

MAE 306 Notes #4a

Engineering Mathematics II

S. H. Lam

March 1, 1999

1 Reading and Homework Assignments

Study §18.6, Numerical Solution, pp. 999-1016 of Michael Greenberg's *Advanced Engineering Mathematics*, and do the problem assigned below.

The problems are due on Tuesday, March 2nd, 1999, 5PM. Please submit your homework to the MAE 306 homework **IN** tray outside D-302, E.Q.

2 Comments on the Age of Computers

All of the materials presented so far were developed before the advent of modern computers. The mathematics developed were beautiful. And they work—provided all the conditions needed for the applicability of the methods are satisfied.

But this is the Brave New World of computers. How would one get solutions of diffusion PDE's when you have access to computers?

The answer is very simple. You *discretize* your domain of interest into closely spaced nodes (distinct points), replace all your derivatives (including time derivatives) by finite difference approximations. Then voila! You discover that what is staring back to you from your worksheet is an algebraic equation. A BIG algebraic equations. How many unknown? The same number as the number of nodes. So if you have 1,000,000 nodes, you have 1,000,000 unknowns.

If you have a linear PDE (or ODE, for that matter), you will be looking at a linear algebraic equation, consisting of a square matrix \mathbf{M} and a forcing

vector \mathbf{f} :

$$\mathbf{M} \cdot \mathbf{x} = \mathbf{f}. \quad (1)$$

The unknown is the vector \mathbf{x} . How do we solve this straightforward problem (assuming that \mathbf{M} is non-singular)? All non-numerical methods oriented people would say: find the inverse of \mathbf{M} , and the solution is:

$$\mathbf{x} = \mathbf{M}^{-1} \cdot \mathbf{f}. \quad (2)$$

There is absolutely nothing wrong with this statement—except that no self-respecting person who is numerical method oriented would ever be caught dead saying this. Finding and using the inverse of a big matrix is a NO NO!

If \mathbf{M} is a diagonal matrix, we all agree the solution is trivially quick. If \mathbf{f} consists of the diagonal plus an adjacent diagonal, we will agree after a bit of thinking that the solution is also trivially quick. If \mathbf{f} consists of the diagonal plus two adjacent diagonal, we have a so-called tri-diagonal matrix. After a lot of thinking (wait for the presentation in the lecture), you will agree that the tri-diagonal problem is also very quick.

The `tridag.m` file, downloadable from the course web page and included at the end of these Notes, is short and sweet. Use it in good cheer.

3 Explicit and Implicit Procedures

For the diffusion problem, one can use either the explicit procedure or the implicit procedure (Crank-Nicholson). In the world of real engineering, few people would admit that they use the explicit scheme. You will see that from the programming point of view, one is just as easy as the other—while the Crank-Nicholson scheme has many superior qualities.

4 Homework Assignment

There is only one problem assigned, but it is a programming assignment. Its time consuming (all programming assignments are), but its a lot of fun once you get your program going.

Consider the linear one-dimensional diffusion equaton:

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial y^2}. \quad (3)$$

The domain of interest is $t \geq 0$, and $0 \leq y \leq \ell$. The initial and boundary conditions are as follows:

$$\text{initial condition:} \quad u(0, y) = \sin\left(\frac{\pi y}{\ell}\right) \quad (4)$$

$$\text{boundary condition:} \quad u(t, 0) = L, \quad u(t, \ell) = R, \quad (5)$$

where L and R are constants.

Solve it numerically using the forward difference in time and central difference in y , using either Matlab or Mathematica or any software package you prefer.

We shall discretise the y -space into N nodes (only interior points are counted as nodes) Δy apart. The left boundary point is at $y = 0$, the right boundary point is at $y = (N + 1)\Delta y = \ell$.

$$u(t, y) = u(m, n); \quad t = m\Delta t; \quad y = n\Delta y \quad (6)$$

The initial and boundary conditions are now as follows: Initial condition

$$\text{initial condition:} \quad u(0, n) = \sin\left(\frac{n\pi}{N + 1}\right), \quad (7)$$

$$\text{initial condition:} \quad u(m, 0) = L, \quad u(N + 1, n) = R. \quad (8)$$

The original PDE, after the discretization, becomes a simple linear matrix equation for $u(m + 1, n)$ in terms of $u(m, n)$ and a single dimensionless parameter r defined by

$$r = \frac{\alpha^2 \Delta t}{(\Delta y)^2}. \quad (9)$$

Note that you do not need to know the actual value of α^2 or Δt or Δy to do the computation. However, you must pick a value for r .

