Math232 - Numerical Analysis II

Dr. Chrysoula Tsogka

HW7

Matteo Polimeno, Alex Ho

Department of Applied Mathematics, UC Merced

(Dated: April 3, 2020)

PROBLEM 1

Here we consider the following method

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

used to solve the heat equation.

Part (a)

To determine the order of accuracy of the scheme in both space and time, we need to make use of Taylor expansion. In order to make our work less strenuous, we are going to re-shift our indices by letting $n+1 = m$ so that Eq. (1) becomes

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

Now we let $U_i^n \approx u(x_i, t_n)$, $t_{n+1} = t_n + k$, $x_{i+1} = x_i + h$, and Taylor expand (Note: we will drop the intermediate variables (x,t) from the derivative terms for ease of notation)

$$\begin{aligned}
U_i^{m+1} &= u(x, t+k) = u(x,t) + ku_t + \frac{k^2}{2}u_{tt} + \frac{k^3}{6}u_{ttt} + \frac{k^4}{24}u_{tttt} + \text{h.o.t} \\
U_i^{m-1} &= u(x, t-k) = u(x,t) - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{6}u_{ttt} + \frac{k^4}{24}u_{tttt} + \text{h.o.t} \\
U_{i-1}^{m} &= u(x-h, t) = u(x,t) - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \text{h.o.t} \\
U_{i+1}^{m} &= u(x+h, t) = u(x,t) + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \text{h.o.t.}
\end{aligned}$$

Then we replace U_i^n with $u(x_i, t_n)$ and define the local truncation error for this method as

$$\tau(x,t) = \frac{u(x,t+k) - u(x,t-k)}{2k} - \frac{1}{h^2}(u(x-h,t) - 2u(x,t) + u(x+h,t)). \tag{3}$$

After plugging our Taylor expansions into Eq. (3) and work out some algebra, we find

$$\begin{aligned}
\tau(x,t) &= \frac{1}{2k}\left(2ku_t + \frac{k^3}{3}u_{ttt}\right) - \frac{1}{h^2}\left(h^2 u_{xx} + \frac{h^4}{12}u_{xxxx}\right) + \text{h.o.t} \\
&= u_t + \frac{k^2}{6}u_{ttt} - u_{xx} - \frac{h^2}{12}u_{xxxx} + \text{h.o.t.}
\end{aligned}$$

Now, we can use the heat equation to notice that $u_t = u_{xx}$ to finally obtain to leading order

$$\tau(x,t) = \frac{k^2}{6}u_{ttt} - \frac{h^2}{12}u_{xxxx}, \tag{4}$$

which shows that the method is second order both in space and in time. And we are done.

Part (b)

A linear method of the form

$$\mathbf{U}^{n+1} = B(k)\mathbf{U}^n + \mathbf{b}^n(k), \tag{5}$$

is Lax-Richtmeyer stable if, for each time T, there is constant C_T such that $\|\ B(k)^n\ \| \leq C_T$, for any $k > 0$ and any integer n for which $kn \leq T$.
Thus, for the given scheme (after letting $n+1 = m$) we have

$$\mathbf{U}^{m+1} = \frac{2k}{h^2}A\mathbf{U}^m + \mathbf{U}^{m-1},$$

where A is a $m \times m$ a tridiagonal matrix with -2's on the main diagonal, 1's on the upper and lower diagonal and 0's everywhere else. Then, letting $\mathbf{b}^{m-1} = \mathbf{U}^{m-1}$ and $B(k) = \frac{2k}{h^2}A\mathbf{U}^m$ we have the same form as in Eq. (5). The eigenvalues of A are known to be

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

and for the scheme to be Lax-Richtmeyer stable we require the spectral radius $\rho(B(k)) < 1$. Therefore,

$$\begin{aligned}
\rho(B(k)) &= \max_{1\leq 1\leq m} \mid \frac{2k}{h^2}\lambda_p \mid \\
&= \max_{1\leq p\leq m} \mid \frac{2\alpha}{h^2}2(-2) \mid,\ k = \alpha h^2 \\
&= \frac{8\alpha}{h^2}.
\end{aligned}$$

Thus, for the scheme to be Lax-Richtmeyer stable we would need

$$\alpha < \frac{h^2}{8}. \tag{6}$$

Part (c)

This is not a useful scheme. Despite being second order in both space and time, for the scheme to be Lax-Richtmeyer stable we would have to require our time step k to be smaller than $h^4/8$, assuming we take $k = \alpha h^2$, with $0 < \alpha < h^2/8$. Thus, for any h such that $0 < h < 1$, we would have a tiny time step. This would make the scheme very computationally costly and, therefore, not really useful in practice.

PROBLEM 2

In this problem we modify `heat_CN.m` to solve the heat equation

$$u(x,t)_t = \kappa u(x,t)_{xx}, \tag{7}$$

with $\kappa = 0.02$, over $0 < x < 1$, $0 < t < 1$. The boundary and initial conditions are defined through the given analytical solution

$$u(x,t) = \frac{\exp(-(x-0.4)^2/(4\kappa t + 1/\beta))}{\sqrt{(4\beta\kappa t + 1)}}, \tag{8}$$

with $\beta = 150$. We report on our results in Tables I and II, as well as in Figs. (1)-(4). As can readily be seen from the tables, the Crank-Nicolson scheme is second order accurate in both space and time, confirming what was expected from the theory. Moreover Fig.(1)(d) shows how the scheme (blue line) does a good job of approximating the analytical solution (red line), using as few as 20 spatial grid points.
We use $k = 4h$ and $\kappa = 0.02$ and run the scheme using various numbers of grid points.

Table of Error - Space			
h	error	ratio	observed order
0.11111	4.00939e-03	NaN	NaN
0.05263	1.05412e-03	3.80353	1.78788
0.02564	1.62002e-04	6.50685	2.60436
0.01449	5.38371e-05	3.00912	1.93087
0.01010	2.55505e-05	2.10709	2.06448

TABLE I. Table of the error and order of accuracy in space for the Crank-Nicolson scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the Crank-Nicolson scheme is second order accurate in space, as expected from the theory.

Table of Error - Time			
k	error	ratio	observed order
0.44444	4.00939e-03	NaN	NaN
0.21053	1.05412e-03	3.80353	1.78788
0.10256	1.62002e-04	6.50685	2.60436
0.05797	5.38371e-05	3.00912	1.93087
0.04040	2.55505e-05	2.10709	2.06448

TABLE II. Table of the error and order of accuracy in time for the Crank-Nicolson scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the Crank-Nicolson scheme is second order accurate in time, as expected from the theory.

