Brown/Paris Numerical Analysis: Problem set 12 (last one!)

Nat Trask

July 21, 2015

1 Numerical methods for solving PDE: Pulling everything together

Up to this point we've learned a lot of different methods (linear solvers, root finding, ODE discretization, PDE discretization). In this assignment we will combine all of our previous work to solve the unsteady heat equation. You should be able to pull subroutines from previous homeworks to complete this (and feel a feeling of satisfaction) but if you weren't able to get the previous assignments working feel free to use the native MATLAB routines where necessary.

- We seek a discretization of the unsteady heat equation:

$$\partial_t u + \partial_{xx} u = 0$$

- Following the method of lines, we discretize separately in the space and time dimensions. Here you can pick your favorite ODE method:

$$\frac{u^{n+1} - u^n}{\Delta t} + \theta \partial_{xx} u^{n+1} + (1 - \theta) \partial_{xx} u^n = 0$$

  i.e. pick $\theta \in \{1, 0.5, 0\}$ to obtain implicit Euler, Crank-Nicolson, and explicit Euler respectively.

- Use your favorite PDE discretization (finite difference, spectral collocation, spectral Galerkin) to discretize the above system of equations.

$$\mathbf{M} \frac{\vec{u}^{n+1} - \vec{u}^n}{\Delta t} + \mathbf{L} \theta \vec{u}^{n+1} + (1 - \theta) \vec{u}^n = 0$$

  i.e. $\mathbf{M}$ is the identity matrix in finite difference or the mass matrix in the spectral methods, and $\mathbf{L}$ is either the finite difference Laplacian matrix or the spectral Laplacian operator.

- Use your favorite linear system solver to invert this system of equations. Think about the properties of your matrix and what you can exploit. Make sure that if you're using MATLAB routines you use the right data structures (i.e. for sparse solvers you should be using sparse matrix format).

- March on in time! If everything is working and you follow stability restrictions, you should see your initial condition decay over time.

Advice:
Pulling a bunch of mathematical pieces together is difficult and part of the art of scientific programming. Try to structure your code in a modular way so that you can easily debug which part is giving you trouble.

Your choice of discretization will come in handy here. For some combination of methods you can prove on paper that the method should converge and at what rate. This makes debugging much easier!