Second Year Engineering Mathematics Laboratory
Hilary Term 2000

Ian Reid, 21/12/1999

Exercise 6: PDEs

The purpose of this exercise is to have a look at solutions to partial differential equations by computer. You have seen in lectures how second order PDEs can be classified as elliptic, parabolic and hyperbolic. The prototypical equations for these type are respectively, *Poisson's equation* (of which *Laplace's equation* is a special case), the *diffusion equation*, and the *wave equation*.

From a computational point of view a more meaningful classification divides the problems into *initial value* (or *Cauchy*) problems and *boundary value* problems. In the former, we are interested in the time evolution of a system (typically parabolic or hyperbolic), given a set of initial conditions. In the latter, typically arising from elliptic equations, we are usually interested in finding a "static" function which satisfies the equation within a region of interest, given some desired behaviour on the region boundary.

You have seen example of all three in lectures. In this lab, we concentrate on initial value problems (arising from parabolic and hyperbolic equations), and extend these from the one-dimensional versions you have seen, into two dimensions.

1 Preparation

2D diffusion equation

The two-dimensional (unsteady) heat equation governing the diffusion of heat in a two-dimensional slab is given by

$$\frac{\partial u}{\partial t} = \frac{K}{c\rho} \bigtriangledown^2 u$$

where K, c and ρ are (respectively) the thermal conductivity, the specific heat capacity, and the density of the material.

By writing the left-hand side as a forward difference (in time) and the right hand side as two central differences in x and y (assuming that $\delta x = \delta y = h$), show that an explicit update scheme for the time evolution of the temperature in a slab is given by

$$u_{i,j}(k+1) = r(u_{i-1,j}(k) + u_{i+1,j}(k) + u_{i,j-1}(k) + u_{i,j+1}(k) - 4u_{i,j}(k)) + u_{i,j}(k)$$

where

$$r = \frac{K\delta t}{c\rho h^2}$$

The stability criterion for this explicit scheme is that $r \leq 0.25$. If we are welding a slab of mild steel, and model the spatial dimensions at intervals of 10cm, what is the largest time step over which we can observe the evolution while maintaining stability of the iteration?

2D wave equation

The two-dimensional wave equation is given by

$$\frac{\partial^2 u}{\partial t^2} = c^2 \bigtriangledown^2 u$$

By approximating both sides of the equation with appropriate finite differences, show that an explicit update for the time evolution of the wave is given by

$$u_{i,j}(k+1) = r(u_{i-1,j}(k) + u_{i+1,j}(k) + u_{i,j-1}(k) + u_{i,j+1}(k) - 4u_{i,j}(k)) + 2u_{i,j}(k) - u_{i,j}(k-1)$$

where

$$r = \frac{c^2 \delta t^2}{h^2}$$

and $h = \delta x = \delta y$.

The highest value of r for stability is $\frac{1}{2}$ Show that when $r = \frac{1}{2}$ the equation above simplifies considerably.

Show that if $g_{i,j}(0)$ is a *central* difference approximation to the initial velocity at node (i, j), then the first update is given by

$$u_{i,j}(1) = \frac{1}{4}(u_{i-1,j}(0) + u_{i+1,j}(0) + u_{i,j-1}(0) + u_{i,j+1}(0)) + \delta t g_{i,j}$$

2 Lab Work

A. 2D heat equation

In this exercise we look at the diffusion of temperature through a slab of mild steel immediately after welding.

Begin by creating a function called **heatstep** (of course put it in a file called `heatstep.m`) which will evolve the system forward by one time step. Use the code below as a guide, and fill in the missing bits from your preparation:

```
function vnew = heatstep(v,alpha,t,h)
% evolve the system forward one time step
%       alpha: = k/(cp) where k = thermal conductivity
%                               p = density
%                               c = specific heat capacity
% t: size of time step
%       h: size of grid spacing
[m,n]=size(v);
vnew = v;

for j=2:m-1
    for i=2:n-1
        vnew(i,j) = ?????????;
    end
end
```

What boundary conditions are implicit in this code?

