1) Discretizing the advection equation using three different schemes we have:

Upwind:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

Lax-Friedrichs:

$$\frac{u_i^{n+1} - \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right)}{\Delta t} + a \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = 0$$

Lax-Wendroff:

$$1) \quad \frac{u_{i+1/2}^{n+1/2} - \frac{1}{2}\left(u_i^n + u_{i+1}^n\right)}{\frac{1}{2}\Delta t} + a \frac{u_{i+1}^n - u_i^n}{\Delta x} = 0$$

$$2) \quad \frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_{i+1/2}^{n+1/2} - u_{i-1/2}^{n+1/2}}{\Delta x} = 0$$

Using 3 different MATLAB code the following results are achieved.

The error is calculated as the mean error for all nodes: $error = \frac{1}{N}\sum_{i=1}^{N}\left|u(i) - u_{exact}(i)\right|$

Rate of convergence would be:

$$\frac{\log(error_{\max}) - \log(error_{\min})}{\log(h_{\max}) - \log(h_{\min})}$$

Based on the results that are achieved rate of convergence for 3 schemes are as shown in Table 1

Table 1: Rate of convergence for different schemes

Upwind	Lax-Friedrichs	Lax-Wendroff
0.51	0.52	0.59

As results shows the Upwind and Lax-Friedrichs scheme have almost the same rate of convergence, on the other hand Lax-Wendroff scheme show a faster convergence rate.

