Assignment 3

Colin Turner: 250956100

MME 9621: Computational Methods in Mechanical Engineering

Dr. Mohammad Z. Hossain

March 18, 2024

Q1)

A. The following depicts the discretized equation with D representing uav acquired from equation 1.

Q1

a) $\mu \dfrac{d^2u}{dy^2} + \dfrac{12\mu D}{H^2} = 0$ $\qquad V_{ac} = D = \dfrac{KH^2}{12\mu} \qquad \Delta y^2 = \left(\dfrac{H}{n-1}\right)^2$

$$\frac{d^2u}{dy^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta y^2}$$

$$\mu\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta y^2}\right) + \frac{12\mu K H^2}{12H^2\mu}$$

$$\mu\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta y^2}\right) + K = 0$$

$$\boxed{u_{i+1} - 2u_i + u_{i-1} = \frac{-K(\Delta y)^2}{\mu} \rightarrow B = \frac{-K(\Delta y)^2}{\mu}}$$

B & C.

The following Figure depicts the boundary conditions applied and the resulting 5 node discretized solutions along with the resulting matrix A,u & b. Note explanation for B value can be found above. At u=0 at y = +- H/2 we determine the boundaries are zero.

b) $u = 0$ at $y = \pm H/2$

$u(H/2) = 0$ & $u(-H/2) = 0$ $\therefore$ $u_0 = 0$ & $u_N = 0$

c) $\underline{i=1}$ $-2u_1 + u_2 = B$ $\quad$ $\underline{i=3}$ $u_2 - 2u_3 + u_4 = B$ $\quad$ $\underline{i=5}$ $-2u_5 + u_4 = B$

$\underline{i=2}$ $u_1 - 2u_2 + u_3 = B$ $\quad$ $\underline{i=4}$ $u_3 - 2u_4 + u_5 = B$

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}; \quad u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix}; \quad b = \begin{bmatrix} B \\ B \\ B \\ B \\ B \end{bmatrix}$$

D & E.

The following code depicts the initialization of parameters, definition of the matrices along with the numerical, analytical & pde4c solver. Note boundary conditions were manually added in line 18 for i=0 & i=6. N was selected as 5 for the initial trials for the code, however later we will see the grid independent solution to increase performance and reduce error.

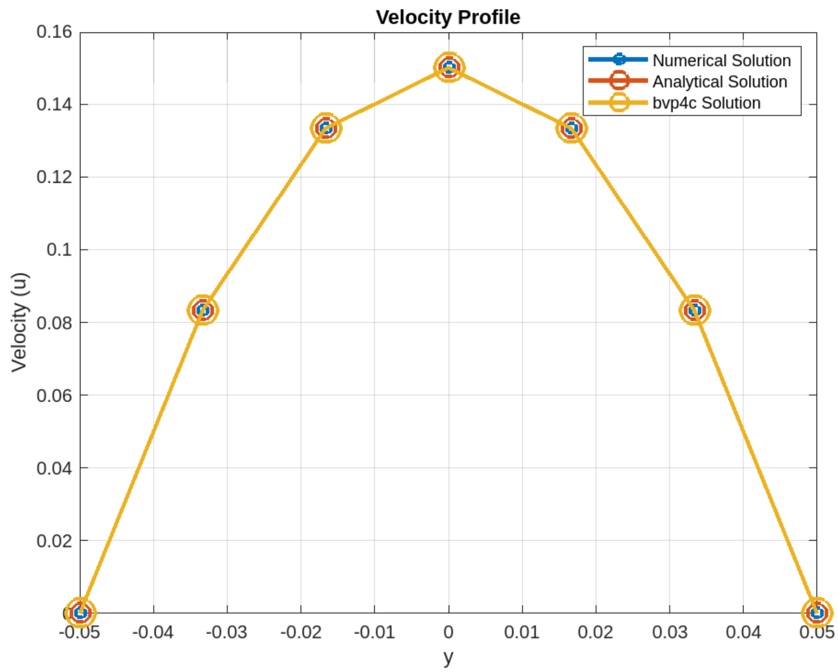
```
1  %% Q1
2  clear all; close all; clc;
3
4  % Given parameters
5  H = 0.1;rho = 1.2;mu = 4e-5;
6  D = 0.1;n =5;K = 12 * D * mu / H^2;
7  delY2 = (H / (n + 1))^2;B = -12*D*delY2/H^2;
8
9  % Define the matrix A
10 A = diag(-2 * ones(n, 1)) + diag(ones(n - 1, 1), 1) + diag(ones(n - 1, 1), -1);
11
12 % Define the vector b
13 b = B * ones(n, 1);
14
15 % Solve the system of equations
16 tic;
17 u_numerical = A \ b;
18 u_numerical = [0; u_numerical; 0];
19 time_numerical = toc;
20
21 % Define the y values corresponding to each node
22 y_values = linspace(-H/2, H/2, n+2)';
23
24 % Analytical solution
25 tic;
26 u_analytical = 3 * D / 2 * (1 - (y_values / (H/2)).^2);
27 time_analytical = toc;
28
29 % bvp4c Solver
30 bvpfcn = @(y,u) [u(2); -12 * D / H^2];
31 bcfcn = @(ua,ub) [ua(1); ub(1)];
32 guess = @(y) [0; 0];
33 solinit = bvpinit(linspace(-H/2, H/2, n+2), guess);
34 tic;
35 sol = bvp4c(bvpfcn, bcfcn, solinit);
36 time_bvp4c = toc;
37 u_bvp4c = deval(sol, y_values);
38
```

