Lecture 6: Finite differences and PDES

Partial Differential Equations (PDEs)

PDE: partial differential equation: ≥ 2 independent variables.

History:

1600s: calculus.

1700s, 1800s: PDEs all over physics, especially linear theory.

1900s: the nonlinear explosion and spread into physiology, biology, electrical engr., finance, and just about everywhere else.

The PDE Coffee Table Book

Unfortunately this doesn't physically exist (yet!). Fortunately, we can read many of the pages online: people.maths.ox.ac.uk/trefethen/pdectb.html

Notation:

- *Partial derivatives* $u_t = \frac{\partial u}{\partial t}$, $u_{xx} = \frac{\partial^2 u}{\partial x^2}$,
- *Gradient* (a vector) $\operatorname{grad} u = \nabla u = (u_x, u_y, u_z)$,
- *Laplacian* (a scalar) $\operatorname{lap} u = \nabla^2 u = \triangle u = \Delta u = u_{xx} + u_{yy} + u_{zz}$,
- *Divergence* (vector input, scalar output)

$\operatorname{div}(u, v, w)^T = \nabla \cdot (u, v, w)^T = u_x + v_y + w_z$.

(and combinations: e.g., $\Delta u = \nabla \cdot \nabla u$).

Examples

Laplace eq: $\Delta u = 0$ (elliptic)

Poisson eq: $\Delta u = f(x, y, z)$ (elliptic)

Heat or diffusion eq: $u_t = \Delta u$ (parabolic)

Wave eq: $u_{tt} = \Delta u$ (hyperbolic)

Burgers eq: $u_t = (u^2)_x + \epsilon u_{xx}$

KdV eq: $u_t = (u^2)_x + u_{xxx}$

Finite differences in space and time

The simplest approach to numerical soln of PDE is finite difference discretization in both space and time (if it's time-dependent).

Consider the heat or diffusion equation in 1D:

$$u_t = u_{xx}, \quad -1 < x < 1,$$

with *initial conditions*: $u(x, 0) = u_0(x)$,

and *boundary conditions*: $u(-1, t) = u(1, t) = 0$.

Set up a regular grid with k = time step , h = spatial step

$$v_j^n \approx u(x, t).$$

A simple finite difference formula:

$$\frac{v_j^{n+1} - v_j^n}{k} = \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2}.$$

The Method-of-lines Discretize in space only, to get a system of ODEs. This reduces the problem to one we've already looked at: numerical solution of ODEs.

For the heat/diffusion equation, we get:

$$\frac{d}{dt} v_j = \frac{v_{j-1} - 2v_j + v_{j+1}}{h^2},$$

where $v_j(t)$ is a continuous function in time and $v_j(t) \approx u(x_j, t)$.

We solve the (coupled) ODEs along lines in time.

```
t
^
|
|  ^    ^    ^          ^
|  |    |    |          |
|  |    |    |          |
|  |    |    |          |
+--+----+----+----------+----> x
  x_0  x_1  x_2     ... x_N
```

Consider v^n as an N-vector. We can get from v^n to v^{n+1} by using a matrix to approximate u_{xx}.

Linear algebra: MOL approach

```
     |       |           |-2    1                 |  |       |
     | v (t) |           |                        |  | v (t) |
     |  1    |           | 1   -2    1            |  |  1    |
     |       |           |                        |  |       |
d    |  .    |           |     .    .   .         |  |  .    |
--   |  .    |  =  1/h^2 |      .    .   .        |  |  .    |
dt   |  .    |           |       .    .   .       |  |  .    |
     |       |           |                        |  |       |
     |       |           |          1   -2    1   |  |       |
     | v (t) |           |                        |  | v (t) |
     |  N    |           |               1   -2   |  |  N    |
```