Do the following cases:

- (a) $r = 1$, $N=49$, $L= 1$, $R=0$, and marches forward in time (one step at a time), take three or four steps. Look at the mess you get near the left boundary. Plot a graph to show the mess. (Note: $u(t = 0, y = 0) = 0$ while $u(t > 0, y = 0) = 1$).
- (b) Repeat (a), but use $L = 0$ (You know the exact analytical solution for this case.). Do it for forty or fifty steps, using a for-loop. Plot a graph to show the mess. What you have done is to show that the failure of the computation is not caused by the discontinuity of $u(t, 0)$ at $t = 0$.

- (c) Repeat (a) and (b), but use $r = 0.5$ (reduce the value of Δt used). Plot the results for $L = 0$ at $m=100$ cycles. All should be well here. (Observation: The explicit scheme is stable only if $r \leq 0.5$)
- (d) Use $r = 0.25$ (halving Δt of (c)) for $L = 0$. To reach the same actual t as in (c) after 100 cycles, we need to compute to $m=200$ cycles. Compare this result with the result obtained in (c)—how do they compare with each other? How much more work the computer has to do if you decide to have twice the resolution (reduce the spatial grid size by a factor of two) and cover the same span of time?
- (e) Repeat either (c) or (d) with $L = 1$ for 10 or 20 cycles. Plot result to show that the calculation is stable and the results look good!
-

Think about the following:

- What are the complications if L and R are functions of time?
- What are the complications if there is a source term in the original PDE?
- What are the complications if there is a convective term in the original PDE?
- What are the complications if you want to use the implicit Crank-Nicholson scheme (see Greenberg) rather than the explicit scheme? (I hope you know the benefits ...).
- What would you do if \mathbf{M} and \mathbf{f} have some dependence on \mathbf{x} —the problem is nonlinear? (Hint: think about the word “iteration.”)

Your thoughts on the above questions are optional as far as homework is concerned.

The Tridag Subroutine for Matlab

A downloadable m-file of this subroutine is available on the web
(<http://www.princeton.edu/~lam/course/tridag.m>)
which is reproduced below (some minor editing was done).

```
% Numerical Recipes, Press, Flannery, Teukolsky and Vetterling,  
% Cambridge University Press (1986), section 2.6, page 40.  
%  
function U=tridag(a,b,c,f);  
%  
% To solve  $\mathbf{M} \cdot \mathbf{x}=\mathbf{f}$  where  $\mathbf{M}$  is a  $n \times n$  tri-diagonal matrix.  
% To call it, the vectors a, b, c, and f are passed.  
% The diagonal elements is b, the left elements a, the right elements is c.  
% All vectors are  $n$ -dimensional. a(1) and c(n) are not called.  
% The  $n$ -dimensional vector U is returned.  
%  
% Checking to see if the calling protocol is followed:  
%  
if nargin< 4  
    disp('needs a, b, c and f as inputs.')end  
n=length(b);  
if length(a)==n-1  
    a=[0 a];  
else if length(a)==n  
    a(1)=0;  
end  
if length(f) ~= n  
    disp('forcing vector must have same dimesion as matrix.')end  
if length(c) < n-1  
    disp('c vector too short.')end  
bet=b(1);  
if bet==0  
    disp('b1 cannot be zero ...');  
    bet=eps;  
end
```

```

%
% The rest is from Numerical Recipes
%
U(1)=f(1)/bet;
for j=2:n
    gam(j)=c(j-1)/bet;
    bet=b(j)-a(j)*gam(j);
    if bet==0
        disp('... probably need pivoting.');
```

$$\text{bet} = \text{eps};$$

```

    end
    U(j)=(f(j)-a(j)*U(j-1))/bet;
end
for j=n-1:-1:1
    U(j)=U(j)-gam(j+1)*U(j+1);
end;
```