MATH 517 Finite Differences Homework 6

Caleb Logemann

April 5, 2016

1. Consider the following method for solving the heat equation $u_t = u_{xx}$:

$$U_i^{n+2} = U_i^n + \frac{2k}{h^2}\left(U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}\right)$$

(a) Determine the order of accuracy of this method (in both space and time).

In order to find the order of accuracy of this method, the local truncation error for this method must be found. The local truncation error for this method is given as

$$\tau = \frac{1}{2k}(u(t+k,x) - u(t-k,x)) - \frac{1}{h^2}(u(t,x-h) - 2u(t,x) + u(t,x+h))$$

I will consider the time and space terms seperately at first

$$u(t+k,x) = u(t,x) + ku_t(t,x) + \frac{1}{2}k^2u_{tt}(t,x) + \frac{1}{6}k^3u_{ttt}(t,x) + O(k^4)$$
$$u(t-k,x) = u(t,x) - ku_t(t,x) + \frac{1}{2}k^2u_{tt}(t,x) - \frac{1}{6}k^3u_{ttt}(t,x) + O(k^4)$$
$$\frac{1}{2k}(u(t+k,x) - u(t-k,x)) = \frac{1}{2k}\left(2ku_t(t,x) + \frac{1}{3}k^3u_{ttt}(t,x) + O(k^4)\right)$$
$$\frac{1}{2k}(u(t+k,x) - u(t-k,x)) = u_t(t,x) + \frac{1}{6}k^2u_{ttt}(t,x) + O(k^3)$$
$$u(t,x-h) = u(t,x) - hu_x(t,x) + \frac{1}{2}h^2u_{xx}(t,x) - \frac{1}{6}h^3u_{xxx}(t,x) + \frac{1}{24}h^4u_{xxxx}(t,x) + O(h^5)$$
$$u(t,x+h) = u(t,x) + hu_x(t,x) + \frac{1}{2}h^2u_{xx}(t,x) + \frac{1}{6}h^3u_{xxx}(t,x) + \frac{1}{24}h^4u_{xxxx}(t,x) + O(h^5)$$
$$\frac{1}{h^2}(u(t,x-h) - 2u(t,x) + u(t,x+h)) = \frac{1}{h^2}\left(h^2u_{xx}(t,x) + \frac{1}{12}h^4u_{xxxx} + O(h^6)\right)$$
$$\frac{1}{h^2}(u(t,x-h) - 2u(t,x) + u(t,x+h)) = u_{xx}(t,x) + \frac{1}{12}h^2u_{xxxx} + O(h^4)$$

Note that in the spacial terms the 5th order error canceled out.

Using these two expressions the full truncation error is

$$\tau = u_t(t,x) + \frac{1}{6}k^2u_{ttt}(t,x) + O(k^3) - u_{xx}(t,x) - \frac{1}{12}h^2u_{xxxx} + O(h^4)$$

The heat equation states that $u_t(t,x) = u_{xx}(t,x)$

$$\tau = \frac{1}{6}k^2u_{ttt}(t,x) + O(k^3) - u_{xx}(t,x) - \frac{1}{12}h^2u_{xxxx} + O(h^4)$$
$$\tau = O(k^2 + h^2)$$

(b) Suppose we take $k = \alpha h^2$ for some fixed $\alpha > 0$ and refine the grid. For what values of α will this method be Lax-Richtmyer stable and hence convergent?

(c) Is this method useful?

2. Consider the one-dimensional heat equation:

$$PDE : u_t = \kappa u_{xx} \text{ for } (t,x) \in [0,T] \times [0,1]$$
$$BCs : u(t,0) = g_0,\ u(t,1) = g_1$$
$$ICs : u(0,x) = f(x)$$

(a) The m-file heat_CN.m solves the heat equation $u_t = \kappa u_{xx}$ using the Crank-Nicolson method. Run this code and by changing the number of grid points, confirm that it is second order accurate. Create a table of errors for varius h values, and $k = 4h$ at $T = 1$.

The following script creates a table that verifies that the Crank-Nicolson is second order accurate.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create test function
u = @(x) exp(x);
% Create exact second derivative
u2d = @(x) exp(x);

% randomly generate h
sz = [500,1];
H = exp(linspace(-7,2,500)');
h1 = H.*rand(sz);
h2 = H.*rand(sz);
h3 = H.*rand(sz);

% create x values from h values
x1 = -1*h1;
x2 = 0*H;
x3 = h2;
x4 = h2 + h3;

% compute weights
w1 = (2*(2*h2 + h3))./(h1.*(h1 + h2).*(h1 + h2 + h3));
w2 = (2*h1 - 4*h2 - 2*h3)./(h1.*h2.^2 + h1.*h2.*h3);
w3 = (2*(-1*h1 + h2 + h3))./(h2.*(h1 + h2).*h3);
w4 = (2*(h1 - h2))./(h3.*(h2 + h3).*(h1 + h2 + h3));

% approximate second derivative
u2da = w1.*u(x1) + w2.*u(x2) + w3.*u(x3) + w4.*u(x4);

