AMa105b

Homework #3 (Elliptic and Parabolic Equations)

Handed out: Tuesday, May 21, 1996
Due in my office: Wednesday, June 5, 1996
(Note the due date is later than originally announced.)

A general second-order diffusion equation in an open set $\Omega \subset \mathbb{R}^2$ with Dirichlet boundary conditions can be written in the form:

$$\rho \frac{\partial u}{\partial t} = \nabla \cdot (a \nabla u) - cu + f \quad \text{in } \Omega \times (0, T],$$

$$u(x, y, t) = g(x, y, t) \quad \text{on } \partial\Omega \times (0, T],$$

$$u(x, y, 0) = u_0(x, y) \quad \text{on } \Omega,$$

where $a(x, y) > 0$, $c(x, y) \geq 0$, $\forall (x, y) \in \Omega \subset \mathbb{R}^2$.

In this problem set, we will solve special forms of this parabolic equation in MATLAB using finite differences (and finite elements, if you do the extra-credit part!). In particular, we will consider the case $\Omega = (0, 1) \times (0, 1)$ (the open unit square), $g \equiv 0$ (homogeneous Dirichlet conditions), and $a \equiv 1, c \equiv 0$ (the heat operator), so that our problem becomes simply the heat equation

$$\rho \frac{\partial u}{\partial t} = \nabla^2 u + f \quad \text{in } \Omega \times (0, T],$$

$$u(x, y, t) = 0 \quad \text{on } \partial\Omega \times (0, T],$$

$$u(x, y, 0) = u_0(x, y) \quad \text{on } \Omega,$$

where $\Omega = (0, 1) \times (0, 1)$. To complete the following problems, I will give you some of the trickier pieces of code, and you will put the pieces together to solve the problems.

- **Problem 1.** (EQUILIBRIUM CASE WITH FINITE DIFFERENCES)

  If we first take $\rho = 0$ (the equilibrium case) then we have an elliptic rather than parabolic equation:

  $$-\nabla^2 u = f \quad \text{in } \Omega,$$

  $$u(x, y) = 0 \quad \text{on } \partial\Omega.$$

  To solve this problem numerically, we place a uniform mesh of n points in each of the two coordinate directions (x and y) so that

  $$0 = x_0 < x_1 < \ldots < x_n < x_{n+1} = 1, \qquad 0 = y_0 < y_1 < \ldots < y_n < y_{n+1} = 1.$$

  The discrete mesh Ω_h then consists of the points (x_i, y_j) lying in the interior of the domain Ω, and the closed set $\bar{\Omega}_h$ then includes the boundary points. With a uniform mesh spacing of $h = 1/(n + 1)$, the mesh points forming $\bar{\Omega}_h$ are characterized by

  $$\{(x_i, y_j) \mid \quad x_i = ih, \quad y_j = jh, \quad i = 0, 1, \ldots, n + 1, \quad j = 0, 1, \ldots, n + 1\}.$$

  Your assignment is to discretize the elliptic problem on this mesh, and solve the resulting matrix equations $L_h u_h = f_h$ to yield an approximate solution u_h. In particular, do the following:

1. Use the source function:

$$f(x,y) = (p^2+q^2)\pi^2 \sin(p\pi x)\sin(q\pi y),$$

with $p = q = 1$ for this part.

2. Generate the matrix equation $L_h u_h = f_h$ in MATLAB for a specified number of input mesh points n.
3. Implement the simple CG algorithm from last quarter for solving the matrix equation (we won't use a preconditioner).
4. Solve the resulting matrix equation using CG and also using MATLAB's backslash operator (for comparison) three times each (total of 6), using: $n = 15, 31, 63$.
5. To decide when to stop CG, drive the residual down to $1.0e-7$:

$$\frac{\|f_h - L_h u_h^i\|_2}{\|f_h\|_2} < 1.0e-7,$$

where u_h^i is the approximation computed by CG after i iterations, and $\|\cdot\|_2$ is the usual Euclidean 2-norm. Make sure the iterative (CG) and the direct (MATLAB/backslash) solutions agree to several digits.

6. Plot the three solutions using one of MATLAB's functions (mesh, surf, surfc, ...).
7. Compare your numerical results to the analytical solution, which is:

$$u(x,y) = \sin(p\pi x)\sin(q\pi y).$$

In particular, monitor the discretization error:

$$\|Ru - u_h\|_2 = \left(\sum_{i=1}^{n}\sum_{j=1}^{n}\left(u(x_i,y_j) - u_h(x_i,y_j)\right)^2\right)^{1/2},$$

where R is the canonical restriction that simply samples the continuous solution u at the discrete mesh points (x_i, y_j), u_h is our discrete approximation at the mesh points, and $\|\cdot\|_2$ is the usual Euclidean 2-norm. As you double the number of mesh points n (halve the mesh spacing h), does the error behave like $O(h^2)$ as predicted by the theory from class? Measure also the relative discretization error:

$$\frac{\|Ru - u_h\|_2}{\|Ru\|_2}.$$

Does the relative error behave like $O(h^2)$? Can you explain the different behaviors?

8. If you redo the earlier parts, but take $p = q = 5$ in the definition of the source function $f(x,y)$, what happens to the (relative) discretization error? Why does this happen?

Hint 1: To make this problem easier, I'll give you a very simple way to make the matrix L_h in MATLAB, so that all you need to do is to construct the discrete source term f_h, and get all the bookkeeping correct with h^2 factors and so on. The following MATLAB routine will generate the finite difference discretization of the Laplace operator on the unit square given a specified number of mesh points to use in both directions on the square. I.e., you specify the number of mesh points n in the x- and y-directions, and back comes our $N \times N$ matrix L_h, where $N = n^2$. Note that the factor of $1/h^2$, with $h = 1/(n+1)$, is *not* included in the matrix returned by this function; you must multiply the right-hand-side of your matrix equations by h^2 as described in class.

