Rui Xu
UID: 005230642
CEE/MAE M20
December 14, 2018

CEE/MAE M20
Introduction to Computer Programming with MATLAB

Fall 2018

FINAL PROJECT

Overview

This project focuses on solving equations of motion of type $m_i\ddot{x}_i = f_i(\mathbf{x})$ implicitly and explicitly for numerical simulation.

$\mathbf{x}(k)$ is used to represent the positions of motion at time t_k and $\dot{\mathbf{x}}(k)$ is used to represent the velocities of motion at time t_k.

For implicit simulation, the motion equations are equivalent to the following equation:

$$f_i \equiv \frac{m_i}{\Delta t}\left(\frac{x_i(k+1)-x_i(k)}{\Delta t}-\dot{x}_i(k)\right)-f_i\left(\mathbf{x}(k+1)\right)=0$$

For explicit simulation, the motion equations are equivalent to the following equation:

$$f_i \equiv \frac{m_i}{\Delta t}\left(\frac{x_i(k+1)-x_i(k)}{\Delta t}-\dot{x}_i(k)\right)-f_i\left(\mathbf{x}(k)\right)=0$$

The numerical simulation with Newton's method is as shown below:

Set initial conditions, $\mathbf{x}(0)$ and $\dot{\mathbf{x}}(0)$.

for k = 1 : timesteps

For each timestep, to solve the $\mathbf{x}(k+1)$ based on the known $\mathbf{x}(k)$ clearer with Newton's method, a vector, $\mathbf{q}$, is used to represent the $\mathbf{x}(k+1)$.

Newton's method is as shown below:

Start with a guess that $\mathbf{q} = \mathbf{x}(k)$ and keep updating the solution until the value of the elements in $\mathbf{f}$ is close to zero (e.g. `sum(abs(f))<1e-2`):

Set an initial error, `error` and tolerance, `tol`.

Set the starting points, $\mathbf{q} = \mathbf{x}(k)$.

while `error` < `tol`

Evaluate $\mathbf{f}$ % A function matrix

Evaluate $\mathbf{J}$ %A Jacobian matrix

$\triangle\mathbf{q} = \mathbf{J} \setminus \mathbf{f}$

$\mathbf{q} = \mathbf{q} - \triangle\mathbf{q}$

```
error=sum(abs(f))
```

end of while

J is Jacobian matrix calculated with implicit or explicit equation, as shown below:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_{i-1}} & \frac{\partial f_1}{\partial x_i} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_{i-1}} & \frac{\partial f_2}{\partial x_i} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_{i-1}} & \frac{\partial f_3}{\partial x_i} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \frac{\partial f_{i-1}}{\partial x_1} & \frac{\partial f_{i-1}}{\partial x_2} & \frac{\partial f_{i-1}}{\partial x_3} & \cdots & \frac{\partial f_{i-1}}{\partial x_{i-1}} & \frac{\partial f_{i-1}}{\partial x_i} \\ \frac{\partial f_i}{\partial x_1} & \frac{\partial f_i}{\partial x_2} & \frac{\partial f_i}{\partial x_3} & \cdots & \frac{\partial f_i}{\partial x_{i-1}} & \frac{\partial f_i}{\partial x_i} \end{bmatrix}$$

After solving **x**(k+1) with Newton's method, $\dot{\mathbf{x}}(k+1)$ can be obtained as shown below:

$\mathbf{x}(k+1) = \mathbf{q}$

$$\dot{\mathbf{x}}(k+1) = \frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{\Delta t}$$

end

The explicit simulation can also be achieved directly with the motion equations as shown below:

Set initial conditions, $\mathbf{x}(0)$ and $\dot{\mathbf{x}}(0)$.

for k = 1 : timesteps

$$x_i(k+1) = \left[\frac{\Delta t}{m_i} f_i(\mathbf{x}(k)) + \dot{x}_i(k)\right]\Delta t + x_i(k)$$

$$\dot{\mathbf{x}}(k+1) = \frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{\Delta t}$$

end

In this project, four problems in four parts shown below are used to demonstrate the application.

1 Rigid Sphere Falling in Viscous Flow

1.1 Introduction

The goal in this problem is to simulate the position and velocity of the sphere vs. time both implicitly and explicitly with the given final time and initial conditions and then plot the position vs. time and the velocity vs. time. The accuracy of the two simulations should be verified using the given criteria. Following these calculations, the results will be printed to the screen.

1.2 Model and Methods

To make the implicit simulations of this project's different parts consistent, the position and velocity of the sphere are calculated using Newton's method, and to make the calculation clearer, the position y_c is substituted with q.

Then for implicit simulation, the given Equation (2) is equivalent to the following equation:

$$f \equiv \frac{m}{\Delta t}\left[\frac{q-y_c(k)}{\Delta t}-\dot{y}_c(k)\right]+W+(6\pi\mu R)\frac{q-y_c(k)}{\Delta t}=0$$

The Jacobian is obtained by calculating the derivative of the equation above.

$$J=\frac{m}{\Delta t^2}+\frac{6\pi\mu R}{\Delta t}$$

For explicit simulation, the position can be directly obtained by the following equation, which is equivalent to the given Equation (4):

$$y_c(k+1)=\left[\frac{\Delta t}{m}\left(-W-6\pi\mu R\dot{y}_c(k)\right)+\dot{y}_c(k)\right]\Delta t+y_c(k)$$

First, a MATLAB function named `WeightMass` is defined to use given equations to calculate the weight and mass of a sphere in fluid, `W` and `m`, with input radius, density of the sphere and density of the fluid, `R`, `rho_metal`, and `rho_fluid`, as shown below.

```
function [W,m] = WeightMass(R,rho_metal,rho_fluid)
%W is the weight of a sphere in fluid
%m is the mass of a sphere in fluid
%R is the radius of the sphere
%rho_metal is the density of the sphere
%rho_fluid is the density of the fluid
g=9.8; %gravitational pull
W=4/3*pi*R^3*(rho_metal-rho_fluid)*g;
m=4/3*pi*R^3*rho_metal;
end
```

The script first assigns values for the attributes of the fluid and sphere, `1000` for the viscosity of the fluid, `mu`, `1000` for the density of the fluid, `rho_fluid`, `7000` for the density of the sphere, `rho_metal`, and `0.035` for the radius of the sphere, `R`. The weight and mass of the sphere in fluid, `W` and `m`, are obtained by calling `WeightMass` MATLAB function, as shown below.

```
[W,m]=WeightMass(R,rho_metal,rho_fluid);
```

For implicit simulation, the script assigns `0.01` for the timestep size, `dt`, and `1` for the final time, `tfinal`. The steps in for-loop, `tstep`, is obtained by dividing `tfinal` by `dt`, as shown below.

```
tstep=ceil(tfinal/dt);
```

The value of the sphere position, `yc`, and the sphere velocity, `yc_dot`, should be initialized before the execution of for-loop. A single sample of these is as shown below.

```
yc=zeros(1,tstep+1);
```

Then the initial conditions are assigned to the first entry in the `yc` and `yc_dot`, as shown below.

```
yc(1)=0;
yc_dot(1)=0;
```

Inside for-loop, the Newton's method is used to solve the `q` shown at the beginning with a while-loop.

Before solving the value of `q`, the error, `error`, and the tolerance, `tol`, should be

initialized. The error is assigned a value of `10` to make while-loop start. The `q` is assigned with an initial value of `yc(k)`. While `error` is bigger than `tol`, the iteration is kept running in while-loop. First, inside the while-loop, the values of the function, `f`, and the Jacobian matrix, `J`, are calculated using denoted equations (matrixes) with `q` and other known variables. Second, the value of `q` is updated by subtracting `dq=J\f` from itself. Third, the error, `error`, is updated by adding up the absolute values of elements in `f`. This is the Newton's method. Its general pseudocode is shown at the beginning of this report.

After the value of `q` is obtained by Newton's method, it should be assigned to `yc(k+1)` and then the `yc_dot(k+1)` should be calculated with the given Equation (3).

The for-loop is as shown below.

```
%Calculate the position and velocity of the sphere implicitly
for k=1:tstep
    %Initialize the error
    error=10;
    %Initialize the tolerance
    tol=1e-3;
    %Set the starting point
    q=yc(k);
    while error>tol
        %Compute the function
        f=m/dt*((q-yc(k))/dt-yc_dot(k))+W+(6*pi*mu*R)*(q-yc(k))/dt;
        %Compute the Jacobian
        J=m/(dt^2)+(6*pi*mu*R)/dt;
        %Newton's update
        dq=J\f;
        q=q-dq;
        %Compute the error
        error=abs(f);
    end
        %Position
        yc(k+1)=q;
        %Velocity
        yc_dot(k+1)=(yc(k+1)-yc(k))/dt;
end
```

After these calculations, the plot of the sphere position over time and the plot of the sphere velocity over time for implicit simulation are printed to the screen using MATLAB's `plot` function. After being created, they can be formatted with appropriate titles and axis labels,

as shown below.

```
%Plot the position and velocity vs. time for implicit simulation
t=0:dt:tfinal;
%Create the plot
figure(1)
plot(t,yc);
%Format the plot
xlabel('time (sec)');
ylabel('position (m)');
title('the Position vs. Time for Implicit Simulation');
%Create the plot
figure(2)
plot(t,yc_dot);
%Format the plot
xlabel('time (sec)');
ylabel('velocity (m/sec)');
title('the Velocity vs. Time for Implicit Simulation');
```

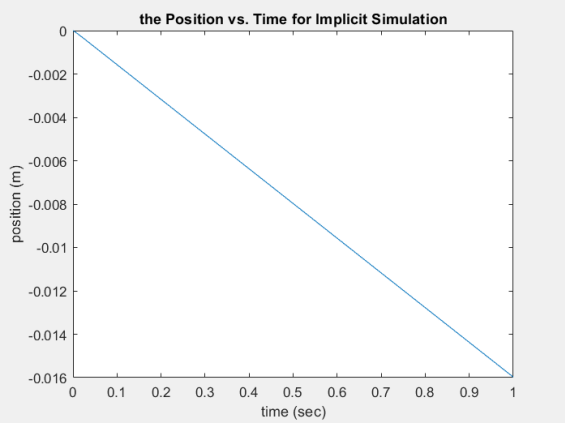
For explicit simulation, the script assigns `0.0001` for the timestep size, `dT`. The value of the sphere position, `YC`, and the sphere velocity, `YC_dot`, are calculated with the equation shown at the beginning. The for-loop is as shown below.

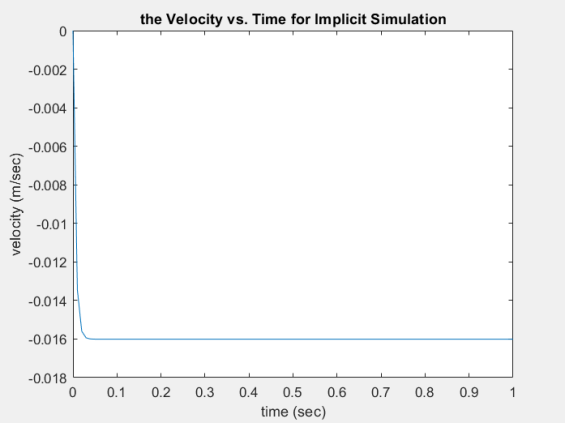
```
for k=1:Tstep
       %Position
       YC(k+1)=((dT/m)*(-W-6*pi*mu*R*YC_dot(k))+YC_dot(k))*dT+YC(k);
       %Velocity
       YC_dot(k+1)=(YC(k+1)-YC(k))/dT;
end
```

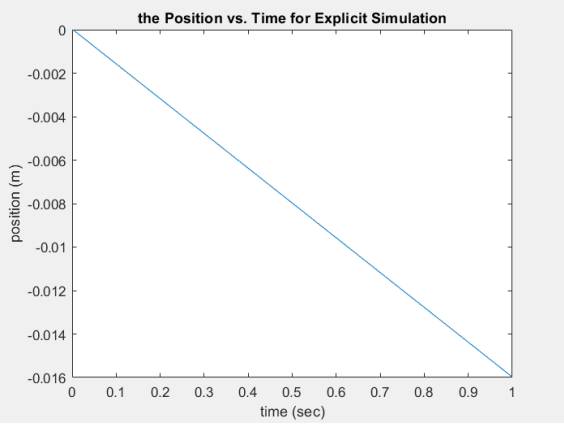
After these calculations, the plot of the sphere position over time and the plot of the sphere velocity over time for explicit simulation are generated with the same method as the implicit simulation.

