

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

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

- a. Determine the order of accuracy of this method (in both space and time). b. Suppose we take  $k = \alpha h^2$  for some fixed  $\alpha > 0$  and refine the grid. For what values of  $\alpha$  (if any) will this method be Lax-Richtmyer stable and hence convergent? Hint: Consider the MOL interpretation and the stability region of the time-discretization being used.
- c. Is this a useful method?

*Solution.*

- a. We will need to make use of Taylor series expansions. Note that the local truncation error comes out to be

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

Now, we will expand  $u(x, t + 2k)$ ,  $u(x - h, t + k)$ ,  $u(x, t + k)$ , and  $u(x + h, t + k)$ . Observe that

$$\begin{aligned} u(x, t + 2k) &= u(x, t) + 2ku_t + 2k^2u_{tt} + \frac{8}{6}k^3u_{ttt} + \mathcal{O}(k^4) \\ u(x - h, t + 2k) &= u(x, t) - hu_x + ku_t + \frac{h^2}{2}u_{xx} - hku_{xt} + \frac{k^2}{2}u_{tt} - \frac{h^3}{6}u_{xxx} \\ &\quad + \frac{1}{2}h^2ku_{xxt} - \frac{1}{2}hk^2u_{xtt} + \frac{k^3}{6}u_{ttt} \\ u(x, t + k) &= u(x, t) + ku_t + \frac{1}{2}k^2u_{tt} + \frac{1}{6}k^3u_{ttt} + \mathcal{O}(k^4) \end{aligned}$$

and

$$\begin{aligned} u(x + h, t + 2k) &= u(x, t) + hu_x + ku_t + \frac{h^2}{2}u_{xx} + hku_{xt} + \frac{k^2}{2}u_{tt} + \frac{h^3}{6}u_{xxx} \\ &\quad + \frac{1}{2}h^2ku_{xxt} + \frac{1}{2}hk^2u_{xtt} + \frac{k^3}{6}u_{ttt} \end{aligned}$$

Hence,

$$\begin{aligned} \tau &= u_t + ku_{tt} + \frac{2}{3}k^2u_{ttt} + \mathcal{O}(k^3) - \frac{1}{h^2}(h^2u_{xx} + h^2ku_{xxt} + \mathcal{O}(k^4) + \mathcal{O}(h^4)) \\ &= \frac{2}{3}k^2u_{ttt} + \mathcal{O}(h^2) + \mathcal{O}(k^3) \end{aligned}$$

So, we see that the above method is second order accurate in both time and space.

b. Using the MOL, we get that  $U'_i = \frac{1}{h^2}(U_{i-1} - 2U_i + U_{i+1})$ . From this, we can get to  $U^{n+1} = AU^n + \frac{k}{h^2}G^n$  by discretizing time using the trapazoidal method where

$$A = I + \frac{k}{h^2} \begin{pmatrix} -2 & 1 & 0 & \cdots \\ 1 & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 \end{pmatrix}$$

Using the spectral norm of  $A$ , we get that the above method is LR stable if and only if  $|1 + \frac{k}{h^2}\lambda_i| \leq 1$  where  $\lambda_i$  is the  $i^{th}$  eigenvalue of  $A$ . Since the eigenvalues of  $A$  are  $\lambda_i = 2(\cos(i\pi h) - 1)$  for  $1 \leq i \leq m$ , we have that the method is LR stable if and only if  $2\frac{k}{h^2} \leq 1$ . Since  $k = \alpha h^2$ , we need  $2\alpha \leq 1$  or  $\alpha \leq \frac{1}{2}$ .

c. Since we need  $\alpha \leq \frac{1}{2}$ , this method is limited in its use.

- 9.2 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. (Observe how the error at some fixed time such as  $T = 1$  behaves as  $k$  and  $h$  go to zero with a fixed relation between  $k$  and  $h$ , such as  $k = 4h$ .) You might want to use the function `error_table.m` to print out this table and estimate the order of accuracy, and `error_loglog.m` to produce a log-log plot of the error vs.  $h$ . See `bvp_2.m` for an example of how these are used.
- b. Modify `heat_CN.m` to produce a new m-file `heat_trbdf2.m` that implements the TR-BDF2 method on the same problem. Test it to confirm that it is also second order accurate. Explain how you determined the proper boundary conditions in each stage of this Runge-Kutta method.

