MATH 517 PROJECT 2: DIFFUSION EQUATION, EFFECT OF BOUNDARY CONDITIONS

Turn in on Blackboard by the end of Sunday 9/28.

Goal: obtain a numeric approximation to the solution of the PDE $u_t = ku_{xx}$ with Dirichlet boundary condition at one end and Neumann boundary condition at another.

Recommended software: Matlab (available in SU computer clusters) or its free alternative Scilab (available for download from `www.scilab.org`).

Answer the following questions, based on output:

(1) How does the Dirichlet boundary condition (immediate escape) affect the solution?
(2) How does the Neumann boundary condition (no escape) affect the solution?
(3) What happens if you change the time step Δt to a number such that $2k\Delta t = (\Delta x)^2$?
(4) What happens if you change the time step Δt to a number such that $2k\Delta t > (\Delta x)^2$?

You can include the answers as comments when submitting the file on Blackboard.

Data (posted on Blackboard among the grades): diffusion coefficient k and space step Δx.

Method: First, choose a value of time step Δt so that $2k\Delta t < (\Delta x)^2$. Don't go for extremely small values of Δt, or the computations will take very long.

Set up the matrix of u values (so far, filled with zero), computing its size using the time and space steps, similar to Project 1.

The solution should be on space interval $-1 \le x \le 1$ and time interval $0 \le t \le 1$.

For the initial values of u, only one entry of the matrix should be assigned a nonzero value: put $1/\Delta x$ in the entry that corresponds to $x = 0$ and $t = 0$. That entry is $U(1, 1 + 1/\Delta x)$; I use capital letter U for the matrix to distinguish it from the function u.

Use the boundary conditions $u = 0$ when $x = -1$ and $u_x = 0$ when $x = 1$. The Dirichlet condition $u = 0$ when $x = -1$ is enforced simply by leaving all entries $U(i, 1)$ equal to zero in the process of calculation. To enforce the Neumann condition $u_x = 0$ when $x = 1$, we need another way: see below.

For most of the matrix, the difference scheme

$$U(i+1, j) = U(i, j) + \frac{c\Delta t}{(\Delta x)^2} \left(U(i, j-1) - 2U(i, j) + U(i, j+1) \right)$$

is used to calculate the solution. Exceptions: this formula should not be applied when j is at the right edge of the matrix, since $j + 1$ would go out of bounds. Instead, replace $j + 1$ by $j - 1$ in this case: this expresses the Neumann boundary condition:

$$U(i+1, j) = U(i, j) + \frac{c\Delta t}{(\Delta x)^2} \left(U(i, j-1) - 2U(i, j) + U(i, j-1) \right)$$

Sample elements of program. The set up of parameters such as timeStep, numSpaceSteps, etc, is as in Project 1, except of course the numbers are different. You will also need to define $k =$ (the diffusion coefficient), and impose the initial condition as described above. Then run the double loop that calculates the solution using a difference scheme:

```
for i = 1 : numTimeSteps-1
    for j = 2 : numSpaceSteps-1
         U(i+1,j) = U(i,j) + k*timeStep/(spaceStep)^2*(U(i,j-1)-2*U(i,j)+U(i,j+1));
    end
    j = numSpaceSteps;
    U(i+1,j) = U(i,j) + k*timeStep/(spaceStep)^2*(U(i,j-1)-2*U(i,j)+U(i,j-1));
end
```

Finally, plot the solution. To avoid clutter, the loop below plots every 100th time step. Also, the very first rows have quite large values of u, which would distort the scale if they were shown. For this reason plotting begins only with 100th step of calculation:

```
hold on                  .....     omit this if using Scilab
for i = 1 : numTimeSteps/100
    plot(a:spaceStep:b, U(100*i,:));
    pause(.5)         .....       replace with xpause(.5e6)  if using Scilab
end
```