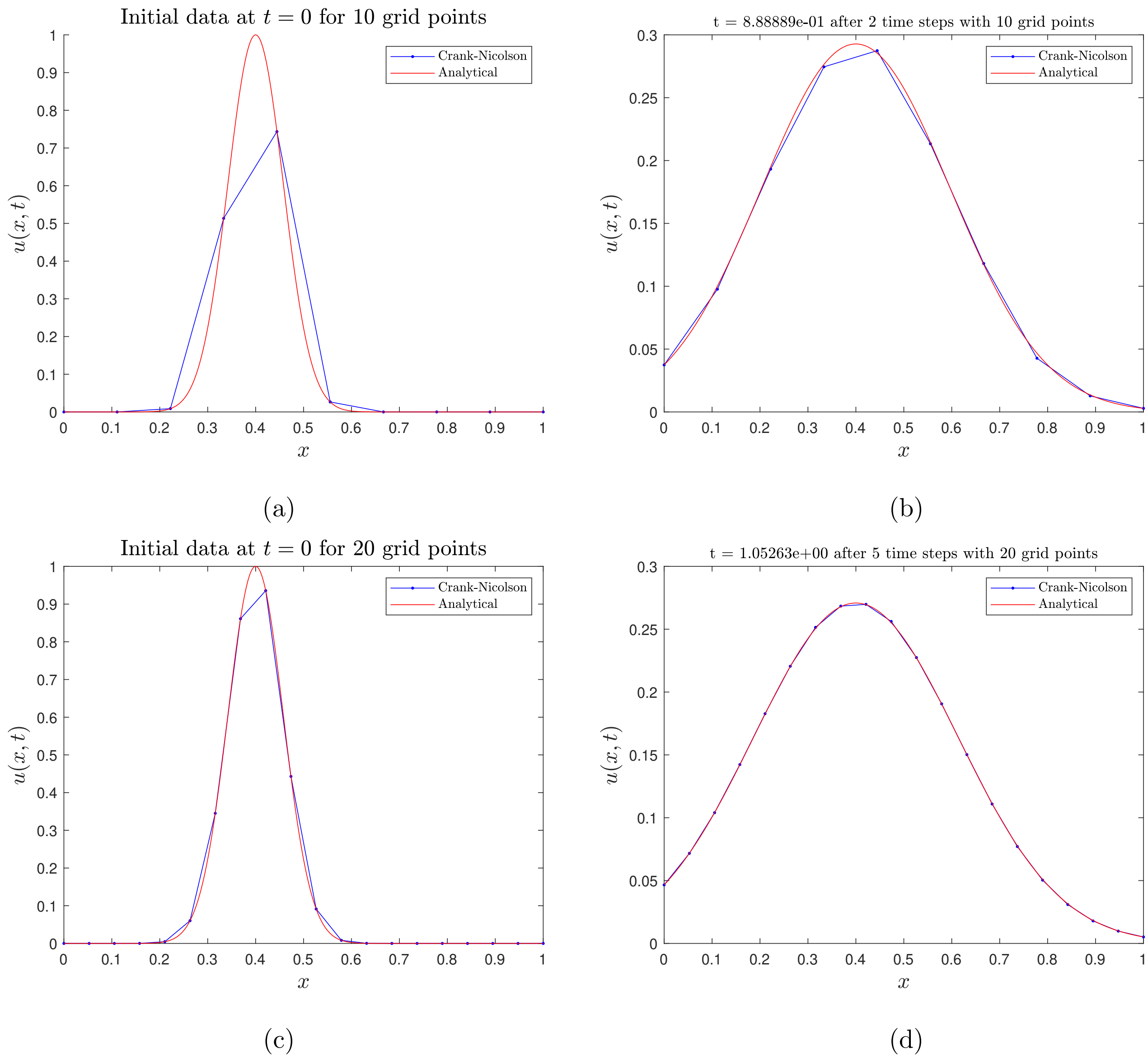


FIG. 1. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). From panel (d) it is readily apparent how the Crank-Nicolson scheme accurately mimics the analytical solution using only 20 spatial grid points.

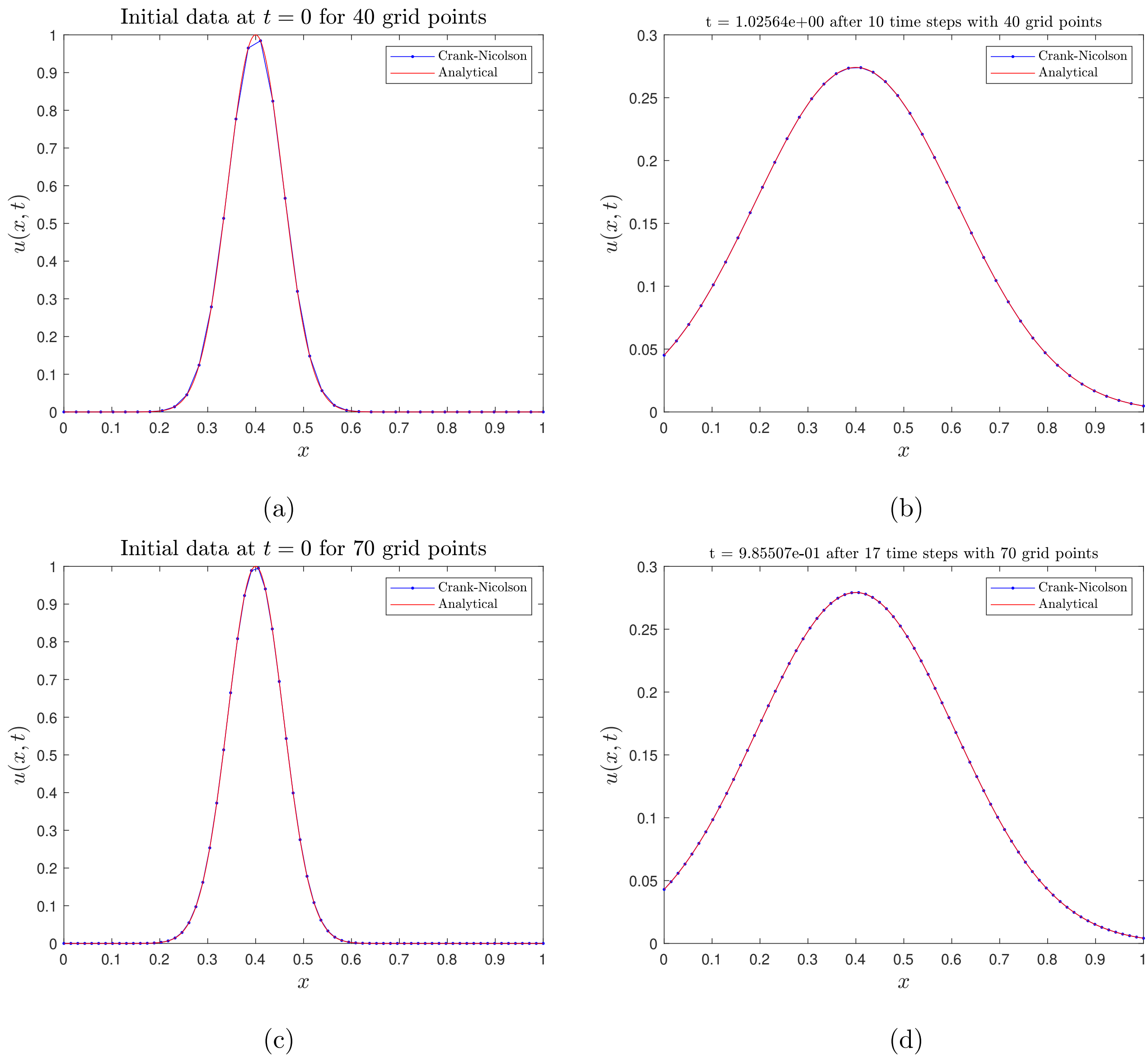
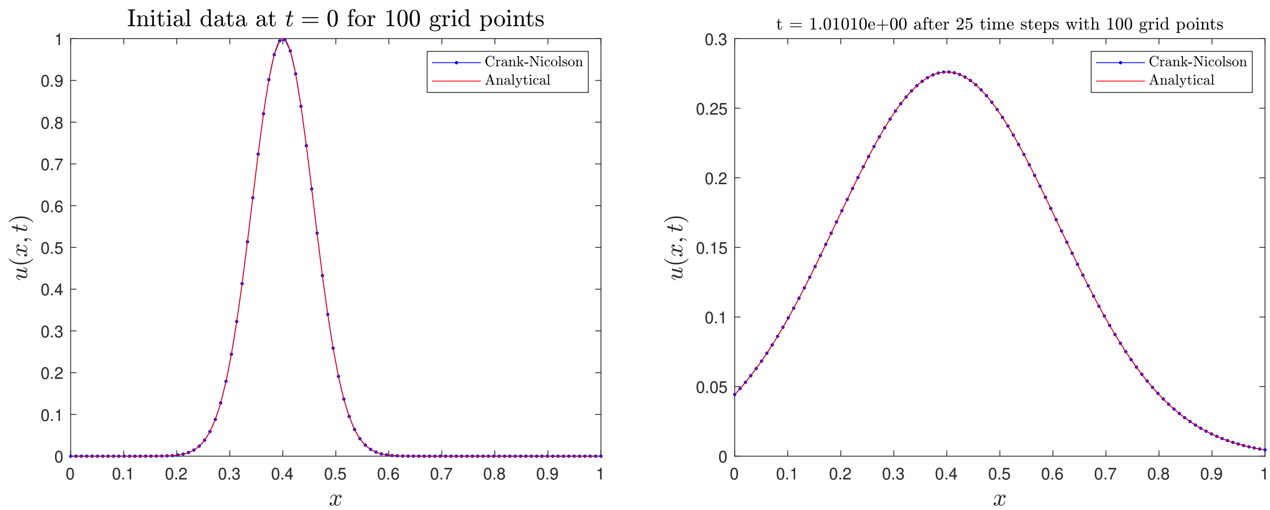


FIG. 2. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). The solutions overlap.



(a) (b)

FIG. 3. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$ (a) and at a later time t (b). The solutions essentially overlap.