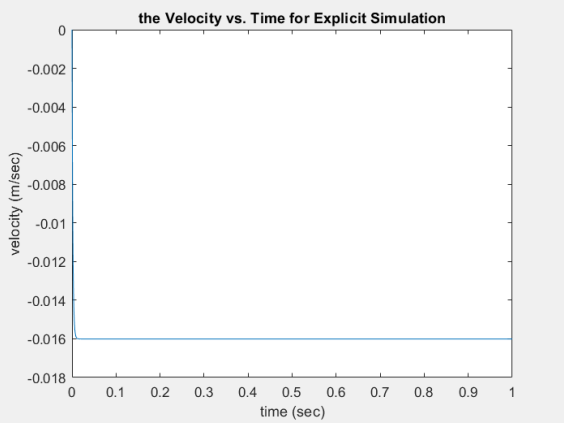
1.3 Calculations and Results

When the program is executed, the following output is printed to the screen.









As is shown in the plots, the difference between the implicit simulation with timestep size of 0.01 and explicit simulation with timestep size of 0.0001 is quite small. Only the velocity of explicit simulation decreases faster.

1.4 Discussion

To verify the accuracy of these simulations, the terminal velocities obtained from the `yc_dot`, `YC_dot` and theoretical velocity, `velocity_terminal`, obtained with mathematic method should be acquired, as shown below.

```
%%Calculte the theoretical terminal velocity
velocity_terminal=-W/(6*pi*mu*R);
%Display the terminal velocities
fprintf('The terminal velocity from implicit simulation is %f.\n',yc_dot(tstep+1));
fprintf('The terminal velocity from explicit simulation is %f.\n',YC_dot(Tstep+1));
fprintf('The theoretical terminal velocity is %f.\n',velocity_terminal);
```

When the program is executed, the following output is printed to the screen.

```
The terminal velocity from implicit simulation is -0.016007.
The terminal velocity from explicit simulation is -0.016007.
The theoretical terminal velocity is -0.016007.
```

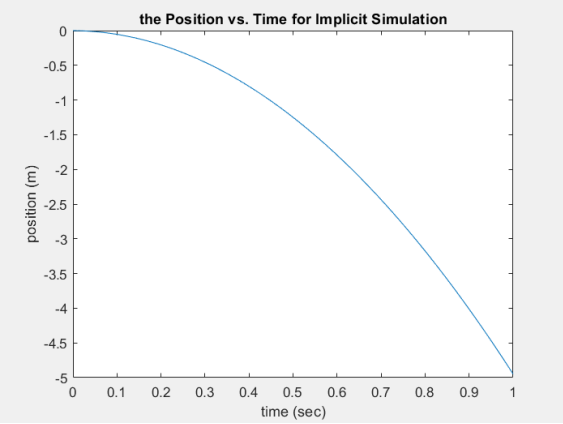
With errors of 0% and 0%, it can be concluded that from the aspect of terminal velocity, both simulations are accurate.

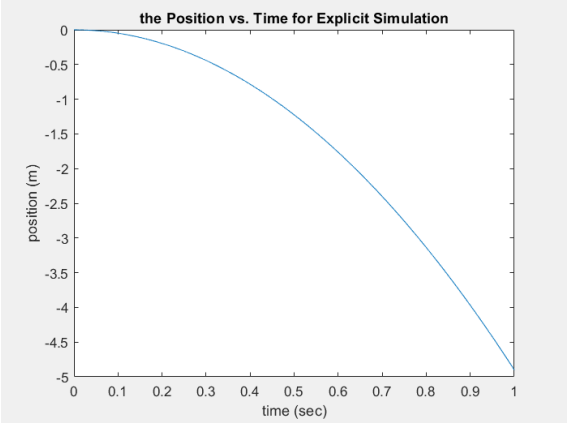
The freefall sphere position, `yc_freefall`, can be calculated using the given equation and a time array, `tt`. After this calculation, the plot of it can printed to the screen, as shown below.

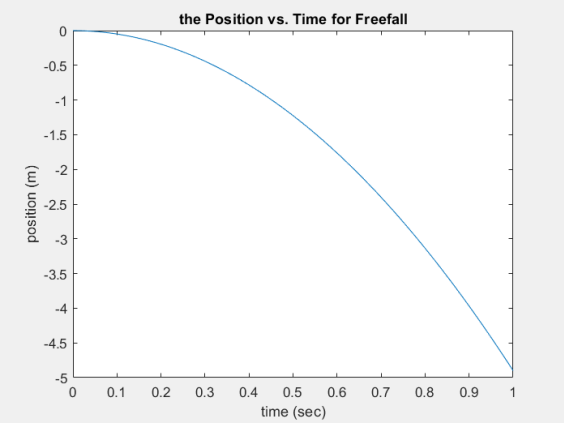
```
%Calculate the position of the freefall sphere
tt=0:0.01:1;
g=9.8;
yc_freefall=-1/2*g*tt.^2;
%%Plot the position of the freefall sphere
%Create the plot
figure(5)
plot(tt,yc_freefall);
%Format the plot
xlabel('time (sec)');
ylabel('position (m)');
title('the Position vs. Time for Freefall');
```

Both the viscosity of the fluid, `mu`, and the density of the fluid, `rho_fluid`, should be assigned `0` to make comparisons among the position plots of implicit simulation, explicit simulation and freefall mathematic method.

When the program is executed, the three position plots are as shown below.

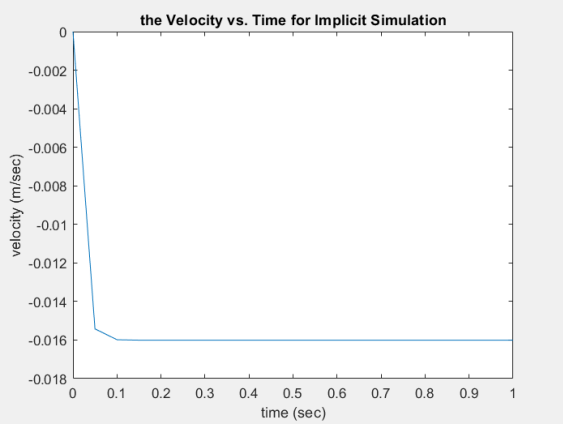


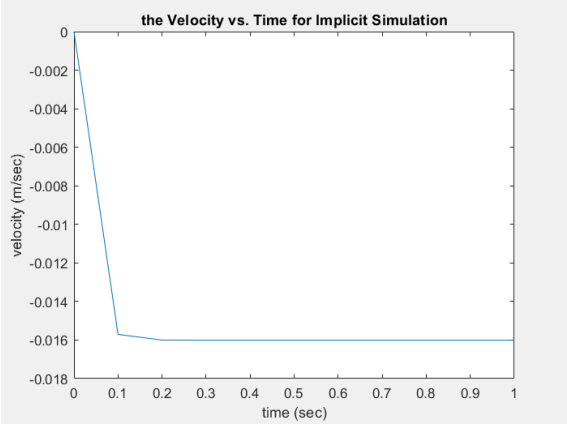




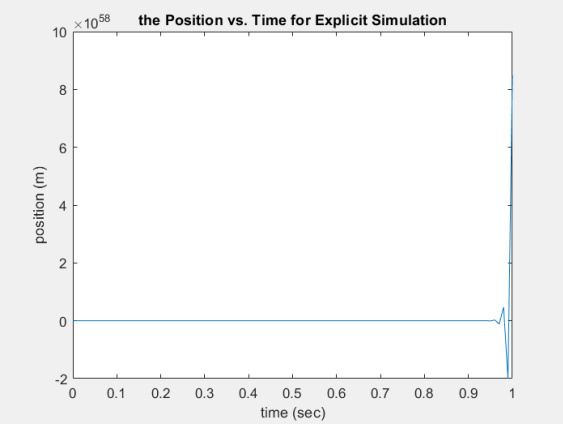
Since there is no big difference between them, it can be concluded that both simulations are accurate enough.

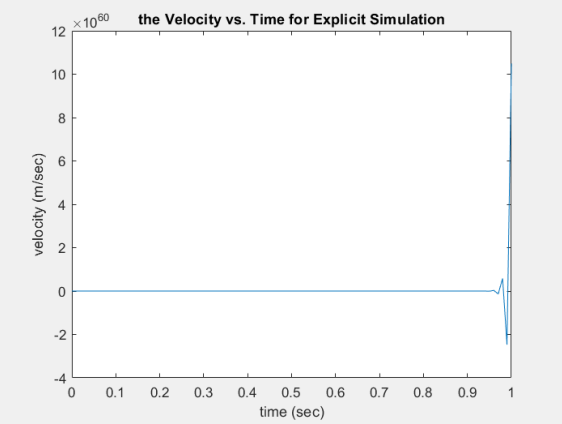
After changing the timestep size of implicit simulation from 0.01 to 0.05 and 0.1, the position plot does not change much but the velocity plot changes as shown below.

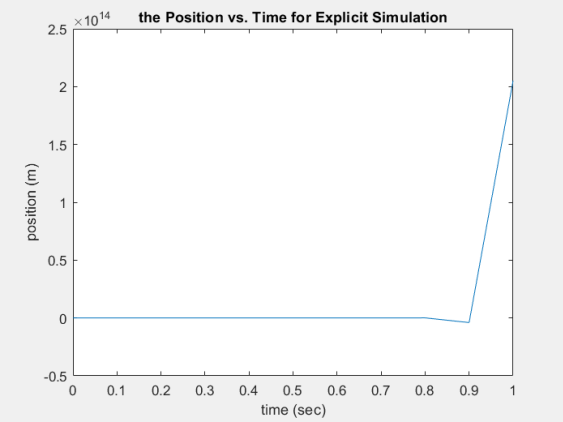


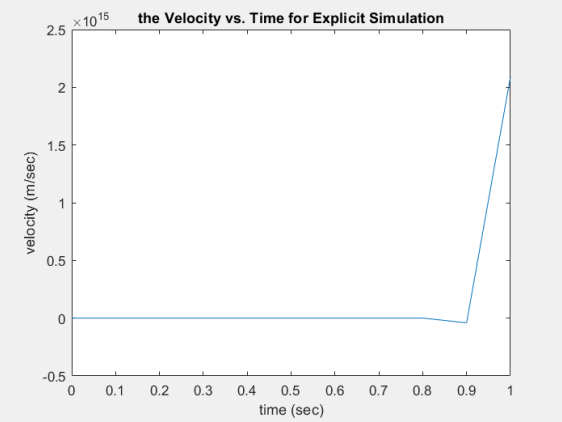


After changing the timestep size of explicit simulation from 0.0001 to 0.001, the position plot and velocity plot does not change much but when the timestep size is changed to 0.01 and 0.1, both position plot and velocity plot are obviously incorrect as shown below.









With the same timestep size, the computing time of implicit simulation is larger than that of explicit simulation. For example, the results obtained using MATLAB's `tic-toc` function for timestep 0.01 are as shown below.

```
Elapsed time is 0.237781 seconds.
Elapsed time is 0.102823 seconds.
```

It can be concluded that both implicit and explicit simulations need enough timestep size to obtain reasonable results and this will cost more computing time. The computing time of implicit simulation is larger than that of explicit simulation with the same timestep size; however, explicit simulation needs much smaller timestep size to make the simulation results accurate enough.

2 Rigid Spheres and Elastic Beam Falling in Viscous Flow

2.1 Introduction

The goal in this problem is to simulate the positions and velocities of three spheres vs. time both implicitly and explicitly with the given final time and initial conditions and then plot the position vs. time and the velocity vs. time of the middle node and the turning angle vs. time. Following these calculations, the results will be printed to the screen.

2.2 Model and Methods

For implicit simulation, the positions and velocities of three spheres are calculated using Newton's method, and to make the calculation clearer, the positions (x_1, y_1), (x_2, y_2) and (x_3, y_3) is constructed in a vector and substituted with $\boldsymbol{q}$.