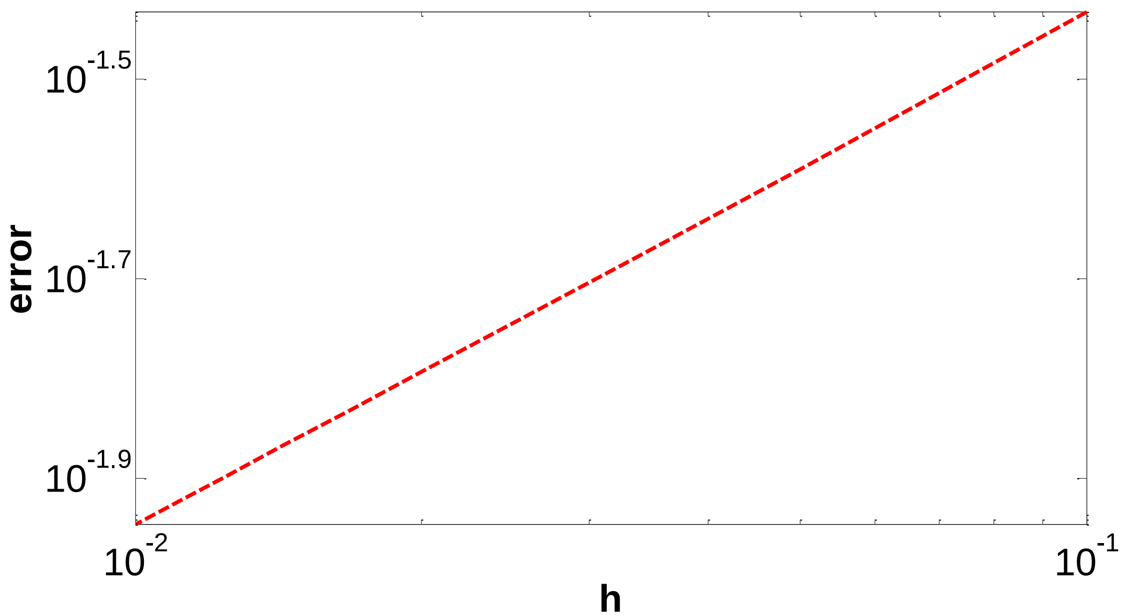

Figure 1: The error vs h for Upwind scheme

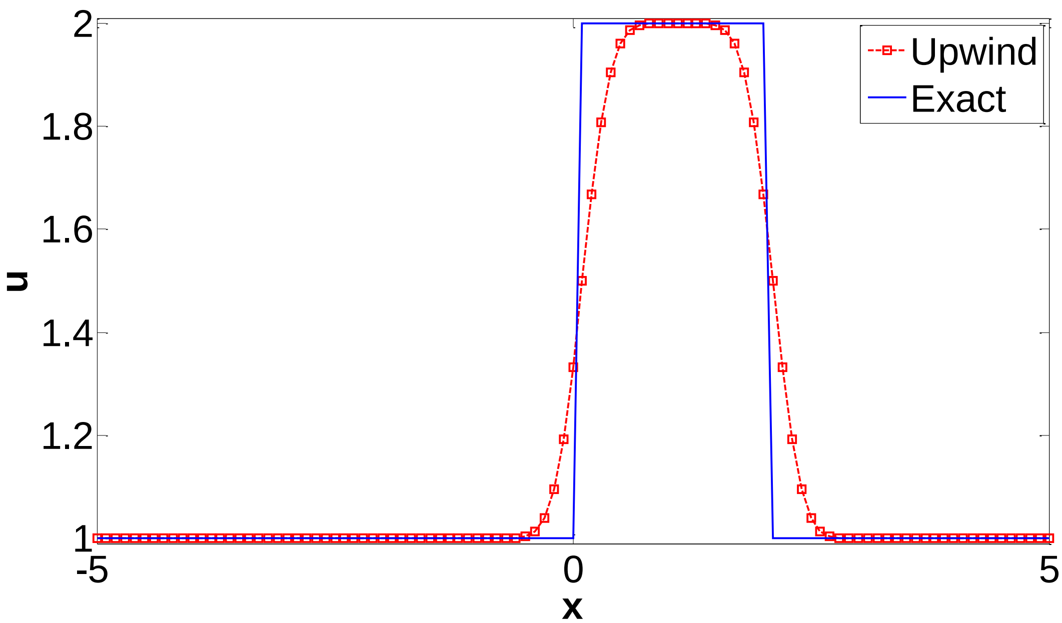

Figure 2: Final result for Upwind scheme using h = 0.1

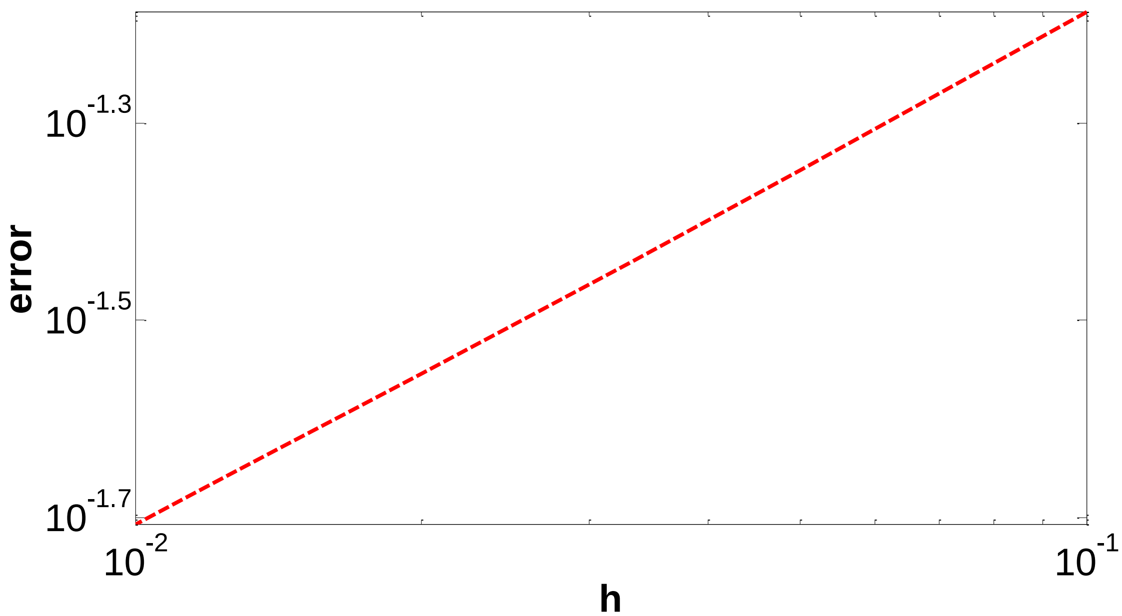

Figure 3: The error vs h for Lax-Friedrichs scheme

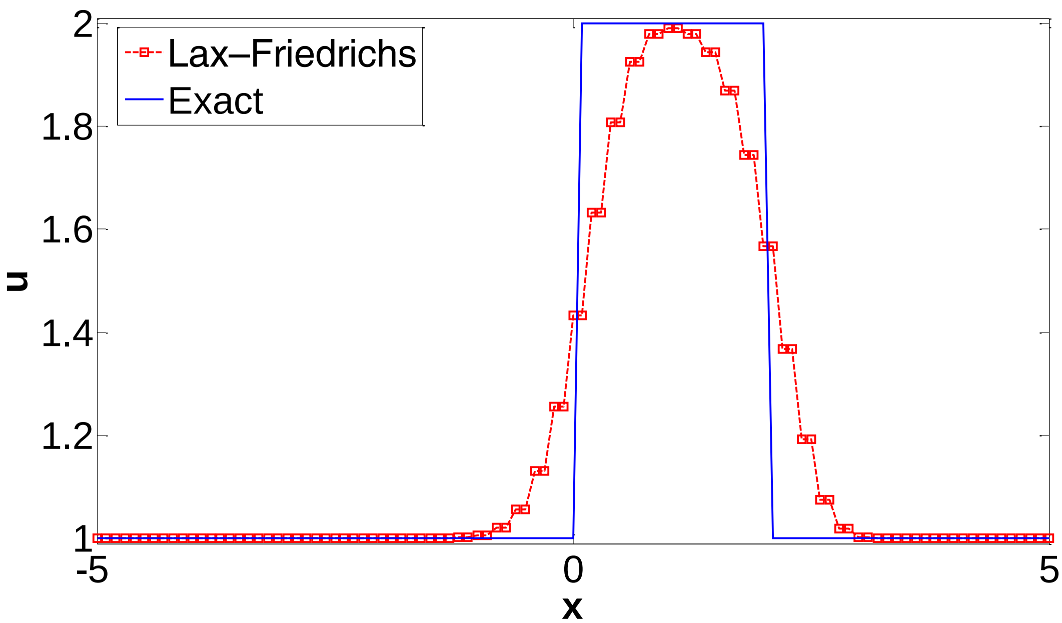

Figure 4: Final result for Lax-Friedrichs scheme using h = 0.1

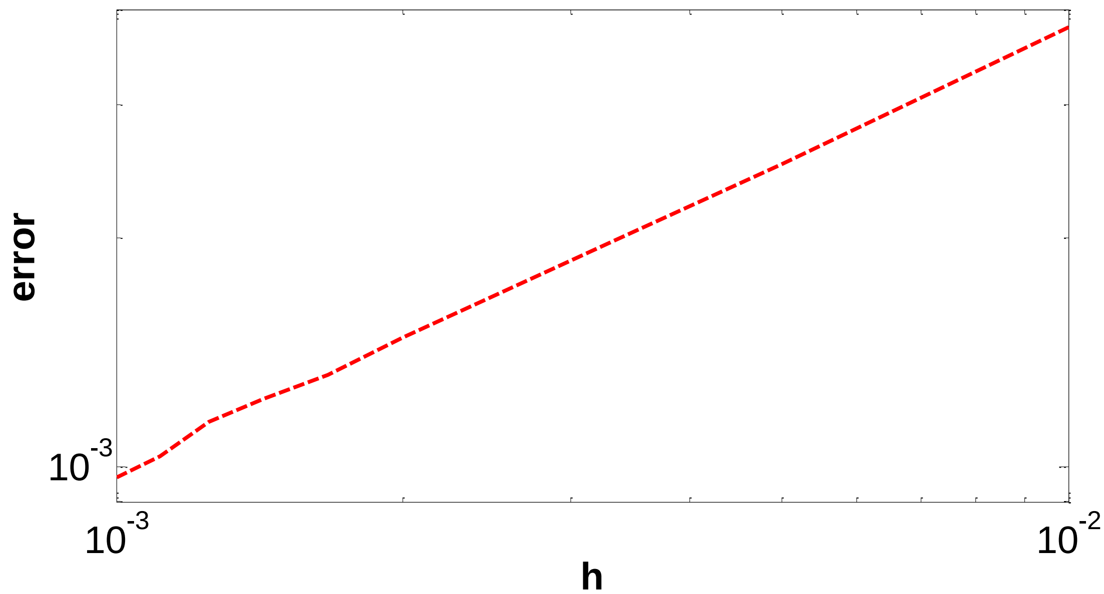

Figure 5: The error vs h for Lax-Wendroff scheme

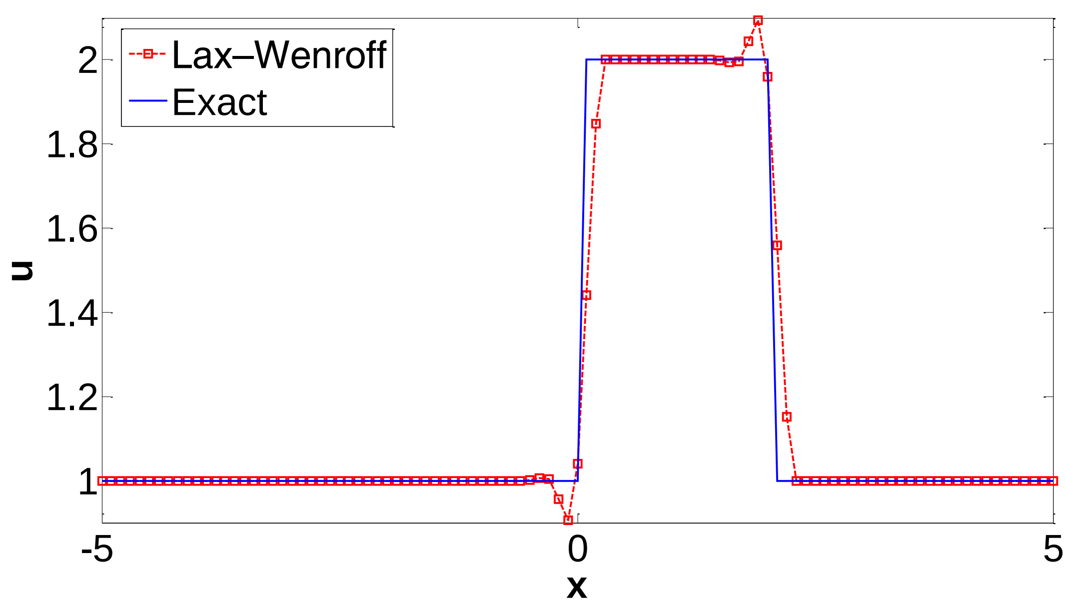

Figure 6: Final result for Lax-Wendroff scheme using h = 0.1

MATLAB Code

Upwind:

```
clc;
clear all;
close all;

a = 1;
j = 0;
for j=100:100:1000
    j
    %j = j+1;
    h = 10./j;
    k = 0.5*h;
    n = j+1;
    x = h*[0:j]-5;
    for i=1:n
        uexact(i) = 1 + H(x(i)+1-1) - H(x(i)-1-1);
    end
    for i=1:n
        u(i) = 1 + H(x(i)+1) - H(x(i)-1);
    end
    uo = u;
    for t = 0:k:1
        for i = 2:n-1
            u(i) = uo(i) - a*k*(uo(i)-uo(i-1))/h;
        end
        uo = u;
    end
    plot(x,u,'--rs','LineWidth',2)
    hold on;
    plot(x,uexact,'-b','LineWidth',2)
    axis([-5 5 0.99 2.01])
    set(gca,'FontSize',30);
    xLabel('x','FontSize',30,'fontweight','b');
    yLabel('u','FontSize',30,'fontweight','b');