*Solution.*

- a. We can see from the output that this method is indeed second order accurate. See code below.
- b. See code below.

```

1 clear
2 close all
3 clc
4 for i=1:15
5 M=10*i+9;
6 heat_CN(M);
7 end
8 function [h,k,error] = heat_CN(m)
9 %
10 % heat_CN.m
11 %
12 % Solve  $u_t = \kappa u_{xx}$  on  $[a,b]$  with Dirichlet boundary ...
    conditions,
13 % using the Crank-Nicolson method with m interior points.
14 %
15 % Returns k, h, and the max-norm of the error.
16 % This routine can be embedded in a loop on m to test the accuracy,
17 % perhaps with calls to error_table and/or error_loglog.
18 %
19 % From http://www.amath.washington.edu/~rjl/fdmbook/ (2007)
20
21 clf % clear graphics
22 hold on % Put all plots on the same graph (comment out if ...
    desired)
23
24 ax = 0;
25 bx = 1;
26 kappa = .02; % heat conduction coefficient:
27 tfinal = 1; % final time
28
29 h = (bx-ax)/(m+1); % h =  $\Delta x$ 
30 x = linspace(ax,bx,m+2)'; % note x(1)=0 and x(m+2)=1
31 % u(1)=g0 and u(m+2)=g1 are known from BC's
32 k = 4*h; % time step
33
34 nsteps = round(tfinal / k); % number of time steps
35 nplot = 1; % plot solution every nplot time steps
36 % (set nplot=2 to plot every 2 time steps, etc.)
37 nplot = nsteps; % only plot at final time
38
39
40 % true solution for comparison:
41 % For Gaussian initial conditions  $u(x,0) = \exp(-\beta (x-0.4)^2)$ 
42 beta = 150;
43 utrue = @(x,t) exp(-(x-0.4).^2 / (4*kappa*t + 1/beta)) / ...
    sqrt(4*beta*kappa*t+1);
44
45 % initial conditions:
46 u0 = utrue(x,0);
47
48
49 % Each time step we solve MOL system  $U' = AU + g$  using the ...
    Trapezoidal method

```

```

50
51 % set up matrices:
52 r = (1/2) * kappa * k / (h^2);
53 e = ones(m,1);
54 A = spdiags([e -2*e e], [-1 0 1], m, m);
55 A1 = eye(m) - r * A;
56 A2 = eye(m) + r * A;
57
58
59 % initial data on fine grid for plotting:
60 xfine = linspace(ax,bx,1001);
61 ufine = utrue(xfine,0);
62
63 % initialize u and plot:
64 tn = 0;
65 u = u0;
66
67 plot(x,u,'b.-', xfine,ufine,'r')
68 legend('computed','true')
69 title('Initial data at time = 0')
70
71 %input('Hit <return> to continue ');
72
73
74 % main time-stepping loop:
75
76 for n = 1:nsteps
77     tnp = tn + k;    % = t_{n+1}
78
79     % boundary values u(0,t) and u(1,t) at times tn and tnp:
80
81     g0n = u(1);
82     gln = u(m+2);
83     g0np = utrue(ax,tnp);
84     glnp = utrue(bx,tnp);
85
86     % compute right hand side for linear system:
87     uint = u(2:(m+1));    % interior points (unknowns)
88     rhs = A2*uint;
89     % fix-up right hand side using BC's (i.e. add vector g to ...
        A2*uint)
90     rhs(1) = rhs(1) + r*(g0n + g0np);
91     rhs(m) = rhs(m) + r*(gln + glnp);
92
93     % solve linear system:
94     uint = A1\rhs;
95
96     % augment with boundary values:
97     u = [g0np; uint; glnp];
98
99     % plot results at desired times:
100    if mod(n,nplot)==0 || n==nsteps
101        ufine = utrue(xfine,tnp);
102        plot(x,u,'b.-', xfine,ufine,'r')