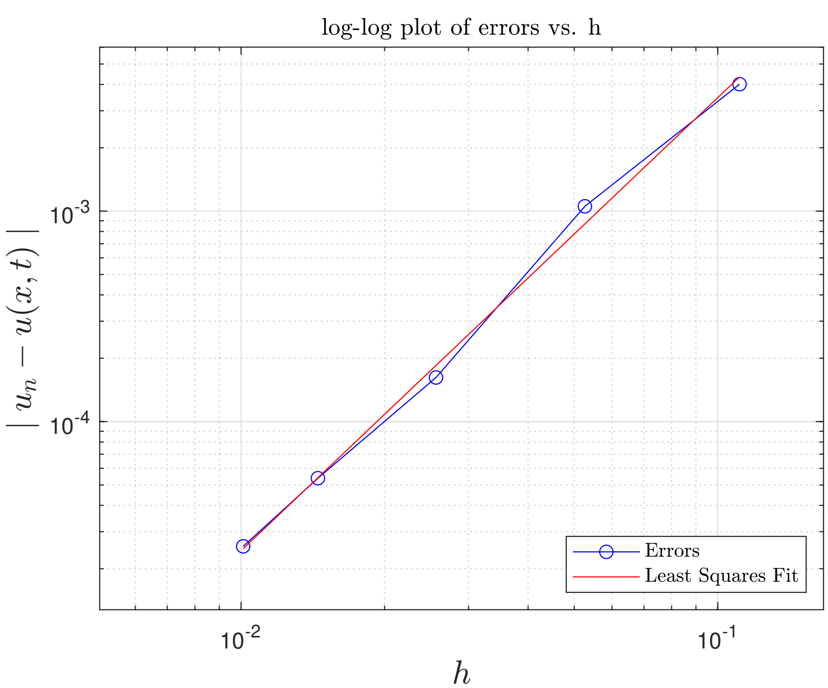


FIG. 4. Plot of the error in space for the Crank-Nicolson method a function of the mesh width h.

Table of Error - Space			
h	error	ratio	observed order
0.11111	5.41354e-03	NaN	NaN
0.05263	1.20580e-03	4.48960	2.00981
0.02564	2.93774e-04	4.10450	1.96362
0.01449	9.71405e-05	3.02422	1.93964
0.01010	4.64038e-05	2.09337	2.04640

TABLE III. Table of the error and order of accuracy in space for the Forward Euler scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the TR-BDF2 scheme is second order accurate in space, as expected from the theory.

PROBLEM 3

Here we solve Eq. ((7)) numerically using the TR-BDF2 schme:

$$\begin{aligned} U_i^{n+1/2} &= U_i^n + \frac{k}{4}(f(U_i^n) + f(U_i^{n+1/2})) \\ U_i^{n+1} &= \frac{1}{3}(4U_i^{n+1/2} - U_i^n + kf(U_i^{n+1})). \end{aligned} \tag{9}$$

Here $f(U) = D_2(U)$ represents the central finite different scheme that approximates the second derivative operator and $U_i^n \approx u(x_i, t_n)$. Effectively, we use the solution at time the discrete time t_n and position x_i to iterate forward in time by half a step from t_n to $t_{n+1/2}$ using the implicit trapezoid method. Then, we use the computed $U_i^{n+1/2}$ and iterate forward in time by another half a step from $t_{n+1/2}$ to t_{n+1} using the BDF2 scheme to compute U_i^{n+1}. Given that both schemes are implicit and then we push the scheme forward in time by half a step in each case, and that we are using central finite difference for the space discretization, we expect our method to be second order in both space and time.
Tables III, IV and Figs.(5)-(8) confirm what was expected from the theory. We use $k = 4h$ and $\kappa = 0.02$ and run the scheme using various numbers of grid points.

Table of Error - Time			
k	error	ratio	observed order
0.44444	5.41354e-03	NaN	NaN
0.21053	1.20580e-03	4.48960	2.00981
0.10256	2.93774e-04	4.10450	1.96362
0.05797	9.71405e-05	3.02422	1.93964
0.04040	4.64038e-05	2.09337	2.04640

TABLE IV. Table of the error and order of accuracy in time for the Crank-Nicolson scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the TR-BDF2 scheme is second order accurate in time, as expected from the theory.

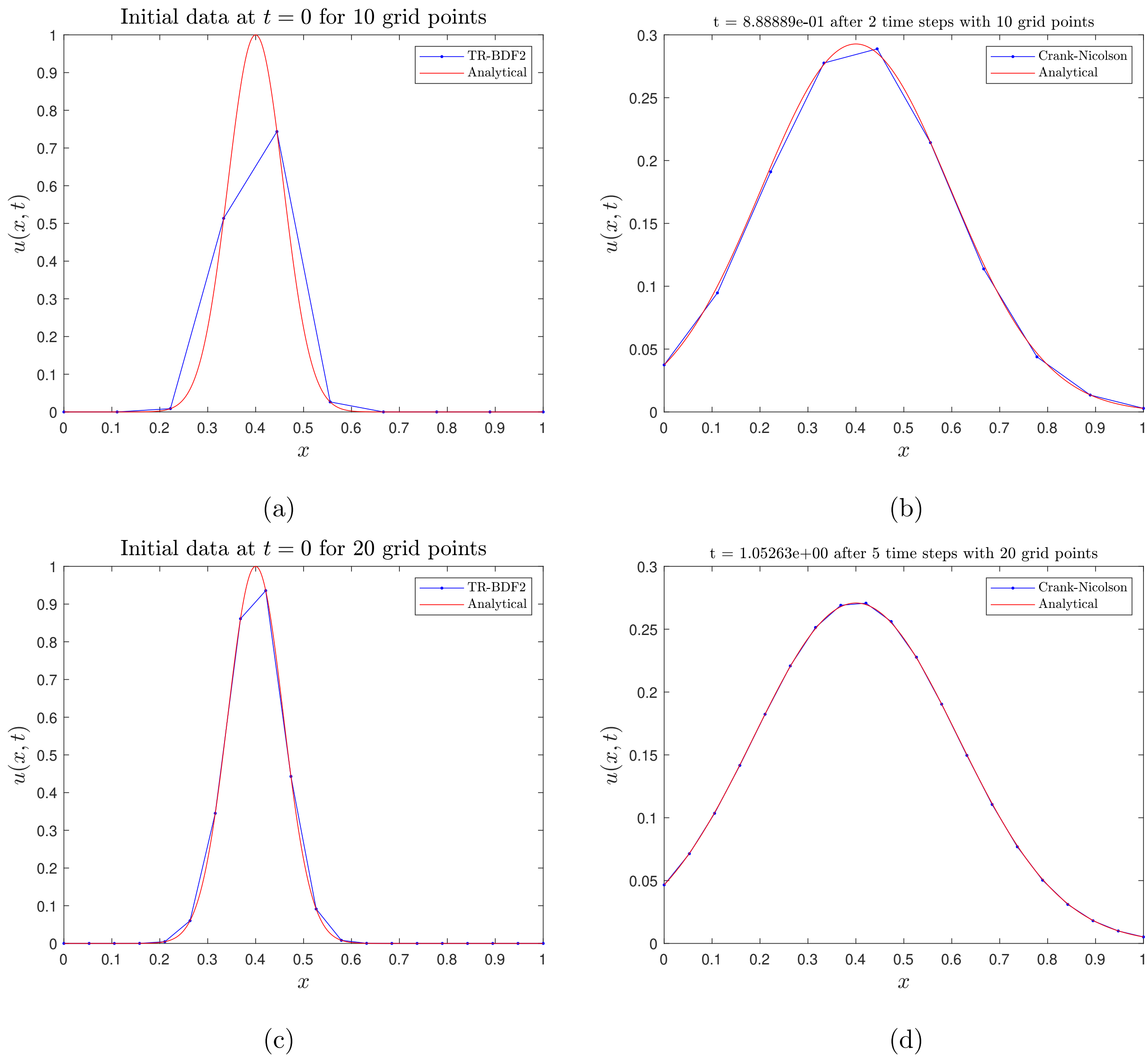


FIG. 5. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different numbers of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). Panel (d) show how the TR-BDF2 method already approximates the solution well, using only 20 spatial grid points.

