MMF 2021F ASSIGNMENT #5 Due: 13 December 2019.

This assignment is due on Friday, 13 December 2019. Put in in my mailbox in the MMF lunch room and I'll stop by late in the afternoon on Friday, 13 December 2019, to pick-up the assignments. We finalize the details about this by email.

This assignment asks you to write some MatLab programs. Hand-in the programs and their output as well as any written answers requested in the assignment. Your programs and their output, as well as your written answers, will be marked. Your programs should be "well-commented" and use good programming style, etc. When first learning to program in MatLab, students often produce long, messy output, but you should be an experienced programmer now. So, try to format the output from your programs so that it is easy for your TA to read and to understand your results.

1. Consider the $n \times n$ *symmetric, tridiagonal* matrix (it could also be called a *sparse or banded* matrix)

$$A = \begin{pmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & & & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & -2 \end{pmatrix}$$

   for which

   - $A_{i,i} = -2$, for $i = 1, \dots, n$,
   - $A_{i,i-1} = 1$, for $i = 2, \dots, n$,
   - $A_{i,i+1} = 1$, for $i = 1, \dots, n-1$ and
   - $A_{i,j} = 0$, otherwise.

   There are functions in MatLab that take advantage of the sparsity structure of such matrices. For example, to create a 100×100 matrix of the form A in sparse matrix format in MatLab, you can use the commands

   $n = 100;$
   $e =$ ones(n,1);
   $A =$ spdiags($[e, -2 * e, e]$, -1:1, n, n);

Read "help spdiags" in MatLab for more information about spdiags.

Using this sparse matrix format, MatLab stores only the $3n - 2$ non-zero elements in the matrix A, rather than all n^2 elements.

You can create a *full* matrix B, equivalent to A, using the MatLab command

$B = \text{full}(A);$

Similarly, you can create a matrix, C, identical to A in sparse matrix format using the MatLab command

$C = \text{sparse}(B);$

Read "help full" and "help sparse" in MatLab for more information about these functions.

In addition to saving storage, many operations are much faster if you perform them with a matrix in sparse format, rather than full format. To see this, create a vector b of length n (e.g., b = ones(n,1)) and time

(a) how long it takes MatLab to solve $Ax = b$ using the MatLab command $x = A\backslash b$, and

(b) how long it takes MatLab to solve $Bx = b$ using the MatLab command $x = B\backslash b$,

where A is the sparse format version of the matrix A shown at the beginning of this question and $B = \text{full}(A)$ is the full format version of the matrix A discussed above.

Use the MatLab tic and toc functions (read "help tic" in MatLab) to do the timing. For several different values of n, record the times that it takes to perform these solves. For the solves with A, try to estimate parameters c_1 and p_1 such that the time taken to solve $Ax = b$ for a system of size n is approximately $c_1 n^{p_1}$. An easy way to do this is to plot the time taken to solve $Ax = b$ versus n on a log–log plot, since, if $T(n) \approx c_1 n^{p_1}$ is the time to solve the system of size n, then $\log(T(n)) \approx \log(c_1) + p_1 \log(n)$. So, p_1 will be the slope of the approximating line in the log–log plot and $\log(c_1)$ will be its y-intercept. Similarly, for the solves with B, try to estimate parameters c_2 and p_2 such that the time taken to solve $Bx = b$ for a system of size n is approximately $c_2 n^{p_2}$.

Note: n must be quite large for the solve $Ax = b$ to take enough time for tic–toc to time it accurately. Because solves with A are much faster than solves with B, you might find it helpful to use larger n's for A than for B. (When I ran this, I used 5 values for n between 50,000 and 150,000 for A and 5 values for n between 1,000 and 5,000 for B.) Also, run your program a few times. Computer clocks are not as accurate as you might initially expect them to be.

Hand in your MatLab program that you used to solve this problem, your table of times that MatLab took to solve $Ax = b$ and $Bx = b$, your plots, and your estimates of c_1, c_2, p_1 and p_2.