```

```
103         title(sprintf('t = %9.5e  after %4i time steps with %5i ...  
            grid points',...  
104                     tnp,n,m+2))  
105         error = max(abs(u-uttrue(x,tnp)));  
106         disp(sprintf('at time t = %9.5e  max error = ...  
            %9.5e',tnp,error))  
107         if n<nsteps, input('Hit <return> to continue  ');  
108         end  
109     end  
110  
111     tn = tnp;      % for next time step  
112  
113 end  
114 error_loglog(h,error);  
115 end
```

```

1 clear
2 close all
3 clc
4 for i=1:15
5 M=10*i+9;
6 heat_CN(M);
7 end
8 function [h,k,error] = heat_CN(m)
9 % heat_CN.m
10 %
11 % Solve  $u_t = \kappa u_{xx}$  on  $[a,b]$  with Dirichlet boundary ...
    conditions,
12 % using the Crank-Nicolson method with m interior points.
13 %
14 % Returns k, h, and the max-norm of the error.
15 % This routine can be embedded in a loop on m to test the accuracy,
16 % perhaps with calls to error_table and/or error_loglog.
17 %
18 % From http://www.amath.washington.edu/~rjl/fdmbook/ (2007)
19
20 clf % clear graphics
21 hold on % Put all plots on the same graph (comment out if ...
    desired)
22
23 ax = 0;
24 bx = 1;
25 kappa = .02; % heat conduction coefficient:
26 tfinal = 1; % final time
27
28 h = (bx-ax)/(m+1); % h =  $\Delta x$ 
29 x = linspace(ax,bx,m+2)'; % note x(1)=0 and x(m+2)=1
30 % u(1)=g0 and u(m+2)=g1 are known from BC's
31 k = 4*h; % time step
32
33 nsteps = round(tfinal / k); % number of time steps
34 nplot = 1; % plot solution every nplot time steps
35 % (set nplot=2 to plot every 2 time steps, etc.)
36 nplot = nsteps; % only plot at final time
37
38
39 % true solution for comparison:
40 % For Gaussian initial conditions  $u(x,0) = \exp(-\beta (x-0.4)^2)$ 
41 beta = 150;
42 utrue = @(x,t) exp(-(x-0.4).^2 / (4*kappa*t + 1/beta)) / ...
    sqrt(4*beta*kappa*t+1);
43
44 % initial conditions:
45 u0 = utrue(x,0);
46
47
48 % Each time step we solve MOL system  $U' = AU + g$  using the ...
    Trapezoidal method
49

```

```

50 % set up matrices:
51 r = (1/4) * kappa* k/(h^2);
52 e = ones(m,1);
53 A = spdiags([e -2*e e], [-1 0 1], m, m);
54 A1 = eye(m) - r * A;
55 A2 = eye(m) + r * A;
56
57
58 % initial data on fine grid for plotting:
59 xfine = linspace(ax,bx,1001);
60 ufine = utrue(xfine,0);
61
62 % initialize u and plot:
63 tn = 0;
64 u = u0;
65
66 plot(x,u,'b.-', xfine,ufine,'r')
67 legend('computed','true')
68 title('Initial data at time = 0')
69
70 %input('Hit <return> to continue ');
71
72
73 % main time-stepping loop:
74
75 for n = 1:nsteps
76     tnp = tn + k;    % = t_{n+1}
77
78     % boundary values u(0,t) and u(1,t) at times tn and tnp:
79
80     g0n = u(1);
81     gln = u(m+2);
82     g0np = utrue(ax,tnp);
83     glnp = utrue(bx,tnp);
84
85     % compute right hand side for linear system:
86     uint = u(2:(m+1));    % interior points (unknowns)
87     rhs = 1/3*( (4*inv(A1)*A2-eye(m))*uint);
88     % fix-up right hand side using BC's (i.e. add vector g to ...
89         A2*uint)
89     rhs(1) = rhs(1) + r*(g0n + g0np);
90     rhs(m) = rhs(m) + r*(gln + glnp);
91
92     % solve linear system:
93     uint = A1\rhs;
94
95     % augment with boundary values:
96     u = [g0np; uint; glnp];
97
98     % plot results at desired times:
99     if mod(n,nplot)==0 || n==nsteps
100         ufine = utrue(xfine,tnp);
101         plot(x,u,'b.-', xfine,ufine,'r')
102         title(sprintf('t = %9.5e after %4i time steps with %5i ...