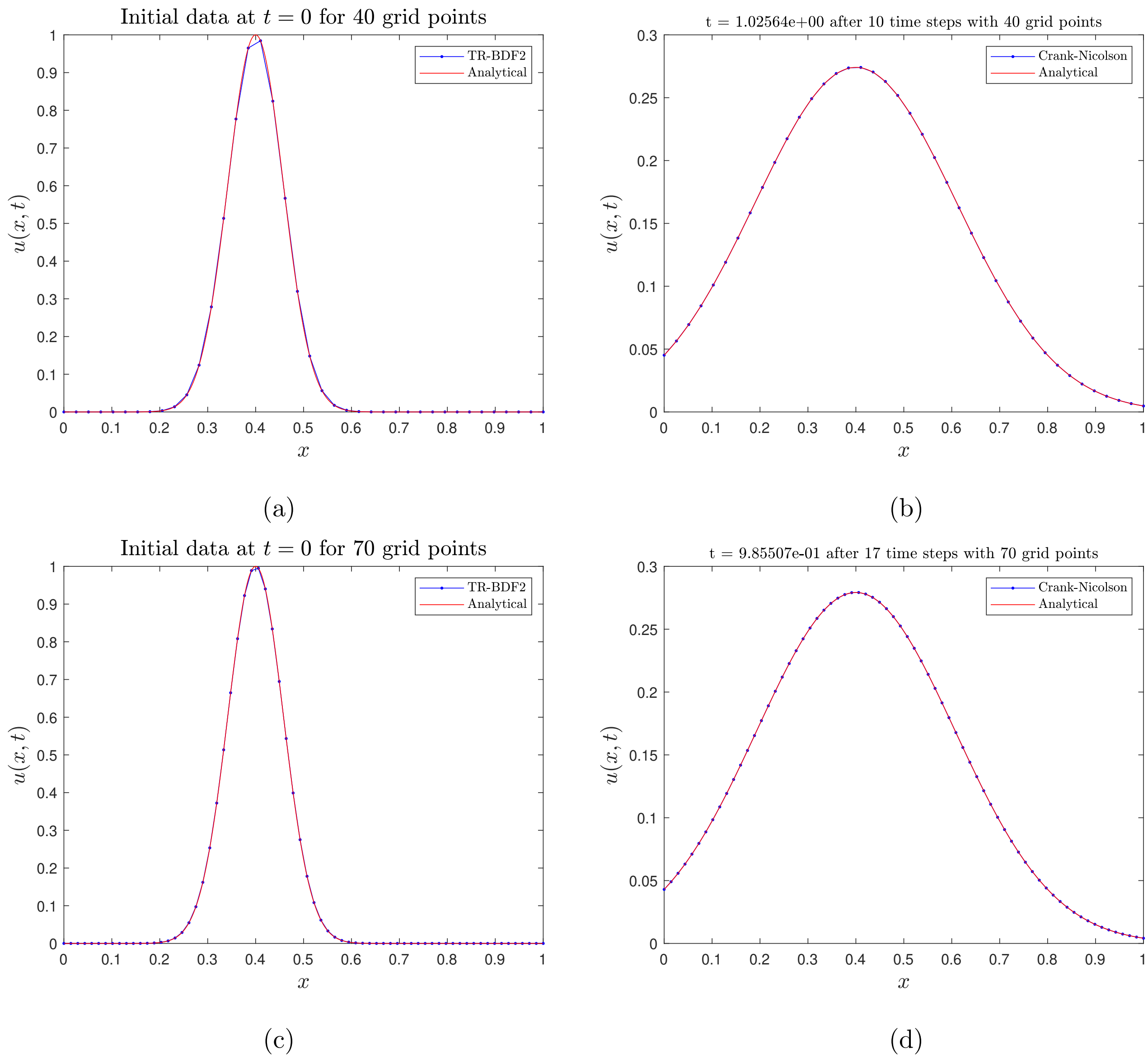


FIG. 6. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different numbers of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). The solutions essentially overlap.

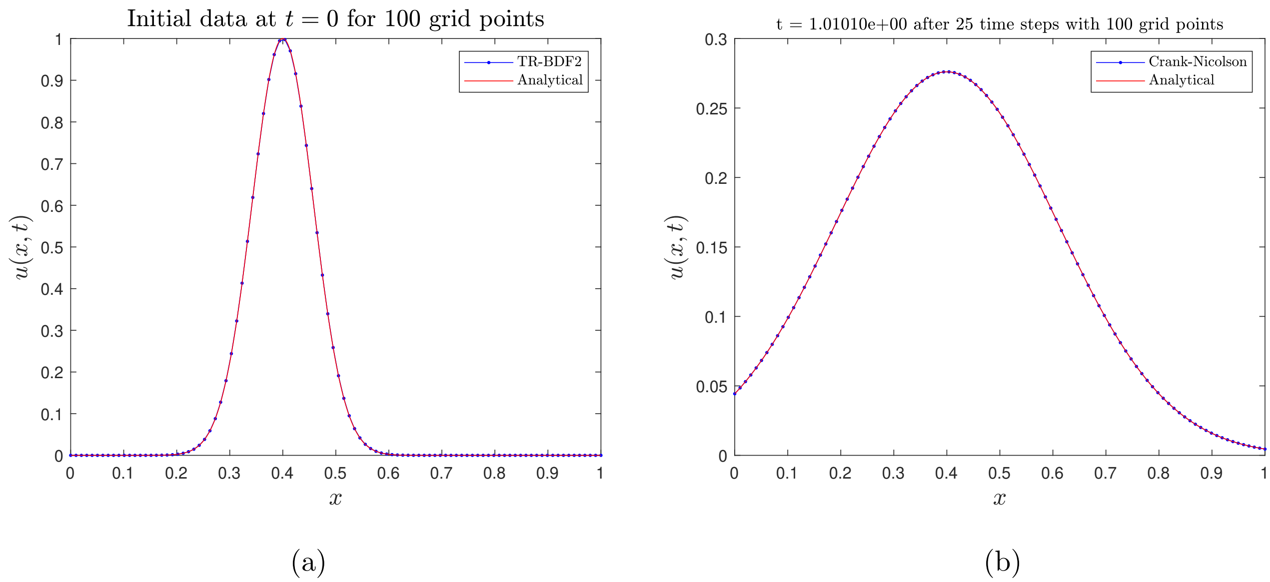


FIG. 7. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different numbers of grid points, at the initial time $t = 0$ (a) and at a later time t (b). The solutions overlap.

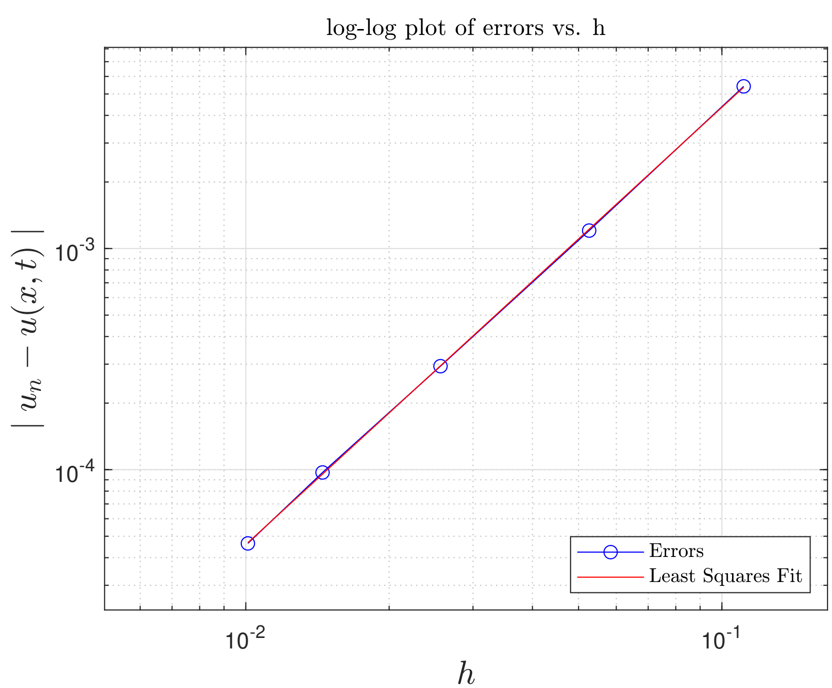


FIG. 8. Plot of the error in space for the TR-BDF2 method as a function of the mesh width h.

Table of Error - Space			
h	error	ratio	observed order
0.11111	6.36377e-02	NaN	NaN
0.05263	3.96699e-03	16.04181	3.71406
0.02564	9.08020e-04	4.36883	2.05041
0.01449	2.89623e-04	3.13518	2.00280
0.01010	1.40626e-04	2.05953	2.00125

TABLE V. Table of the error and order of accuracy in space for the Forward Euler scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the Forward Euler scheme is second order accurate in space, as expected from the theory.

PROBLEM 4

Here we modify `heat_CN.m` to solve Eq. (7) using the Forward Euler method in time, while keeping central finite difference in space. Since Forward Euler is an explicit first order method, we expect the method to be less accurate than both Crank-Nicolson and TR-BDF2. Tables V, VI and Figs.(9)-(12) confirm that the method is first order in time and second order in space. We use $k = 24h^2$ and $\kappa = 0.02$ run the scheme using various numbers of grid points.

Table of Error - Time			
k	error	ratio	observed order
0.29630	6.36377e-02	NaN	NaN
0.06648	3.96699e-03	16.04181	1.85703
0.01578	9.08020e-04	4.36883	1.02520
0.00504	2.89623e-04	3.13518	1.00140
0.00245	1.40626e-04	2.05953	1.00062

TABLE VI. Table of the error and order of accuracy in time for the Forward-Euler scheme, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results confirm the the Forward-Euler scheme is first order accurate in time, as expected from the theory.

