MATH 517 Finite Differences Homework 6

Caleb Logemann

April 7, 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?

This method will be Lax-Richtmyer stable if the eigenvalues of the matrix for B are inside the region of absolute stability for the time stepping method. Equivalently this method will be Lax-Richtmyer stable if $\|B(k)^n\| \leq C_T$ for all $k > 0$ and integers n such that $kn \leq T$. For this method

$$B = I + 2kD_x^2$$

In this method we are using the Leap-Frog time stepping method, and the 2nd order central finite difference for the spatial derivative.

The eigenvalues for the spatial derivative are

$$\lambda_p = \frac{2}{h^2}(\cos(p\pi h) - 1).$$

The right hand side matrix is $B = I + 2kD_x^2$, so the eigenvalues of the right hand side are

$$\lambda_p = 1 + \frac{4k}{h^2}(\cos(p\pi h) - 1)$$

Since $k = \alpha h^2$

$$\lambda_p = 1 + 4\alpha\cos(p\pi h) - 4\alpha$$

Theerefore the maximum eigenvalue or spectral radius of B is

$$\rho(B) = \lambda_1 = 1 + 4\alpha\cos(\pi h) - 4\alpha$$

Therefore for Lax-Richtmyer stability

$$\begin{aligned}\|B(k)^n\| &= \rho(B)^n \\ &= (1 + 4\alpha\cos(\pi h) - 4\alpha)^n \\ &= (1 + 4(\cos(\pi h) - 1)\alpha)^n\end{aligned}$$

If $\alpha = \beta k$, then

$$\begin{aligned}&= (1 + 4(\cos(\pi h) - 1)\beta k)^n \\ &= (1 + \gamma k)^n\end{aligned}$$

where $\gamma = 4(\cos(\pi h) - 1)\beta$. Therefore if $\alpha = O(k) = O(h^2)$, then this method is Lax-Richtmyer stable.

(c) Is this method useful?

This method isn't useful because the time step, k, has to scale with h^4, which means that the time step must be extremely small to guarantee stability. This makes the method extremely inefficient.

2. Consider the one-dimensional heat equation:

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

(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 (a)
    H = [];
    E = [];
    for m = [11, 23, 47, 95, 191, 383]
        [h, k, err] = heat_CN(m, 4);
        H = [H; h];
        E = [E; err];
    end
    hRatios = H(1:end-1)./H(2:end);
    errorRatios = E(1:end-1)./E(2:end);
    order = log(errorRatios)./log(hRatios);
    table(hRatios, errorRatios, order)
```

```
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) The follwing method implements the TR-BDF2 method.

```
function [h,k,err] = heat_trbdf2(m, a)
%
% heat_trbdf2.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 = 0;
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 = a*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)
beta = 150;
utrue = @(x,t) exp(-(x-0.4).^2 / (4*kappa*t + 1/beta)) / sqrt(4*beta*kappa*t+1);
```

```
% initial conditions:
u0 = utrue(x,0);

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

% set up matrices:
r1 = kappa*k/(4*h^2);
r2 = kappa*k/(3*h^2);
e = ones(m,1);
Dx2 = spdiags([e -2*e e], [-1 0 1], m, m);
A1 = eye(m) - r1 * Dx2;
A2 = eye(m) + r1 * Dx2;
A3 = eye(m) - r2 * Dx2;

% 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, tn + k/2, and tnp:
     g0n = u(1);
     g1n = u(m+2);
     g0npHalf = utrue(ax,tn+k/2);
     g1npHalf = utrue(bx,tn+k/2);
     g0np = utrue(ax,tnp);
     g1np = utrue(bx,tnp);

     % compute right hand side for linear system:
     uint = u(2:(m+1));    % interior points (unknowns)

     % first stage
     rhs = A2*uint;
     % fix-up right hand side using BC's (i.e. add vector g to A2*uint)
     rhs(1) = rhs(1) + r1*(g0n + g0npHalf);
     rhs(m) = rhs(m) + r1*(g1n + g1npHalf);

     % solve linear system for first stage:
     ustar = A1\rhs;

     % second stage
     rhs = 1/3 * (4*ustar - uint);
     rhs(1) = rhs(1) + r2 * g0np;
     rhs(m) = rhs(m) + r2 * g1np;

     % solve linear system for second stage
     uint = A3\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
```