Or using $v(t)$ as N-vector:

```
                      |-2    1              |
                      |                     |
                      | 1   -2    1         |
                      |                     |
    d                 |    .    .   .       |
    -- v(t)  =  1/h^2 |     .    .   .      | v(t)
    dt                |      .    .   .     |
                      |                     |
                      |         1   -2    1 |
                      |                     |
                      |              1  -2  |
```

Let's call this matrix L (with the $\frac{1}{h^2}$ factor). If we then discretize time with forward Euler we get:

```
                      |-2    1              |  |  n |
                      |                     |  | v  |
                      | 1   -2    1         |  |  1 |
                      |                     |  |    |
 n+1    n             |    .    .   .       |  | .  |
v    = v  + k * 1/h^2 |     .    .   .      |  | .  |
                      |      .    .   .     |  | .  |
                      |                     |  |    |
                      |         1   -2    1 |  |  n |
                      |                     |  | v  |
                      |              1  -2  |  |  N |
```

Where v^{n+1} represents the discrete vector of solution at time t_{n+1}:

$$v^{n+1} \approx v(t_{n+1}).$$

We can also write the above as $v^{n+1} = Av^n$ with $A := I + kL$.

Can also convince ourselves that the above fully discrete matrix system is the same as the earlier fully discrete equations:

$$\frac{v_j^{n+1} - v_j^n}{k} = \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2}.$$

(Note however that not all finite difference schemes are methods-of-lines).

Implementation of these matrices and schemes

[m10_heat_mol.m]

Discuss constructing the matrix L using sparse matrices. Sparse matrices save storage space here.

Matlab: discuss $k * L * u$ versus the better $k * (L * u)$.

You can also implement using for-loops. [m11_heat_loops.m] This can be much slower in Matlab (and other high-level languages), although "JIT-compilers" often speed it up.

Here its the choice between a for/do loop, and sparse matrices. Personally, I like the abstraction of constructing a discrete operator to approximate my derivatives

Boundary conditions We have $v_0^n = v_{N+1}^n = 0$. Note choice of x grid in code. And how to change to periodic conditions.

Brownian motion Random walk / Brownian motion, the physical implementation of the heat/diffusion equation. [m12_random_walk.m]

We move back-and-forth between discrete and continuous!

```
DISCRETE                      CONTINUOUS
--------                      ----------
bouncing molecules            continuum models of physics
finite-difference approx      PDE
random walk                   Brownian motion
dots on computer screen       our perception of movie
floating-point arithmetic     real arithmetic
```

Note that these are all happening on different scales, which need not be tied to one another. Notably the PDE is a continuum model of what molecules do. We then discretize this to solve a discrete problem on a grid on the computer, but this grid has little (or nothing) to do with the molecular scaler.

Nonlinear PDE

The heat equation was linear, dating to Fourier in 1807, and it can be more or less solved analytically.

Let's consider now nonlinear PDE, intractable analytically but easy to solve numerically.

Fisher-KPP Equation Independent 1937 discoveries for biological applications (spread of species): Fisher; Kolmogorov, Petrovsky, and Piscounov

$$u_t = \epsilon u_{xx} + u - u^2.$$

Solutions: traveling waves. Similar more complicated behaviour: Hodgkin-Huxley eqs for neural impulses (1952).

Explicit finite difference model is similar to heat equation. Note we have "nonhomogeneous" BCs $u(0) = 1$, $u(20) = 0$; the former is implemented by using an extra vector "BC". [m13_fisher_kpp.m]

```
                  +--                                                       --+
                  |         |-2   1                |  |  n |     |1/h^2|          |
                  |         |                      |  | v  |     |     |          |
                  |         | 1  -2   1            |  |  1 |     |  0  |          |
                  |         |                      |  |    |     |     |          |
  n+1    n        |         |   .   .   .          |  |    |     |  .  |        2 |
 v     = v  + k * | eps/h^2 |    .   .   .         |  |    |  +  |  .  |  +  (v -v ) |
                  |         |     .   .   .        |  |    |     |  .  |      j  j  |
                  |         |                      |  |    |     |     |          |
                  |         |       1   -2   1     |  |  n |     |     |          |
                  |         |                      |  | v  |     |     |          |
                  |         |            1   -2    |  |  N |     |  0  |          |
                  +--                                                       --+
                                                                BC
```