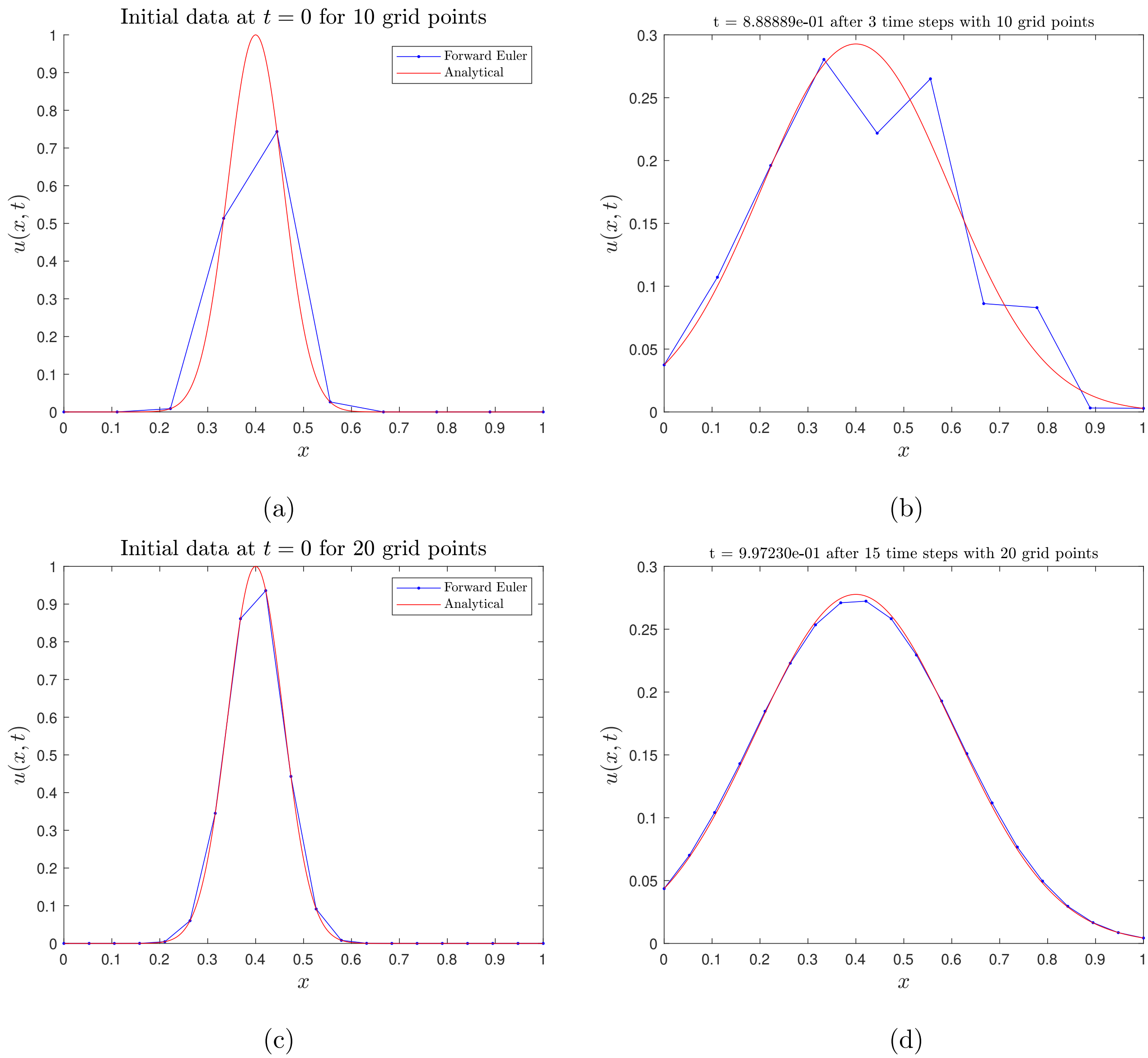


FIG. 9. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). Panels (a)-(d) shows that the Forward-Euler method does not fair well when compared with the Crank-Nicolson Method and the TR-BDF2, for the same number of points being used.

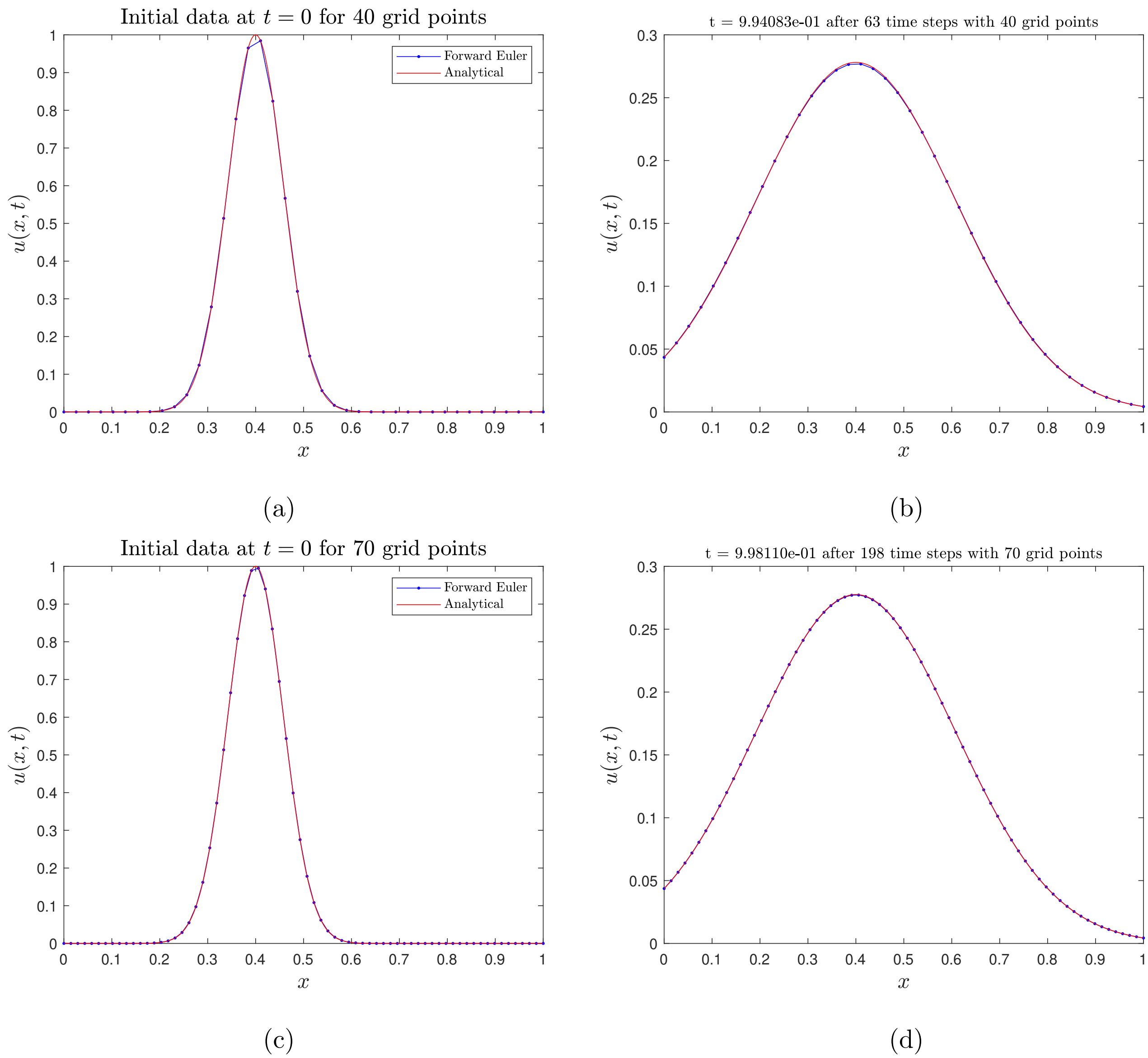
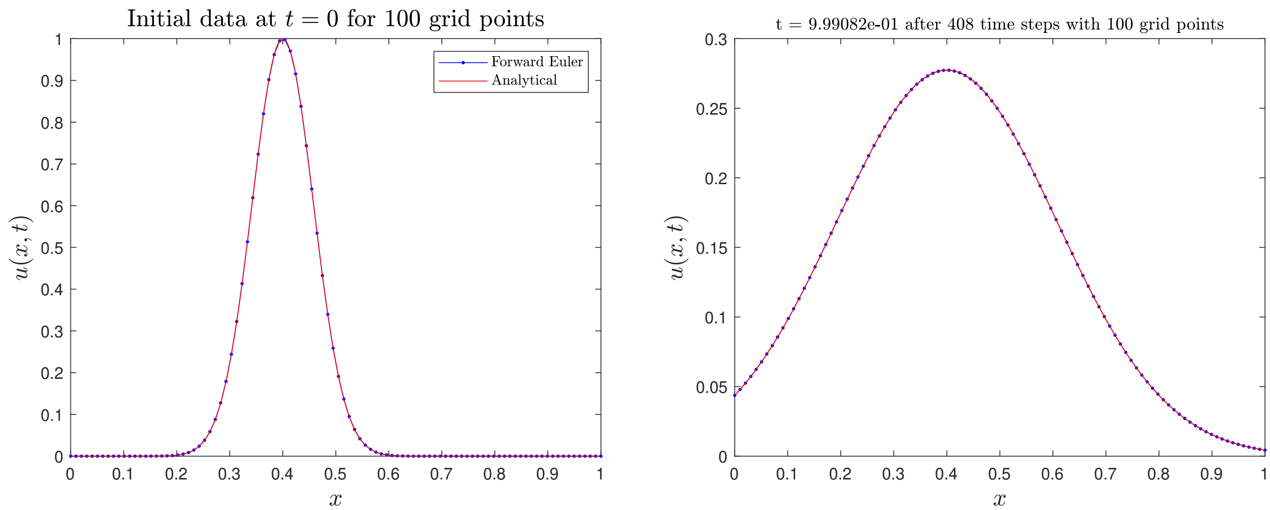