The order of accuracy is checked with this script.

```
%% Problem 2 (b)
    H = [];
    E = [];
    for m = [11, 23, 47, 95, 191, 383]
        [h, k, err] = heat_trbdf2(m, 4);
        H = [H; h];
        E = [E; err];
    end
    hRatios = H(1:end-1)./H(2:end);
    errorRatios = E(1:end-1)./E(2:end);
    order = log(errorRatios)./log(hRatios);
    table(hRatios, errorRatios, order)
```

The method is second order accurate as is shown below.

```
ans =

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

    2          4.0302         2.0109
    2          3.9812         1.9932
    2          3.9898         1.9963
    2           3.986         1.9949
    2          3.9943         1.9979
```

The boundary conditions were chosen based on the formulation of the TR-BDF2 method in section 8.5 of the book. For the first stage the boundaries of U^n are given by $g(t = t^n, x = 0, 1)$. The boundaries of U^* are given by $g(t = t^n + k/2, x = 0, 1)$, because U^* is an approximation of the solution at time $t = t^n + k/2$. Finally the boundaries of U^{n+1} are given by $g(t = t^n + k = t^{n+1}, x = 0, 1)$

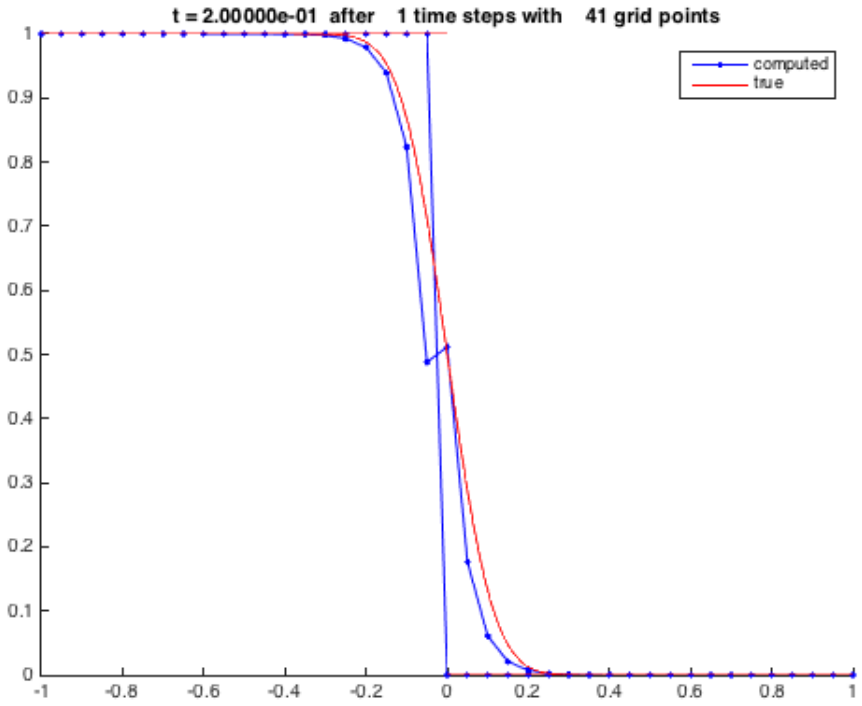
3. Again, consider the 1D heat equation.

   (a) (i) The following function uses the new

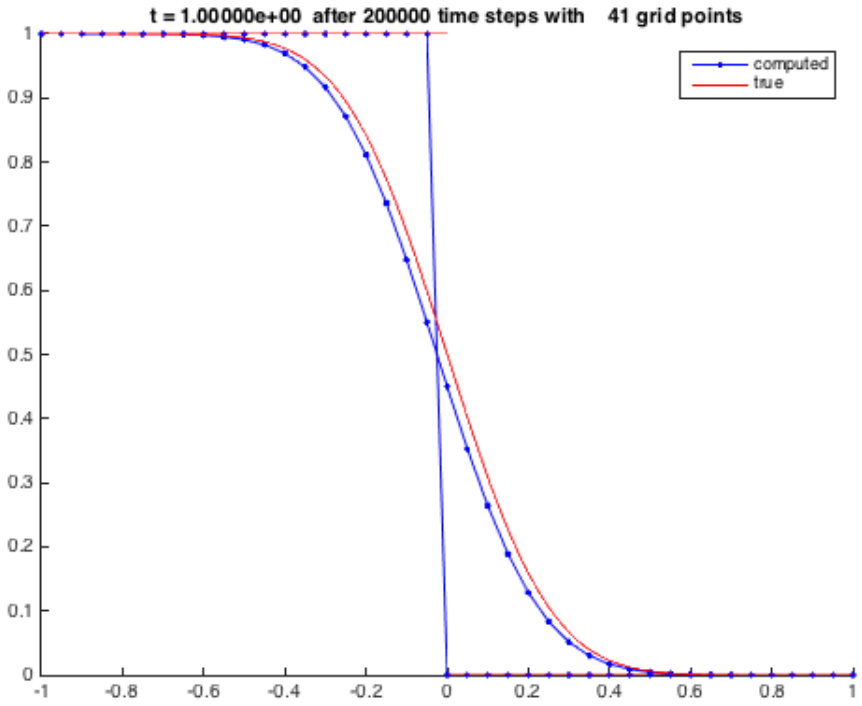
```
function [h,k,err] = heat_CN3(m, a)
%
% heat_CN.m
%
% Solve u_t = kappa * u_{xx} on [ax,bx] with Dirichlet boundary conditions,
% using the Crank-Nicolson method with m interior points.
%
% Returns k, h, and the max-norm of the error.
% This routine can be embedded in a loop on m to test the accuracy,
% 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 = a*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:
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
```