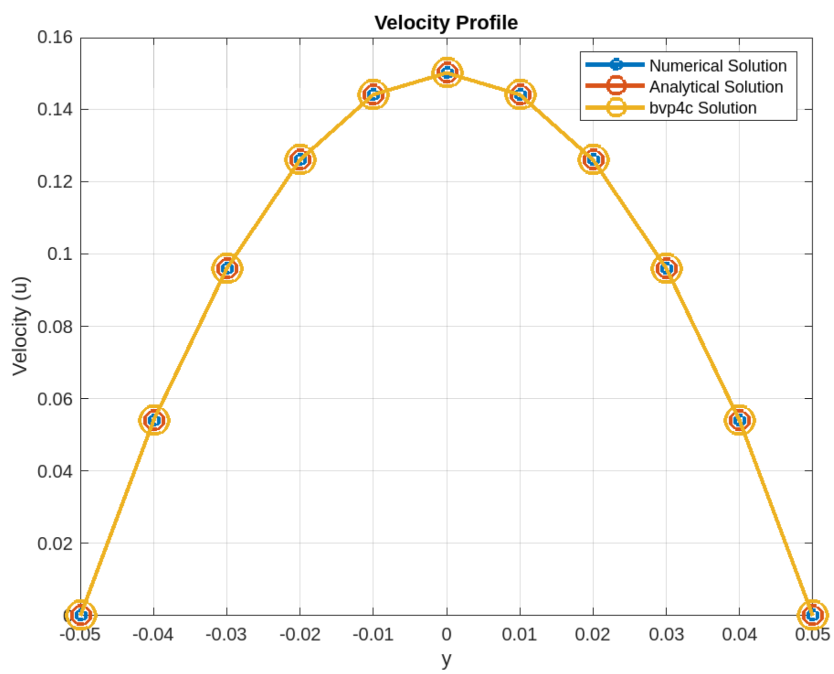
The following code depicts the plotting process for the 3 graphs with varying node sizes for improved visual analysis. Additionall we capture the Max velocities for comparison to the analytical solution. Finally we include a maximum absolute error to determine the accuracy of the solutions in regards to the analytical solution.

```
% Plotting
figure;
plot(y_values, u_numerical, '-o', 'LineWidth', 2, 'MarkerSize', 5, 'DisplayName', 'Numerical Solut
hold on;
plot(y_values, u_analytical, '-o', 'LineWidth', 2, 'MarkerSize', 10, 'DisplayName', 'Analytical So
hold on;
plot(y_values, u_bvp4c(1,:), '-o','LineWidth', 2, 'MarkerSize', 15, 'DisplayName', 'bvp4c Solution
xlabel('y');
ylabel('Velocity (u)');
title('Velocity Profile');
legend;
grid on; hold off;

% Evaluate the velocity at y=0 from the numerical solution
u_max_numerical = u_numerical((n+3)/2);

% Calculate the maximum velocity using the given formula
D_max = 3 * D / 2;

error_numerical = norm(u_numerical - u_analytical, 'inf'); % Maximum absolute error
error_bvp4c = norm(u_bvp4c(1,:) - u_analytical', 'inf'); % Maximum absolute error

% Display results analytical
fprintf('Analytical Method:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_analytical(i));
end
fprintf('Maximum velocity (analytical): %.6f m/s\n', D_max);
fprintf('Computational time: %.6f seconds\n', time_analytical);
% Display results Numerical
fprintf('Numerical Method:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_numerical(i));
end
fprintf('Maximum velocity (numerical): %.6f m/s\n', u_max_numerical);
fprintf('Maximum absolute error: %12.5e\n', error_numerical);
```

```
fprintf('Computational time: %.6f seconds\n', time_numerical);
% Display results bvp4c
fprintf('\nbvp4c Solver:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_bvp4c(1,i));
end
fprintf('Maximum velocity (bvp4c): %.6f m/s\n', u_bvp4c(1,(n+3)/2));
fprintf('Maximum absolute error: %12.5e\n', error_bvp4c);
fprintf('Computational time: %.6f seconds\n', time_bvp4c);
```

The following 2 graphs depicts the output of the previous code. The first figure depicts the use of n =5, while the second solution depicts the grid independent solution utilizing n =9. As we can see both graphs line up almost perfectly however we will see later that there is a slight difference as minor errors arise.





The following to figures depict the terminal outputs for n = 5 and n =9 respectively. As we can see both solutions yield almost identical results however in n = 5 the error in the numerical method is significantly higher than pde4c method, however when n = 9 the error falls to almost the exact same as pde4c. the value of 9 was determined through trial and error until the minimum error was acquired. Additionally, through analysis of the computation costs we can see for n=5 the analytical method is fastest while numerical is a close second and finally pde4c method in last. When n=9 is used we find different computational costs with numerical method produces the fastest results with analytical second and pde4c in last. The is due to the function utilizing a more optimized set of values for computation. additionally pde4c solver utilizes more steps and complex processes to yield results which explains why it produces the slowest results.

```
Command Window
Analytical Method:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.08333
Node 1 Velocity 3 =      0.13333
Node 1 Velocity 4 =      0.15000
Node 1 Velocity 5 =      0.13333
Node 1 Velocity 6 =      0.08333
Node 1 Velocity 7 =      0.00000
Maximum velocity (analytical): 0.150000 m/s
Computational time: 0.001027 seconds
Numerical Method:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.08333
Node 1 Velocity 3 =      0.13333
Node 1 Velocity 4 =      0.15000
Node 1 Velocity 5 =      0.13333
Node 1 Velocity 6 =      0.08333
Node 1 Velocity 7 =      0.00000
Maximum velocity (numerical): 0.150000 m/s
Maximum absolute error:  8.32667e-17
Computational time: 0.026262 seconds

bvp4c Solver:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.08333
Node 1 Velocity 3 =      0.13333
Node 1 Velocity 4 =      0.15000
Node 1 Velocity 5 =      0.13333
Node 1 Velocity 6 =      0.08333
Node 1 Velocity 7 =      0.00000
Maximum velocity (bvp4c): 0.150000 m/s
Maximum absolute error:  2.77556e-17
Computational time: 0.179725 seconds
>>
```

```
Command Window
Analytical Method:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.05400
Node 1 Velocity 3 =      0.09600
Node 1 Velocity 4 =      0.12600
Node 1 Velocity 5 =      0.14400
Node 1 Velocity 6 =      0.15000
Node 1 Velocity 7 =      0.14400
Maximum velocity (analytical): 0.150000 m/s
Computational time: 0.000228 seconds
Numerical Method:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.05400
Node 1 Velocity 3 =      0.09600
Node 1 Velocity 4 =      0.12600
Node 1 Velocity 5 =      0.14400
Node 1 Velocity 6 =      0.15000
Node 1 Velocity 7 =      0.14400
Maximum velocity (numerical): 0.150000 m/s
Maximum absolute error:  2.77556e-17
Computational time: 0.000168 seconds

bvp4c Solver:
Node 1 Velocity 1 =      0.00000
Node 1 Velocity 2 =      0.05400
Node 1 Velocity 3 =      0.09600
Node 1 Velocity 4 =      0.12600
Node 1 Velocity 5 =      0.14400
Node 1 Velocity 6 =      0.15000
Node 1 Velocity 7 =      0.14400
Maximum velocity (bvp4c): 0.150000 m/s
Maximum absolute error:  2.77556e-17
Computational time: 0.082771 seconds
>>
```