```



```
        grid points',...
103         tnp,n,m+2))
104     error = max(abs(u-uttrue(x,tnp)));
105     disp(sprintf('at time t = %9.5e  max error = ...
        %9.5e',tnp,error))
106     if n<nsteps, input('Hit <return> to continue ');
107     end
108     end
109
110     tn = tnp;    % for next time step
111 end
112 error_loglog(h,error);
113 end
```

9.3 9.3.a. Modify heat\_CN.m to solve the heat equation for  $-1 \leq x \leq 1$  with step function initial data

$$u(x, 0) = \begin{cases} 1 & x < 0 \\ 0 & x \geq 0 \end{cases}$$

With appropriate Dirichlet boundary conditions, the exact solution is

$$u(x, t) = \frac{1}{2} \operatorname{erfc}(x/\sqrt{4\kappa t})$$

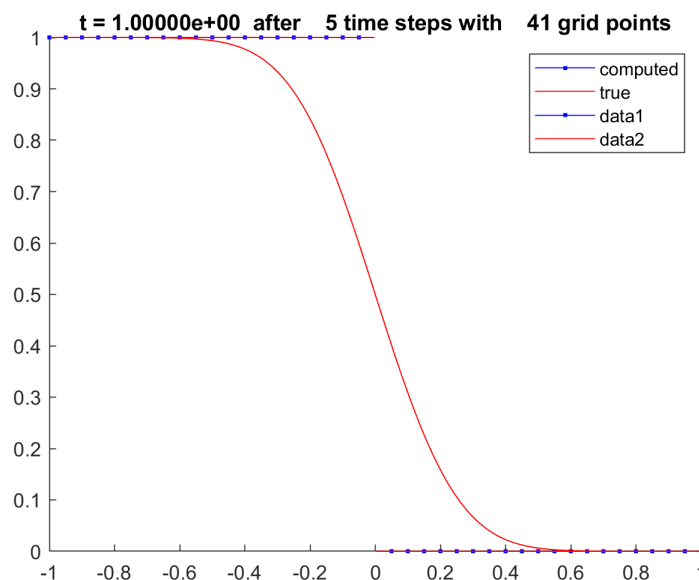
where erfc is the complementary error function

$$\operatorname{erfc} = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-z^2} dz.$$

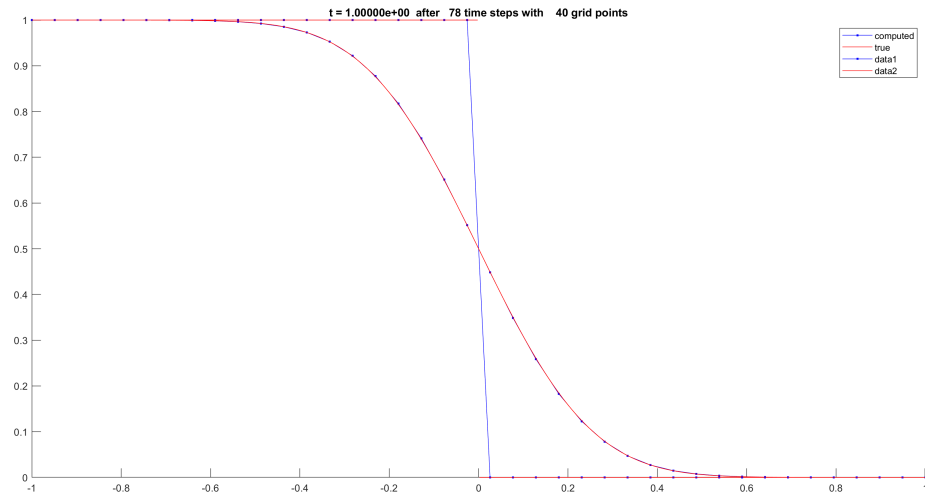
- i. Test this routine  $m = 39$  and  $k = 4h$ . Note that there is an initial rapid transient decay of the high wave numbers that is not captured well with this size time step.
- ii. How small do you need to take the time step to get reasonable results? For a suitably small time step, explain why you get much better results by using  $m = 38$  than  $m = 39$ . What is the observed order of accuracy as  $k \rightarrow 0$  when  $k = \alpha h$  with  $\alpha$  suitably small and  $m$  even?
- b. Modify heat\_trbdf2.m (see Exercise 9.2) to solve the heat equation for  $-1 \leq x \leq 1$  with step function initial data as above. Test this routine using  $k = 4h$  and estimate the order of accuracy as  $k \rightarrow 0$  with  $m$  even. Why does the TR-BDF2 method work better than Crank-Nicolson?