2. For this question, you will solve the heat equation

$$\frac{\partial u(t,x)}{\partial t} = \frac{\partial^2 u(t,x)}{\partial x^2} \tag{1}$$

for $t \in [0,1]$ and $x \in [0,1]$ with the boundary conditions

$$u(t,0) = u(t,1) = 0 \qquad \text{for } t \in [0,1] \tag{2}$$

and with the initial condition

$$u(0,x) = 4x(1-x) \qquad \text{for } x \in [0,1] \tag{3}$$

You will also solve the heat equation (1) with the boundary conditions (2) and the initial condition

$$u(0,x) = \begin{cases} 2x & \text{for } x \in [0, \frac{1}{2}] \\ 2(1-x) & \text{for } x \in [\frac{1}{2}, 1] \end{cases} \tag{4}$$

As noted in class, the exact solution of either of these problems (i.e. either (1)–(2)–(3) or (1)–(2)–(4)) can be written as

$$u(t,x) = \sum_{k=1}^{\infty} c_k \, \mathrm{e}^{-k^2\pi^2 t} \sin(k\pi x) \tag{5}$$

where

$$c_k = 2 \int_0^1 u(0,x) \sin(k\pi x) \, dx$$

For the initial condition (3),

$$c_k = \begin{cases} 0 & \text{for } k \text{ even} \\ \frac{32}{k^3\pi^3} & \text{for } k \text{ odd} \end{cases} \tag{6}$$

while, for the initial condition (4),

$$c_k = \begin{cases} 0 & \text{for } k \text{ even} \\ (-1)^{\frac{k+3}{2}} \frac{8}{k^2\pi^2} & \text{for } k \text{ odd} \end{cases} \tag{7}$$

Thus, you can compute an accurate approximation to the solution of either of these problems (i.e., either (1)–(2)–(3) or (1)–(2)–(4)) by choosing an appropriate value of K and approximating the solution (5) by

$$u(t,x) = \sum_{k=1}^{K} c_k \, \mathrm{e}^{-k^2\pi^2 t} \sin(k\pi x) \tag{8}$$

You will need to choose K so that the approximation to $u(t,x)$ given by (8) is significantly more accurate than the approximations generated by the numerical methods described below. Because the series (5) converges faster with the coefficients (6) than

with the coefficients (7), you may have to choose K larger when you are solving the heat equation (1) with initial condition (4) than with initial condition (3).

The accurate approximation to the true solution computed by (8) is referred to as the "exact" solution below (even though it is not really exact).

Hand in a brief discussion of how you chose K for the two problems, (1)–(2)–(3) and (1)–(2)–(4), and why you believe the values you chose are appropriate for the "exact" solutions.

(a) Use the Explicit Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (3). Also use the Explicit Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (4). Let $\rho = 0.3$ and solve each problem several times with different values of δt and δx satisfying $\delta t/\delta x^2 = \rho$.

For each problem (1)–(2)–(3) and (1)–(2)–(4), plot the "exact" solution $u(1, x)$ versus x for $x \in [0, 1]$ and the numerical solution approximating it for the different stepsizes. On a separate graph, plot the associated errors (i.e., for each of the different stepsizes you used, plot the "exact" solution $u(1, x)$ minus the numerical solution approximating it versus x for $x \in [0, 1]$). Consider if it might be better to plot the log of the absolute value of the error rather than the error itself.

For both problems (i.e., (1)–(2)–(3) and (1)–(2)–(4)), try to estimate how the error in the numerical solution at $t = 1$ decreases as δt and δx decrease, while maintaining the relationship $\delta t/\delta x^2 = \rho$.

Does the rate of convergence you observe agree with the rate of convergence for the Explicit Method we discussed in class?

Hand in the program that you developed for this question, your plots, your analysis of the rate of convergence of the numerical method observed from the numerical results, and your discussion of how this observed rate of convergence agrees with the theoretical rate of convergence of the Explicit Method.

(b) Use the Explicit Method again to solve the heat equation (1) with the boundary conditions (2) and the initial condition (3). Also use the Explicit Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (4). However, in this case, let $\rho = 2/3$ and solve each problem several times with different values of δt and δx satisfying $\delta t/\delta x^2 = \rho$.

You should get a very bad solution in this case. What does this confirm about the stability of the Explicit Method?

Hand in the program that you developed for this question, the numerical solutions that it computed, and your answer to the question in the paragraph above.

(c) Use the Fully Implicit Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (3). Also, use the Fully Implicit Method to solve the heat equation (1) with the boundary conditions (2) and the initial

condition (4). Let $\rho = 1$ and solve each problem several times with different values of δt and δx satisfying $\delta t/\delta x = \rho$.