Q2)

The following figure depicts the derivation of the recursion formula utilizing explicit method

Q2

a) $$\frac{du}{dt} - \nu\frac{d^2u}{dy^2} = 0 \qquad t=0 \begin{cases} u=u_0, & \text{at } y=0 \\ u=0, & \text{at } 0<y\leq h \end{cases}$$

$$u(y,\infty)=u_0(1-y/h) \qquad t>0 \begin{cases} u=u_0, & \text{at } y=0 \\ u=0, & \text{at } y=h \end{cases}$$

i) Explicit

$$\beta = \frac{\nu(\Delta t)}{(\Delta y)^2}$$

$$\frac{du}{dt} = \frac{u_i^{n+1}-u_i^n}{\Delta t}$$

$$\frac{d^2u}{dy^2} = \frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{(\Delta y)^2}$$

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \nu\left[\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{(\Delta y)^2}\right]$$

$$u_i^{n+1} = u_i^n + \frac{\nu(\Delta t)}{(\Delta y)^2}\left[u_{i+1}^n-2u_i^n+u_{i-1}^n\right]$$

$$u_i^{n+1} = u_i^n + \beta\left[u_{i+1}^n-2u_i^n+u_{i-1}^n\right]$$

$$u_i^{n+1} = u_i^n + \beta u_{i+1}^n - 2\beta u_i^n + \beta u_{i-1}^n$$

$$\boxed{u_i^{n+1} = \beta u_{i+1}^n + (1-2\beta)u_i^n + \beta u_{i-1}^n}$$

The following figure depicts the derivation of the recursion formula utilizing Implicit method

ii) Implicit $\qquad \beta = \frac{\nu(\Delta t)}{(\Delta y)^2}$

$$\frac{du}{dt} = \frac{u_i^n - u_i^{n-1}}{\Delta t} + O(\Delta t)$$

$$\frac{d^2u}{dy^2} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta y)^2} + O(\Delta y^2)$$

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = \nu\left[\frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta y)^2}\right]$$

$$u_i^n = u_i^{n+1} + \beta\left[u_{i-1}^n - 2u_i^n + u_{i+1}^n\right]$$

$$\boxed{-u_i^{n-1} = \beta u_{i-1}^n - (1+2\beta)u_i^n + \beta u_{i+1}^n}$$

Matrix Form

$$Au^n = -u^{n-1} - \beta S^n$$

The following figure depicts the derivation of the recursion formula utilizing C-N Method

iii) C-N $\qquad \beta = \frac{\nu(\Delta t)}{(\Delta y)^2}$

$$\frac{du}{dt} = \frac{u_i^n - u_i^{n-1}}{\Delta t}$$

$$\frac{d^2u}{dy^2} = \frac{1}{2}\left(\frac{d^2u}{dy^2}\Big|^n + \frac{d^2u}{dy^2}\Big|^{n-1}\right) = \frac{1}{2(\Delta y)^2}\left[(u_{i-1}^n - 2u_i^n + u_{i+1}^n) + (u_{i-1}^{n-1} - 2u_i^{n-1} + u_{i+1}^{n-1})\right]$$

$$2u_i^n - 2u_i^{n-1} = \beta\left[u_{i-1}^n - 2u_i^n + u_{i+1}^n + u_{i-1}^{n-1} - 2u_i^{n-1} + u_{i+1}^{n-1}\right]$$

$$2u_i^n - 2u_i^{n-1} = \beta u_{i-1}^n - 2\beta u_i^n + \beta u_{i+1}^n + \beta u_{i-1}^{n-1} - 2\beta u_i^{n-1} + \beta u_{i+1}^{n-1}$$

$$2u_i^n + 2\beta u_i^n - \beta u_{i-1}^n - \beta u_{i+1}^n = 2u_i^{n-1} - 2\beta u_i^{n-1} + \beta u_{i-1}^{n-1} + \beta u_{i+1}^{n-1}$$

$$\boxed{\beta u_{i-1}^n - 2(1+\beta)u_i^n + \beta u_{i+1}^n = -\beta u_{i-1}^{n-1} - 2(1-\beta)u_i^{n-1} - \beta u_{i+1}^{n-1}}$$

The following figure depicts the 3 recursive formulas derived from each method respectively.

i) $u_i^{n+1} = \beta u_{i+1}^n + (1-2\beta)u_i^n + \beta u_{i-1}^n$

ii) $-u_i^{n-1} = \beta u_{i-1}^n - (1+2\beta)u_i^n + \beta u_{i+1}^n$

iii) $\beta u_{i-1}^n - 2(1+\beta)u_i^n + \beta u_{i+1}^n = -\beta u_{i-1}^{n-1} - 2(1-\beta)u_i^{n-1} - \beta u_{i+1}^{n-1}$

The following figure depicts the derived matrices for explicit and implicit method.

i) $M=5$

$$\begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \end{bmatrix} = \begin{bmatrix} 1-2\beta & \beta & 0 & 0 \\ \beta & 1-2\beta & \beta & 0 \\ 0 & \beta & 1-2\beta & \beta \\ 0 & 0 & \beta & 1-2\beta \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \end{bmatrix} + \begin{bmatrix} \beta u_0 \\ 0 \\ 0 \\ \beta u_5 \end{bmatrix}$$

ii) $M=5 \quad Au^n = -u^{n-1} - \beta S^n$

$$\begin{bmatrix} -(1+2\beta) & \beta & 0 & 0 \\ \beta & -(1+2\beta) & \beta & 0 \\ 0 & \beta & -(1+2\beta) & \beta \\ 0 & 0 & \beta & -(1+2\beta) \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \end{bmatrix} = \begin{bmatrix} -u_1^{n-1} \\ -u_2^{n-1} \\ -u_3^{n-1} \\ -u_4^{n-1} \end{bmatrix} - \beta \begin{bmatrix} u_0^n \\ 0 \\ 0 \\ u_5^n \end{bmatrix}$$

The following figure depicts the derived matrices for C-N method.