Then for implicit simulation, the discretized governing equations are equivalent to the following equations:

$$f_1 \equiv \frac{m_1}{\Delta t}\left[\frac{q(1)-x_1(k)}{\Delta t}-\dot{x}_1(k)\right]+(6\pi\mu R_1)\frac{q(1)-x_1(k)}{\Delta t}+\frac{\partial E_1^s}{\partial x_1}+\frac{\partial E^b}{\partial x_1}=0 \tag{1}$$

$$f_2 \equiv \frac{m_1}{\Delta t}\left[\frac{q(2)-y_1(k)}{\Delta t}-\dot{y}_1(k)\right]+W_1+(6\pi\mu R_1)\frac{q(2)-y_1(k)}{\Delta t}+\frac{\partial E_1^s}{\partial y_1}+\frac{\partial E^b}{\partial y_1}=0 \tag{2}$$

$$f_3 \equiv \frac{m_2}{\Delta t}\left[\frac{q(3)-x_2(k)}{\Delta t}-\dot{x}_2(k)\right]+(6\pi\mu R_2)\frac{q(3)-x_2(k)}{\Delta t}+\frac{\partial E_1^s}{\partial x_2}+\frac{\partial E_2^s}{\partial x_2}+\frac{\partial E^b}{\partial x_2}=0 \tag{3}$$

$$f_4 \equiv \frac{m_2}{\Delta t}\left[\frac{q(4)-y_2(k)}{\Delta t}-\dot{y}_2(k)\right]+W_2+(6\pi\mu R_2)\frac{q(4)-y_2(k)}{\Delta t}+\frac{\partial E_1^s}{\partial y_2}+\frac{\partial E_2^s}{\partial y_2}+\frac{\partial E^b}{\partial y_2}=0 \tag{4}$$

$$f_5 \equiv \frac{m_3}{\Delta t}\left[\frac{q(5)-x_3(k)}{\Delta t}-\dot{x}_3(k)\right]+(6\pi\mu R_3)\frac{q(5)-x_3(k)}{\Delta t}+\frac{\partial E_2^s}{\partial x_3}+\frac{\partial E^b}{\partial x_3}=0 \tag{5}$$

$$f_6 \equiv \frac{m_3}{\Delta t}\left[\frac{q(6)-y_3(k)}{\Delta t}-\dot{y}_3(k)\right]+W_3+(6\pi\mu R_3)\frac{q(6)-y_3(k)}{\Delta t}+\frac{\partial E_2^s}{\partial y_3}+\frac{\partial E^b}{\partial y_3}=0 \tag{6}$$

To make use of the functions given in the project, the equations above is categorized as shown below:

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} \equiv \begin{bmatrix} \frac{m_1}{\Delta t}\left[\frac{q(1)-x_1(k)}{\Delta t}-\dot{x}_1(k)\right]+(6\pi\mu R_1)\frac{q(1)-x_1(k)}{\Delta t} \\ \frac{m_1}{\Delta t}\left[\frac{q(2)-y_1(k)}{\Delta t}-\dot{y}_1(k)\right]+W_1+(6\pi\mu R_1)\frac{q(2)-y_1(k)}{\Delta t} \\ \frac{m_2}{\Delta t}\left[\frac{q(3)-x_2(k)}{\Delta t}-\dot{x}_2(k)\right]+(6\pi\mu R_2)\frac{q(3)-x_2(k)}{\Delta t} \\ \frac{m_2}{\Delta t}\left[\frac{q(4)-y_2(k)}{\Delta t}-\dot{y}_2(k)\right]+W_2+(6\pi\mu R_2)\frac{q(4)-y_2(k)}{\Delta t} \\ \frac{m_3}{\Delta t}\left[\frac{q(5)-x_3(k)}{\Delta t}-\dot{x}_3(k)\right]+(6\pi\mu R_3)\frac{q(5)-x_3(k)}{\Delta t} \\ \frac{m_3}{\Delta t}\left[\frac{q(6)-y_3(k)}{\Delta t}-\dot{y}_3(k)\right]+W_3+(6\pi\mu R_3)\frac{q(6)-y_3(k)}{\Delta t} \end{bmatrix} + \begin{bmatrix} \frac{\partial E_1^s}{\partial x_1} \\ \frac{\partial E_1^s}{\partial y_1} \\ \frac{\partial E_1^s}{\partial x_2} \\ \frac{\partial E_1^s}{\partial y_2} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{\partial E_2^s}{\partial x_2} \\ \frac{\partial E_2^s}{\partial y_2} \\ \frac{\partial E_2^s}{\partial x_3} \\ \frac{\partial E_2^s}{\partial y_3} \end{bmatrix} + \begin{bmatrix} \frac{\partial E^b}{\partial x_1} \\ \frac{\partial E^b}{\partial y_1} \\ \frac{\partial E^b}{\partial x_2} \\ \frac{\partial E^b}{\partial y_2} \\ \frac{\partial E^b}{\partial x_3} \\ \frac{\partial E^b}{\partial y_3} \end{bmatrix} = 0$$

Then the Jacobian is calculated using matrixes above and their derivatives as shown below.

$$\mathbf{J} = \mathbf{J}_{g+v} + \mathbf{J}_{s1} + \mathbf{J}_{s2} + \mathbf{J}_b$$

$$\mathbf{J}_{g+v} = \begin{bmatrix} \frac{m_1}{\Delta t^2}+\frac{6\pi\mu R_1}{\Delta t} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{m_1}{\Delta t^2}+\frac{6\pi\mu R_1}{\Delta t} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{m_2}{\Delta t^2}+\frac{6\pi\mu R_2}{\Delta t} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{m_2}{\Delta t^2}+\frac{6\pi\mu R_2}{\Delta t} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{m_3}{\Delta t^2}+\frac{6\pi\mu R_3}{\Delta t} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{m_3}{\Delta t^2}+\frac{6\pi\mu R_3}{\Delta t} \end{bmatrix}$$

$$\mathbf{J}_{s1}=\begin{bmatrix}
\frac{\partial^2 E_1^s}{\partial x_1 \partial x_1} & \frac{\partial^2 E_1^s}{\partial x_1 \partial y_1} & \frac{\partial^2 E_1^s}{\partial x_1 \partial x_2} & \frac{\partial^2 E_1^s}{\partial x_1 \partial y_2} & 0 & 0 \\
\frac{\partial^2 E_1^s}{\partial y_1 \partial x_1} & \frac{\partial^2 E_1^s}{\partial y_1 \partial y_1} & \frac{\partial^2 E_1^s}{\partial y_1 \partial x_2} & \frac{\partial^2 E_1^s}{\partial y_1 \partial y_2} & 0 & 0 \\
\frac{\partial^2 E_1^s}{\partial x_2 \partial x_1} & \frac{\partial^2 E_1^s}{\partial x_2 \partial y_1} & \frac{\partial^2 E_1^s}{\partial x_2 \partial x_2} & \frac{\partial^2 E_1^s}{\partial x_2 \partial y_2} & 0 & 0 \\
\frac{\partial^2 E_1^s}{\partial y_2 \partial x_1} & \frac{\partial^2 E_1^s}{\partial y_2 \partial y_1} & \frac{\partial^2 E_1^s}{\partial y_2 \partial x_2} & \frac{\partial^2 E_1^s}{\partial y_2 \partial y_2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{J}_{s2}=\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\partial^2 E_2^s}{\partial x_2 \partial x_2} & \frac{\partial^2 E_2^s}{\partial x_2 \partial y_2} & \frac{\partial^2 E_2^s}{\partial x_2 \partial x_3} & \frac{\partial^2 E_2^s}{\partial x_2 \partial y_3} \\
0 & 0 & \frac{\partial^2 E_2^s}{\partial y_2 \partial x_2} & \frac{\partial^2 E_2^s}{\partial y_2 \partial y_2} & \frac{\partial^2 E_2^s}{\partial y_2 \partial x_3} & \frac{\partial^2 E_2^s}{\partial y_2 \partial y_3} \\
0 & 0 & \frac{\partial^2 E_2^s}{\partial x_3 \partial x_2} & \frac{\partial^2 E_2^s}{\partial x_3 \partial y_2} & \frac{\partial^2 E_2^s}{\partial x_3 \partial x_3} & \frac{\partial^2 E_2^s}{\partial x_3 \partial y_3} \\
0 & 0 & \frac{\partial^2 E_2^s}{\partial y_3 \partial x_2} & \frac{\partial^2 E_2^s}{\partial y_3 \partial y_2} & \frac{\partial^2 E_2^s}{\partial y_3 \partial x_3} & \frac{\partial^2 E_2^s}{\partial y_3 \partial y_3}
\end{bmatrix}$$

$$\mathbf{J}_{b}=\begin{bmatrix}
\frac{\partial^2 E^b}{\partial x_1 \partial x_1} & \frac{\partial^2 E^b}{\partial x_1 \partial y_1} & \frac{\partial^2 E^b}{\partial x_1 \partial x_2} & \frac{\partial^2 E^b}{\partial x_1 \partial y_2} & \frac{\partial^2 E^b}{\partial x_1 \partial x_3} & \frac{\partial^2 E^b}{\partial x_1 \partial y_3} \\
\frac{\partial^2 E^b}{\partial y_1 \partial x_1} & \frac{\partial^2 E^b}{\partial y_1 \partial y_1} & \frac{\partial^2 E^b}{\partial y_1 \partial x_2} & \frac{\partial^2 E^b}{\partial y_1 \partial y_2} & \frac{\partial^2 E^b}{\partial y_1 \partial x_3} & \frac{\partial^2 E^b}{\partial y_1 \partial y_3} \\
\frac{\partial^2 E^b}{\partial x_2 \partial x_1} & \frac{\partial^2 E^b}{\partial x_2 \partial y_1} & \frac{\partial^2 E^b}{\partial x_2 \partial x_2} & \frac{\partial^2 E^b}{\partial x_2 \partial y_2} & \frac{\partial^2 E^b}{\partial x_2 \partial x_3} & \frac{\partial^2 E^b}{\partial x_2 \partial y_3} \\
\frac{\partial^2 E^b}{\partial y_2 \partial x_1} & \frac{\partial^2 E^b}{\partial y_2 \partial y_1} & \frac{\partial^2 E^b}{\partial y_2 \partial x_2} & \frac{\partial^2 E^b}{\partial y_2 \partial y_2} & \frac{\partial^2 E^b}{\partial y_2 \partial x_3} & \frac{\partial^2 E^b}{\partial y_2 \partial y_3} \\
\frac{\partial^2 E^b}{\partial x_3 \partial x_1} & \frac{\partial^2 E^b}{\partial x_3 \partial y_1} & \frac{\partial^2 E^b}{\partial x_3 \partial x_2} & \frac{\partial^2 E^b}{\partial x_3 \partial y_2} & \frac{\partial^2 E^b}{\partial x_3 \partial x_3} & \frac{\partial^2 E^b}{\partial x_3 \partial y_3} \\
\frac{\partial^2 E^b}{\partial y_3 \partial x_1} & \frac{\partial^2 E^b}{\partial y_3 \partial y_1} & \frac{\partial^2 E^b}{\partial y_3 \partial x_2} & \frac{\partial^2 E^b}{\partial y_3 \partial y_2} & \frac{\partial^2 E^b}{\partial y_3 \partial x_3} & \frac{\partial^2 E^b}{\partial y_3 \partial y_3}
\end{bmatrix}$$

For explicit simulation, the positions can be directly obtained by the following equations:

$$x_1(k+1)=\left[\frac{\Delta t}{m_1}\left(-6\pi\mu R_1\dot{x}_1(k)-\frac{\partial E_1^s}{\partial x_1}-\frac{\partial E_b}{\partial x_1}\right)+\dot{x}_1(k)\right]\Delta t+x_1(k) \tag{1}$$

$$y_1(k+1)=\left[\frac{\Delta t}{m_1}\left(-W_1-6\pi\mu R_1\dot{y}_1(k)-\frac{\partial E_1^s}{\partial y_1}-\frac{\partial E_b}{\partial y_1}\right)+\dot{y}_1(k)\right]\Delta t+y_1(k) \tag{2}$$

$$x_2(k+1)=\left[\frac{\Delta t}{m_2}\left(-6\pi\mu R_2\dot{x}_2(k)-\frac{\partial E_1^s}{\partial x_2}-\frac{\partial E_2^s}{\partial x_2}-\frac{\partial E_b}{\partial x_2}\right)+\dot{x}_2(k)\right]\Delta t+x_2(k) \tag{3}$$

$$y_2(k+1)=\left[\frac{\Delta t}{m_2}\left(-W_2-6\pi\mu R_2\dot{y}_2(k)-\frac{\partial E_1^s}{\partial y_2}-\frac{\partial E_2^s}{\partial y_2}-\frac{\partial E_b}{\partial y_2}\right)+\dot{y}_2(k)\right]\Delta t+y_2(k) \tag{4}$$

$$x_3(k+1)=\left[\frac{\Delta t}{m_3}\left(-6\pi\mu R_3\dot{x}_3(k)-\frac{\partial E_2^s}{\partial x_3}-\frac{\partial E_b}{\partial x_3}\right)+\dot{x}_3(k)\right]\Delta t+x_3(k) \tag{5}$$

$$y_3(k+1)=\left[\frac{\Delta t}{m_3}\left(-W_3-6\pi\mu R_3\dot{y}_3(k)-\frac{\partial E_2^s}{\partial y_3}-\frac{\partial E_b}{\partial y_3}\right)+\dot{y}_3(k)\right]\Delta t+y_3(k) \tag{6}$$

The script first assigns values for the attributes of the beam, `0.001` for the cross-sectional radius of the beam, `r0`, and `1e9` for Young's modulus, `E`. Then the stretching stiffness and bending stiffness of the beam, `EA` and `EI`, can be calculated using the given equations with `r0` and `E`, as shown below.

```
EA=E*pi*r0^2;   %stretching stiffness
EI=E*pi*r0^4/4; %bending stiffness
```

The length of the beam, `l`, is assigned with `0.1`. Then the length of each segment between two nodes, `l_k`, can be assigned with `l/2`.

