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 α (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

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

and

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

Hence,

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

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 λ_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.

```
clear
close all
clc
for i=1:15
M=10*i+9;
heat_CN(M);
end
function [h,k,error] = heat_CN(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 = 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 = 4*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


% 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:
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))
        error = max(abs(u-utrue(x,tnp)));
        disp(sprintf('at time t = %9.5e  max error =  ... %9.5e',tnp,error))
        if n<nsteps, input('Hit <return> to continue  ');
        end
    end

    tn = tnp;   % for next time step

end
error_loglog(h,error);
end
```

```
clear
close all
clc
for i=1:15
M=10*i+9;
heat_CN(M);
end
function [h,k,error] = heat_CN(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 = 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 = 4*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


% 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:
r = (1/4) * 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 = 1/3*((4*inv(A1)*A2-eye(m))*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))
          error = max(abs(u-utrue(x,tnp)));
          disp(sprintf('at time t = %9.5e  max error =  ...
             %9.5e',tnp,error))
          if n<nsteps, input('Hit <return> to continue  ');
          end
      end

      tn = tnp;   % for next time step
end
error_loglog(h,error);
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

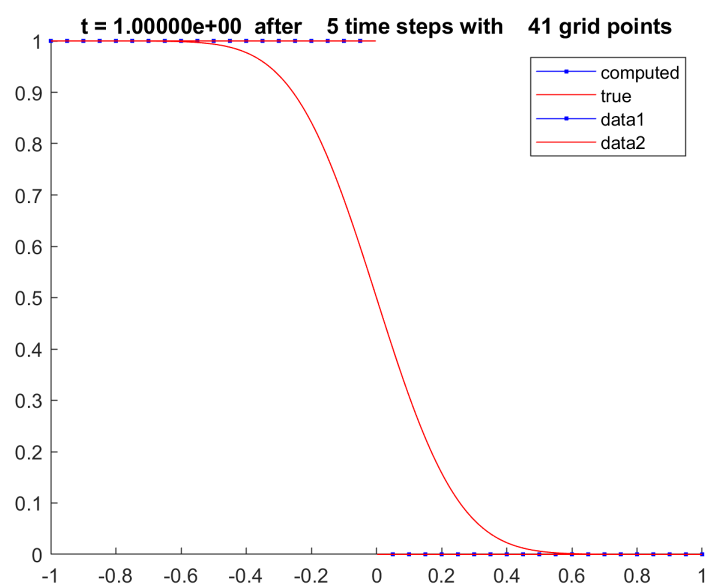
$$u(x,t) = \frac{1}{2}\text{erfc}(x/\sqrt{4\kappa t})$$

where erfc is the complementary error function

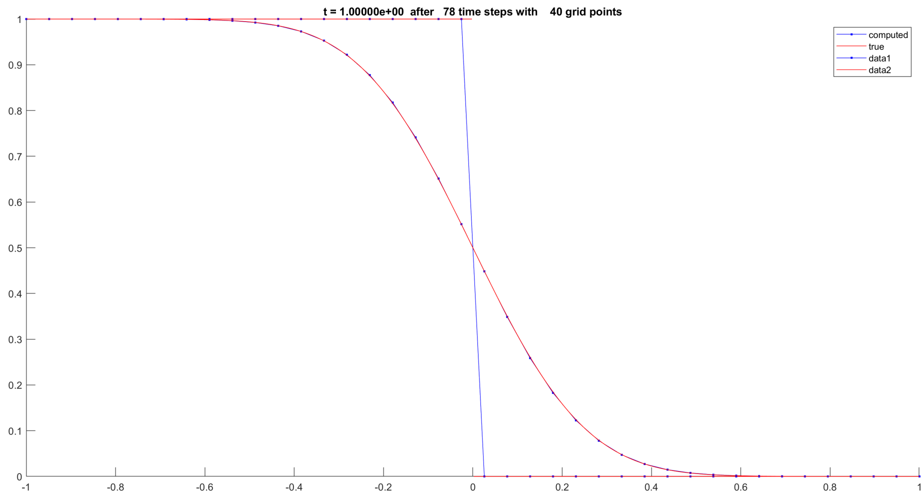
$$\text{erfc} = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-z^2} dz.$$

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 $k \to 0$ when $k = \alpha h$ with α 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 $k \to 0$ with m even. Why does the TR-BDF2 method work better than Crank-Nicolson?

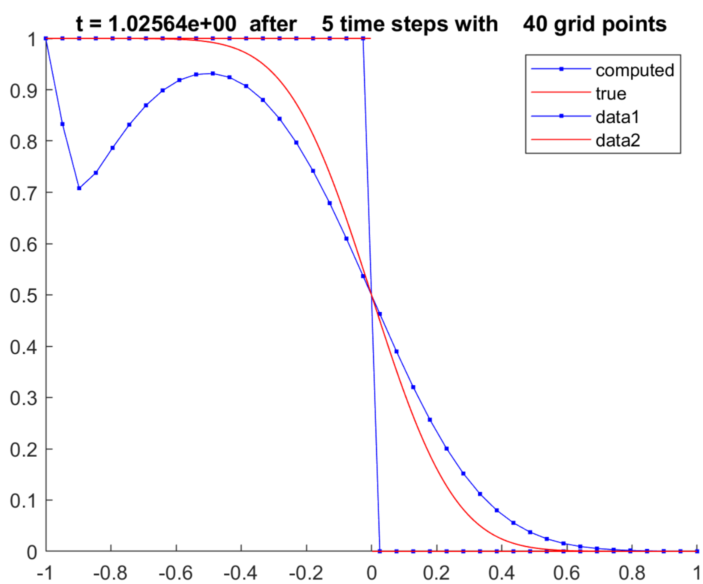
Solution.
a.i.



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.



```
clear
close all
clc
M=39;    %Comment out one of these 2 lines to test differnt grids
%M=38;
heat_CN(M);
function [h,k,error] = heat_CN(m)

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 = 4*h;                     % Comment out one of these lines to ...
    test different time steps.
%k=(1/4)*h;

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


% true solution for comparison:
utrue = @(x,t) 1/2 * erfc(x/(sqrt(4*kappa*t)));

% initial conditions:
u0 = utrue(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))
        error = max(abs(u-utrue(x,tnp)));
        disp(sprintf('at time t = %9.5e  max error =  ...
           %9.5e',tnp,error))
        if n<nsteps, input('Hit <return> to continue  ');
        end
     end

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

```
clear
close all
clc
M=38;
heat_CN(M);
function [h,k,error] = heat_CN(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 = 4*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

% true solution for comparison:
utrue = @(x,t) 1/2 * erfc(x/(sqrt(4*kappa*t)));

% initial conditions:
u0 = utrue(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 = 1/3*((4*inv(A1)*A2-eye(m))*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))
        error = max(abs(u-utrue(x,tnp)));
        disp(sprintf('at time t = %9.5e  max error =  ...
            %9.5e',tnp,error))
```

```
        if n<nsteps, input('Hit <return> to continue  ');
        end
    end

    tn = tnp;   % for next time step
end
error_loglog(h,error);
end
```