*Solution.*

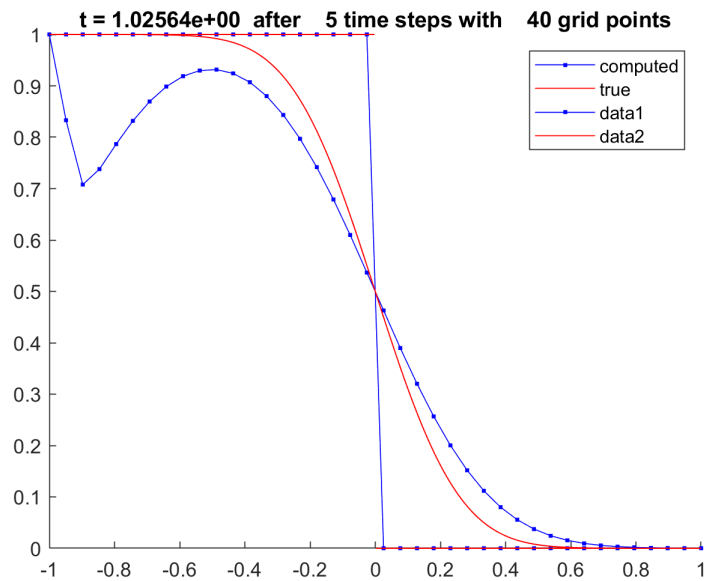
a.i.



ii. Here we are taking a time step of  $k = \frac{1}{10}h$ .



b. The TR-BDF2 works better than the Crank-Nicolson due to the better performance with the same time step.



```

1 clear
2 close all
3 clc
4 M=39;    %Comment out one of these 2 lines to test differnt grids
5 %M=38;
6 heat_CN(M);
7 function [h,k,error] = heat_CN(m)
8
9 clf      % clear graphics
10 hold on % Put all plots on the same graph (comment out if ...
    desired)
11
12 ax = -1;
13 bx = 1;
14 kappa = .02;      % heat conduction coefficient:
15 tfinal = 1;      % final time
16
17 h = (bx-ax)/(m+1); % h = Δ x
18 x = linspace(ax,bx,m+2)'; % note x(1)=0 and x(m+2)=1
19 % u(1)=g0 and u(m+2)=g1 are known from BC's
20 k = 4*h;          % Comment out one of these lines to ...
    test different time steps.
21 %k=(1/4)*h;
22
23 nsteps = round(tfinal / k); % number of time steps
24 %nplot = 1; % plot solution every nplot time steps
25 % (set nplot=2 to plot every 2 time steps, etc.)
26 nplot = nsteps; % only plot at final time
27
28
29 % true solution for comparison:
30 utrue = @(x,t) 1/2 * erfc(x/(sqrt(4*kappa*t)));
31
32 % initial conditions:
33 u0 = utrue(x,0);
34
35
36 % Each time step we solve MOL system U' = AU + g using the ...
    Trapezoidal method
37
38 % set up matrices:
39 r = (1/2) * kappa * k / (h^2);
40 e = ones(m,1);
41 A = spdiags([e -2*e e], [-1 0 1], m, m);
42 A1 = eye(m) - r * A;
43 A2 = eye(m) + r * A;
44
45
46 % initial data on fine grid for plotting:
47 xfine = linspace(ax,bx,1001);
48 ufine = utrue(xfine,0);
49
50 % initialize u and plot:

```

```

51 tn = 0;
52 u = u0;
53
54 plot(x,u,'b.-', xfine,ufine,'r')
55 legend('computed','true')
56 title('Initial data at time = 0')
57
58 %input('Hit <return> to continue ');
59
60
61 % main time-stepping loop:
62
63 for n = 1:nsteps
64     tnp = tn + k;    % = t_{n+1}
65
66     % boundary values u(0,t) and u(1,t) at times tn and tnp:
67
68     g0n = u(1);
69     gln = u(m+2);
70     g0np = utrue(ax,tnp);
71     glnp = utrue(bx,tnp);
72
73     % compute right hand side for linear system:
74     uint = u(2:(m+1));    % interior points (unknowns)
75     rhs = A2*uint;
76     % fix-up right hand side using BC's (i.e. add vector g to ...
77     A2*uint)
78     rhs(1) = rhs(1) + r*(g0n + g0np);
79     rhs(m) = rhs(m) + r*(gln + glnp);
80
81     % solve linear system:
82     uint = A1\rhs;
83
84     % augment with boundary values:
85     u = [g0np; uint; glnp];
86
87     % plot results at desired times:
88     if mod(n,nplot)==0 || n==nsteps
89         ufine = utrue(xfine,tnp);
90         plot(x,u,'b.-', xfine,ufine,'r')
91         title(sprintf('t = %9.5e after %4i time steps with %5i ...
92                     grid points',...
93                     tnp,n,m+2))
94         error = max(abs(u-utrace(x,tnp)));
95         disp(sprintf('at time t = %9.5e max error = ...
96                     %9.5e',tnp,error))
97         if n<nsteps, input('Hit <return> to continue ');
98         end
99     end
100     tn = tnp;    % for next time step
101 end
102 end