Here is the discrete Laplace matrix generator:

```
function Lh = laplace(n)
   N = n*n;
   B = spdiags(ones(n,1),1:1, n, n);
   B = B+B';
   if (n == 1)
      Lh = 4*speye(N) ;
   else
      Lh = 4*speye(N) - kron(B,speye(n)) - kron(speye(n),B);
   end
end
```

Note that this routine generates L_h in MATLAB's internal sparse format; this allows you to solve a very large problem in MATLAB. (If you represented L_h as a full matrix, you wouldn't be able to complete this problem for $n = 63$ on your workstation...)

Hint 2: You will need to move between the "vector" form of the problem, as in the matrix equation $L_h u_h = f_h$, and the "mesh" form of the problem, such as when you wish to plot the discrete solution over the unit square. MATLAB's "reshape" function allows you to move between the two representations (do a "help reshape" in MATLAB).

- **Problem 2.** (EVOLUTION CASE WITH FINITE DIFFERENCES)

We now will take $\rho \equiv 1$ to yield the heat equation:

$$\frac{\partial u}{\partial t} = \nabla^2 u + f \quad \text{in } \Omega \times (0, T],$$

$$u(x, y, t) = 0 \quad \text{on } \partial\Omega \times (0, T],$$

$$u(x, y, 0) = u_0(x, y) \quad \text{on } \Omega,$$

where $\Omega = (0, 1) \times (0, 1)$.

Since we have routines for discretizing the spatial derivatives in the equation above, and for solving the resulting matrix equations, we can easily put together explicit or implicit time-stepping methods for the heat equation. If we take the "method-of-lines" view of semi-discretizing in space, then in fact we have a system of ODEs to solve:

$$\dot{u_h} = L_h u_h + f_h,$$

where $\dot{u_h} = du_h/dt$. (Note that the matrix L_h could in fact be generated by finite elements instead, although we must also then place a "mass" matrix M_h in front of the time-derivative term...see me if you decide to do the extra-credit problem.)

Using a fixed time-step k for discretizing the remaining time derivative, the three methods we examined in class take the form:

$$\text{Forward Euler}: \quad u_h^{n+1} = [I + kL_h]u_h^n + kf_h$$

$$\text{Backward Euler}: \quad [I - kL_h]u_h^{n+1} = u_h^n + kf_h$$

$$\text{Crank} - \text{Nicolson}: \quad [I - \frac{k}{2}L_h]u_h^{n+1} = [I + \frac{k}{2}L_h]u_h^n + kf_h$$

Your assignment is to:

1. Write three simple MATLAB routines to implement the three methods above, using the matrix L_h generated by the LAPLACE routine above. (You can just use MATLAB's backslash operator for the implicit methods rather than your CG routine to simplify this part.)
2. Using the source function:
$$f(x,y) = (p^2+q^2)\pi^2 \sin(p\pi x)\sin(q\pi y),$$
and the initial condition:
$$u(x,y,0) = u_0(x,y) = \sin(p\pi x)\sin(q\pi y),$$
with $p = q = 2$, integrate the problem with each of the three methods, using $n = 31$ spatial mesh points, and $m = 100$ time steps. Since all three of these methods behave like $O(k, h^2)$, you will probably want to take $k = Ch^2$ for some constant C for accuracy considerations. If you violate the stability limit derived in class for Forward Euler, namely
$$k \leq \frac{h^2}{2},$$
what happens? What happens to the other two methods as you similarly increase the time-step in relation to the spatial mesh-width? Plot (mesh/surfc/surfl/...) your solutions for each methods for a few early and late time steps.
3. Using now the source function:
$$f(x,y) = 0,$$
and the initial condition:
$$u(x,y,0) = u_0(x,y) = \sin(p\pi x)\sin(q\pi y),$$
with $p = q = 2$, integrate the problem with each of the three methods, using $n = 31$ spatial mesh points, and $m = 100$ time steps. If you violate the stability limit derived in class for Forward Euler again, what happens? What happens again to the other two methods as you similarly increase the time-step in relation to the spatial mesh-width? Plot (mesh/surfc/...) your solutions for each method for a few early and late time steps.

- **Problem 3.** (FINITE ELEMENTS – EXTRA CREDIT)

  Repeat Problems 1 and 2 with a finite element matrix generator which I will supply to you (contact me if you want to do this problem, and I'll arrange for you to get the MATLAB finite element code that I have written).

  This problem is actually very easy once you have done Problems 1 and 2 with finite differences; the only changes you will need to make to your code are:

  1. Call the finite element matrix generator rather than the LAPLACE routine above.
  2. Use my finite element function plotter rather than MATLAB's mesh/surf routines to plot the solutions (since the mesh points can lie anywhere in $\mathbb{R}^2$, we can't use uniform-mesh-based things).
  3. Include the "mass" matrix M_h in the case of the evolution problem, or illiminate it by "mass-lumping".

  There will be a little time involved in figuring out how to call my finite element matrix generator, but it is pretty straight-forward. After you get the matrix, then you can forget where it came from and just use the code you have written for the finite difference case (again, except for the plotting of the solution).

  In this problem, you will be able to work with arbitrary polyhedral domains in $\mathbb{R}^2$ rather than just the unit square; simplicial meshes in 2D and 3D allow for great generality.