    legend('Upwind','Exact');
    hold off
    pause(0.1);
    error(j/100) = sum(abs(u-uexact))/j;
end

figure;
loglog(10./([100:100:1000]),error,'--r','LineWidth',3);
hold on

set(gca,'FontSize',30);
xLabel('h','FontSize',30,'fontweight','b');
yLabel('error','FontSize',30,'fontweight','b');
```

Lax-Friedrichs:

```
clc;
clear all;
close all;

a = 1;
j = 0;
for j=100:100:1000
    j
    h = 10./j;
    k = 0.5*h;
    n = j+1;
    x = h*[0:j]-5;
    for i=1:n
        uexact(i) = 1 + H(x(i)+1-1) - H(x(i)-1-1);
    end
    for i=1:n
        u(i) = 1 + H(x(i)+1) - H(x(i)-1);
    end
    uo = u;
    for t = 0:k:1
        for i = 2:n-1
            u(i) = 0.5*(uo(i+1)+uo(i-1)) - a*k*(uo(i+1)-uo(i-1))/(2*h);
        end
        uo = u;
    end
    plot(x,u,'--rs','LineWidth',2)
    hold on;
    plot(x,uexact,'-b','LineWidth',2)
    hold off;
    axis([-5 5 0.99 2.01])
    set(gca,'FontSize',30);
    xLabel('x','FontSize',30,'fontweight','b');
    yLabel('u','FontSize',30,'fontweight','b');
    pause(0.01);
    legend('Lax-Friedrichs','Exact');
    error(j/100) = sum(abs(u-uexact))/j;
end

figure;
loglog(10./([100:100:1000]),error,'--r','LineWidth',3);
hold on

set(gca,'FontSize',30);
xLabel('h','FontSize',30,'fontweight','b');