Then the script assigns values for the attributes of the fluid and sphere, the same as the previous part for `mu`, `rho_fluid`, and `rho_metal`, and `0.005`, `0.025`, `0.005` for the radii of the spheres, `R1`, `R2`, and `R3`. The weights and masses of the spheres in fluid, `W1`, `W2`, `W3`, and `m1`, `m2`, `m3`, are obtained by calling `WeightMass` MATLAB function. A sample of these is as shown below.

```
[W1,m1]=WeightMass(R1,rho_metal,rho_fluid);
```

For implicit simulation, the script assigns `1e-2` for the timestep size, `dt`, and `10` for the final time, `tfinal`. The steps in for-loop, `tstep` is obtained the same as the previous

part.

The value of the sphere positions, `x1`, `y1`, `x2`, `y2`, `x3`, `y3`, and the sphere velocities, `x1_dot`, `y1_dot`, `x2_dot`, `y2_dot`, `x3_dot`, `y3_dot`, should initialized before the execution of for-loop, the same as the previous part.

Then the initial conditions are assigned to the first entry in the `x1`, `y1`, `x2`, `y2`, `x3`, `y3`, and `x1_dot`, `y1_dot`, `x2_dot`, `y2_dot`, `x3_dot`, `y3_dot`, as shown below.

```
x1(1)=0;x1_dot(1)=0;
y1(1)=0;y1_dot(1)=0;
x2(1)=l_k;x2_dot(1)=0;
y2(1)=0;y2_dot(1)=0;
x3(1)=2*l_k;x3_dot(1)=0;
y3(1)=0;y3_dot(1)=0;
```

Inside for-loop, the Newton's method is used to solve the `q` shown at the beginning with a while-loop.

After the value of `q` is obtained by Newton's method, it should be assigned to `x1(k+1)`, `y1(k+1)`, `x2(k+1)`, `y2(k+1)`, `x3(k+1)`, `y3(k+1)`, and then the `x1_dot(k+1)`, `y1_dot(k+1)`, `x2_dot(k+1)`, `y2_dot(k+1)`, `x3_dot(k+1)`, `y3_dot(k+1)`, should be calculated.

The for-loop is as shown below.

```
for k=1:tstep
    ...
    %Set the starting points
    q=[x1(k);y1(k);x2(k);y2(k);x3(k);y3(k)];
    while error>tol
        %Initialize the functions
        f=zeros(6,1);
        Fs1=zeros(4,1);
        Fs2=zeros(4,1);
        Fb=zeros(6,1);
        %Initialize the Jacobian
        J=zeros(6,6);
        Js1=zeros(4,4);
        Js2=zeros(4,4);
        Jb=zeros(6,6);
```

```
        %Compute the functions
        Fs1=gradEs(q(1),q(2),q(3),q(4),l_k,EA);
        Fs2=gradEs(q(3),q(4),q(5),q(6),l_k,EA);
        Fb=gradEb(q(1),q(2),q(3),q(4),q(5),q(6),l_k,EI);
        f(1)=m1/dt*((q(1)-x1(k))/dt-x1_dot(k))+...
            (6*pi*mu*R1)*((q(1)-x1(k))/dt);
        f(2)=m1/dt*((q(2)-y1(k))/dt-y1_dot(k))+...
            W1+(6*pi*mu*R1)*((q(2)-y1(k))/dt);
        ...
        f(1:4,1)=f(1:4,1)+Fs1;
        f(3:6,1)=f(3:6,1)+Fs2;
        f=f+Fb;
        %Compute the Jacobian
        Js1=hessEs(q(1),q(2),q(3),q(4),l_k,EA);
        Js2=hessEs(q(3),q(4),q(5),q(6),l_k,EA);
        Jb=hessEb(q(1),q(2),q(3),q(4),q(5),q(6),l_k,EI);
        J(1,1)=m1/(dt^2)+(6*pi*mu*R1)/dt;
        J(2,2)=m1/(dt^2)+(6*pi*mu*R1)/dt;
        ...
        J(1:4,1:4)=J(1:4,1:4)+Js1';
        J(3:6,3:6)=J(3:6,3:6)+Js2';
        J=J+Jb';
        %Newton's update
        dq=J\f;
        q=q-dq;
        %Compute the error
        error=sum(abs(f));
    end
    %Position
    x1(k+1)=q(1);
    ...
    %Velocity
    x1_dot(k+1)=(x1(k+1)-x1(k))/dt;
    ...
end
```

After these calculations, the plot of the position and velocity of the middle node over time for implicit simulation are printed to the screen using MATLAB's `plot` function. After being created, they can be formatted with appropriate titles and axis labels, as shown below.

```
%%Plot the position and velocity of the middle node vs. time
```

```
t=0:dt:tfinal;
%Create the plot
figure(1)
plot(t,x2);
hold on
plot(t,y2);
hold off
%Format the plot
xlabel('time (sec)');
ylabel('position (m)');
legend('x-position','y-position','Location','Best');
title('the Middle Node Position for Implicit Simulation');
%Create the plot
figure(2)
plot(t,x2_dot);
hold on
plot(t,y2_dot);
hold off
%Format the plot
xlabel('time (sec)');
ylabel('velocity (m/sec)');
ylim([-9e-3,1e-3]);
legend('x-velocity','y-velocity','Location','Best');
title('the Middle Node Velocity for Implicit Simulation');
```

The turning angle θ, which is represented by `theta`, can be calculated using method of symmetrical coordinates and law of cosines, as shown below.

$$\theta = \arccos \frac{2\Delta l^2 - \left[\left(2x_2 - x_1 - x_3\right)^2 + \left(2y_2 - y_1 - y_3\right)^2 \right]}{2\Delta l^2}$$

After the calculation, the plot of the turning angle over time for implicit simulation are printed to the screen using MATLAB's `plot` function. After being created, they can be formatted with appropriate titles and axis labels, as shown below.

```
%%Plot the turning angle
%Calculate the turning angle
theta=acos((2*l_k^2-((2*x2-x1-x3).^2+(2*y2-y1-y3).^2))/(2*l_k^2));
%Create the plot
figure(3)
```

```
plot(t,theta);
%Format the plot
xlabel('time (sec)');
ylabel('angle (rad)');
ylim([0,1.2]);
title('the Turning Angle vs. Time for Implicit Simulation');
```

For explicit simulation, the script assigns `1e-5` for the timestep size, `dT`. The value of the sphere positions, `X1`, `Y1`, `X2`, `Y2`, `X3`, `Y3`, and the sphere velocities, `X1_dot`, `Y1_dot`, `X2_dot`, `Y2_dot`, `X3_dot`, `Y3_dot`, are calculated with the equations shown at the beginning. The for-loop is as shown below.

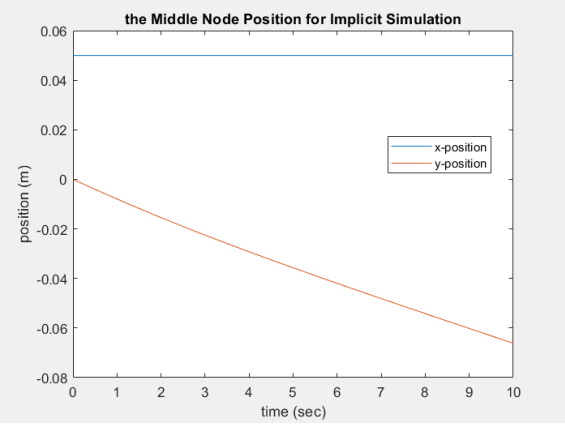
```
for k=1:Tstep
    Fs1=gradEs(X1(k),Y1(k),X2(k),Y2(k),l_k,EA);
    Fs2=gradEs(X2(k),Y2(k),X3(k),Y3(k),l_k,EA);
    Fb=gradEb(X1(k),Y1(k),X2(k),Y2(k),X3(k),Y3(k),l_k,EI);
    %Position
    X1(k+1)=((dT/m1)*(-6*pi*mu*R1*X1_dot(k)-Fs1(1)-Fb(1))+X1_dot(k))*dT+X1(k);
    Y1(k+1)=((dT/m1)*(-W1-6*pi*mu*R1*Y1_dot(k)-Fs1(2)-Fb(2))+Y1_dot(k))*dT+Y1(k);
    X2(k+1)=((dT/m2)*(-6*pi*mu*R2*X2_dot(k)-Fs1(3)-Fs2(1)-Fb(3))+X2_dot(k))*dT+X2(k);
    Y2(k+1)=((dT/m2)*(-W2-6*pi*mu*R2*Y2_dot(k)-Fs1(4)-Fs2(2)-Fb(4))+Y2_dot(k))*dT+Y2(k);
    X3(k+1)=((dT/m3)*(-6*pi*mu*R3*X3_dot(k)-Fs2(3)-Fb(5))+X3_dot(k))*dT+X3(k);
    Y3(k+1)=((dT/m3)*(-W3-6*pi*mu*R3*Y3_dot(k)-Fs2(4)-Fb(6))+Y3_dot(k))*dT+Y3(k);
    %Velocity
    X1_dot(k+1)=(X1(k+1)-X1(k))/dT;
    Y1_dot(k+1)=(Y1(k+1)-Y1(k))/dT;
    X2_dot(k+1)=(X2(k+1)-X2(k))/dT;
    Y2_dot(k+1)=(Y2(k+1)-Y2(k))/dT;
    X3_dot(k+1)=(X3(k+1)-X3(k))/dT;
    Y3_dot(k+1)=(Y3(k+1)-Y3(k))/dT;
end
```

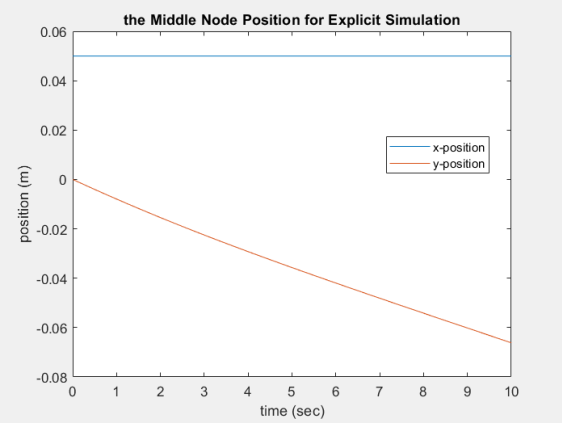
After these calculations, the plot of the position and velocity of the middle node over time for explicit simulation are generated with the same method as the implicit simulation.

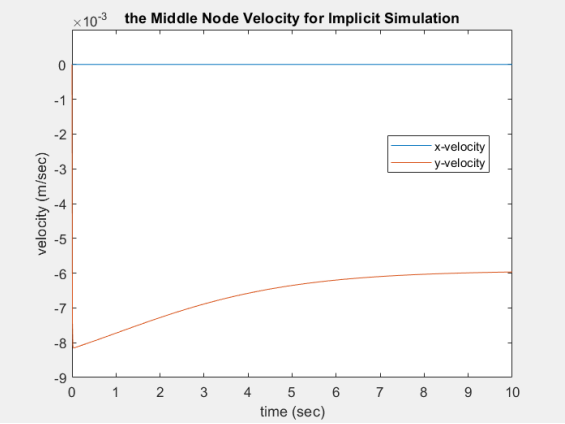
The turning angle, `THETA`, is also calculated and plotted with the same method as the implicit simulation.

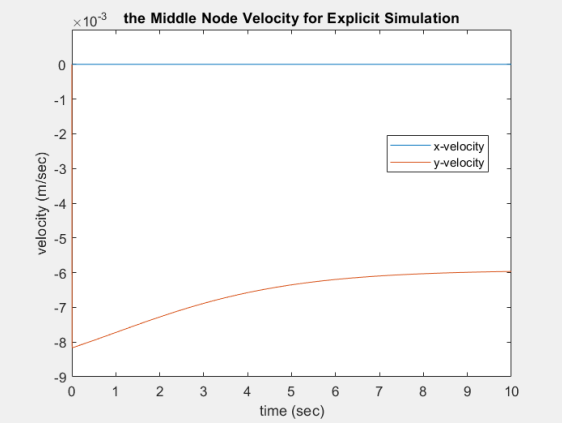
2.3 Calculations and Results

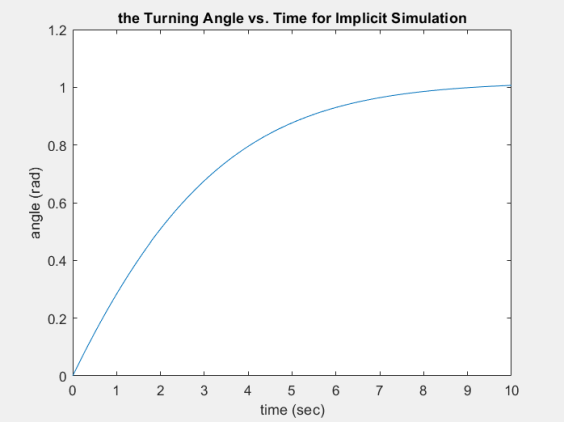
When the program is executed, the following output is printed to the screen.

