The formulation of many implicit methods for a scalar PDE results in the following equation :

$$a_i^n u_{i-1}^{n+1} + b_i^n u_i^{n+1} + c_i^n u_{i+1}^{n+1} = D_i^n \tag{A.1}$$

Tridiagonal Matrix Algorithm (TDMA) (Thomas Algorithm)

The solver is really a formula for recursive use of solving a matrix equation using Gauss-Elimination. The finite volume discretization gives a tri-diagonal (the diagonal plus two off-diagonals) equation system in 1D, a pentadiagonal system in 2D, and a septa-diagonal system in 3D.

In TDMA, Equation (A.1) can be rewritten as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = D_i \tag{A.2}$$

Equation 1.63 is solved from i = 2 to i = n-1 and i = 1and i = n are boundary nodes. Writing Eq. A.2 on the form

$$u_i = -P_i u_{i+1} + Q_i \tag{A.3}$$

In order to derive Eq. A.3 we write Eq. A.2 on matrix form so that

$$\begin{bmatrix} b_2 & c_2 & & & & \\ a_3 & b_3 & c_3 & & & \\ & a_4 & b_4 & c_4 & & \\ & & & & & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} D_2 - a_2 u_1 \\ D_3 \\ D_4 \\ \\ D_{n-1} \\ D_n - c_n u_n \end{bmatrix} \tag{A.4}$$

Start by dividing the first row by b_2 so that (see Eq. A.3)

$$\begin{bmatrix} 1 & P_2 & 0 & \dots & & \\ a_3 & b_3 & c_3 & 0 & \dots & \\ 0 & a_4 & b_4 & c_4 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} Q_2 \\ D_3 \\ D_4 \\ \vdots \end{bmatrix} \tag{A.5}$$

where

$$P_2 = \frac{c_2}{b_2}, Q_2 = \frac{D_2 - a_2 u_1}{b_2} \tag{A.6}$$

Now ,eliminate a's . Multiply row 1 by a_3, subtract it from row 2 and after that divide row 2 by $b_3 - a_3P_2$.,

we obtain

$$\begin{bmatrix} 1 & P_2 & 0 & \dots & & \\ 0 & 1 & P_3 & 0 & \dots & \\ 0 & a_4 & b_4 & c_4 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} Q_2 \\ Q_3 \\ d_4 \\ \vdots \end{bmatrix} \tag{A.7}$$

where

$$P_3 = \frac{c_3}{b_3 - a_3P_2}, Q_3 = \frac{D_3 - a_3Q_2}{b_3 - a_3P_2} \tag{A.8}$$

We see that Eq. A.8 becomes an recursive equation for P_i and Q_i on the form

$$P_i = \frac{c_i}{b_i - a_iP_{i-1}} \quad , \quad Q_i = \frac{D_i - a_iQ_{i-1}}{b_i - a_iP_{i-1}} \tag{A.9}$$

Now the P_i and Q_i coefficients can be computed.

1. for $i = 2$, use Eq. A.6.
2. for $i = 3$ to $i = n$-1, use Eq. A.8.
3. Compute u from Eq. A.3 starting from $i = n$-1 to $i = 2$.

The TDMA is essentially a forward elimination (implicit in the recurrence relations) and backward substitution procedure in which velocities at all i are updated simultaneously.

```
clc
clear all
close all

dt = 0.002; % time step
dy = 0.001; % Grid spacing
ly = 0.04;  % Distance between the walls
n = ly/dy;  % Number of intervals

nstep = 540;% Number of timestep

U0 = 40; % velocity of lower wall

nu = 0.000217;% kinematic viscosity of oil
```

```matlab
% grid generation

   for i = 1:n+1
    y(i) = (i-1)*dy;
   end




% Initialization(velocity at t = 0)
for j = 2:n
    u(1,j) = 0;
end


% Boundary conditions
for i = 1:nstep+1
    for j = 1:n+1
        if j == 1
            u(i,j) = U0;
        end
        if j == n+1
            u(i,j) = 0;
        end
    end
end

p = (nu*dt)/(dy)^2;

for j = 2:n-1
 a(j) = p;
 b(j) = -(2*p+1);
 c(j)= p;
end

% Applying Laasonen implicit to solve set of linear
equations formed using
% equation 3.12 at each node
```

```
for i = 1:nstep
   % Performs Gauss elimination
        for j = 2:n-1
            D(j) = -u(i,j);
            if j == 2
                P(i,j) = c(j)/b(j);
                Q(i,j) = (D(j)-a(j)*u(i+1,j-1))/b(j);
            else
                P(i,j) = c(j)/(b(j)-a(j)*P(i,j-1));
                Q(i,j) = (D(j)-a(j)*Q(i,j-1))/(b(j)-
a(j)*P(i,j-1));
            end

        end
  % Performs backward substitution

        for j = n-1:-1:2
           u(i+1,j) = -P(i,j)*u(i+1,j+1)+Q(i,j);
        end
    end


% creating file and writing data on it
 fid = fopen( 'couette_flow_Laasonen1.txt', 'wt' );
  fprintf(fid,'        y         t = 0     t = 0.18    t =
0.36   t = 0.54   t = 0.72   t = 0.90   t = 1.08 \n');
  for i = 1:n+1
  A =
[y(i);u(1,i);u(91,i);u(181,i);u(271,i);u(361,i);u(451,i
);u(541,i)];
   fprintf(fid,'%10.3f %10.3f %10.3f %10.3f %10.3f
%10.3f %10.3f %10.3f\n',A);
  end

fclose(fid)

% plotting velocity profile at different time steps
figure(1)
plot(u(1,:),y,'-*r',u(91,:),y,'-or',u(181,:),y,'-
sr',u(271,:),y,'-xr',u(361,:),y,'-+r',u(451,:),y,'-
pr',u(541,:),y,'-dr');
```

```
legend('T=0.00 sec','T=0.18 sec','T=0.36 sec','T=0.54
sec','T=0.72 sec','T=0.90 sec','T=1.08 sec')
 xlabel('u (m/s)','fontsize', 15)
 ylabel('y (m)','fontsize', 15)

% Plotting Syntax to animate the profile
% at different time steps
figure(2)
for i = 1:nstep
        plot(u(i,:),y(:),'ok','MarkerFaceColor','r')
        drawnow
end
```