iii) $M=5 \quad AT^n = BT^{n-1} - \beta(S^{n-1}+S^n)$

$$\begin{bmatrix} -2(1-\beta) & -\beta & 0 & 0 \\ -\beta & -2(1-\beta) & -\beta & 0 \\ 0 & -\beta & -2(1-\beta) & -\beta \\ 0 & 0 & -\beta & -2(1-\beta) \end{bmatrix} \begin{bmatrix} u_1^{n-1} \\ u_2^{n-1} \\ u_3^{n-1} \\ u_4^{n-1} \end{bmatrix} - \beta \begin{bmatrix} u_0^{n-1}+u_0^n \\ 0 \\ 0 \\ u_4^{n-1}+u_4^n \end{bmatrix} = \begin{bmatrix} -2(1+\beta) & \beta & 0 & 0 \\ \beta & -2(1+\beta) & \beta & 0 \\ 0 & \beta & -2(1+\beta) & \beta \\ 0 & 0 & \beta & -2(1+\beta) \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \end{bmatrix}$$

The following code depicts the initialization steps in the code and setting the boundary conditions and implementing the solutions along with tracking the Comp time. Along with the pdepe solver function and its application to solve the defined problem

```
%% Q2

clc; clear all; close all;

% Given parameters
delt=0.01; v=0.000217; h=40e-3;
yL=0; time_final=2; my = 5;
dely=h/my; u0 = 40;
nt=time_final/delt;dely2=dely*dely;
beta=v*delt/dely2;
Tini = 4e-2;
y = linspace(0, Tini, 6);
t = linspace(0, 2, 200);
xmt = y';
uM=0;
f_initial=@(y) (0);

i=1:(my-1);
u_initial(i)=f_initial(y(i+1));
% explicit method
tic;
[usol_1]=Explicit_1D(beta,u0,uM,my,nt,u_initial);
explicit_time = toc;
% implicit method
tic;
[usol_2]=Implicit_1D(beta,u0,uM,my,nt,u_initial);
implicit_time = toc;
% crank-nicolson method
tic;
[usol_3]=CN_1D(beta,u0,uM,my,nt,u_initial);
C_N_time = toc;
% pdepe solver
tic
sol = pdepe(0, @pdefun1, @pdeIC1, @pdeBC1, y, t, [], v, Tini, u0);
time_pdepe = toc;

fprintf('Comp Time %es\n', time_pdepe);
fprintf('Explicit Method Comp Time: %fs\n', explicit_time);
fprintf('Implicit Method Comp Time: %fs\n', implicit_time);
```

The following code depicts the plotting process for the graphs with the accuracy comparison graph. This accuracy comparison graph allows us to find the n value that provides the lowest error resulting in grid independence.

```
fprintf('Crank-Nicolson Method Comp Time: %fs\n', C_N_time);

% Boundary Points
for jj=2:nt
    u_1(:,jj)=[u0;usol_1(:,jj);uM];
    u_2(:,jj)=[u0;usol_2(:,jj);uM];
    u_3(:,jj)=[u0;usol_2(:,jj);uM];
end

% plotting explicit method
figure(1);
plot(u_1(:,10),y,'-o',u_1(:,30),y,'-o',u_1(:,60),y,'-o',u_1(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Explicit method');

% plotting implicit method
figure(2);
plot(u_2(:,10),y,'-o',u_2(:,30),y,'-o',u_2(:,60),y,'-o',u_2(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Implicit method');

% plotting C-N method
figure(3);
plot(u_3(:,10),y,'-o',u_3(:,30),y,'-o',u_3(:,60),y,'-o',u_3(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Crank-Nicolson method');

figure(4);
plot(sol(10,:), xmt, '-o', sol(30,:), xmt, '-o', sol(60,:), xmt, '-o', sol(end,:), xmt, '-o');
xlim([0, u0]);
legend('t=0.1', 't=0.3', 't=0.6', 't=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
```

The following figure depicts the code utilized to determine the accuracy of the systems.

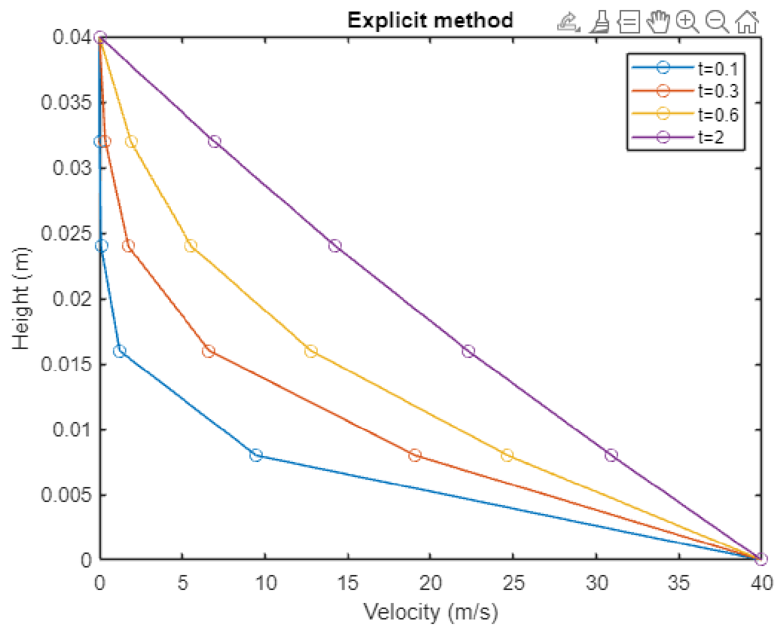
```matlab
%Accuracy comparison
figure(5);
timesteps = [10, 30, 60, 200];
for idx = 1:length(timesteps)
    subplot(2, 2, idx);
    hold on;
    plot(u_1(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 5);
    plot(u_2(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 10);
    plot(u_3(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 15);
    hold off;
    title(['Timestep: ', num2str(timesteps(idx))]);
    legend('Explicit', 'Implicit', 'Crank-Nicolson');
    xlabel('Velocity (m/s)'); ylabel('Height (m)');
end

% error calculation at each point compared to pdepe solver
u_1error(1) = norm(u_1(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_2error(1) = norm(u_2(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_3error(1) = norm(u_3(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_1error(2) = norm(u_1(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_2error(2) = norm(u_2(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_3error(2) = norm(u_3(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_1error(3) = norm(u_1(:,end) - sol(end,:), 'inf'); % Maximum absolute error
u_2error(3) = norm(u_2(:,end) - sol(end,:), 'inf'); % Maximum absolute error
u_3error(3) = norm(u_3(:,end) - sol(end,:), 'inf'); % Maximum absolute error

fprintf('Error Explicit Method\n');
for i=1:1:3
%formatpec = '%5f\n';
fprintf('%5f\n',u_1error(i));
end
fprintf('Error Implicit Method\n');
for i=1:1:3
%formatpec = '%5f\n';
fprintf('%5f\n',u_2error(i));
end
fprintf('Crank-Nicolson Method\n');
for i=1:1:3
%formatpec = '%5f\n';
```

The following 2 figures depicts the resulting functions that were created.