For each problem (1)–(2)–(3) and (1)–(2)–(4), plot the "exact" solution $u(1, x)$ versus x for $x \in [0, 1]$ and the numerical solution approximating it for the different stepsizes. On a separate graph, plot the associated errors (i.e., for each of the different stepsizes you used, plot the "exact" solution $u(1, x)$ minus the numerical solution approximating it versus x for $x \in [0, 1]$). Consider if it might be better to plot the log of the absolute value of the error rather than the error itself.

Try to estimate how the error in the numerical solution at $t = 1$ decreases as δt and δx decrease.

Does the rate of convergence you observe agree with the rate of convergence for the Fully Implicit Method we discussed in class?

Justify your answer.

What do these numerical results confirm about the stability of the Fully Implicit Method?

Hand in the program that you developed for this question, your plots, your analysis of the rate of convergence of the numerical method observed from the numerical results, your discussion of how this observed rate of convergence agrees with the theoretical rate of convergence of the Fully Implicit Method and your discussion of the stability of the Fully Implicit Method.

(d) Use the Crank-Nicolson Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (3). Also use the Crank-Nicolson Method to solve the heat equation (1) with the boundary conditions (2) and the initial condition (4).

Let $\rho = 1$ and solve the problem several times with different values of δt and δx satisfying $\delta t/\delta x = \rho$.

For each problem (1)–(2)–(3) and (1)–(2)–(4), plot the "exact" solution $u(1, x)$ versus x for $x \in [0, 1]$ and the numerical solution approximating it for the different stepsizes. On a separate graph, plot the associated errors (i.e., for each of the different stepsizes you used, plot the "exact" solution $u(1, x)$ minus the numerical solution approximating it versus x for $x \in [0, 1]$). Consider if it might be better to plot the log of the absolute value of the error rather than the error itself.

For each problem, try to estimate how the error in the numerical solution at $t = 1$ decreases as δt and δx decrease.

Does the rate of convergence you observe agree with the rate of convergence for the Crank-Nicolson Method we discussed in class?

Justify your answer.

What do these numerical results confirm about the stability of the Crank-Nicolson Method?

Hand in the program that you developed for this question, your plots, your analysis of the rate of convergence of the numerical method observed from the numerical

results, your discussion of how this observed rate of convergence agrees with the theoretical rate of convergence of the Crank-Nicolson Method and your discussion of the stability of the Crank-Nicolson Method.

(e) Repeat question 2(d), but use *Rannacher smoothing*. That is, for each numerical integration, take the first 5 time-steps with the Fully Implicit Method and then switch to the Crank-Nicolson Method for the remaining time-steps.

For each problem (1)–(2)–(3) and (1)–(2)–(4), plot the "exact" solution $u(1,x)$ versus x for $x \in [0,1]$ and the numerical solution approximating it for the different stepsizes. On a separate graph, plot the associated errors (i.e., for each of the different stepsizes you used, plot the "exact" solution $u(1,x)$ minus the numerical solution approximating it versus x for $x \in [0,1]$). Consider if it might be better to plot the log of the absolute value of the error rather than the error itself.

For each problem, try to estimate how the error in the numerical solution at $t = 1$ decreases as δt and δx decrease.

Does the rate of convergence you observe agree with the rate of convergence for the Crank-Nicolson Method we discussed in class?

Justify your answer.

Does Rannacher smoothing improve the numerical results for either or both of the problems (1)–(2)–(3) and (1)–(2)–(4)?

Can you think of a theoretical explanation of your observations to the question above?

Hand in the program that you developed for this question, your plots, your analysis of the rate of convergence of the numerical method observed from the numerical results, your discussion of how this observed rate of convergence agrees with the theoretical rate of convergence of the Crank-Nicolson Method and your discussion the effectiveness of Rannacher smoothing.

You can use the numerical solutions above to approximate the the hedging parameters $\partial u(1,x)/\partial x$ and $\partial^2 u(1,x)/\partial x^2$ by approximating these derivatives by finite differences. I had planned to ask you to do this too in this assignment, but I dropped this part of the assignment because the assignment seems to be quite big already. However, think about how you could use the numerical solutions computed above to approximate the hedging parameters $\partial u(1,x)/\partial x$ and $\partial^2 u(1,x)/\partial x^2$.