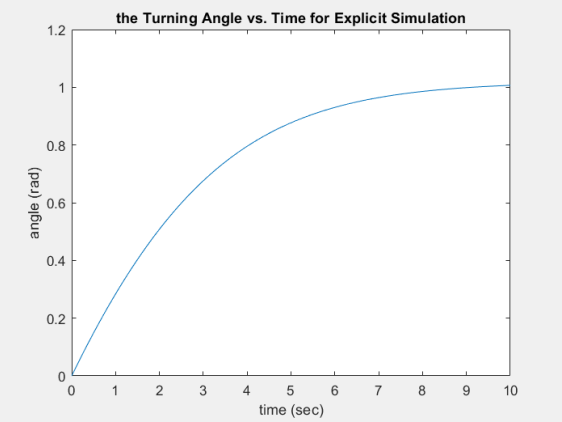












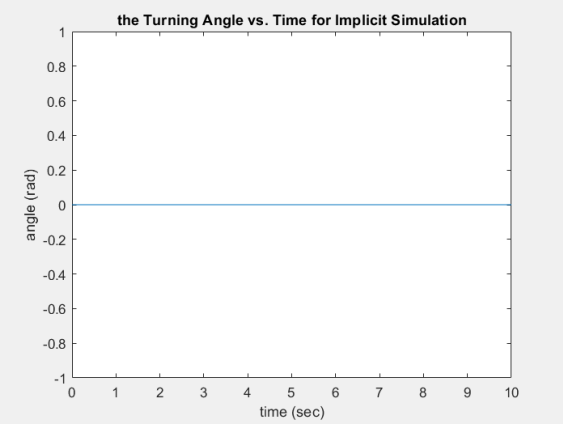
As is shown in the plots, the difference between the implicit simulation with timestep size of 10^{-2} and explicit simulation with timestep size of 10^{-5} is quite small.

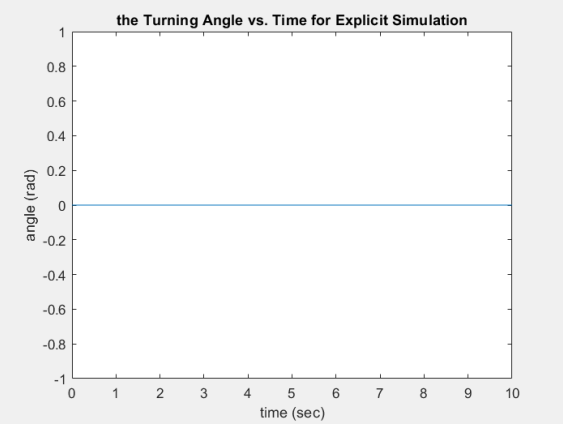
2.4 Discussion

If the all the radii are the same, the beam will fall down with a shape of line, and then the

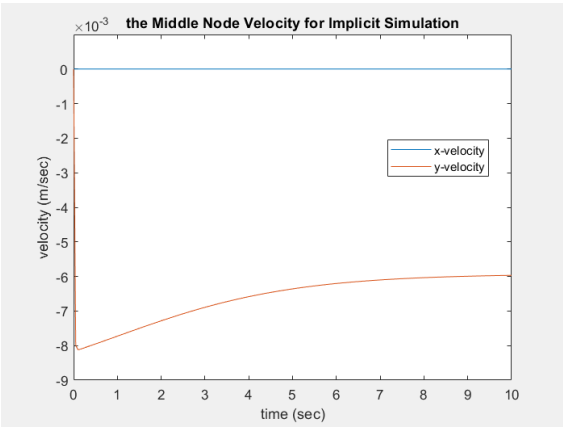
turning angle should be 0.

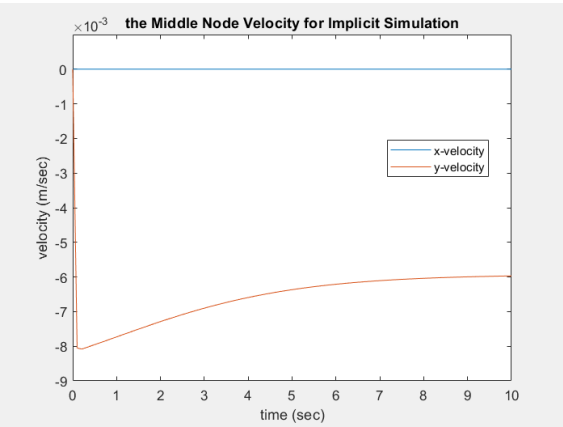
When the `R1`, `R2` and `R3` are all assigned with `0.005`, after the program is executed, the turning angle plots for implicit and explicit simulations are as shown below, which agree with my intuition.



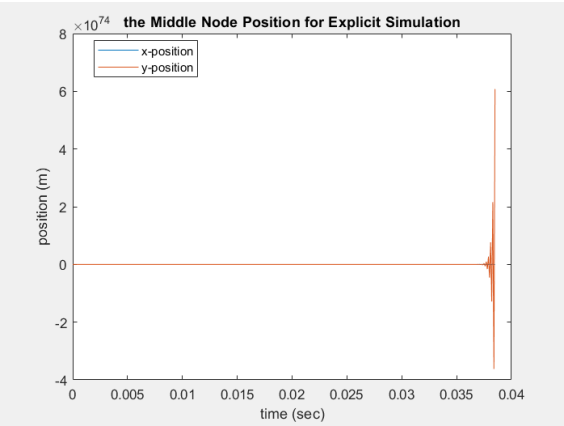


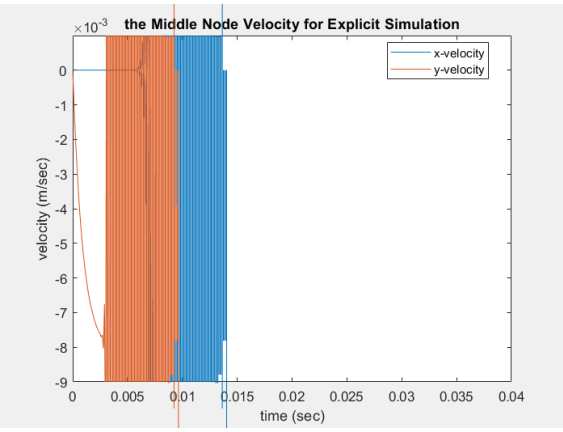
After changing the timestep size of implicit simulation from 0.01 to 0.05 and 0.1, the position plot and turning angle plot do not change much but the velocity plot changes as shown below.

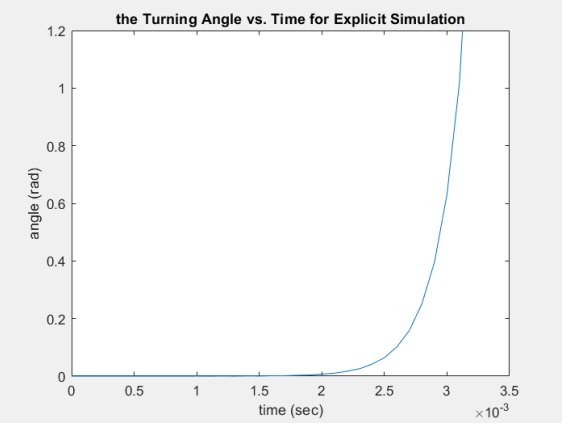




After changing the timestep size of explicit simulation from 10^{-5} to $5x10^{-5}$, the three plots does not change much, but after just changing the timestep size to 10^{-4}, the three plots already appear wrong massages as shown below.







It can be concluded that results from both implicit and explicit simulations will be less accurate if the timestep size increases. Explicit simulation needs much smaller timestep size to make the simulation results accurate enough while implicit simulation does not need very small timestep size.

After decreasing the timestep sizes of both implicit and explicit simulations, the plots do not change much but obviously the computing time increases. That means to obtain more accurate results, more computing time is needed.

Then after making the timestep sizes of implicit and explicit simulations the same, 10^{-5}, it can be concluded that the explicit simulation costs much less time. The results obtained using MATLAB's `tic-toc` function are as shown below.

```
Elapsed time is 110.030250 seconds.
Elapsed time is 3.134298 seconds.
```

In conclusion, implicit simulation needs more computing time while explicit simulation needs smaller timestep size.

3 Generalized Case of Elastic Beam Falling in Viscous Flow

3.1 Introduction

The goal in this problem is to generalize the previous problem -- simulate the positions and velocities of n nodes of a beam vs. time implicitly with the given final time and initial conditions and then plot the vertical position and velocity of the middle node vs. time. Following these calculations, the results will be printed to the screen.

3.2 Model and Methods

The positions and velocities of the nodes are calculated using Newton's method, and to make the calculation clearer, the positions (x_1, y_1), (x_2, y_2), (x_3, y_3) … is constructed in a vector and substituted with $\boldsymbol{q}$. The stretching elastic forces are denoted from 1 to n-1 and the bending elastic forces are denoted from 2 to n-1.

Then for implicit simulation, to make use of the functions given in the project, the discretized governing equations are equivalent to the following categorized equations:

$$\mathbf{f} \equiv \mathbf{f}_{g+v} + \mathbf{f}_s + \mathbf{f}_b = 0$$

$$\mathbf{f}_{g+v} = \begin{bmatrix} \frac{m_1}{\Delta t}\left[\frac{q(1)-x_1(k)}{\Delta t} - \dot{x}_1(k)\right] + (6\pi\mu R_1)\frac{q(1)-x_1(k)}{\Delta t} \\ \frac{m_1}{\Delta t}\left[\frac{q(2)-y_1(k)}{\Delta t} - \dot{y}_1(k)\right] + W_1 + (6\pi\mu R_1)\frac{q(2)-y_1(k)}{\Delta t} \\ \frac{m_2}{\Delta t}\left[\frac{q(3)-x_2(k)}{\Delta t} - \dot{x}_2(k)\right] + (6\pi\mu R_2)\frac{q(3)-x_2(k)}{\Delta t} \\ \frac{m_2}{\Delta t}\left[\frac{q(4)-y_2(k)}{\Delta t} - \dot{y}_2(k)\right] + W_2 + (6\pi\mu R_2)\frac{q(4)-y_2(k)}{\Delta t} \\ \frac{m_3}{\Delta t}\left[\frac{q(5)-x_3(k)}{\Delta t} - \dot{x}_3(k)\right] + (6\pi\mu R_3)\frac{q(5)-x_3(k)}{\Delta t} \\ \frac{m_3}{\Delta t}\left[\frac{q(6)-y_3(k)}{\Delta t} - \dot{y}_3(k)\right] + W_3 + (6\pi\mu R_3)\frac{q(6)-y_3(k)}{\Delta t} \\ \vdots \\ \frac{m_n}{\Delta t}\left[\frac{q(2n-1)-x_n(k)}{\Delta t} - \dot{x}_n(k)\right] + (6\pi\mu R_n)\frac{q(2n-1)-x_n(k)}{\Delta t} \\ \frac{m_n}{\Delta t}\left[\frac{q(2n)-y_n(k)}{\Delta t} - \dot{y}_n(k)\right] + W_n + (6\pi\mu R_n)\frac{q(2n)-y_n(k)}{\Delta t} \end{bmatrix}_{2n\times 1}$$

$$\mathbf{f}_s = \begin{bmatrix} \frac{\partial E_1^s}{\partial x_1} \\ \frac{\partial E_1^s}{\partial y_1} \\ \frac{\partial E_1^s}{\partial x_2} \\ \frac{\partial E_1^s}{\partial y_2} \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \begin{bmatrix} 0 \\ 0 \\ \frac{\partial E_2^s}{\partial x_2} \\ \frac{\partial E_2^s}{\partial y_2} \\ \frac{\partial E_2^s}{\partial x_3} \\ \frac{\partial E_2^s}{\partial y_3} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{\partial E_3^s}{\partial x_3} \\ \frac{\partial E_3^s}{\partial y_3} \\ \frac{\partial E_3^s}{\partial x_4} \\ \frac{\partial E_3^s}{\partial y_4} \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \cdots + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \frac{\partial E_{n-1}^s}{\partial x_{n-1}} \\ \frac{\partial E_{n-1}^s}{\partial y_{n-1}} \\ \frac{\partial E_{n-1}^s}{\partial x_n} \\ \frac{\partial E_{n-1}^s}{\partial y_n} \end{bmatrix}_{2n\times 1}$$

$$\mathbf{f}_b = \begin{bmatrix} \frac{\partial E_2^b}{\partial x_1} \\ \frac{\partial E_2^b}{\partial y_1} \\ \frac{\partial E_2^b}{\partial x_2} \\ \frac{\partial E_2^b}{\partial y_2} \\ \frac{\partial E_2^b}{\partial x_3} \\ \frac{\partial E_2^b}{\partial y_3} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \begin{bmatrix} 0 \\ 0 \\ \frac{\partial E_3^b}{\partial x_2} \\ \frac{\partial E_3^b}{\partial y_2} \\ \frac{\partial E_3^b}{\partial x_3} \\ \frac{\partial E_3^b}{\partial y_3} \\ \frac{\partial E_3^b}{\partial x_4} \\ \frac{\partial E_3^b}{\partial y_4} \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \frac{\partial E_{n-1}^b}{\partial x_{n-2}} \\ \frac{\partial E_{n-1}^b}{\partial y_{n-2}} \\ \frac{\partial E_{n-1}^b}{\partial x_{n-1}} \\ \frac{\partial E_{n-1}^b}{\partial y_{n-1}} \\ \frac{\partial E_{n-1}^b}{\partial y_n} \\ \frac{\partial E_{n-1}^b}{\partial y_n} \end{bmatrix}_{2n\times 1}$$