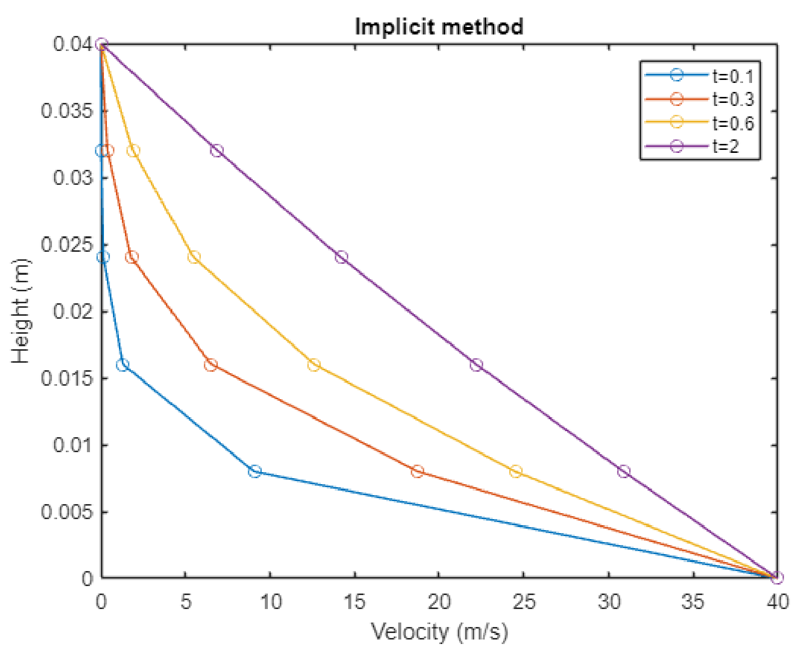
```matlab
% explicit method function
function [u]=Explicit_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1);
u(:,1)=u_initial';
S=[u0; zeros(my-3,1); uM]; % BC
for i=1:my-1
    A(i,i)=1-2*beta;
    if i<my-1
        A(i,i+1)=beta;
    end
    if i>=2
        A(i,i-1)=beta;
    end
end
for n=2:nt % time loop
    u(:,n)=A*u(:,n-1)+beta*S;
end
end

% implicit method function
function [u]=Implicit_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1);
u(:,1)=u_initial';
S=[u0; zeros(my-3,1); uM]; %BC
for i=1:my-1
    A(i,i)=-(1+2*beta);
    if i<my-1
        A(i,i+1)=beta;
    end
    if i>=2
        A(i,i-1)=beta;
    end
end
for n=2:nt % time loop
    b=-u(:,n-1)-beta*S;
    u(:,n)=A\b;
end
end
```

```matlab
% crank nicolson method funciton defined
function [u]=CN_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1); B=A;
u(:,1)=u_initial';
S=[u0; zeros(my-3,1); uM]; %BC
for i=1:my-1
    A(i,i)=-2*(1+beta);
    B(i,i)=-2*(1-beta);
    if i<my-1
        A(i,i+1)=beta;
        B(i,i+1)=-beta;
    end
    if i>=2
        A(i,i-1)=beta;
        B(i,i-1)=-beta;
    end
end
for n=2:nt %time loop
    b=B*u(:,n-1)-beta*(S+S);
    u(:,n)=A\b;
end
end
function [c, f, s] = pdefun1(y, t, u, dudy, v, Tini, u0)
    c = 1/v;
    f = dudy;
    s = 0;
end
function u0 = pdeIC1(y, v, Tini, u0)
    u0 = zeros(size(y));
    u0(1) = u0;
end
function [pl, ql, pr, qr] = pdeBC1(yL, uL, yR, uR, t, v, Tini, u0)
    pl = uL - u0;   ql = 0;
    pr = uR;        qr = 0;
end
```

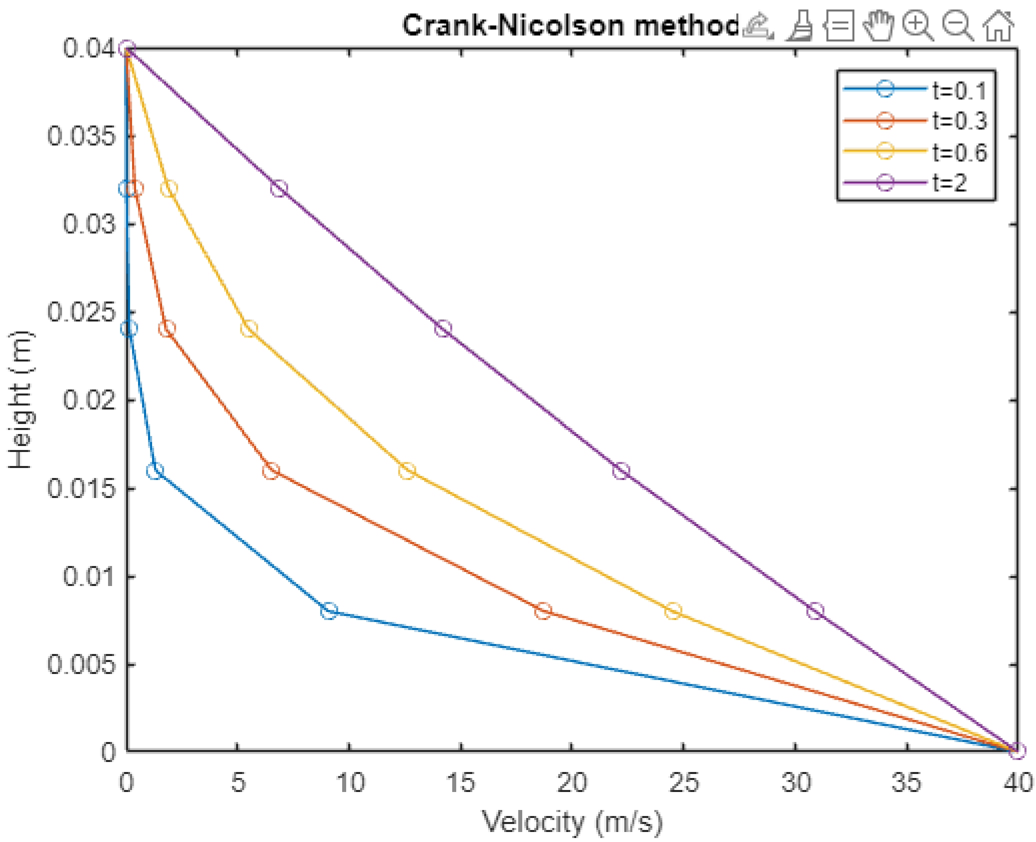
The following figure depicts the resulting graph from Explicit method.