yLabel('error','FontSize',30,'fontweight','b');
```

Lax-Wendroff:

```
clc;
clear all;
close all;
a = 1;
j = 0;
for j=1000:1000:10000
    j
    h = 10./j;
    k = 0.9*h;
    n = j+1;
    x = h*[0:j]-5;
    for i=1:n
        uexact(i) = 1 + H(x(i)+1-1) - H(x(i)-1-1);
    end
    for i=1:n
        u(i) = 1 + H(x(i)+1) - H(x(i)-1);
    end
    ut = u;
    uo = u;
    for t = 0:k:1
        for i = 2:n-1
            ut(i) = 0.5*(uo(i+1)+uo(i)) - a*k*(uo(i+1)-uo(i))/(2*h);
        end

        for i = 2:n-1
            u(i) = uo(i) - a*k*(ut(i)-ut(i-1))/h;
        end
        uo = u;
    end
    plot(x,u,'--rs','LineWidth',2)
    hold on;
    plot(x,uexact,'-b','LineWidth',2)
    hold off;
    axis([-5 5 0.99 2.01])
    set(gca,'FontSize',30);
    xLabel('x','FontSize',30,'fontweight','b');
    yLabel('u','FontSize',30,'fontweight','b');
    pause(0.01);
    legend('Lax-Wenroff','Exact');
    error(j/1000) = sum(abs(u-uexact))/j;
end
figure;
loglog(10./([1000:1000:10000]),error,'--r','LineWidth',3);
hold on
set(gca,'FontSize',30);
xLabel('h','FontSize',30,'fontweight','b');
yLabel('error','FontSize',30,'fontweight','b');

function h = H(x)
if x<=0
    h=0;
else
    h=1;
end
```

2)

Repeat Exercise 2 using the Crank-Nicolson method.