Then the Jacobian is calculated using matrixes above and their derivatives as shown below.

$$\mathbf{J} = \mathbf{J}_{g+v} + \mathbf{J}_{s1} + \mathbf{J}_{s2} + \mathbf{J}_{s3} + \cdots + \mathbf{J}_{s(n-1)} + \mathbf{J}_{b2} + \mathbf{J}_{b3} + \cdots + \mathbf{J}_{b(n-1)}$$

$$\mathbf{J}_{g+v}=\begin{bmatrix}
\frac{m_1}{\Delta t^2}+\frac{6\pi\mu R_1}{\Delta t} & 0 & 0 & 0 & \cdots & 0 & 0\\
0 & \frac{m_1}{\Delta t^2}+\frac{6\pi\mu R_1}{\Delta t} & 0 & 0 & \cdots & 0 & 0\\
0 & 0 & 0 & 0 & \cdots & 0 & 0\\
\vdots & & & & \ddots & & \\
0 & 0 & 0 & 0 & \cdots & 0 & 0\\
0 & 0 & 0 & 0 & \cdots & \frac{m_n}{\Delta t^2}+\frac{6\pi\mu R_n}{\Delta t} & 0\\
0 & 0 & 0 & 0 & \cdots & 0 & \frac{m_n}{\Delta t^2}+\frac{6\pi\mu R_n}{\Delta t}
\end{bmatrix}_{2n\times 2n}$$

$$\mathbf{J}_{s1}=\begin{bmatrix}
\frac{\partial^2 E_1^s}{\partial x_1\partial x_1} & \frac{\partial^2 E_1^s}{\partial x_1\partial y_1} & \frac{\partial^2 E_1^s}{\partial x_1\partial x_2} & \frac{\partial^2 E_1^s}{\partial x_1\partial y_2} & 0 & 0 & 0 & \cdots & 0 & 0\\
\frac{\partial^2 E_1^s}{\partial y_1\partial x_1} & \frac{\partial^2 E_1^s}{\partial y_1\partial y_1} & \frac{\partial^2 E_1^s}{\partial y_1\partial x_2} & \frac{\partial^2 E_1^s}{\partial y_1\partial y_2} & 0 & 0 & 0 & \cdots & 0 & 0\\
\frac{\partial^2 E_1^s}{\partial x_2\partial x_1} & \frac{\partial^2 E_1^s}{\partial x_2\partial x_1} & \frac{\partial^2 E_1^s}{\partial x_2\partial x_2} & \frac{\partial^2 E_1^s}{\partial x_2\partial y_2} & 0 & 0 & 0 & \cdots & 0 & 0\\
\frac{\partial^2 E_1^s}{\partial y_2\partial x_1} & \frac{\partial^2 E_1^s}{\partial y_2\partial y_1} & \frac{\partial^2 E_1^s}{\partial y_2\partial x_2} & \frac{\partial^2 E_1^s}{\partial y_2\partial y_2} & 0 & 0 & 0 & \cdots & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0\\
\vdots & & & & & & & \ddots & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}_{2n\times 2n}$$

$$\mathbf{J}_{s2}=\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0\\
0 & 0 & \frac{\partial^2 E_2^s}{\partial x_2\partial x_2} & \frac{\partial^2 E_2^s}{\partial x_2\partial y_2} & \frac{\partial^2 E_2^s}{\partial x_2\partial x_3} & \frac{\partial^2 E_2^s}{\partial x_2\partial y_3} & 0 & \cdots & 0 & 0\\
0 & 0 & \frac{\partial^2 E_2^s}{\partial y_2\partial x_2} & \frac{\partial^2 E_2^s}{\partial y_2\partial y_2} & \frac{\partial^2 E_2^s}{\partial y_2\partial x_3} & \frac{\partial^2 E_2^s}{\partial y_2\partial y_3} & 0 & \cdots & 0 & 0\\
0 & 0 & \frac{\partial^2 E_2^s}{\partial x_3\partial x_2} & \frac{\partial^2 E_2^s}{\partial x_3\partial y_2} & \frac{\partial^2 E_2^s}{\partial x_3\partial x_3} & \frac{\partial^2 E_2^s}{\partial x_3\partial y_3} & 0 & \cdots & 0 & 0\\
0 & 0 & \frac{\partial^2 E_2^s}{\partial y_3\partial x_2} & \frac{\partial^2 E_2^s}{\partial y_3\partial y_2} & \frac{\partial^2 E_2^s}{\partial y_3\partial x_3} & \frac{\partial^2 E_2^s}{\partial y_3\partial y_3} & 0 & \cdots & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0\\
\vdots & & & & & & & \ddots & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}_{2n\times 2n}$$

$$\mathbf{J}_{s3}=\begin{bmatrix}
0&0&0&0&0&0&0&0&0&\cdots&0\\
0&0&0&0&0&0&0&0&0&\cdots&0\\
0&0&0&0&0&0&0&0&0&\cdots&0\\
0&0&0&0&0&0&0&0&0&\cdots&0\\
0&0&0&0&\frac{\partial^2 E_3^s}{\partial x_3\partial x_3}&\frac{\partial^2 E_3^s}{\partial x_3\partial y_3}&\frac{\partial^2 E_3^s}{\partial x_3\partial x_4}&\frac{\partial^2 E_3^s}{\partial x_3\partial y_4}&0&\cdots&0\\
0&0&0&0&\frac{\partial^2 E_3^s}{\partial y_3\partial x_3}&\frac{\partial^2 E_3^s}{\partial y_3\partial y_3}&\frac{\partial^2 E_3^s}{\partial y_3\partial x_4}&\frac{\partial^2 E_3^s}{\partial y_3\partial y_4}&0&\cdots&0\\
0&0&0&0&\frac{\partial^2 E_3^s}{\partial x_4\partial x_3}&\frac{\partial^2 E_3^s}{\partial x_4\partial y_3}&\frac{\partial^2 E_3^s}{\partial x_4\partial x_4}&\frac{\partial^2 E_3^s}{\partial x_4\partial y_4}&0&\cdots&0\\
0&0&0&0&\frac{\partial^2 E_3^s}{\partial y_4\partial x_3}&\frac{\partial^2 E_3^s}{\partial y_4\partial y_3}&\frac{\partial^2 E_3^s}{\partial y_4\partial x_4}&\frac{\partial^2 E_3^s}{\partial y_4\partial y_4}&0&\cdots&0\\
0&0&0&0&0&0&0&0&0&&0\\
\vdots&&&&&&&&&\ddots&\\
0&0&0&0&0&0&0&0&&0&0
\end{bmatrix}_{2n\times 2n}$$

…

$$\mathbf{J}_{s(n-1)}=\begin{bmatrix}
0&0&\cdots&0&0&0&0&0&0&0\\
&&\ddots&&&&&&&\vdots\\
0&0&\cdots&0&0&0&0&0&0&0\\
0&0&\cdots&0&0&0&\frac{\partial^2 E_1^s}{\partial x_1\partial x_1}&\frac{\partial^2 E_1^s}{\partial x_1\partial y_1}&\frac{\partial^2 E_1^s}{\partial x_1\partial x_2}&\frac{\partial^2 E_1^s}{\partial x_1\partial y_2}\\
0&0&\cdots&0&0&0&\frac{\partial^2 E_1^s}{\partial y_1\partial x_1}&\frac{\partial^2 E_1^s}{\partial y_1\partial y_1}&\frac{\partial^2 E_1^s}{\partial y_1\partial x_2}&\frac{\partial^2 E_1^s}{\partial y_1\partial y_2}\\
0&0&\cdots&0&0&0&\frac{\partial^2 E_1^s}{\partial x_2\partial x_1}&\frac{\partial^2 E_1^s}{\partial x_2\partial x_1}&\frac{\partial^2 E_1^s}{\partial x_2\partial x_2}&\frac{\partial^2 E_1^s}{\partial x_2\partial y_2}\\
0&0&\cdots&0&0&0&\frac{\partial^2 E_1^s}{\partial y_2\partial x_1}&\frac{\partial^2 E_1^s}{\partial y_2\partial y_1}&\frac{\partial^2 E_1^s}{\partial y_2\partial x_2}&\frac{\partial^2 E_1^s}{\partial y_2\partial y_2}
\end{bmatrix}_{2n\times 2n}$$

$$\mathbf{J}_{b2} = \begin{bmatrix}
\frac{\partial^2 E_2^b}{\partial x_1 \partial x_1} & \frac{\partial^2 E_2^b}{\partial x_1 \partial y_1} & \frac{\partial^2 E_2^b}{\partial x_1 \partial x_2} & \frac{\partial^2 E_2^b}{\partial x_1 \partial y_2} & \frac{\partial^2 E_2^b}{\partial x_1 \partial x_3} & \frac{\partial^2 E_2^b}{\partial x_1 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
\frac{\partial^2 E_2^b}{\partial y_1 \partial x_1} & \frac{\partial^2 E_2^b}{\partial y_1 \partial y_1} & \frac{\partial^2 E_2^b}{\partial y_1 \partial x_2} & \frac{\partial^2 E_2^b}{\partial y_1 \partial y_2} & \frac{\partial^2 E_2^b}{\partial y_1 \partial x_3} & \frac{\partial^2 E_2^b}{\partial y_1 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
\frac{\partial^2 E_2^b}{\partial x_2 \partial x_1} & \frac{\partial^2 E_2^b}{\partial x_2 \partial y_1} & \frac{\partial^2 E_2^b}{\partial x_2 \partial x_2} & \frac{\partial^2 E_2^b}{\partial x_2 \partial y_2} & \frac{\partial^2 E_2^b}{\partial x_2 \partial x_3} & \frac{\partial^2 E_2^b}{\partial x_2 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
\frac{\partial^2 E_2^b}{\partial y_2 \partial x_1} & \frac{\partial^2 E_2^b}{\partial y_2 \partial y_1} & \frac{\partial^2 E_2^b}{\partial y_2 \partial x_2} & \frac{\partial^2 E_2^b}{\partial y_2 \partial y_2} & \frac{\partial^2 E_2^b}{\partial y_2 \partial x_3} & \frac{\partial^2 E_2^b}{\partial y_2 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
\frac{\partial^2 E_2^b}{\partial x_3 \partial x_1} & \frac{\partial^2 E_2^b}{\partial x_3 \partial y_1} & \frac{\partial^2 E_2^b}{\partial x_3 \partial x_2} & \frac{\partial^2 E_2^b}{\partial x_3 \partial y_2} & \frac{\partial^2 E_2^b}{\partial x_3 \partial x_3} & \frac{\partial^2 E_2^b}{\partial x_3 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
\frac{\partial^2 E_2^b}{\partial y_3 \partial x_1} & \frac{\partial^2 E_2^b}{\partial y_3 \partial y_1} & \frac{\partial^2 E_2^b}{\partial y_3 \partial x_2} & \frac{\partial^2 E_2^b}{\partial y_3 \partial y_2} & \frac{\partial^2 E_2^b}{\partial y_3 \partial x_3} & \frac{\partial^2 E_2^b}{\partial y_3 \partial y_3} & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
\vdots & & & & & & & & \ddots & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0
\end{bmatrix}_{2n \times 2n}$$