The following figure depicts the resulting graph from Implicit method.

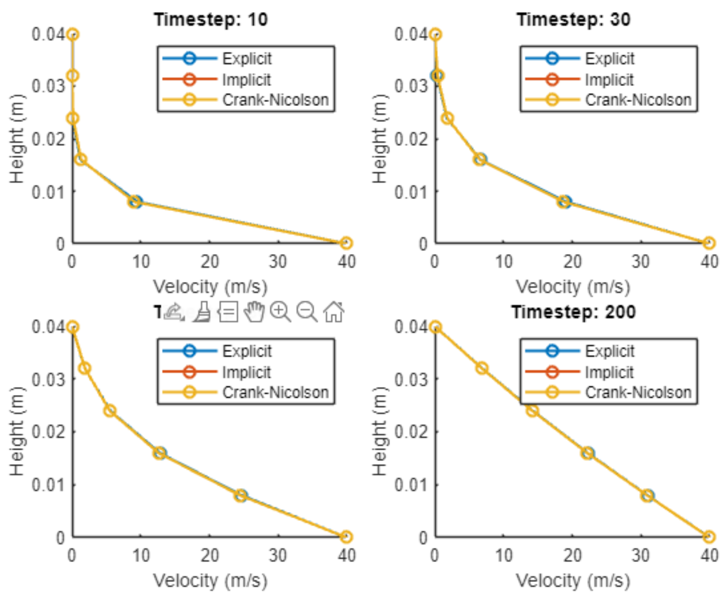


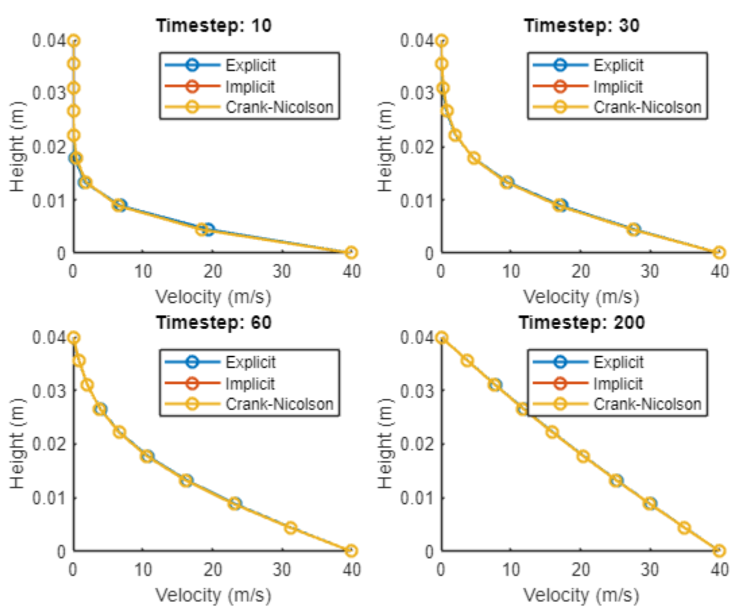
The following figure depicts the resulting graph from C-N method.



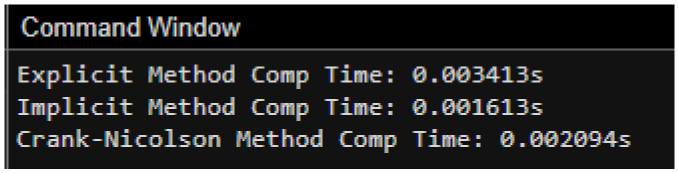
Additionally, it can be seen that the three provided solutions yield almost identical results, however they tend to vary slightly.

The following two figures depicts the resulting accuracy per time step, this graph allows us to change n to obtain a grid independent solution. We can see that the graphs do not match the previous graphs for any method. However, in the second graph we can see that by utilizing n =9 we obtain a more accurate and efficient process.

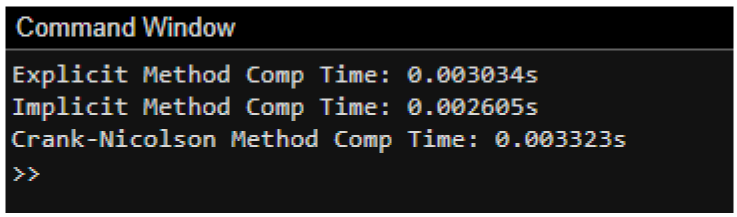




In the following figures we can see that when n=5 C-N method yields the fastest results with Implicit method second and explicit method last, when n=9 Implicit method yields the fastest results with Explicit in second and finally C-N method in last.