SOLUTION: For this problem, we have $\alpha = 1/4$, $m = 3$, $T = 0.1$, $N = 2$, $l = 1$, $h = 1/3$, and $k = 0.05$.

The difference equations are given by

$$\frac{w_{i,j+1} - w_{i,j}}{0.05} - \frac{1}{32}\left[\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{\frac{1}{9}} + \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{\frac{1}{9}}\right] = 0.$$

For $j = 0$, we have, for i =1 and 2,

$$-\frac{9}{640}w_{i-1,1} + \frac{329}{320}w_{i,1} - \frac{9}{640}w_{i+1,1} = \frac{9}{640}w_{i-1,0} + \frac{311}{320}w_{i,0} + \frac{9}{640}w_{i+1,0}.$$

Thus,

$$\begin{aligned}
-\frac{9}{640}w_{0,1} + \frac{329}{320}w_{1,1} - \frac{9}{640}w_{2,1} &= \frac{9}{640}w_{0,0} + \frac{311}{320}w_{1,0} + \frac{9}{640}w_{2,0}\\
-\frac{9}{640}w_{1,1} + \frac{329}{320}w_{2,1} - \frac{9}{640}w_{3,1} &= \frac{9}{640}w_{1,0} + \frac{311}{320}w_{2,0} + \frac{9}{640}w_{3,0}.
\end{aligned}$$

Since $w_{0,0} = w_{3,0} = w_{0,1} = w_{3,1} = 0$, $w_{1,0} = \sqrt{3}$, and $w_{2,0} = -\sqrt{3}$, we have the linear system

$$\begin{bmatrix} \frac{329}{320} & -\frac{9}{640} \\ -\frac{9}{640} & \frac{329}{320} \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{2,1} \end{bmatrix} = \begin{bmatrix} \frac{311}{320}\sqrt{3} - \frac{9}{640}\sqrt{3} \\ \frac{9}{640}\sqrt{3} - \frac{311}{320}\sqrt{3} \end{bmatrix},$$

which has the solution $w_{1,1} = 1.591825$ and $w_{2,1} = -1.591825$.

For $j = 1$, we have, for $i = 1$ and 2,

$$\begin{aligned}
-\frac{9}{640}w_{0,2} + \frac{329}{320}w_{1,2} - \frac{9}{640}w_{2,2} &= \frac{9}{640}w_{0,1} + \frac{311}{320}w_{1,1} + \frac{9}{640}w_{2,1}\\
-\frac{9}{640}w_{1,2} + \frac{329}{320}w_{2,2} - \frac{9}{640}w_{3,2} &= \frac{9}{640}w_{1,1} + \frac{311}{320}w_{2,1} + \frac{9}{640}w_{3,1}.
\end{aligned}$$

Since $w_{0,1} = w_{3,1} = w_{0,2} = w_{3,2} = 0$, we have the linear system

$$\begin{bmatrix} \frac{329}{320} & -\frac{9}{640} \\ -\frac{9}{640} & \frac{329}{320} \end{bmatrix} \begin{bmatrix} w_{1,2} \\ w_{2,2} \end{bmatrix} = \begin{bmatrix} \frac{311}{320}(1.591825) - \frac{9}{640}(1.591825) \\ \frac{9}{640}(1.591825) - \frac{311}{320}(1.591825) \end{bmatrix},$$

which has the solution $w_{1,2} = 1.462951$ and $w_{2,2} = -1.462951$.

The following table summarizes the results.

i	j	x_i	t_j	w_{ij}	$u(x_i, t_j)$
1	1	$\frac{1}{3}$	0.05	1.591825	1.53102
2	1	$\frac{2}{3}$	0.05	−1.591825	−1.53102
1	2	$\frac{1}{3}$	0.1	1.462951	1.35333
2	2	$\frac{2}{3}$	0.1	−1.462951	−1.35333

3) After rearranging the equation we have:

$$\frac{\partial u}{\partial t}=\frac{1}{K}\frac{\partial^2 u}{\partial x^2}+\frac{r}{\rho c}$$

And using Crank-Nicolson we have:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\frac{1}{2(\Delta x)^2 K}\left(\left(u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}\right)+\left(u_{i+1}^n-2u_i^n+u_{i-1}^n\right)\right)+\frac{r}{\rho C}$$

