x}_{k+1} = M^{-1}\left[(M-A)\vec{x}_k + \vec{b}\right]} = B\vec{x}_k + M^{-1}\vec{b}$$

What are our choices for M? The popular ones are:

- $M = D$, where D are the diagonal elements of A (not the eigenvalues in diagonalized D matrix!). This is the **Jacobi** method.
- $M = L + D$, which is the lower triangular part + the diagonal (L is *not* the LU factorization!). This is the **Gauss-Seidel** method.
- $M = L + \dfrac{D}{\omega}$, where ω is a tuning parameter to minimize $\lambda_{\max}(B)$. This is the **SOR** method.

3.1 Stability and convergence

We looked at two properties of iterative schemes, stability and convergence. Stability says that if we run our iterative scheme, then our solution will converge to (something similar to) the true solution, while (rate of) convergence asks, knowing that we'll get the true solution, how fast will we get there?

We found that the error at the k-th iteration step is given by $\vec{e}_k = B^k\vec{e}_0$, where $\vec{e}_0$ is the initial error, and that $|\vec{e}_k| \sim \lambda_{\max}^k$ where $\lambda_{\max} \overset{\text{def}}{=} \max_i |\lambda_i(B)|$ (not the eigenvalues of A or M!). Since we hope that our error decreases as we perform more iterations, we see that it must be the case

$$\boxed{\lambda_{\max} < 1}$$

and this is our *stability* criteria.

The *convergence*, as previous mentioned, scales as $\lambda_{\max}$. Note, however, that this is an **exponential** scaling, not a multiplicative scaling. For example:

Problem 3: Scheme A has $\lambda_{\max} = \dfrac{1}{3}$ and scheme B has $\lambda_{\max} = \dfrac{1}{9}$. How many times faster is B's rate of convergence than A's?

Solution: Let's do a couple of examples. For scheme A:

$$\begin{aligned} e_1 &\sim \left(\frac{1}{3}\right)^1 = \frac{1}{3} \\ e_2 &\sim \left(\frac{1}{3}\right)^2 = \frac{1}{9} \\ e_3 &\sim \left(\frac{1}{3}\right)^3 = \frac{1}{27} \\ e_4 &\sim \left(\frac{1}{3}\right)^4 = \frac{1}{81} \\ e_5 &\sim \left(\frac{1}{3}\right)^5 = \frac{1}{243} \\ e_6 &\sim \left(\frac{1}{3}\right)^6 = \frac{1}{729} \end{aligned}$$

while for scheme B:

$$\begin{aligned} e_1 &\sim \left(\frac{1}{9}\right)^1 = \frac{1}{9} \\ e_2 &\sim \left(\frac{1}{9}\right)^2 = \frac{1}{81} \\ e_3 &\sim \left(\frac{1}{9}\right)^3 = \frac{1}{729} \end{aligned}$$

thus we see that scheme B is *twice* as fast as scheme A, which makes sense as $\left(\frac{1}{3}\right)^2 = \frac{1}{9}$. In other words, every two iterations of scheme A will reduce my error by as much as one iteration of scheme B. $\square$

3.2 Point-wise iterative updates

In some cases, it might be easier (implementation-wise and readability-wise) to perform iterations on a point-by-point basis as opposed to solving a huge matrix system (e.g. Problem 2 here). If we take our final expression we obtained in Problem 2,

$$\frac{1}{h^2}\left[u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)\right] = f(x,y)$$

we can isolate $u(x_i, y_j)$ to obtain

$$\begin{aligned} u(x_i,y_j) &= \frac{1}{4}\left[u(x_i+h,y_j) + u(x_i-h,y_j) + u(x_i,y_j+h) + u(x_i,y_j-h)\right] - \frac{f(x_i,y_j)\cdot h^2}{4} \\ u_{i,j}^{k+1} &= \frac{1}{4}\left[u_{i+1,j}^{?} + u_{i-1,j}^{?} + u_{i,j+1}^{?} + u_{i,j-1}^{?}\right] - \frac{f_{ij}\cdot h^2}{4} \end{aligned}$$

For Jacobi, which uses only old values to update the new values, everything on the RHS is from the k-th iteration, which means implementation wise you have to store a copy of the old matrix, update everything in the new one, and then throw the old one away (make the "new" matrix your "old" matrix). For Gauss-Seidel, assuming a linear sweep through each row from top to bottom, the "-1" terms will have the updated values from the $(k+1)$th iteration. Note that you do not have to explicitly mark these in MATLAB. Since you've already updated your matrix in place, you can just directly call the values from your same matrix [**illustrate on board**]. The old matrix only has to be stored for the purpose of calculating iteration error.