Stability in finite difference calculations

A fourth-order problem

$$u_t = -u_{xxxx}.$$

How to discretize? Think $(u_{xx})_{xx}\ldots$, this leads, eventually, to

$$v_j^{n+1} = v_j^n - \frac{k}{h^4}\left(v_{j-2}^n - 4v_{j-1}^n + 6v_j^n - 4v_{j+1}^n + v_{j+2}^n\right)$$
$$\frac{d}{dt}v_j = \frac{1}{h^4}\left(v_{j-2} - 4v_{j-1} + 6v_j - 4v_{j+1} + v_{j+2}\right)$$
$$\frac{d}{dt}v = -Hv$$

[m14_biharmonic.m] Note ridiculously small time steps required. Let's try to see why (a stability issue) and what we can do about it (implicit A-stable ODE methods).

von Neumann analysis *[For your interest, will not be tested]* One approach is *von Neumann Analysis* of the finite difference formula, also known as *discrete Fourier analysis*, invented in the late 1940s.

Suppose we have periodic boundary conditions and that at step n we have a (complex) sine wave

$$v_j^n = \exp(i\xi x_j) = \exp(i\xi jh),$$

for some wave number ξ. Higher ξ is more oscillatory. We will analysis whether this wave grows in amplitude or decays (for each ξ). For stability, we want all waves to decay.

For the biharmonic diffusion equation, we substitute this wave into the finite difference scheme above, and factor out $\exp(i\xi h)$ to get

$$v_j^{n+1} = g(\xi)v_j^n,$$

with the *amplification factor*

$$g(\xi) = 1 - \frac{k}{h^4}\left(e^{-i2\xi h} - 4e^{-i\xi h} + 6 - 4e^{i\xi h} + e^{i2\xi h}\right).$$

This can be simplified to:

$$g(\xi) = 1 - \frac{16k}{h^4}\sin(\xi\, h/2).$$

As ξ ranges over various values sin is bounded by 1 so we have $1 - 16k/h^4 \le g(\xi) \le 1$.

A mode will blow up if $|g(\xi)| > 1$. Thus for stability we want to ensure $|g(\xi)| \le 1$ for all ξ , i.e.,

$$1 - 16k/h^4 \ge -1, \qquad \text{or} \qquad \boxed{k \le h^4/8.}$$

For $h = 0.025$, as in the Matlab code, this gives $\boxed{k \le 4.883e-08}$. This matches our experiment convincingly, but confirms that this finite difference formula is not really practical.

Method-of-lines

As an alternative to von Neumann analysis, we follow the linear stability analysis for the ODE methods. The spatial discretization gives us (numerically anyway) the eigenvalues of the *semidiscrete* system. Need these eigenvalues to lie inside the absolute stability region of the ODE method.

Note: this involves the eigenvalues of the semidiscrete system, not the original right-hand-side of the PDE.

Demo: in Matlab, run m14_biharmonic, then use eigs to compute 'largest magnitude' eigenvalues of the *discretized* biharmonic operator: need k times these less than 2 for forward Euler stability. Note this gives almost the same restriction as observed in practice (and calulated with von Neumann analysis)

Implicit methods for PDEs

Apply implicit (A stable) methods to the semidiscrete method-of-lines form. For example, let's look at the heat equation $u_t = \Delta u$. If we apply backward Euler:

$$\frac{v^{n+1} - v^n}{k} = Lv^{n+1},$$

(c.f., forward Euler which has Lv^n on the RHS.)

Recall backward Euler is A-stable: stable for all eigenvalues in the left-half plane.