Now copy the file `heat2d.m` from the directory `/packages/demo/comp/lab6` into your own `lab6` directory. You are now almost in a position to try it out. Start by defining a parameter `alpha` which will equal the *thermal diffusivity*; i.e. $\frac{K}{c\rho}$ for mild steel.

```
>> alpha = ??????;
```

Now assume that a spot weld has been made in the centre of a sheet of steel 2m by 2m. We will model this by a 20 by 20 grid at 18^oC with an array of elements at 1600^oC in the centre:

```
>> v0 = 18*ones(20,20);
>> v0(9:12,9:12) = 1600*ones(4,4);
```

Call the function **heat2d** and observe the temperature evolution over 10 time steps. You will need first to look at the code in `heat2d.m` or to type `help heat2d` to determine the parameter meanings.

```
>> heat2d(v0,alpha,??,??,10)
```

We can turn this into an animation by replacing a few lines in the file `heat2d.m`. Add two new (identical) lines, one after each of the `pause` statements:

```
m(:,i) = getframe;
```

and change the function header so that it returns the movie `m`:

```
function m = heat2d(v0,alpha,t,h,n)
```

If you now run the function with the same parameters as before, it will return an animation of the time evolution of the temperature in the slab. You can view a movie using the `movie` command.

```
>> m = heat2d(??,??,??,??,20);
>> movie(m,10);
```

Try out a value for δt which is greater than the maximum allowable for stability. Note down what you observe.

Consider different initial conditions such as (a) the left half of the slab at $100\,^oC$ and the right half at 0^oC; (b) a weld along a line from top to bottom, and note down your observations.

[Optional] Consider how you would modify the code in `heat2d.m` for the more realistic assumption that the weld in (b) above is made sequentially in time, so that at successive time intervals different points are raised to 1600^oC. Note down you thoughts in your lab book but don't implement it (unless you have time).

B. Two dimensional wave equation

In this exercise we model the vibration of a membrane stretched across a rectangular frame.

Write a function called **wavestep** which implements the explicit update scheme you derived in part B of the preparation. The code below gives a template.

```
function unew = wavestep(u1,u0)
     % u1, u0: previous two time steps
[m,n] = size(u1);
unew = u1;

for i=2:m-1
    for j=2:n-1
         unew(i,j) = ????????;
    end
end
```

Copy the file `wave2d.m` from the usual place and have a look at the code.

What are the initial conditions, and how does this code implement them?

Try out the function **wave2d** (which calls your function **wavestep**) for a membrane stretched over a square frame occupying the region $0 \leq x \leq 2, \quad 0 \leq y \leq 2$, with the starting position of the membrane given by $u = x(2-x)y(2-y)$:

```
>> x = [0:0.5:2];
>> y = [0:0.5:2];
>> u0 = x'*y;
>> for i=1:length(y)
         for j=1:length(x)
             u0(i,j) = x(i)*(2-x(i))*y(j)*(2-y(j));
         end
   end
>> uexp = wave2d(u0,20);
```

You can view each time step using the **surf** function. For example, type

```
>> surf(uexp(:,:,1);
```

to view the first time step.


If $c^2 = 3$, what is the maximum value of δt to ensure stability?

An analytic solution to this problem is given by

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} B_m n \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\cos(c\pi t\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}$$

$$\text{where} \quad B_{mn} = \frac{16a^2b^2}{\pi^6 m^3 n^3}(1-\cos m\pi)(1-\cos n\pi)$$

and the membrane is attached to a frame $0 \le x \le a,\ \ 0 \le y \le b$ (see Kreyszig 11.8 for details).

The file `membranesoln.m` implements a function **membranesoln** which computes the first 625 fourier components to provide an approximate analytic solution from the usual place. You can obtain an estimate of the solution by typing

```
>> t = [0:????:????*10];
>> utrue = membranesoln(2,2,sqrt(3),x,y,t);
```

Where the "`????`" should be replaced by your calculation above for maximum δt for stability.

You can view the analystic solution for the different times in the array `t` as you did for the numerical solution:

```
>> surf(utrue(:,:,1));
```

to view the solution at $t = 0$.

Unfortunately, our forward differencing scheme for this membrane problem, although stable, introduces errors. We can view these by finding the difference between the numerical and analytic solutions.

```
>> diff = uexp(:,:,1:10) - utrue(:,:,1:10);
```

Plot the maximum errors for the first ten time steps:

```
>> for i=1:10
       r(i) = norm(diff(:,:,i),inf);
   end
>> plot(r);
```


Note down your observations in your log.

It turns out that to improve the numerical accuracy of our explicit scheme we must use a much smaller quantisation of the x and y directions (i.e. make h smaller). In order to maintain stability this also measn that δt must decrease in proportion.

In practice this is not a great idea, and one would be better off with an *implicit* method (analogous to Crank-Nicholson for parabolic equations). However here just to finish off, we will increase the spatial resolution and create a couple of movies. Code to do this is provided in the usual place in a file called `wave2dmov.m`. Copy this to your area, then create an array `u0` as you did before, but this time with $h = 0.1$ (i.e. `x=[0:0.1:2]` and `y=[0:0.1:2]`). Now type

```
>> m = wave2dmov(u0,20);
>> movie(m,20);
```


What value of δt maintains stability?

Finally (**optional**), try out **wave2dmov** for asymmetric initial conditions such as $u = x^6(2-x)y(2-y)$, and view the resulting movie.