% compute error
E = abs(u2d(x2) - u2da);
loglog(H, E, 'ko');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% do a least squares fit on the error in terms of H
A = [ones(sz), log(H)];
b = log(E);

% least squares fit Ax = b
lsf = (A'*A)\(A'*b);
lsf2 = A\b;
K = lsf(1);
p = lsf(2);
hold on
x = min(H):.01:max(H);
y = exp(K)*x.^p;
loglog(x,y,'r', 'LineWidth', 3);
set(gca, 'FontSize', 16);
ylabel('Error');
xlabel('H');
```

```
title('H vs Error')

p
K
```

```
ans =

    hRatios    errorRatios    order
    -------    -----------    ------

    2          8.1811         3.0323
    2          4.4039         2.1388
    2           3.999         1.9996
    2          3.9953         1.9983
    2               4              2
    2               4              2
    2               4              2
```

(b)

3. Again, consider the 1D heat equation.

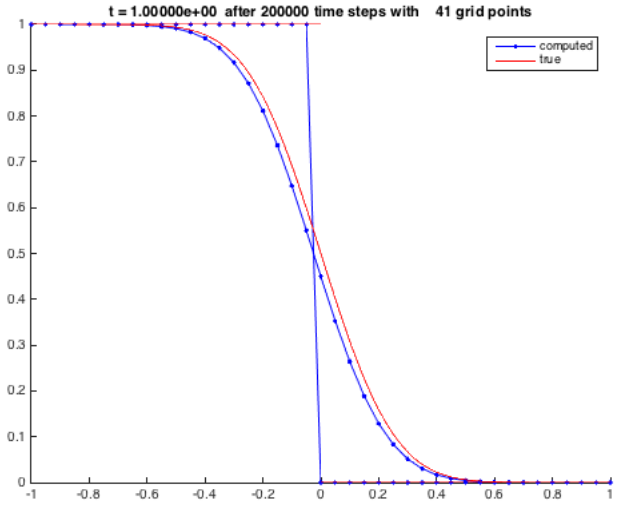
   (a) The following function uses the new

```
function [h,k,err] = heat_CN2(m)
%
% heat_CN.m
%
% Solve u_t = kappa * u_{xx} on [ax,bx] with Dirichlet boundary conditions,
% using the Crank-Nicolson method with m interior points.
%
% Returns k, h, and the max-norm of the error.
% This routine can be embedded in a loop on m to test the accuracy,
% perhaps with calls to error_table and/or error_loglog.
%
% From  http://www.amath.washington.edu/¬rjl/fdmbook/  (2007)

clf               % clear graphics
hold on           % Put all plots on the same graph (comment out if desired)

ax = -1;
bx = 1;
kappa = .02;                % heat conduction coefficient:
tfinal = 1;                 % final time

h = (bx-ax)/(m+1);          % h = Δ x
x = linspace(ax,bx,m+2)';   % note x(1)=0 and x(m+2)=1
                            % u(1)=g0 and u(m+2)=g1 are known from BC's
k = .0001*h;                   % time step

nsteps = round(tfinal / k);     % number of time steps
%nplot = 1;       % plot solution every nplot time steps
                  % (set nplot=2 to plot every 2 time steps, etc.)
nplot = nsteps;   % only plot at final time

if abs(k*nsteps - tfinal) > 1e-5
```

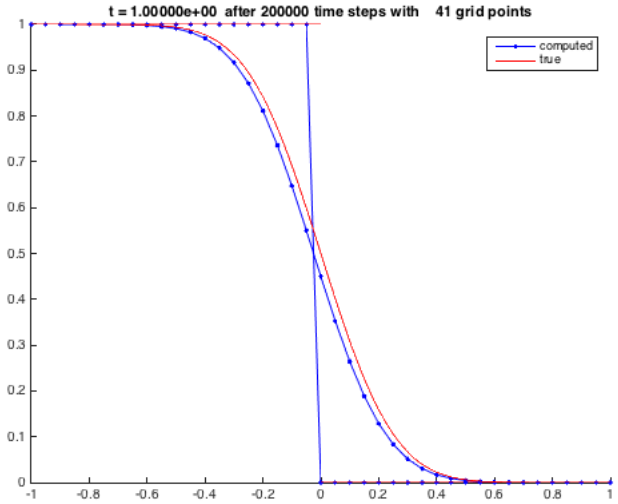
```
   % The last step won't go exactly to tfinal.
   disp(' ')
   disp(sprintf('WARNING *** k does not divide tfinal, k = %9.5e',k))
   disp(' ')
   end


% true solution for comparison:
% For Gaussian initial conditions u(x,0) = exp(-beta * (x-0.4)^2)
utrue = @(x,t) 1/2*erfc(x/sqrt(4*kappa*t));

% initial conditions:
u0 = x < 0;


% Each time step we solve MOL system U' = AU + g using the Trapezoidal method

% set up matrices:
r = (1/2) * kappa* k/(h^2);
e = ones(m,1);
A = spdiags([e -2*e e], [-1 0 1], m, m);
A1 = eye(m) - r * A;
A2 = eye(m) + r * A;