FIG. 10. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points. Increasing the number of points indubitably increases the accuracy of the Forward-Euler method, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). In panel (d) the numerical solution matches the analytical solution.



(a) (b)

FIG. 11. Plot of the analytical solution (blue line) and the numerical solutions (red line) for different number of grid points, at the initial time $t = 0$ (a) and at a later time t (b). The solutions overlap.

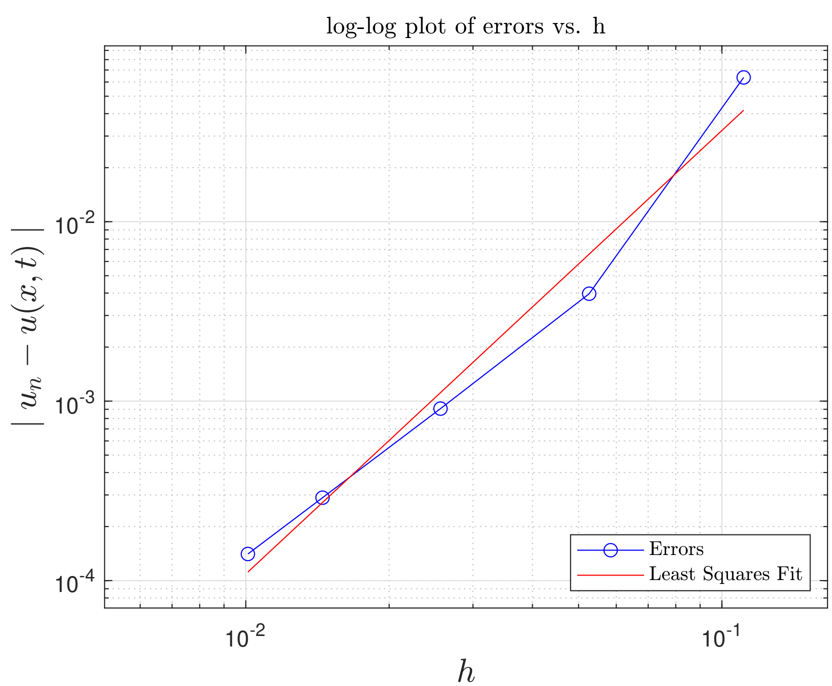


FIG. 12. Plot of the error in space for the Forward-Euler method as a function of the mesh width h.

PROBLEM 5

In this problem we use the Crank-Nicolson method to solve the heat equation for $-1 < x < 1$ and $0 < t < 1$, with step function initial data

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

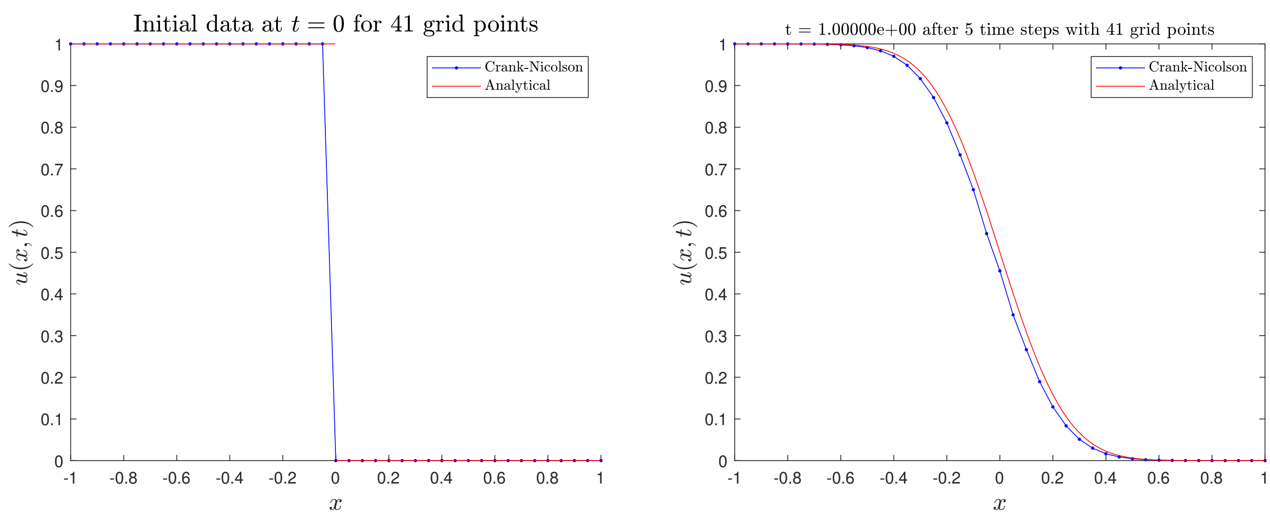
With appropriate Dirichlet boundary conditions, the exact solution is given by

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

where $\text{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-z^2}dz$ is the complimentary error function.

Part (a)

We test our routine with $m = 39$ interior points and time-step $k = 4h$.



(a)

(b)

FIG. 13. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$ (a) and at a later time t (b). The numerical method fails to capture the initial transient decay of the high wave numbers.

Table of Error - Space			
h	error	ratio	observed order
0.05128	2.03699e-03	NaN	NaN
0.03390	8.98505e-04	2.26709	1.97716
0.02020	3.25692e-04	2.75875	1.96061
0.01005	7.93384e-05	4.10510	2.02272

TABLE VII. Table of the error and order of accuracy in space for the Crank-Nicolson scheme implemented to solve the heat equation with piecewise constant initial data, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results show that the scheme is second order in space

Table of Error - Time			
k	error	ratio	observed order
0.05128	2.03699e-03	NaN	NaN
0.03390	8.98505e-04	2.26709	1.97716
0.02020	3.25692e-04	2.75875	1.96061
0.01005	7.93384e-05	4.10510	2.02272

TABLE VIII. Table of the error and order of accuracy in time for the Crank-Nicolson scheme implemented to solve the heat equation with piecewise constant initial data, for $N = 10$ (first row), $N = 20$ (second row), $N = 40$ (third row), $N = 70$ (fourth row) and $N = 100$ (fifth row). Here N represents the total number of spatial points in the grid used. Our results show that the scheme is second order in time.

Part (b)

To get reasonable results with let $k = h$. We observe that for this choice of time-step and for an even number of grid points, the method is second-order accurate in both space and time.