So if we want to create a tridiagonal matrix to solve this system the coefficients are as follows:

	Coefficient
u_{i+1}^{n+1}	$\frac{-1}{2(\Delta x)^2 K}$
u_i^{n+1}	$\frac{1}{\Delta t}+\frac{1}{(\Delta x)^2 K}$
u_{i-1}^{n+1}	$\frac{-1}{2(\Delta x)^2 K}$
Right Hand Side (RHS)	$\frac{u_i^n}{\Delta t}+\frac{1}{2(\Delta x)^2 K}\left(u_{i+1}^n-2u_i^n+u_{i-1}^n\right)+\frac{r}{\rho C}$

A a MATLAB code is written to solve the problem. To solve the tridiagonal matrix a written code from MATLAB website is used that solves the tridiagonal systems of equations. The results for different time are included in Figure 7.

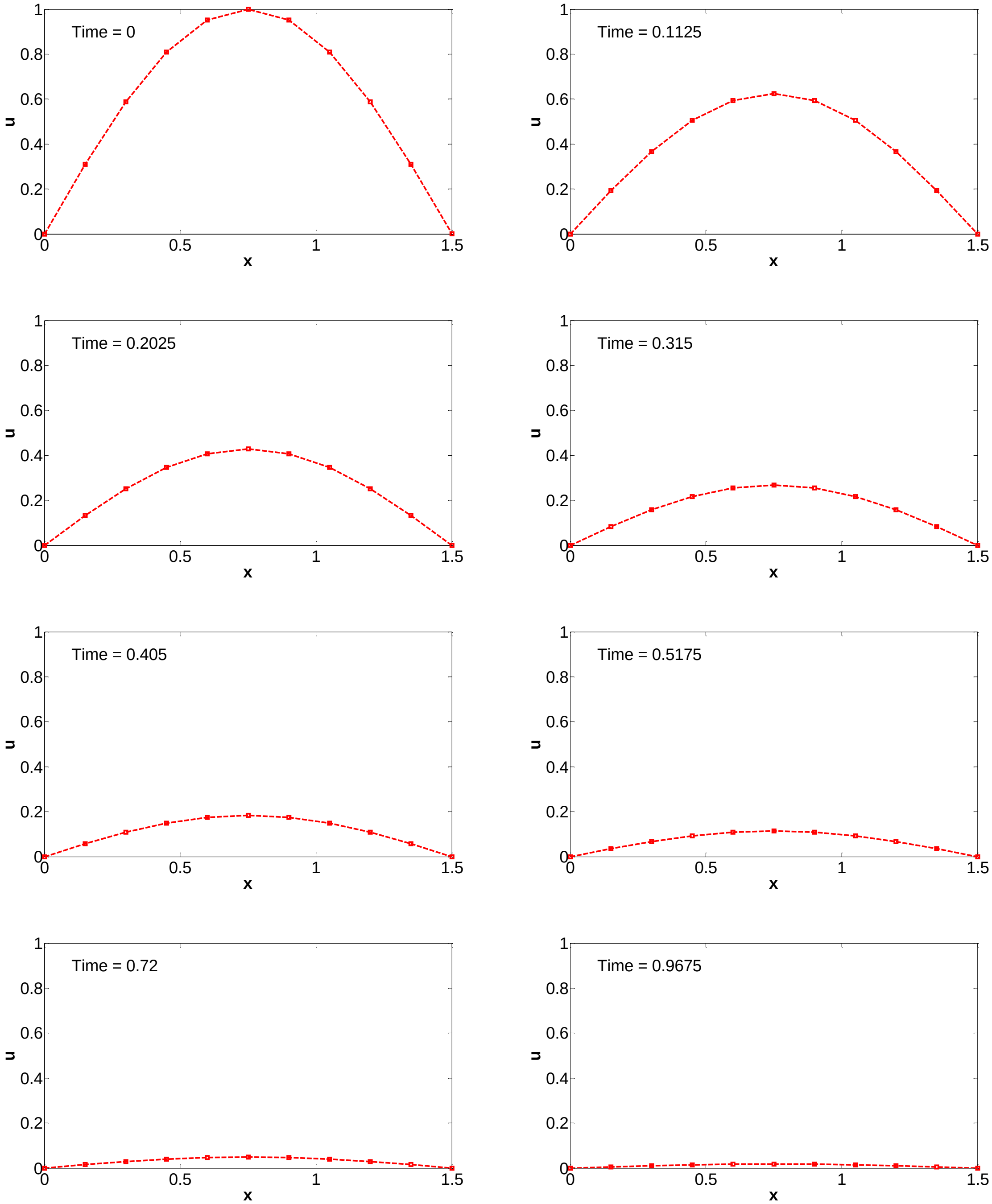

Figure 7: Evolution of u by time

MATLAB Code

```
clc;
clear all;
close all;

j = 10;
l = 1.5;
dx = l/j;
dt = 0.0225;
n = j+1;
x = dx*[0:j];
K = 1.04;
ro = 10.6;
C = 0.056;
r = 5.0;

u = sin(pi*x/l);

a = zeros(n,1);
b = zeros(n-1,1);
c = zeros(n-1,1);
RHS = zeros(n,1);

% Fill in diagonals of matrix A
a(1) = 1; RHS(1) = 0;
a(n) = 1; RHS(n) = 0;
for i = 2:n-1
    a(i) = 1/dt + 1/(dx^2*K);
    b(i) = -1/(2*dx^2*K);
    c(i-1) = -1/(2*dx^2*K);
end
for t = dt:dt:1
    t
    hold off
    plot(x,u,'--rs','LineWidth',3)
    set(gca,'FontSize',30);
    axis([0 l 0 1])
    xLabel('x','FontSize',30,'fontweight','b');
    yLabel('u','FontSize',30,'fontweight','b');
    text(0.1,0.9,['Time = ' num2str(t-dt)],'FontSize',30);
    if (mod(t-dt,0.1)<dt)
        pause;
    end
    pause(0.1);
    for i = 2:n-1
        RHS(i) = u(i)/dt + 1/(2*dx^2*K)*(u(i+1)-2*u(i)+u(i-1));
    end
    u = thomas(a,b,c,RHS);
end
```

Function Thomas from MATLAB Wesite

```
function x = thomas(varargin)
% THOMAS    Solves a tridiagonal linear system
%
%    x = THOMAS(A,d) solves a tridiagonal linear system using the very
efficient
%    Thomas Algorith. The vector x is the returned answer.
%
%       A*x = d;    /  a1  b1   0   0   0   ...   0  \   / x1 \    / d1 \
%                   |  c1  a2  b2   0   0   ...   0  |   | x2 |    | d2 |
%                   |   0  c2  a3  b3   0   ...   0  | x | x3 | =  | d3 |
%                   |   :   :   :   :   :    :    :  |   | x4 |    | d4 |
%                   |   0   0   0   0 cn-2 an-1 bn-1 |   | :  |    |  : |
%                   \   0   0   0   0   0  cn-1  an /    \ xn /    \ dn /
%
%   - The matrix A must be strictly diagonally dominant for a stable
solution.
%   - This algorithm solves this system on (5n-4) multiplications/divisions
and
%      (3n-3) subtractions.
%