For the final, you'll likely be asked to do some calculations by hand, similar to what Prof. Khayms did in lecture. Let's take a look at Review Problem 9 for example:

Problem 4: A square $3\,\text{m} \times 3\,\text{m}$ plate has three boundaries kept at $T = 0$ and at $x = 3$ heat is injected into the plate such that $\left.\frac{\mathrm{d}T}{\mathrm{d}x}\right|_{x=3} = 10\,\text{K/m}$. Using 16 equally spaced grid points and choosing:

$$T^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

perform the first three steps of the Jacobi iteration to solve for the steady-state temperature distribution.

Solution: The BCs tell us that the top, left, and bottom boundaries are always kept at $T = 0$. For the right, we want to use the second-order, backward difference scheme for the Neumann BC, which we found to be

$$\left.\frac{\mathrm{d}T}{\mathrm{d}x}\right|_{x=3} = 10 = \frac{T_{1,j} - 4T_{2,j} + 3T_{3,j}}{2h}, \text{ for } j = 2, 3$$

and we can solve for $T_{3,j}$ (since we want to use this to figure out the boundary values) to obtain

$$T_{3,j}^{k+1} = \frac{1}{3}\left[20h - T_{1,j}^{k} + 4T_{2,j}^{k}\right], \text{ for } j = 2, 3$$

The interior points follow the update rule

$$T_{i,j}^{k+1} = \frac{1}{4}\left[T_{i+1,j}^{k} + T_{i-1,j}^{k} + T_{i,j+1}^{k} + T_{i,j-1}^{k}\right], \text{ for } i, j = 2, 3$$

Since this is the Jacobi iteration scheme, we will only use the values from the previous iteration to perform updates, and consequently, it doesn't really matter what order we perform updates in. Since the changes are happening at the boundary, we might as well start from the right side ($x = 3$).

$$T^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20/3 \\ 0 & 0 & 0 & 20/3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 5/3 & 20/3 \\ 0 & 0 & 5/3 & 20/3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T^{(4)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 5/12 & 25/12 & 80/9 \\ 0 & 5/12 & 25/12 & 80/9 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

□

4 von Neumann stability analysis

The most recent topic we covered was von Neumann stability analysis, which is useful for any time-marching scheme, since we now have both Δt and Δx and it's likely that one depends on the other (we can't arbitrarily choose step sizes). Instead of looking at the eigenvalues of the B matrix, we instead looked at the behavior of the discretization scheme directly by plugging in a candidate solution of the form $T_j^n = G^n e^{ikx_j}$. We call G the *amplification factor* and we impose the constraint $|G| \leq 1$ so that our solution remains bounded.

Now the game we play is we plug this candidate solution into the discretization scheme, taking care to replace j with x_j, $j+1$ with $x_j + \Delta x$, $j-1$ with $x_j - \Delta x$, etc (don't mix up indices with actual positions). Then we solve for G, take the magnitude, and see what the restrictions on Δt are in terms of Δx and any other constants. This will determine how fast we're able to propagate our solution while still guaranteeing stability. Let's look at an example problem:

Problem 5: Using a second-order central difference scheme in space and a first-order *backward* difference scheme in time for the heat equation $\dfrac{\partial u}{\partial t} = \lambda^2 \dfrac{\partial^2 u}{\partial x^2}$, perform von Neumann stability analysis to determine the criterion for the maximum allowable time step to ensure the time-marching scheme remains stable.

Solution: We discretize our solution at the point x_j and simplify to obtain

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \lambda^2 \frac{\partial^2 u}{\partial x^2} \\
\frac{u_j^n - u_j^{n-1}}{\Delta t} &= \lambda^2 \cdot \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \\
G^n e^{ikx_j} - G^{n-1} e^{ikx_j} &= \frac{\lambda^2 \Delta t}{\Delta x^2} \left(G^n e^{ik(x_j - \Delta x)} - 2G^n e^{ikx_j} + G^n e^{ik(x_j + \Delta x)} \right) \\
1 - \frac{1}{G} &= \frac{\lambda^2 \Delta t}{\Delta x^2} \left(e^{-ik\Delta x} + e^{ik\Delta x} - 2 \right) \\
1 - \frac{1}{G} &= \frac{2\lambda^2 \Delta t}{\Delta x^2} \left(\cos(k\Delta x) - 1 \right) \\
G &= \left(1 - \frac{2\lambda^2 \Delta t}{\Delta x^2} (\cos(k\Delta x) - 1) \right)^{-1} \\
|G| &= \left| \underbrace{\left(1 - \frac{2\lambda^2 \Delta t}{\Delta x^2} \underbrace{(\cos(k\Delta x) - 1)}_{[-2,0]} \right)}_{\left[1, 1 + \frac{4\lambda^2 \Delta t}{\Delta x^2}\right]} \right|^{-1} \leq 1
\end{aligned}$$

from which we see that using a backward difference scheme in time results in unconditional stability for any choice of Δt and Δx.

Note: Forward difference is often called an *explicit* scheme and using backward difference is called *implicit*. What are the advantages of using an implicit scheme? Disadvantages? Does unconditionally stable mean that we can literally use any time-step and be fine?