$$\mathbf{J}_{b3} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial x_2 \partial x_2} & \frac{\partial^2 E_3^b}{\partial x_2 \partial y_2} & \frac{\partial^2 E_3^b}{\partial x_2 \partial x_3} & \frac{\partial^2 E_3^b}{\partial x_2 \partial y_3} & \frac{\partial^2 E_3^b}{\partial x_2 \partial x_4} & \frac{\partial^2 E_3^b}{\partial x_2 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial y_2 \partial x_2} & \frac{\partial^2 E_3^b}{\partial y_2 \partial y_2} & \frac{\partial^2 E_3^b}{\partial y_2 \partial x_3} & \frac{\partial^2 E_3^b}{\partial y_2 \partial y_3} & \frac{\partial^2 E_3^b}{\partial y_2 \partial x_4} & \frac{\partial^2 E_3^b}{\partial y_2 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial x_3 \partial x_2} & \frac{\partial^2 E_3^b}{\partial x_3 \partial y_2} & \frac{\partial^2 E_3^b}{\partial x_3 \partial x_3} & \frac{\partial^2 E_3^b}{\partial x_3 \partial y_3} & \frac{\partial^2 E_3^b}{\partial x_3 \partial x_4} & \frac{\partial^2 E_3^b}{\partial x_3 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial y_3 \partial x_2} & \frac{\partial^2 E_3^b}{\partial y_3 \partial y_2} & \frac{\partial^2 E_3^b}{\partial y_3 \partial x_3} & \frac{\partial^2 E_3^b}{\partial y_3 \partial y_3} & \frac{\partial^2 E_3^b}{\partial y_3 \partial x_4} & \frac{\partial^2 E_3^b}{\partial y_3 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial x_4 \partial x_2} & \frac{\partial^2 E_3^b}{\partial x_4 \partial y_2} & \frac{\partial^2 E_3^b}{\partial x_4 \partial x_3} & \frac{\partial^2 E_3^b}{\partial x_4 \partial y_3} & \frac{\partial^2 E_3^b}{\partial x_4 \partial x_4} & \frac{\partial^2 E_3^b}{\partial x_4 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & \frac{\partial^2 E_3^b}{\partial y_4 \partial x_2} & \frac{\partial^2 E_3^b}{\partial y_4 \partial y_2} & \frac{\partial^2 E_3^b}{\partial y_4 \partial x_3} & \frac{\partial^2 E_3^b}{\partial y_4 \partial y_3} & \frac{\partial^2 E_3^b}{\partial y_4 \partial x_4} & \frac{\partial^2 E_3^b}{\partial y_4 \partial y_4} & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
\vdots & & & & & & & & & \ddots & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 0
\end{bmatrix}_{2n \times 2n}$$

…

$$\mathbf{J}_{b(n-1)} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & \ddots & & & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-2}\partial y_n} \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-2}\partial y_n} \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial x_{n-1}\partial y_n} \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial y_{n-1}\partial y_n} \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial x_n\partial y_n} \\ 0 & 0 & \cdots & 0 & 0 & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial x_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial y_{n-2}} & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial x_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial y_{n-1}} & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial x_n} & \frac{\partial^2 E^b_{n-1}}{\partial y_n\partial y_n} \end{bmatrix}_{2n\times 2n}$$

The script first assigns a value `21` for the number of nodes, `N`. Then the script assigns the same values as the previous script for the attributes of the beam, fluid and sphere, `r0`, `E`, `EA`, `EI`, `l`, `mu`, `rho_fluid`, `rho_metal`. The length of each segment between two nodes, `l_k`, and the weights and masses of spheres, `W` and `m`, are calculated using the given equations, as shown below.

```
l_k=l/(N-1);    %length of each segment between two nodes
R=ones(N,1)*l_k/10; %radii of the spheres
R((N+1)/2)=0.025;
[W1,m1]=WeightMass(R(1),rho_metal,rho_fluid); %weights and masses
[W_mid,m_mid]=WeightMass(R((N+1)/2),rho_metal,rho_fluid);
W=ones(N,1)*W1;
W((N+1)/2)=W_mid;
m=ones(N,1)*m1;
m((N+1)/2)=m_mid;
```

For implicit simulation, the script assigns `1e-2` for the timestep size, `dt`, and `50` for the final time, `tfinal`. The steps in for-loop, `tstep` is obtained the same as the first part.

The values of the sphere positions at each timestep, `x` and `y`, and the sphere velocities at each timestep, `x_dot` and `y_dot`, should be initialized before the execution of for-loop.

A sample of these is as shown below.

```
x=zeros(N,1);
```

The values of the positions of the middle node, `x_mid` and `y_mid`, and the velocities of the middle node, `x_mid_dot` and `y_mid_dot`, should also be initialized. A sample of these is as shown below.

```
x_mid=zeros(tstep+1,1);
```

Then the initial conditions are assigned to `x`, `y`, `x_dot` and `y_dot`, using MATLAB's for-loop, as shown below.

```
for k=1:N
    x(k)=l_k*(k-1);x_dot(k)=0;
    y(k)=0;y_dot(k)=0;
end
```

The initial positions and velocities of the middle node are also assigned to the first entry in the `x_mid`, `y_mid`, `x_mid_dot` and `y_mid_dot`, as shown below.

```
x_mid(1)=x((N+1)/2);
y_mid(1)=y((N+1)/2);
x_mid_dot(1)=x_dot((N+1)/2);
y_mid_dot(1)=y_dot((N+1)/2);
```

Before the simulation, the video should be created and opened, as shown below.

```
v=VideoWriter('BeamSimulationProblem3.avi');
open(v);
```

Inside for-loop, the Newton's method is used to solve the `q` shown at the beginning with a while-loop.

After the value of `q` is obtained by Newton's method, the velocities of the nodes, `x_dot` and `y_dot`, should be calculated with `x`, `y` and `q` first and then the positions of the nodes, `x` and `y`, should be updated. MATLAB's `plot` and `drawnow` functions are used at each timestep to create an animation of the beam shape. MATLAB's `writeVideo` is used to save each frame at each timestep to the video.

Then the positions of middle node are assigned to `x_mid(k+1)` and `y_mid(k+1)` and

the velocities of middle node are assigned to `x_mid_dot(k+1)` and `y_mid_dot(k+1)`.

The for-loop is as shown below.

```
for k=1:tstep
    ...
    %Set the starting points
    q=zeros(2*N,1);
    for i=1:N
        odd=2*i-1;
        even=2*i;
        q(odd)=x(i);
        q(even)=y(i);
    end
    while error>tol
        %Initialize the functions
        f=zeros(2*N,1);
        Fs=zeros(2*N,1);
        Fb=zeros(2*N,1);
        %Initialize the Jacobian
        J=zeros(2*N,2*N);
        Js=zeros(2*N,2*N);
        Jb=zeros(2*N,2*N);
        %Compute the functions and the Jacobian
        for i=1:N-1
            Fs_s=2*(i-1)+1;
            Fs_e=Fs_s+3;
            Fs(Fs_s:Fs_e,1)=Fs(Fs_s:Fs_e,1)+...
                gradEs(q(Fs_s),q(Fs_s+1),q(Fs_s+2),q(Fs_s+3),l_k,EA);
            Js(Fs_s:Fs_e,Fs_s:Fs_e)=Js(Fs_s:Fs_e,Fs_s:Fs_e)+...
                hessEs(q(Fs_s),q(Fs_s+1),q(Fs_s+2),q(Fs_s+3),l_k,EA)';
        end
        for i=2:N-1
            Fb_s=2*(i-1)-1;
            Fb_e=Fb_s+5;
            Fb(Fb_s:Fb_e,1)=Fb(Fb_s:Fb_e,1)+...
                gradEb(q(Fb_s),q(Fb_s+1),q(Fb_s+2),q(Fb_s+3),q(Fb_s+4),q(Fb_s+5),l_k,EI);
            Jb(Fb_s:Fb_e,Fb_s:Fb_e)=Jb(Fb_s:Fb_e,Fb_s:Fb_e)+...
                hessEb(q(Fb_s),q(Fb_s+1),q(Fb_s+2),q(Fb_s+3),q(Fb_s+4),q(Fb_s+5),l_k,EI)';
        end
```

```
    for i=1:N
        odd=2*i-1;
        even=2*i;
        f(odd)=m(i)/dt*((q(odd)-x(i))/dt-x_dot(i))+...
            (6*pi*mu*R(i))*((q(odd)-x(i))/dt);
        f(even)=m(i)/dt*((q(even)-y(i))/dt-y_dot(i))+...
            W(i)+(6*pi*mu*R(i))*((q(even)-y(i))/dt);
        J(odd)=m(i)/(dt^2)+(6*pi*mu*R(i))/dt;
        J(even)=m(i)/(dt^2)+(6*pi*mu*R(i))/dt;
    end
    f=f+Fs+Fb;
    J=J+Js+Jb;
    %Newton's update
    dq=J\f;
    q=q-dq;
    %Compute the error
    error=sum(abs(f));
end
%Velocity
for i=1:N
    odd=2*(i-1)+1;
    even=2*i;
    x_dot(i)=(q(odd)-x(i))/dt;
    y_dot(i)=(q(even)-y(i))/dt;
end
%Position
for i=1:N
    odd=2*(i-1)+1;
    even=2*i;
    x(i)=q(odd);
    y(i)=q(even);
end

%Plot the shape of beam
figure(1)
plot(x,y,'.-');
xlabel('x (m)');
ylabel('y (m)');
title(num2str(k*dt, 'time %4.2f s'));
axis equal
drawnow
```

```
    %Save frame to video
    writeVideo(v,getframe(gcf));

    x_mid(k+1)=x((N+1)/2);
    y_mid(k+1)=y((N+1)/2);
    x_mid_dot(k+1)=x_dot((N+1)/2);
    y_mid_dot(k+1)=y_dot((N+1)/2);
end
```

The shooting of the video now can be finished, as shown below.

```
close(v);
```

After these calculations, the plot of the positions and velocities of the middle node over time for implicit simulation are printed to the screen using MATLAB's `plot` function. After being created, they can be formatted with appropriate titles and axis labels, as shown below.

```
%%Plot the positions and velocities of the middle node vs. time
t=0:dt:tfinal;
%Create the plot
figure(2)
plot(t,x_mid);
hold on
plot(t,y_mid);
hold off
%Format the plot
xlabel('time (sec)');
ylabel('position (m)');
legend('x-position','y-position','Location','Best');
title('the Middle Node Position for Implicit Simulation');
%Create the plot
figure(3)
plot(t,x_mid_dot);
hold on
plot(t,y_mid_dot);
hold off
%Format the plot
xlabel('time (sec)');
ylabel('velocity (m/sec)');
```

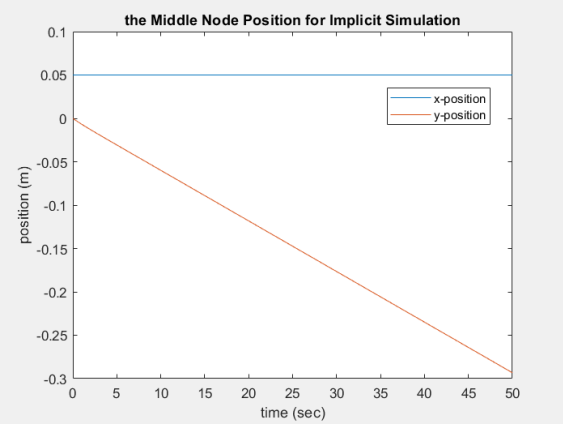
```
legend('x-velocity','y-velocity','Location','Best');
title('the Middle Node Velocity for Implicit Simulation');
```

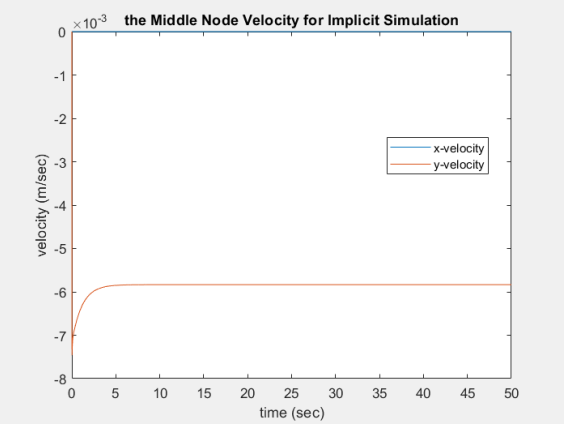
The terminal velocities of the middle node obtained from the `x_mid_dot` and `y_mid_dot` are printed to the screen using MATLAB's `fprintf` function. A sample of these is as shown below.