%   x = THOMAS(a,b,c,d) where a is the diagonal, b is the upper diagonal, and
c is
%       the lower diagonal of A also solves A*x = d for x. Note that a is
size n
%       while b and c is size n-1.
%       If size(a)=size(d)=[L C] and size(b)=size(c)=[L-1 C], THOMAS solves
the C
%       independent systems simultaneously.
%
%
%   ATTENTION : No verification is done in order to assure that A is a
tridiagonal matrix.
%   If this function is used with a non tridiagonal matrix it will produce
wrong results.
%

[a,b,c,d] = parse_inputs(varargin{:});

% Initialization
m = zeros(size(a));
l = zeros(size(c));
y = zeros(size(d));
n = size(a,1);

%1. LU decomposition
______________________________________________________________________________
___
%
% L = / 1                 \     U =  / m1  r1              \
%     | l1 1              |          |     m2 r2           |
%     |    l2 1           |          |        m3 r3        |
%     |     : : :         |          |         :  :  :     |
%     \           ln-1 1 /           \                  mn /
```

```
%
%  ri = bi -> not necessary
m(1,:) = a(1,:);

y(1,:) = d(1,:); %2. Forward substitution (L*y=d, for y)
____________________________

for i = 2 : n
   i_1 = i-1;
   l(i_1,:) = c(i_1,:)./m(i_1,:);
   m(i,:) = a(i,:) - l(i_1,:).*b(i_1,:);

   y(i,:) = d(i,:) - l(i_1,:).*y(i_1,:); %2. Forward substitution (L*y=d, for
y) ____________________________

end
%2. Forward substitution (L*y=d, for y)
______________________________________________________________
%y(1) = d(1);
%for i = 2 : n
%   y(i,:) = d(i,:) - l(i-1,:).*y(i-1,:);
%end

%3. Backward substitutions (U*x=y, for x)
____________________________________________________________
x(n,:) = y(n,:)./m(n,:);
for i = n-1 : -1 : 1
   x(i,:) = (y(i,:) - b(i,:).*x(i+1,:))./m(i,:);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
function [a,b,c,d] = parse_inputs(varargin)

if nargin == 4
   a = varargin{1};
   b = varargin{2};
   c = varargin{3};
   d = varargin{4};
elseif nargin == 2
   A = sparse(varargin{1});
   a = diag(A);
   b = diag(A,1);
   c = diag(A,-1);
   d = varargin{2};
else
   error('Incorrect number of inputs.')
end
```

4) Using backward Euler for time discretization, centered difference for second order term and upwind for the first order term and using a MATLAB code the results in Figure 8 is achieved. We have the final condition of V at time 20 so we should use a negative time step to march backward in time and find the V at initial time.