% initial data on fine grid for plotting:
xfine = linspace(ax,bx,1001);
ufine = utrue(xfine,0);

% initialize u and plot:
tn = 0;
u = u0;

plot(x,u,'b.-', xfine,ufine,'r')
legend('computed','true')
title('Initial data at time = 0')

input('Hit <return> to continue  ');


% main time-stepping loop:

for n = 1:nsteps
     tnp = tn + k;    % = t_{n+1}

     % boundary values u(0,t) and u(1,t) at times tn and tnp:

     g0n = u(1);
     g1n = u(m+2);
     g0np = utrue(ax,tnp);
     g1np = utrue(bx,tnp);

     % compute right hand side for linear system:
     uint = u(2:(m+1));    % interior points (unknowns)
     rhs = A2*uint;
     % fix-up right hand side using BC's (i.e. add vector g to A2*uint)
     rhs(1) = rhs(1) + r*(g0n + g0np);
     rhs(m) = rhs(m) + r*(g1n + g1np);

     % solve linear system:
     uint = A1\rhs;
```

```
% augment with boundary values:
u = [g0np; uint; g1np];

% plot results at desired times:
if mod(n,nplot)==0 | n==nsteps
   ufine = utrue(xfine,tnp);
   plot(x,u,'b.-', xfine,ufine,'r')
   title(sprintf('t = %9.5e  after %4i time steps with %5i grid points',...
                  tnp,n,m+2))
   err = max(abs(u-utrue(x,tnp)));
   disp(sprintf('at time t = %9.5e  max error =  %9.5e',tnp,err))
   if n<nsteps, input('Hit <return> to continue  '); end;
   end

tn = tnp;    % for next time step
end
```

This graph is after one time step, and shows that the decay of the high wave numbers is not captured well.



(b) When m is odd the accuracy is severely limited because the initial data is not symmetrical. As is evident in the image below, the blue initial data does not capture the discontinuity well. At $x = 0$, the discretization is 0, however the point directly to the left of $x = 0$ is skewing the initial data to the left. This causes the approximate solution to always be less than the true solution.



4. Consider the following scheme for solving the 1D heat equation:

$$U_j^{n+1} = U_j^{n-1} + \frac{2k}{h^2}\Big(U_{j+1}^n - \Big(U_j^{n+1} + U_j^{n-1}\Big) + U_{j-1}^n\Big)$$

(a) Determine the local truncation error of this scheme.

The local truncation error for this scheme is given by

$$\tau = \frac{1}{2k}(u(t+k,x) - u(t-k,x)) - \frac{1}{h^2}(u(t,x+h) - (u(t+k,x) + u(t-k,x)) + u(t,x-h))$$

We have shown previously in problem 1(a) that

$$\frac{1}{2k}(u(t+k,x) - u(t-k,x)) = u_t(t,x) + \frac{1}{6}k^2 u_{ttt}(t,x) + O(k^3).$$

We can now consider the second term

$$\begin{aligned}
u(t,x-h) &= u(t,x) - hu_x(t,x) + \frac{1}{2}h^2u_{xx}(t,x) - \frac{1}{6}h^3u_{xxx}(t,x) + \frac{1}{24}h^4u_{xxxx}(t,x) + O(h^5)\\
u(t,x+h) &= u(t,x) + hu_x(t,x) + \frac{1}{2}h^2u_{xx}(t,x) + \frac{1}{6}h^3u_{xxx}(t,x) + \frac{1}{24}h^4u_{xxxx}(t,x) + O(h^5)\\
u(t+k,x) &= u(t,x) + ku_t(t,x) + \frac{1}{2}k^2u_{tt}(t,x) + \frac{1}{6}k^3u_{ttt}(t,x) + O(k^4)\\
u(t-k,x) &= u(t,x) - ku_t(t,x) + \frac{1}{2}k^2u_{tt}(t,x) - \frac{1}{6}k^3u_{ttt}(t,x) + O(k^4)
\end{aligned}$$

$$\begin{aligned}
&\frac{1}{h^2}(u(t,x+h) - (u(t+k,x) + u(t-k,x)) + u(t,x-h))\\
&\quad= \frac{1}{h^2}\Big(h^2u_{xx} - k^2u_{tt} + \frac{1}{12}h^4u_{xxxx} + O(h^6) + O(k^4)\Big)\\
&\quad= u_{xx} + \Big(\frac{1}{12}h^2 - \frac{k^2}{h^2}\Big)u_{xxxx} + O(h^4) + O(\frac{k^4}{h^2})
\end{aligned}$$

The actual truncation error is therefore

$$\tau = u_t(t,x) + \frac{1}{6}k^2 u_{ttt}(t,x) + O(k^3) - u_{xx} - \left(\frac{1}{12}h^2 - \frac{k^2}{h^2}\right)u_{xxxx} + O(h^4) + O(\frac{k^4}{h^2})$$

$$\tau = \frac{1}{6}k^2 u_{ttt}(t,x) - \left(\frac{1}{12}h^2 - \frac{k^2}{h^2}\right)u_{xxxx} + O(h^4) + O(k^3) + O(\frac{k^4}{h^2})$$

(b)