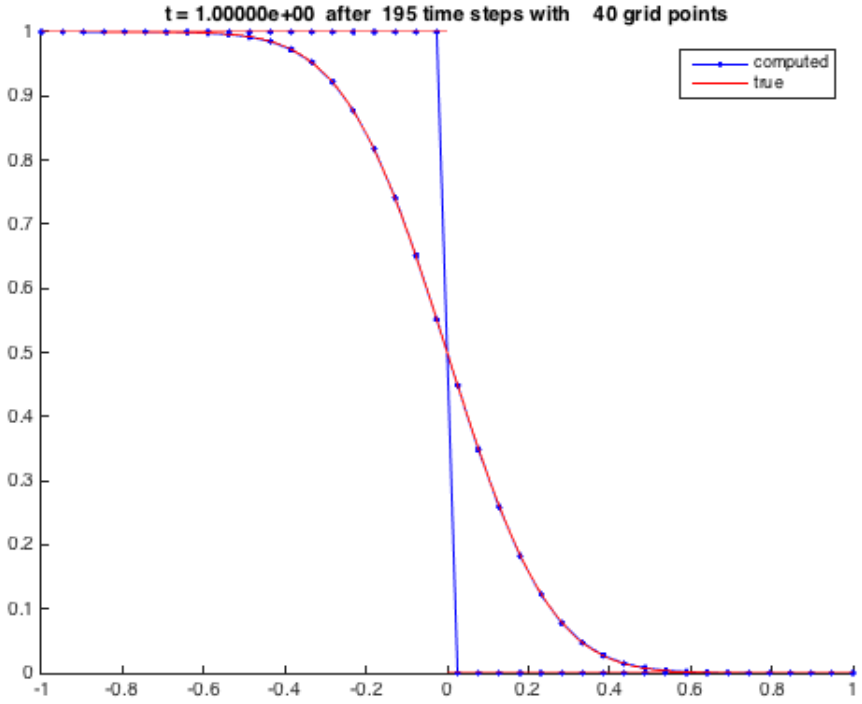
This graph is after one time step, and shows that the decay of the high wave numbers is not captured well.



(ii) If $k = .1h$ instead this high frequency is damped out well, so that the high frequency modes present in part (i) are no longer a problem.
However there is a problem with the accuracy when m is odd. 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.



If m is even, then initial data become symmetrical around $x = 0$, because there is no discritization point at $x = 0$. This result in the following image.



The order of accuracy can be checked to be second order as well with the following script.

```
%% Problem 3 (a)
    H = [];
```

```
E = [];
for m = 38*2.^(0:4)
    [h, k, err] = heat_CN3(m, .1);
    H = [H; h];
    E = [E; err];
end
hRatios = H(1:end-1)./H(2:end);
errorRatios = E(1:end-1)./E(2:end);
order = log(errorRatios)./log(hRatios);
table(hRatios, errorRatios, order)
```

```
ans =

    hRatios    errorRatios    order
    _______    ___________    ______

    1.9744     3.8904          1.9971
    1.987      3.949          2.0003
    1.9935     3.9746          2.0003
    1.9967     3.9872          2.0001
```