Similarly, the biharmonic hyperdiffusion equation with a matrix H:

$$\frac{v^{n+1} - v^n}{k} = -Hv^{n+1}.$$

[m15_be_heat.m][m16_be_biharm.m]

Example: Kuramoto-Sivashinsky equation

$$u_t = -u_{xx} - u_{xxxx} - (u^2/2)_x.$$

Ignore nonlinearity and think about what each "diffusion" term does.

- Long waves grow because of $-u_{xx}$;
- short waves decay because of $-u_{xxxx}$;
- the nonlinear term transfers energy from long to short.

[m17_kuramoto_sivashinsky.m] Note stability and chaotic behaviour of solution.

We treat the linear stiff terms implicitly and the nonlinear (but hopefully nonstiff) terms explicitly—an "IMEX" (implicit/explicit) discretization.

Order of accuracy in PDE finite difference calculations

We looked at this for ODEs before. We apply similar ideas for the PDE.

[m18_be_accuracy.m] Note: we have errors from both h and k. We plot error against k and h (in Figures 2 and 3) but these may not (and here do not) expose the truncation of the spatial and temporal schemes separately.

Local Truncation Error

$$v_j^{n+1} - u(x_j, t_{n+1})$$

where u is smooth solution and v^{n+1} is computed from exact values for v_j^n, v_j^{n-1} etc.

- Replace v_j^{n-1} by Taylor series for equiv. value of u.
- Cancel terms to find local truncation error.
- Divide by one power of k to find global accuracy.

We often (try to) separate h and k dependences and talk about, e.g., $O(k) + O(h^2)$ accuracy. First-order accurate in time and second-order accurate in space.

Example: backward Euler

(all functions evaluated at (x_j, t_n) unless otherwise stated).

Taylor:

$$u(x, t_{n+1}) = u + ku_t + k^2/2u_{tt} + k^3/6u_{ttt} + \ldots$$

And 2D Taylor gives

$$u(x_{j+1}, t^{n+1}) = u \pm hu_x + ku_t + \frac{h^2}{2}u_{xx} + \frac{k^2}{2}u_{tt} \pm hku_{xt} \pm \frac{h^3}{6}u_{xxx} + \frac{k^3}{6}u_{ttt} \pm \frac{k^2h}{2}u_{ttx} + \frac{kh^2}{2}u_{txx} + \frac{h^4}{24}u_{xxxx} + \ldots$$

Now compute the LHS and RHS of the backward Euler method. Then we have:

$$LHS - RHS = ku_t - ku_{xx} + k^2/2u_{tt} + kh^2/12u_{xxxx} + \ldots$$

But $u_t = u_{xx}$ so we have the local truncation error is

$$O(k^2) + O(kh^2)$$

And thus the global error is

$$O(k) + O(h^2)$$

(advanced: there is some subtly about h and k: we assumed $k = O(h)$ in this derivation. . .)

Forward Euler

Forward Euler is also $O(k) + O(h^2)$.

[m19_fe_accuracy.m] These results are less clear. Vary the $k = 0.25h^2$ parameter. . .

Why does this seem to give 2nd-order (in h)? Discuss w.r.t. stability. $k = O(h^2)$ Maybe you only want first-order accuracy, is so, this extra work is wasteful.

(Yet another “definition” of stiffness: if your choice of timestep k is motivated by stability rather than accuracy, you are probably dealing with a stiff problem.)

Higher-order in time

Even if we want second-order perhaps there are better ways, use a better ODE solve: trapezoidal/trapezium rule in time + second order in space. When used on heat equation, this is called "Crank–Nicolson":

$$v^{n+1} = v^n + \frac{k}{2} L v^{n+1} + \frac{k}{2} L v^n.$$

or you can write this as

$$B v^{n+1} = A v^n$$

[m20_cn_accuracy.m] And note we observe 2nd-order clearly in space and time with $k = O(h)$.

Caution Sometimes hard to tell from numerical convergence study which terms are dominating. Can also design tests to isolate the error components in h and k.