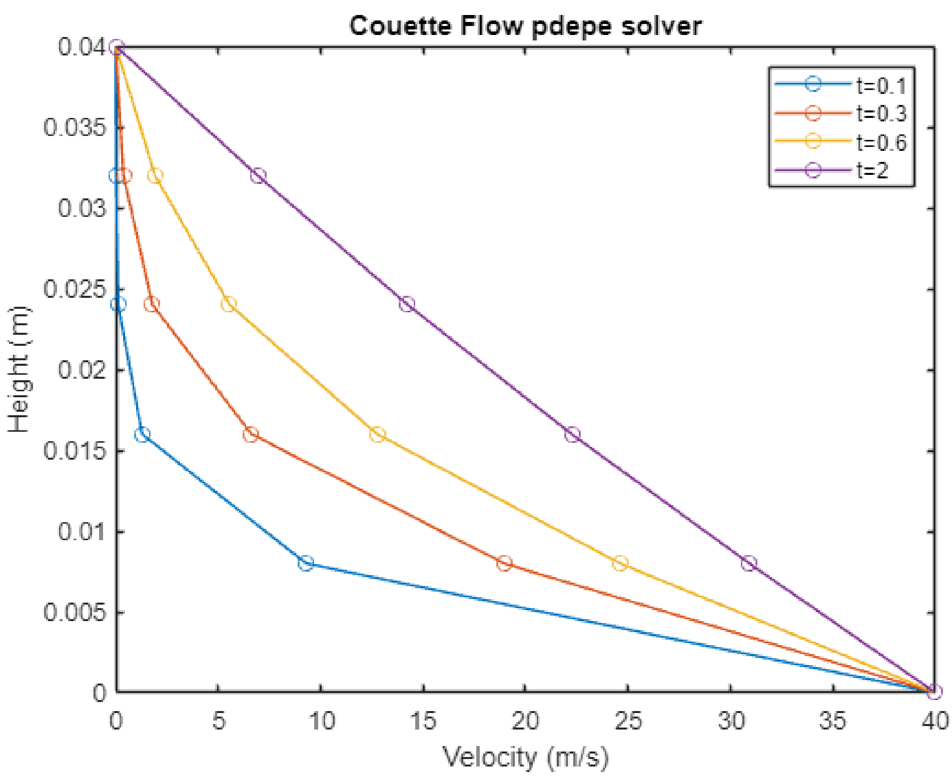


```
Command Window
Explicit Method Comp Time: 0.003413s
Implicit Method Comp Time: 0.001613s
Crank-Nicolson Method Comp Time: 0.002094s
```



```
Command Window
Explicit Method Comp Time: 0.003034s
Implicit Method Comp Time: 0.002605s
Crank-Nicolson Method Comp Time: 0.003323s
>>
```

The following figure depicts the use of MATLAB pdepe.



Finally we can see the computational time for this method, which is the slowest of the other methods however yields the most accurate results. This is due to the complex nature of the matlab function. The error resulting from the outputs shows that all 3 methods are significantly less accurate than pdepe method.

```
Comp Time 1.312370e-01s
Explicit Method Comp Time: 0.001658s
Implicit Method Comp Time: 0.001673s
Crank-Nicolson Method Comp Time: 0.002078s
Error Explicit Method
172.269799
172.269799
125.682237
Error Implicit Method
172.269799
172.269799
125.682237
Crank-Nicolson Method
172.269799
172.269799
125.682237
```