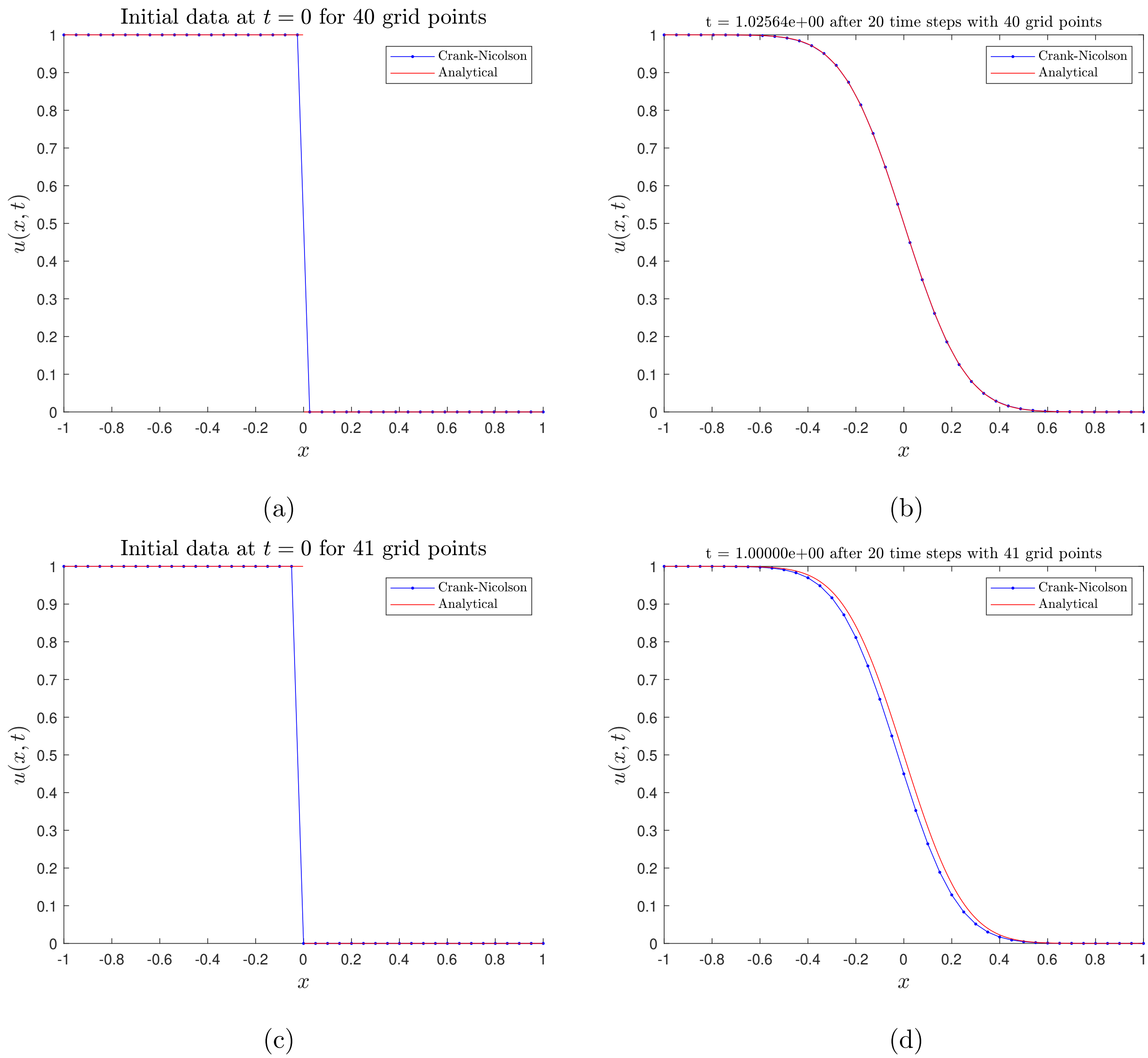
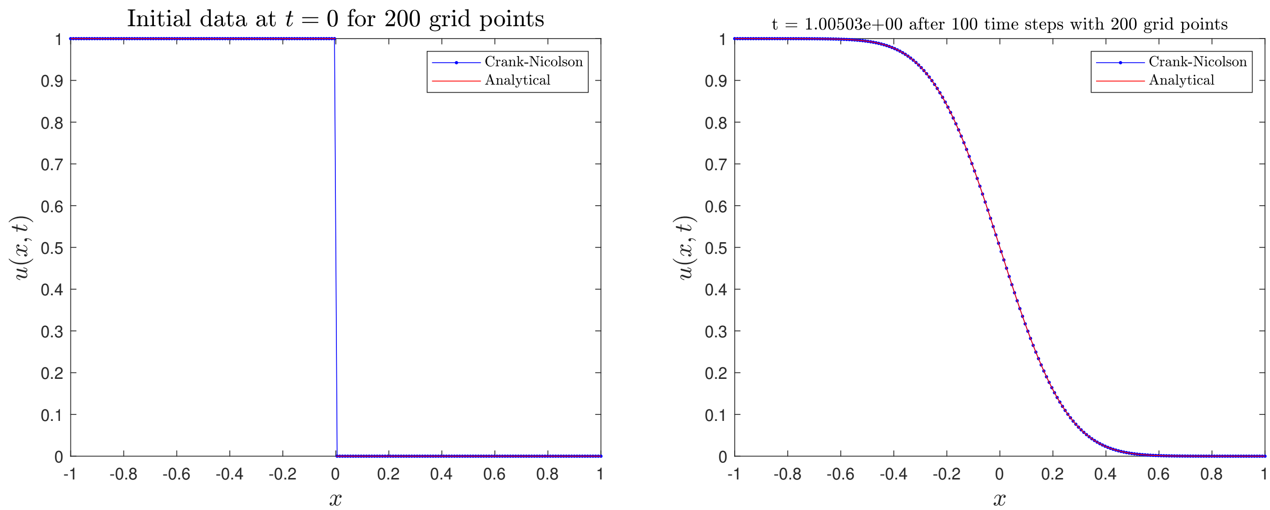


FIG. 14. Plot of the analytical solution (red line) and the numerical solutions (blue line) for different number of grid points, at the initial time $t = 0$, (a) and (c), and at a later time t (b) and (d). In panels (a)-(b) we display the results using m even, while in panels (c)-(d) we display the results for m odd. In panel (b) we see that the solutions overlap for $m = 38$.



(a) (b)

FIG. 15. Plot of the analytical solution (red line) and the numerical solutions (blue line) for $N = 200$ grid points, at the initial time $t = 0$ (a) and at a later time t (b). Panel (b) shows that the solutions overlap.

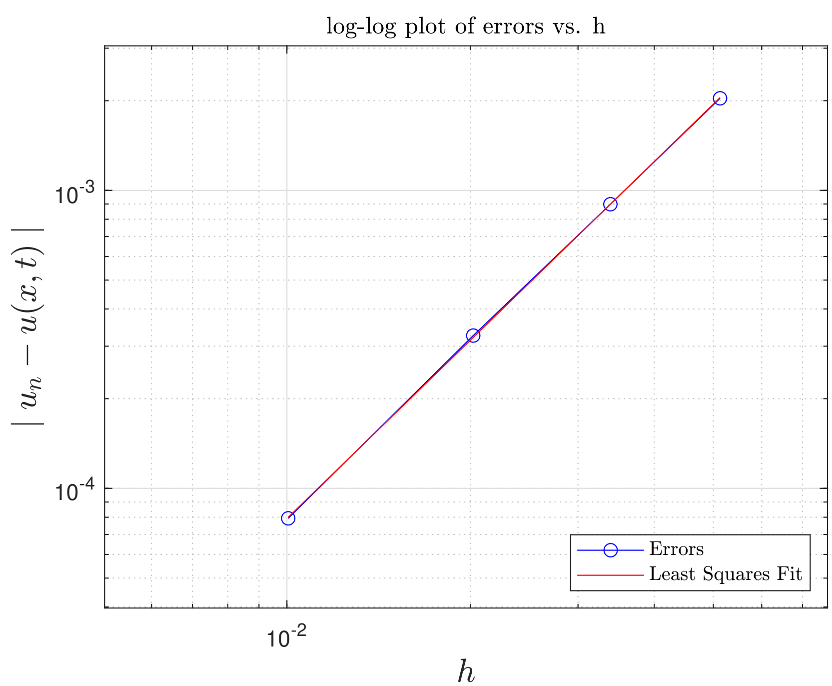


FIG. 16. Plot of the error in space for the Crank-Nicolson method as a function of the mesh width h.

MATLAB CODE

Here is the code used to generate the plots for Problem.

```
%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
errvec = [];

mvals = [8 18 38 68 98];
hvals = [];
kvals = [];
for ii=1:length(mvals)
```

```
    m = mvals(ii);
    h = (bx-ax)/(m+1);           % h = delta x
    hvals = [hvals h];
    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; %just picking a k=O(h)                % time step
    kvals = [kvals k];


    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(' ')
%        fprintf('WARNING *** k does not divide tfinal, k = %9.5e\n',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:
r = (1/2) * kappa* k/(h^2);
e = ones(m,1);
A = spdiags([e -2*e e], -1:1, 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;

figure(ii)
plot(x,u,'b.-', xfine,ufine,'r')
legend('Crank-Nicolson','Analytical','interpreter','latex')
title(sprintf('Initial data at $t = 0$ for %5i grid points',m
    +2),'interpreter','latex','fontsize',15)
xlabel('$x$','interpreter','latex','fontsize',15)
ylabel('$u(x,t)$','interpreter','latex','fontsize',15)