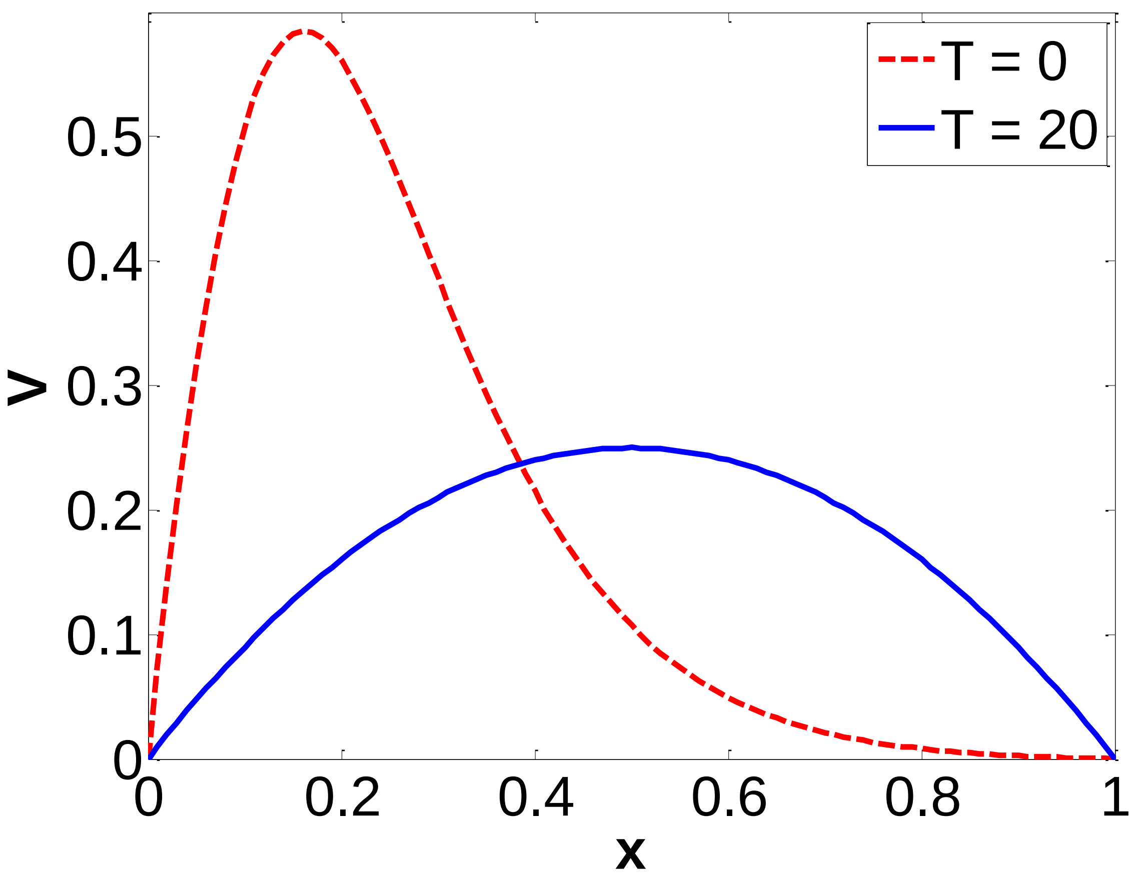

Figure 8: Distribution of V at initial and final time

MATLAB Code

```
clc;
clear all;
close all;
j = 100;
h = 1./j;
k = -0.5*h^2;
n = j+1;
x = h*[0:j];

u = x.*(1-x);
sigma = 1/10;
```

```
r = 1/20;
uo = u;

for t = 20:k:-k
    t
    for i = 2:n-1
        u(i) = uo(i)+k*(-0.5*sigma^2*x(i)^2*(uo(i+1)-2*uo(i)+uo(i-1))/h^2 ...
            -r*uo(i)-r*x(i)*(uo(i)-uo(i-1))/h);
    end
    uo = u;
end

figure;
plot(x,u,'--r','LineWidth',3);
hold on
plot(x,x.*(1-x),'-b','LineWidth',3);
axis([0 1 0 0.6]);
set(gca,'FontSize',30);
xLabel('x','FontSize',30,'fontweight','b');
yLabel('u','FontSize',30,'fontweight','b');
legend('T = 0','T = 20');
```