(b) The modified version of the trbdf2 method for solving this problem is shown below.

```
function [h,k,err] = heat_trbdf23(m, a)
%
% heat_trbdf2.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 = a*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:
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:
r1 = kappa*k/(4*h^2);
r2 = kappa*k/(3*h^2);
e = ones(m,1);
Dx2 = spdiags([e -2*e e], [-1 0 1], m, m);
A1 = eye(m) - r1 * Dx2;
A2 = eye(m) + r1 * Dx2;
A3 = eye(m) - r2 * Dx2;

% 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, tn + k/2, and tnp:
     g0n = u(1);
     g1n = u(m+2);
     g0npHalf = utrue(ax,tn+k/2);
     g1npHalf = utrue(bx,tn+k/2);
     g0np = utrue(ax,tnp);
     g1np = utrue(bx,tnp);

     % compute right hand side for linear system:
     uint = u(2:(m+1));   % interior points (unknowns)

     % first stage
     rhs = A2*uint;
     % fix-up right hand side using BC's (i.e. add vector g to A2*uint)
     rhs(1) = rhs(1) + r1*(g0n + g0npHalf);
     rhs(m) = rhs(m) + r1*(g1n + g1npHalf);

     % solve linear system for first stage:
     ustar = A1\rhs;

     % second stage
     rhs = 1/3 * (4*ustar - uint);
     rhs(1) = rhs(1) + r2 * g0np;
```

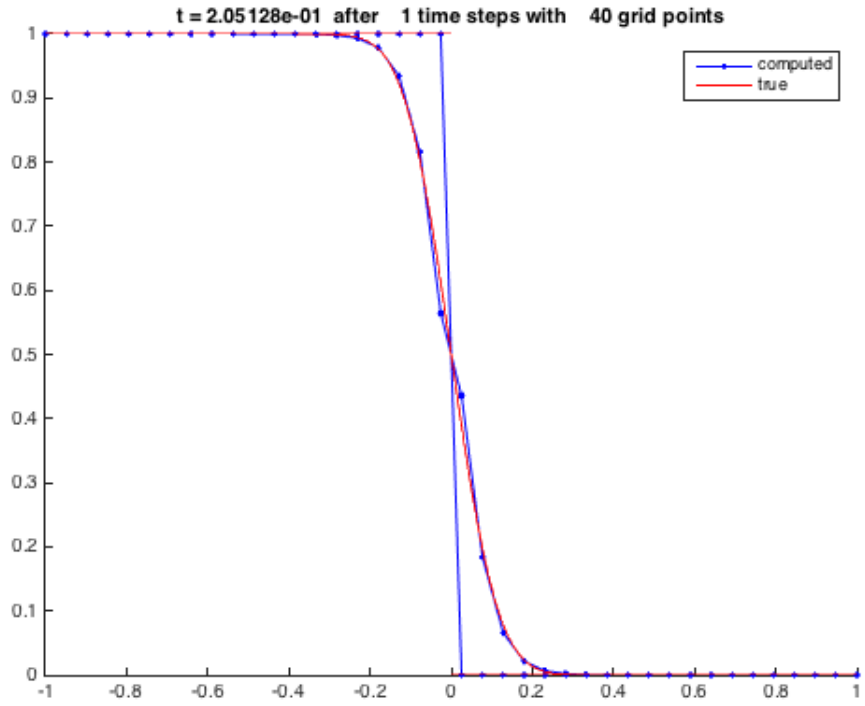
```
     rhs(m) = rhs(m) + r2 * g1np;

     % solve linear system for second stage
     uint = A3\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
```

With $k = 4h$ the order is shown below.

```
%% Problem 3 (b)
    H = [];
    E = [];
    for m = 38*2.^(0:4)
        [h, k, err] = heat_trbdf23(m, 4);
        H = [H; h];
        E = [E; err];
    end
    hRatios = H(1:end-1)./H(2:end);
    errorRatios = E(1:end-1)./E(2:end);
    order = log(errorRatios)./log(hRatios);
    table(hRatios, errorRatios, order)
```

```
ans =

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

    1.9744      3.9894          2.034
    1.987      3.8007         1.9445
    1.9935      3.9826         2.0032
    1.9967      3.9902         2.0012
```

This method is better than the Crank-Nicolson for this problem, because this method is L-stable. This method damps out high frequencies caused by the discontinuity much better than the CN method. This allows for the timestep to be much larger, while still maintaining the same accuracy. Since the timestep is larger, this method is much faster. The damping of high frequency modes can be seen by the following plot.



The equivalent to this plot is shown in 3(a)(i). The high frequency is much less evident for this method, then the CN method.

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

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

(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

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

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

The actual truncation error is therefore

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

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

(b)