```

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


    % main time-stepping loop:

    for n = 1:nsteps
        tnp = tn + k;   % = t_{n+1} %total number of time steps

        % 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);
```

```
            figure(ii+5)
            plot(x,u,'b.-', xfine,ufine,'r')
            legend('Crank-Nicolson','Analytical','interpreter','latex')
            title(sprintf('t = %9.5e  after %4i time steps with %5i grid points',...
                       tnp,n,m+2),'interpreter','latex')
            xlabel('$x$','interpreter','latex','fontsize',15)
            ylabel('$u(x,t)$','interpreter','latex','fontsize',15)
            error = max(abs(u-utrue(x,tnp)));
            errvec = [errvec error];
            fprintf('at time $t$ = %9.5e  max error =  %9.5e\n',tnp,error)
%             if n<nsteps, input('Hit <return> to continue  ');
%             end
        end

     tn = tnp;   % for next time step
    end
end

error_table(hvals, kvals, errvec);   % print tables of errors and ratios

figure(ii+6)
error_loglog(hvals, errvec);  % produce log-log plot of errors and least squares fit
```

```
%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
errvec = [];

mvals = [8 18 38 68 98];
hvals = [];
kvals = [];
for ii=1:length(mvals)
    m = mvals(ii);
    h = (bx-ax)/(m+1);            % h = delta x
```

```
    hvals = [hvals h];
    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; %just picking a k=O(h)                  % time step
    kvals = [kvals k];


    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(' ')
%         fprintf('WARNING *** k does not divide tfinal, k = %9.5e\n',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:
r = kappa* k/(4*h^2);
e = ones(m,1);
A = spdiags([e -2*e e], -1:1, m, m);
A1 = eye(m) - r * A;
A2 = eye(m) + r * A;

%matrix stuff to go from nphalf to np1
r2 = kappa*k/(3*h^2);
A3 = eye(m) - r2*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;

figure(ii)
plot(x,u,'b.-', xfine,ufine,'r')
legend('TR-BDF2','Analytical','interpreter','latex')
title(sprintf('Initial data at $t = 0$ for %5i grid points',m+2),'interpreter','latex','fontsize',15)
xlabel('$x$','interpreter','latex','fontsize',15)
ylabel('$u(x,t)$','interpreter','latex','fontsize',15)
```

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


% main time-stepping loop:

 for n = 1:nsteps
     tnp = tn + k;   % = t_{n+1} %total number of time steps
     tnstar = tn+(k/2); % = t_{n+1/2}
     % 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);
     g0star = utrue(ax,tnstar);
     g1star = utrue(bx,tnstar);
     g1np = utrue(bx,tnp);

     % compute right hand side for linear system:
     uint = u(2:(m+1));   % interior points (unknowns)
     rhsstar = A2*uint;
     % fix-up right hand side using BC's from n to nphalf
     rhsstar(1) = rhsstar(1) + r*(g0n + g0star);
     rhsstar(m) = rhsstar(m) + r*(g1n + g1star);

     % solve linear system:
     uintstar = A1\rhsstar;

     rhsstep = (1/3)*(4*uintstar - uint); %this come from setting up tr-bdf2 and plugging in unphalf
     rhsstep(1) = rhsstep(1) + r2*(g0np);
     rhsstep(m) = rhsstep(m) + r2*(g1np);
```

```

        uint = A3\rhsstep;
        % augment with boundary values:
        u = [g0np; uint; g1np];
        % plot results at desired times:
        if mod(n,nplot)==0 || n==nsteps
            ufine = utrue(xfine,tnp);
            figure(ii+5)
            plot(x,u,'b.-', xfine,ufine,'r')
            legend('Crank-Nicolson','Analytical','interpreter','latex')
            title(sprintf('t = %9.5e  after %4i time steps with %5i grid points',...
                       tnp,n,m+2),'interpreter','latex')
            xlabel('$x$','interpreter','latex','fontsize',15)
            ylabel('$u(x,t)$','interpreter','latex','fontsize',15)
            error = max(abs(u-utrue(x,tnp)));
            errvec = [errvec error];
            fprintf('at time $t$ = %9.5e  max error =  %9.5e\n',tnp,error)
%             if n<nsteps, input('Hit <return> to continue  ');
%             end
        end

     tn = tnp;   % for next time step
    end
end

error_table(hvals, kvals, errvec);   % print tables of errors and ratios

```

```
figure(ii+6)
error_loglog(hvals, errvec);  % produce log-log plot of errors and
    least squares fit
```

```
%function [h,k,error] = heat_FE(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
errvec = [];

mvals = [8 18 38 68 98];
hvals = [];
kvals = [];
for ii=1:length(mvals)
    m = mvals(ii);
    h = (bx-ax)/(m+1);            % h = delta x
```

```
    hvals = [hvals h];
    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 = 24*h^2; %just picking a k=O(h)              % time step
    kvals = [kvals k];


    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(' ')
%         fprintf('WARNING *** k does not divide tfinal, k = %9.5e\n',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:
r = kappa*k/(h^2);
e = ones(m,1);
A = spdiags([e -2*e e], -1:1, 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;

figure(ii)
plot(x,u,'b.-', xfine,ufine,'r')
legend('Forward Euler','Analytical','interpreter','latex')
title(sprintf('Initial data at $t = 0$ for %5i grid points',m+2),'interpreter','latex','fontsize',15)
xlabel('$x$','interpreter','latex','fontsize',15)
ylabel('$u(x,t)$','interpreter','latex','fontsize',15)

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

```
% main time-stepping loop:

for n = 1:nsteps
    tnp = tn + k;   % = t_{n+1} %total number of time steps

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

    g0n = u(1); %this is the initial condition at my first
        spatial grid point
    g1n = u(m+2); %initial condition at my last spatial grid
        point
    g0np = utrue(ax,tnp); %this are the boundary conditions
    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); %this is the only part I am not
         fully sure about
    rhs(m) = rhs(m) + r*(g1n);

    % solve linear system:
    uint = 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);
            figure(ii+5)
            plot(x,u,'b.-', xfine,ufine,'r')
            title(sprintf('t = %9.5e  after %4i time steps with %5i grid points',...
                       tnp,n,m+2),'interpreter','latex')
            xlabel('$x$','interpreter','latex','fontsize',15)
            ylabel('$u(x,t)$','interpreter','latex','fontsize',15)
            error = max(abs(u-utrue(x,tnp)));
            errvec = [errvec error];
            fprintf('at time $t$ = %9.5e  max error =  %9.5e\n',tnp,error)
            if n<nsteps, input('Hit <return> to continue  '); 
            end
        end

      tn = tnp;   % for next time step
    end
end

error_table(hvals, kvals, errvec);   % print tables of errors and ratios

figure(ii+6)
error_loglog(hvals, errvec);  % produce log-log plot of errors and least squares fit
```

```
% function [h,k,error] = HW7Prob5(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
errvec = [];

%mvals = [39 59 99 199];
%mvals = [38 58 98 198];
mvals = [38];
hvals = [];
kvals = [];
for ii=1:length(mvals)
```

```
    m = mvals(ii);
    h = (bx-ax)/(m+1);          % h = delta x
    hvals = [hvals h];
    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 = h; %time step tp get reasonable results                 % time step
    kvals = [kvals k];


    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(' ')
%         fprintf('WARNING *** k does not divide tfinal, k = %9.5e\n',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);
```

```
    utrue = @(x,t) 0.5*erfc(x/sqrt(4*kappa*t));

    % initial conditions:
%       u0 = utrue(x,0);
    u0 = @(x,t) 1*(x<0)+0*(x>=0); %ICs


    % 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], -1:1, 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(x,0);

    figure(ii)
    plot(x,u,'b.-', xfine,ufine,'r')
    legend('Crank-Nicolson','Analytical','interpreter','latex')
    title(sprintf('Initial data at $t = 0$ for %5i grid points',m
        +2),'interpreter','latex','fontsize',15)
```

```
    xlabel('$x$','interpreter','latex','fontsize',15)
    ylabel('$u(x,t)$','interpreter','latex','fontsize',15)

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


    % main time-stepping loop:

    for n = 1:nsteps
        tnp = tn + k;   % = t_{n+1} %total number of time steps

        % 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);
            figure(ii+5)
            plot(x,u,'b.-', xfine,ufine,'r')
            legend('Crank-Nicolson','Analytical','interpreter','latex')
            title(sprintf('t = %9.5e  after %4i time steps with %5i grid points',...
                       tnp,n,m+2),'interpreter','latex')
            xlabel('$x$','interpreter','latex','fontsize',15)
            ylabel('$u(x,t)$','interpreter','latex','fontsize',15)
            error = max(abs(u-utrue(x,tnp)));
            errvec = [errvec error];
            fprintf('at time $t$ = %9.5e  max error =  %9.5e\n',tnp,error)
%             if n<nsteps, input('Hit <return> to continue  ');
%             end
        end

      tn = tnp;   % for next time step
    end
end

error_table(hvals, kvals, errvec);   % print tables of errors and ratios

figure(ii+6)
error_loglog(hvals, errvec);  % produce log-log plot of errors and least squares fit
```