```

```

1 clear
2 close all
3 clc
4 M=38;
5 heat_CN(M);
6 function [h,k,error] = heat_CN(m)
7 %
8 % heat_CN.m
9 %
10 % Solve  $u_t = \kappa \cdot u_{xx}$  on  $[ax,bx]$  with Dirichlet boundary ...
    conditions,
11 % using the Crank-Nicolson method with m interior points.
12 %
13 % Returns k, h, and the max-norm of the error.
14 % This routine can be embedded in a loop on m to test the accuracy,
15 % perhaps with calls to error.table and/or error.loglog.
16 %
17 % From http://www.amath.washington.edu/~rjl/fdmbook/ (2007)
18
19 clf % clear graphics
20 hold on % Put all plots on the same graph (comment out if ...
    desired)
21
22 ax = -1;
23 bx = 1;
24 kappa = .02; % heat conduction coefficient:
25 tfinal = 1; % final time
26
27 h = (bx-ax)/(m+1); % h =  $\Delta x$ 
28 x = linspace(ax,bx,m+2)'; % note x(1)=0 and x(m+2)=1
29 % u(1)=g0 and u(m+2)=g1 are known from BC's
30 k = 4*h; % time step
31
32 nsteps = round(tfinal / k); % number of time steps
33 nplot = 1; % plot solution every nplot time steps
34 % (set nplot=2 to plot every 2 time steps, etc.)
35 nplot = nsteps; % only plot at final time
36
37 % true solution for comparison:
38 utrue = @(x,t) 1/2 * erfc(x/(sqrt(4*kappa*t)));
39
40 % initial conditions:
41 u0 = utrue(x,0);
42
43
44 % Each time step we solve MOL system  $U' = AU + g$  using the ...
    Trapezoidal method
45
46 % set up matrices:
47 r = (1/2) * kappa * k / (h^2);
48 e = ones(m,1);
49 A = spdiags([e -2*e e], [-1 0 1], m, m);
50 A1 = eye(m) - r * A;

```

```

51 A2 = eye(m) + r * A;
52
53
54 % initial data on fine grid for plotting:
55 xfine = linspace(ax,bx,1001);
56 ufine = utrue(xfine,0);
57
58 % initialize u and plot:
59 tn = 0;
60 u = u0;
61
62 plot(x,u,'b.-', xfine,ufine,'r')
63 legend('computed','true')
64 title('Initial data at time = 0')
65
66 %input('Hit <return> to continue ');
67
68
69 % main time-stepping loop:
70
71 for n = 1:nsteps
72     tnp = tn + k;    % = t_{n+1}
73
74     % boundary values u(0,t) and u(1,t) at times tn and tnp:
75
76     g0n = u(1);
77     gln = u(m+2);
78     g0np = utrue(ax,tnp);
79     glnp = utrue(bx,tnp);
80
81     % compute right hand side for linear system:
82     uint = u(2:(m+1));    % interior points (unknowns)
83     rhs = 1/3*((4*inv(A1)*A2-eye(m))*uint);
84     % fix-up right hand side using BC's (i.e. add vector g to ...
85         A2*uint)
86     rhs(1) = rhs(1) + r*(g0n + g0np);
87     rhs(m) = rhs(m) + r*(gln + glnp);
88
89     % solve linear system:
90     uint = A1\rhs;
91
92     % augment with boundary values:
93     u = [g0np; uint; glnp];
94
95     % plot results at desired times:
96     if mod(n,nplot)==0 || n==nsteps
97         ufine = utrue(xfine,tnp);
98         plot(x,u,'b.-', xfine,ufine,'r')
99         title(sprintf('t = %9.5e after %4i time steps with %5i ...
100             grid points',...
101                 tnp,n,m+2))
102         error = max(abs(u-utrace(x,tnp)));
103         disp(sprintf('at time t = %9.5e max error = ...
104             %9.5e',tnp,error))

```

```
102         if n<nsteps, input('Hit <return> to continue ');
103         end
104     end
105
106     tn = tnp;    % for next time step
107 end
108 error_loglog(h,error);
109 end
```