# Lecture 8, Sci. Comp. for DPhil Students II

Nick Trefethen, Tuesday 12.02.19

**Last lecture**

- V.5 Order of accuracy
- V.6 Reaction-diffusion equations and other stiff PDEs

**Today**

- V.7 Finite differencing in general grids
- V.8 Multiple space dimensions

**Handouts**

- Finite difference coefficients from Fornberg
- `m47_BackwardEuler2D.m` - backward Euler for heat equation in 2D
- `m48_CrankNicolson2D.m` - Crank-Nicolson for heat equation in 2D
- `m49_leapfrog2D.m` - leap frog for wave equation in 2D

Assignment 3: due next Tuesday

Reminder: to make the most of this course,

- study the lecture notes!
- download and run the programs!

All this material is pretty brief, but it covers a great deal. You won't find so much covered so concretely and compactly elsewhere.

## V.7 Finite differencing in general grids

How do we derive finite difference approximations to the $k$ th derivative on an arbitrary grid?

Principle:

(1) At each $x_j$, decide which data $v_{j-r}, \ldots, v_{j+s}$ to use.

(2) Interpolate these data by polynomial $p$ of degree $r+s$.

(3) Finite difference approx: $p^{(k)}(x_j)$

We don't do (1)–(3) explicitly; rather, they define differencing coefficients implicitly.

Example: one-sided 1st derivative at $x_0$ based on data $v_0, v_1, v_2$ at equispaced grid points $x_0 = 0, x_1 = h, x_2 = 2h$.

Newton form of interpolant:

$$p(x) = v_0 + (v_1 - v_0)x/h + (v_2 - 2v_1 + v_0)x(x-h)/2h^2$$

This implies

$$p'(x) = (v_1 - v_0)/h + (v_2 - 2v_1 + v_0)(2x-h)/2h^2$$

and thus

$$p'(0) = (v_1 - v_0)/h - (v_2 - 2v_1 + v_0)/2h = [\frac{-1}{2}v_2 + 2v_1 - \frac{3}{2}v_0]/h$$

How to compute these coefficients?

**Old school — operational calculus** (for regular grids)

This goes back to Newton and Gregory; then, a century ago, to people like Heaviside and Milne-Thompson.

Pretty mathematics! — less important nowadays.

For example, let $\Delta$ be the *forward difference operator*:

$$\Delta f(s) = f(s+1) - f(s).$$

Then $1 + \Delta$ is the *forward shift operator*:

$$(1+\Delta)f(s) = f(s+1).$$

To shift by an arbitrary distance $x$, this suggests the formal binomial series

$$f(x) \sim (1+\Delta)^x f(0) = \left[1 + \binom{x}{1}\Delta + \binom{x}{2}\Delta^2 + \cdots\right] f(0).$$

Convergence of this series is problematic for general $f$, but if $f$ is a polynomial of degree $n$, it terminates at step $n$:

$$f(x) = (1+\Delta)^x f(0) = \left[1 + \binom{x}{1}\Delta + \binom{x}{2}\Delta^2 + \cdots + \binom{x}{n}\Delta^n\right] f(0).$$

This is a formula for the polynomial interpolant of degree $n$! People did a lot with such formulas in the old days, now mostly forgotten.

**New school — recursive algorithms** (for arbitrary grids)

Slick algorithms for computing such formulas in general have been developed by Fornberg, and can be found online at `http://amath.colorado.edu/faculty/fornberg/`:

B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Math. Comput.* 51 (1988), 699-706.

B. Fornberg, "Calculation of weights in finite difference formulas", *SIAM Review* 40 (1998), 685-691.

[ handout from Fornberg's 1988 paper ]

Mention of FEM, RBF, RBF-finite diffs.

## V.8 Multiple space dimensions

In two or three space dimensions, the ideas of discretization, stability, convergence etc. are virtually the same as in 1D.

However, something big changes: the matrices are no longer narrowly banded. This means that actually implementing these formulas may become expensive, since implicit formulas may require some difficult linear algebra. In practice the linear algebra aspect of these multidimensional problems is often a dominant feature.

Simplest example: the heat equation on a square:

$$u_t = u_{xx} + u_{yy}, \quad 0 < x, y < 1, \quad u = 0 \text{ on boundaries.}$$

[cf. earlier handout on the heat equation from the *PDE Coffee Table Book*]

Here is a very simple discretisation. Set up a regular grid:

$$x_i = ih, \quad y_j = jh, \quad t_n = nk, \qquad h = 1/J.$$

Now construct a backward Euler finite difference approximation:

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

This is unconditionally stable, whereas forward Euler would have a bad stability restriction: $k < h^2/4$.

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

In principle, our "finite difference Laplacian" maps a matrix of 2D grid data to another such, and is thus a 4D tensor.

However, in practice we stretch out 2D to 1D, so that the tensor becomes a matrix:

$$Bv^{n+1} = v^n, \quad B = \mathbf{I} - kL,$$

$L$ is the discrete Laplacian [cf. Assignments 1 and 2 last term]

```
L  =  kron(I,D) + kron(D,I),   D = h^(-2)*tridiag(1,-2,1).
```

(Recall that this is called a **Kronecker sum**.)

Notation: $I$ = identity of dimension $J-1$, $\mathbf{I}$ = identity of dimension $(J-1)^2$.

Let's examine explicitly what's going on here. The case $J = 4$ will serve to illustrate. We number the interior grid points

```
x7   x8   x9


x4   x5   x6


x1   x2   x3
```

with corresponding unknowns $v_1, \ldots, v_9$. The discrete Laplacian now looks like this:

```
                |-4   1   0    1   0   0    0   0   0 |    | v1 |
                |                                     |    |    |
                | 1  -4   1    0   1   0    0   0   0 |    | v2 |
                |                                     |    |    |
                | 0   1  -4    0   0   1    0   0   0 |    | v3 |
                |                                     |    |    |
                |                                     |    |    |
                | 1   0   0   -4   1   0    1   0   0 |    | v4 |
          -2    |                                     |    |    |
  Lv  =  h      | 0   1   0    1  -4   1    0   1   0 |    | v5 | ,
                |                                     |    |    |
                | 0   0   1    0   1  -4    0   0   1 |    | v6 |
                |                                     |    |    |
                |                                     |    |    |
                | 0   0   0    1   0   0   -4   1   0 |    | v7 |
```

```
|                                                 |   |     |
| 0   0   0   0   1   0   1  -4   1 |   | v8 |
|                                                 |   |     |
| 0   0   0   0   0   1   0   1  -4 |   | v9 |
```

We say that $L$ is **block tridiagonal** with **tridiagonal blocks**.

[`m47_BackwardEuler2D.m` — backward Euler — also periodic, `m47per`]

[`m48_CrankNicolson2D.m` — Crank-Nicolson ] ← removing parentheses has a huge effect, `m48slow` (best to run with $J = 60$ as $J = 90$ takes a long time). Also do periodic, `m48per`

[`m49_leapfrog2D.m` — leap frog for $u_{tt} = u_{xx} + u_{yy}$ ]