Appendix

```
%% Q1
clear all; close all; clc;

% Given parameters
H = 0.1;rho = 1.2;mu = 4e-5;
D = 0.1;n =9;K = 12 * D * mu / H^2;
delY2 = (H / (n + 1))^2;B = -12*D*delY2/H^2;

% Define the matrix A
A = diag(-2 * ones(n, 1)) + diag(ones(n - 1, 1), 1) + diag(ones(n - 1, 1), -1);

% Define the vector b
b = B * ones(n, 1);

% Solve the system of equations
tic;
u_numerical = A \ b;
u_numerical = [0; u_numerical; 0];
time_numerical = toc;

% Define the y values corresponding to each node
y_values = linspace(-H/2, H/2, n+2)';

% Analytical solution
tic;
u_analytical = 3 * D / 2 * (1 - (y_values / (H/2)).^2);
time_analytical = toc;

% bvp4c Solver
bvpfcn = @(y,u) [u(2); -12 * D / H^2];
bcfcn = @(ua,ub) [ua(1); ub(1)];
guess = @(y) [0; 0];
solinit = bvpinit(linspace(-H/2, H/2, n+2), guess);
tic;
sol = bvp4c(bvpfcn, bcfcn, solinit);
time_bvp4c = toc;
u_bvp4c = deval(sol, y_values);

% Plotting
figure;
plot(y_values, u_numerical, '-o', 'LineWidth', 2, 'MarkerSize', 5, 'DisplayName',
'Numerical Solution');
hold on;
plot(y_values, u_analytical, '-o', 'LineWidth', 2, 'MarkerSize', 10, 'DisplayName',
'Analytical Solution');
hold on;
plot(y_values, u_bvp4c(1,:), '-o','LineWidth', 2, 'MarkerSize', 15, 'DisplayName',
'bvp4c Solution');
xlabel('y');
ylabel('Velocity (u)');
title('Velocity Profile');
legend;
```

```
grid on; hold off;

% Evaluate the velocity at y=0 from the numerical solution
u_max_numerical = u_numerical((n+3)/2);

% Calculate the maximum velocity using the given formula
D_max = 3 * D / 2;

error_numerical = norm(u_numerical - u_analytical, 'inf'); % Maximum absolute error
error_bvp4c = norm(u_bvp4c(1,:) - u_analytical', 'inf'); % Maximum absolute error

% Display results analytical
fprintf('Analytical Method:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_analytical(i));
end
fprintf('Maximum velocity (analytical): %.6f m/s\n', D_max);
fprintf('Computational time: %.6f seconds\n', time_analytical);
% Display results Numerical
fprintf('Numerical Method:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_numerical(i));
end
fprintf('Maximum velocity (numerical): %.6f m/s\n', u_max_numerical);
fprintf('Maximum absolute error: %12.5e\n', error_numerical);
fprintf('Computational time: %.6f seconds\n', time_numerical);
% Display results bvp4c
fprintf('\nbvp4c Solver:\n');
for i = 1:1:7
    fomatSpec='Node 1 Velocity %d = %12.5f\n';
    fprintf(fomatSpec,i,u_bvp4c(1,i));
end
fprintf('Maximum velocity (bvp4c): %.6f m/s\n', u_bvp4c(1,(n+3)/2));
fprintf('Maximum absolute error: %12.5e\n', error_bvp4c);
fprintf('Computational time: %.6f seconds\n', time_bvp4c);
%% Q2

clc; clear all; close all;

% Given parameters
delt=0.01; v=0.000217; h=40e-3;
yL=0; time_final=2; my = 5;
dely=h/my; u0 = 40;
nt=time_final/delt;dely2=dely*dely;
beta=v*delt/dely2;
Tini = 4e-2;
y = linspace(0, Tini, 6);
t = linspace(0, 2, 200);
xmt = y';
uM=0;
f_initial=@(y) (0);

i=1:(my-1);
```

```matlab
u_initial(i)=f_initial(y(i+1));
% explicit method
tic;
[usol_1]=Explicit_1D(beta,u0,uM,my,nt,u_initial);
explicit_time = toc;
% implicit method
tic;
[usol_2]=Implicit_1D(beta,u0,uM,my,nt,u_initial);
implicit_time = toc;
% crank-nicolson method
tic;
[usol_3]=CN_1D(beta,u0,uM,my,nt,u_initial);
C_N_time = toc;
% pdepe solver
tic
sol = pdepe(0, @pdefun1, @pdeIC1, @pdeBC1, y, t, [], v, Tini, u0);
time_pdepe = toc;

fprintf('Comp Time %es\n', time_pdepe);
fprintf('Explicit Method Comp Time: %fs\n', explicit_time);
fprintf('Implicit Method Comp Time: %fs\n', implicit_time);
fprintf('Crank-Nicolson Method Comp Time: %fs\n', C_N_time);

% Boundary Points
for jj=2:nt
    u_1(:,jj)=[u0;usol_1(:,jj);uM];
    u_2(:,jj)=[u0;usol_2(:,jj);uM];
    u_3(:,jj)=[u0;usol_2(:,jj);uM];
end

% plotting explicit method
figure(1);
plot(u_1(:,10),y,'-o',u_1(:,30),y,'-o',u_1(:,60),y,'-o',u_1(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Explicit method');

% plotting implicit method
figure(2);
plot(u_2(:,10),y,'-o',u_2(:,30),y,'-o',u_2(:,60),y,'-o',u_2(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Implicit method');

% plotting C-N method
figure(3);
plot(u_3(:,10),y,'-o',u_3(:,30),y,'-o',u_3(:,60),y,'-o',u_3(:,end),y,'-o');
legend('t=0.1','t=0.3','t=0.6','t=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Crank-Nicolson method');

figure(4);
```

```
plot(sol(10,:), xmt, '-o', sol(30,:), xmt, '-o', sol(60,:), xmt, '-o', sol(end,:),
xmt, '-o');
xlim([0, u0]);
legend('t=0.1', 't=0.3', 't=0.6', 't=2');
xlabel('Velocity (m/s)');
ylabel('Height (m)');
title('Couette Flow pdepe solver');

%Accuracy comparison
figure(5);
timesteps = [10, 30, 60, 200];
for idx = 1:length(timesteps)
    subplot(2, 2, idx);
    hold on;
    plot(u_1(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 5);
    plot(u_2(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 10);
    plot(u_3(:, timesteps(idx)), y, '-o', 'LineWidth', 1.5,'MarkerSize', 15);
    hold off;
    title(['Timestep: ', num2str(timesteps(idx))]);
    legend('Explicit', 'Implicit', 'Crank-Nicolson');
    xlabel('Velocity (m/s)'); ylabel('Height (m)');
end

% error calculation at each point compared to pdepe solver
u_1error(1) = norm(u_1(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_2error(1) = norm(u_2(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_3error(1) = norm(u_3(:,10) - sol(30,:), 'inf'); % Maximum absolute error
u_1error(2) = norm(u_1(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_2error(2) = norm(u_2(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_3error(2) = norm(u_3(:,30) - sol(30,:), 'inf'); % Maximum absolute error
u_1error(3) = norm(u_1(:,end) - sol(end,:), 'inf'); % Maximum absolute error
u_2error(3) = norm(u_2(:,end) - sol(end,:), 'inf'); % Maximum absolute error
u_3error(3) = norm(u_3(:,end) - sol(end,:), 'inf'); % Maximum absolute error

fprintf('Error Explicit Method\n');
for i=1:1:3
%formatpec = '%5f\n';
fprintf('%5f\n',u_1error(i));
end
fprintf('Error Implicit Method\n');
for i=1:1:3
%formatpec = '%5f\n';
fprintf('%5f\n',u_2error(i));
end
fprintf('Crank-Nicolson Method\n');
for i=1:1:3
%formatpec = '%5f\n';
fprintf('%5f\n',u_3error(i));
end


% explicit method function
function [u]=Explicit_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1);
u(:,1)=u_initial';
```

```
S=[u0; zeros(my-3,1); uM]; % BC
for i=1:my-1
    A(i,i)=1-2*beta;
    if i<my-1
        A(i,i+1)=beta;
    end
    if i>=2
        A(i,i-1)=beta;
    end
end
for n=2:nt % time loop
    u(:,n)=A*u(:,n-1)+beta*S;
end
end

% implicit method function
function [u]=Implicit_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1);
u(:,1)=u_initial';
S=[u0; zeros(my-3,1); uM]; %BC
for i=1:my-1
    A(i,i)=-(1+2*beta);
    if i<my-1
        A(i,i+1)=beta;
    end
    if i>=2
        A(i,i-1)=beta;
    end
end
for n=2:nt % time loop
    b=-u(:,n-1)-beta*S;
    u(:,n)=A\b;
end
end

% crank nicolson method funciton defined
function [u]=CN_1D(beta,u0,uM,my,nt,u_initial)
A=zeros(my-1,my-1); B=A;
u(:,1)=u_initial';
S=[u0; zeros(my-3,1); uM]; %BC
for i=1:my-1
    A(i,i)=-2*(1+beta);
    B(i,i)=-2*(1-beta);
    if i<my-1
        A(i,i+1)=beta;
        B(i,i+1)=-beta;
    end
    if i>=2
        A(i,i-1)=beta;
        B(i,i-1)=-beta;
    end
end
for n=2:nt %time loop
    b=B*u(:,n-1)-beta*(S+S);
    u(:,n)=A\b;
```

```
end
end
function [c, f, s] = pdefun1(y, t, u, dudy, v, Tini, u0)
    c = 1/v;
    f = dudy;
    s = 0;
end
function u0 = pdeIC1(y, v, Tini, u0)
    u0 = zeros(size(y));
    u0(1) = u0;
end
function [pl, ql, pr, qr] = pdeBC1(yL, uL, yR, uR, t, v, Tini, u0)
    pl = uL - u0;   ql = 0;
    pr = uR;        qr = 0;
end
```