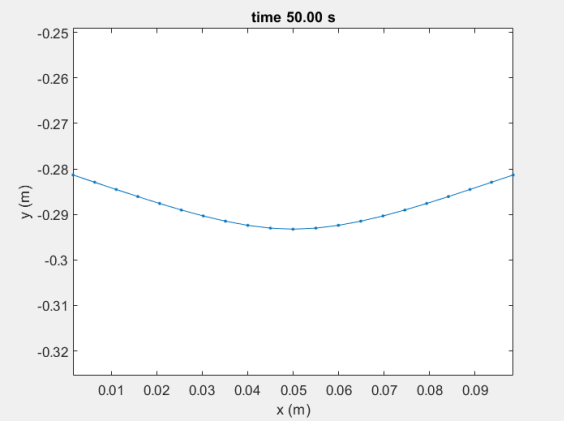
```
fprintf('The terminal velocity along x-axis is %f.\n',x_mid_dot(tstep+1));
```

3.3 Calculations and Results

When the program is executed, the following output is printed to the screen.








```
The terminal velocity along x-axis is 0.000000.
The terminal velocity along y-axis is -0.005834.
```

3.4 Discussion

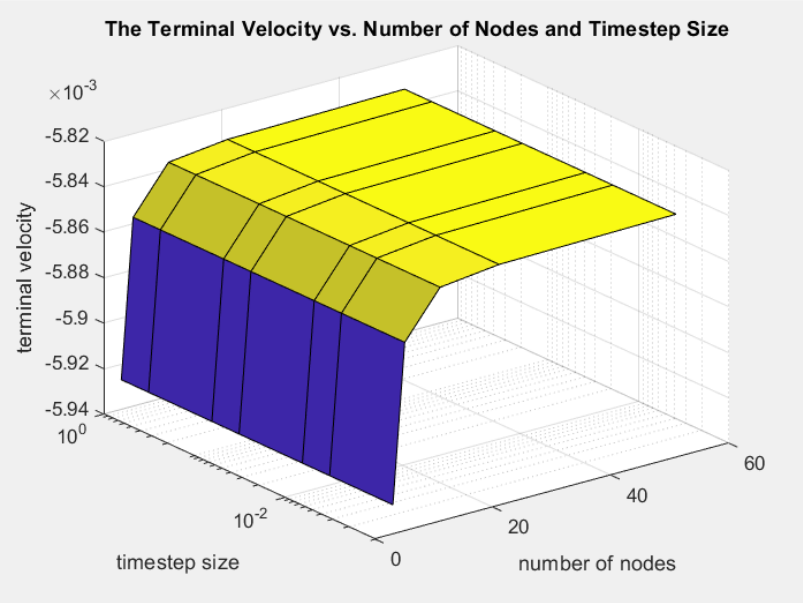
To analyze the significance of spatial discretization and temporal discretization, the non-zero terminal velocity of the middle node along the y-axis should be calculated with different N and $\triangle t$. After manually changing the N and $\triangle t$, the results are as shown below.

$\triangle t$ \ N	3	5	11	21	51
1	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.5	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.1	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.05	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.01	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.005	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833
0.001	-0.005927	-0.005857	-0.005837	-0.005834	-0.005833

After the result data in the table above input in MATLAB, the plot of terminal velocity vs. the number of nodes and the timestep size can be printed to the screen using MATLAB's `surf` function. After being created, it can be formatted with appropriate titles and axis labels, as shown below.

```
dt=[1,0.5,0.1,0.05,0.01,0.005,0.001];
N=[3,5,11,21,51];
TerminalVelocity=[-0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833;
                  -0.005927,-0.005857,-0.005837,-0.005834,-0.005833];
surf(N,dt,TerminalVelocity);
ay=gca;
ay.YScale='log';
xlabel('number of nodes');
ylabel('timestep size');
zlabel('terminal velocity');
title('The Terminal Velocity vs. Number of Nodes and Timestep Size');
```

When the program is executed, the following output is printed to the screen.



As shown in the table and plot above, the terminal velocity changes very little with respect to $\triangle t$ and basically remains the same value. This means that the implicit simulation is quite accurate so $\triangle t$ with very small value is not needed to obtain reasonable results.

On the other hand, the terminal velocity varies with the number of nodes, which means the number of nodes does affect the accuracy of the results a lot compared with the timestep size. The more nodes used to simulate the beam, the more accurate the results will be.

In conclusion, to obtain reasonable results with limited computing cost budget, it is better to change the temporal discretization instead of changing the spatial discretization.

4 Elastic Beam Bending

4.1 Introduction

The goal in this problem is to simulate the beam vs. time implicitly with the given final time and initial conditions and then plot the maximum vertical displacement of the beam vs. time. Following these calculations, the results will be printed to the screen.

4.2 Model and Methods

The beam is represented as a mass-spring system with n nodes. The positions of the nodes are calculated using Newton's method, and to make the calculation clearer, the positions (x_1, y_1), (x_2, y_2), (x_3, y_3) … is constructed in a vector and substituted with $\boldsymbol{q}$. The stretching

elastic forces are denoted from 1 to n-1 and the bending elastic forces are denoted from 2 to n-1.

Since the formulation of the elastic energy remains the same as the previous two problems, then to make use of the functions given in the project, combined with Equation (18), (19) and (20), the discretized governing equations are equivalent to the following categorized equations:

$$\mathbf{f} \equiv \begin{bmatrix} x_1 \\ y_1 \\ 0 \\ \vdots \\ 0 \\ y_n \end{bmatrix}_{2n\times 1} - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ P \\ \vdots \\ 0 \end{bmatrix}_{2n\times 1} + \mathbf{f}_s + \mathbf{f}_b = 0$$

The $\mathbf{f}_s$ and $\mathbf{f}_b$ remain the same as the previous problem, except for the first, second and last rows of their addition should be changed to 0.

Then the Jacobian is calculated using matrixes above and their derivatives as shown below.

$$\mathbf{J} = \mathbf{J}_{cons} + \mathbf{J}_{s1} + \mathbf{J}_{s2} + \mathbf{J}_{s3} + \cdots + \mathbf{J}_{s(n-1)} + \mathbf{J}_{b2} + \mathbf{J}_{b3} + \cdots + \mathbf{J}_{b(n-1)}$$

$$\mathbf{J}_{cons} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}_{2n\times 2n}$$

The $\mathbf{J}_{s1}$, $\mathbf{J}_{s2}$, …, $\mathbf{J}_{s(n\text{-}1)}$ and $\mathbf{J}_{b2}$, $\mathbf{J}_{b3}$, …, $\mathbf{J}_{b(n\text{-}1)}$ remain the same as the previous problem, except for the first, second and last rows of their addition should be changed to 0.

The script first assigns a value `50` for the number of nodes, `N`. Then the script assigns the values for the attributes of the beam, `1` for the length of the beam, `l`, `l/(N-1)` for the length of each segment between two nodes, `l_k`, `0.013` for the outer radius of the circular-tube section of the beam, `R`, `0.011` for the inner radius, `r`, `-2000` for the force, `P`, `0.75` for the distance between the location of the applied force and the left-hand edge, `d`, `7e10` for the modulus of elasticity, `E`, `pi/4*(R^4-r^4)` for the moment of inertia,

`I`, `E*I` for the bending stiffness, `EI`, `E*pi*(R^2-r^2)` for the stretching stiffness, `EA`, `2700` for the density of aluminum, `rho`, and `pi*(R^2-r^2)*l*rho/(N-1)` for the mass of nodes, `m`.

For implicit simulation, the script assigns `1e-2` for the timestep size, `dt`, and `1` for the final time, `tfinal`. The steps in for-loop, `tstep` is obtained the same as the first part.

The initialization of the positions and velocities of the nodes, `x`, `y`, `x_dot`, and `y_dot`, and the assignment of the initial conditions are the same as the previous problem. The value of the maximum vertical displacement, `y_max`, should also be initialized and an initial value should be assigned to its first entry, as shown below.

```
y_max=zeros(tstep+1,1);
y_max(1)=0;
```

Inside for-loop, the Newton's method is used to solve the `q` shown at the beginning with a while-loop.

MATLAB's `min` function is used to find the node that is closest to the force location, `Ind`. The `Ind` then is used to add `P` into the function matrix, `f`.

After the value of `q` is obtained by Newton's method, the positions and velocities of the nodes, `x`, `y`, `x_dot`, and `y_dot`, should be updated. MATLAB's `max` function is used to find the maximum vertical displacement of the beam, `y_max(k+1)`.

The for-loop is as shown below.

```
for k=1:tstep
    ...
    while error>tol
        ...
        %Find the location of the force
        [distance,Ind]=min(abs(x-d));
        f=Fs+Fb;
        f(1)=q(1);
        f(2)=q(2);
        f(2*N)=q(2*N);
        f(Ind*2)=f(Ind*2)-P;
        J=Js+Jb;
        J(1,:)=0;
        J(1,1)=1;
```

```
        J(2,:)=0;
        J(2,2)=1;
        J(2*N,:)=0;
        J(2*N,2*N)=1;
        ...
    end
    ...
    %Find the maximum vertical displacement
    y_max(k+1)=(-1)*max(abs(y));
end
```

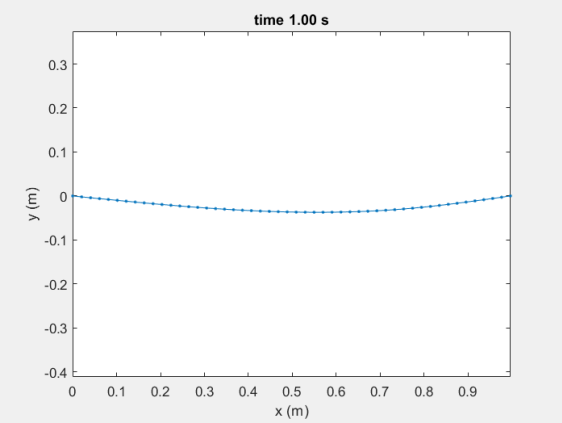
A video named `BeamSimulationProblem4.avi` showing the shape of the beam is created using the same method as the previous problem.

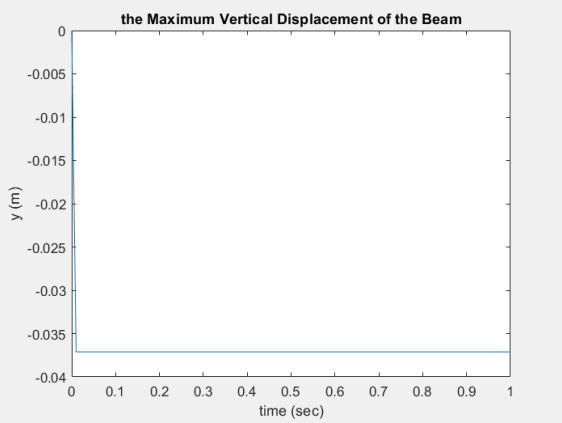
After these calculations, the plot of the maximum vertical displacement of the beam over time for implicit simulation is printed to the screen using MATLAB's `plot` function. After being created, it can be formatted with appropriate titles and axis labels, as shown below.

```
%%Plot the maximum vertical displacement vs. time
t=0:dt:tfinal;
%Create the plot
figure(2)
plot(t,y_max);
%Format the plot
xlabel('time (sec)');
ylabel('y (m)');
title('the Maximum Vertical Displacement of the Beam');
```

4.3 Calculations and Results

When the program is executed, the following output is printed to the screen.





It can be seen that the y_{max} eventually reach a steady value.

4.4 Discussion

To verify the accuracy of this simulation, the maximum displacement obtained from the `y_max` and theoretical prediction from Euler beam theory, `ymax_theo`, obtained with the Equation (21) should be acquired, as shown below.

```
%%Calculate the theoretical solution
c=min(d,l-d);
ymax_theo=P*c*(l^2-c^2)^1.5/(9*sqrt(3)*E*I*l);
%%Display the maximum displacements
fprintf('The theoretical maximum displacement is %f.\n',ymax_theo);
fprintf('The maximum displacement from implicit simulation is %f.\n',y_max(tstep+1));
```

When the program is executed, the following output is printed to the screen.

```
The theoretical maximum displacement is -0.038045.
The maximum displacement from implicit simulation is -0.037109.
```

With the error of 2.460%, it can be concluded that from the aspect of beam theory, this simulation is accurate.

To compare this simulation over the predictions from beam theory, the maximum displacement should be calculated with different P. After manually changing the P, the results are as shown below.

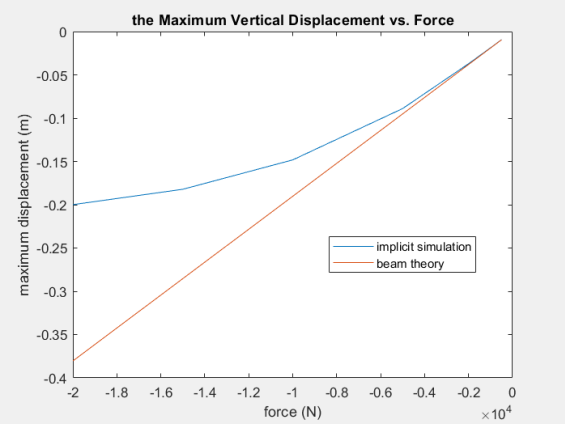
P (N)	Implicit simulation y_{max} (m)	Beam theory y_{max} (m)
-20000	-0.199924	-0.380449

-15000	-0.182167	-0.285337
-10000	-0.148229	-0.190225
-5000	-0.088935	-0.095112
-2000	-0.037109	-0.038045
-1000	-0.018674	-0.019022
-500	-0.009352	-0.009511

After the result data in the table above input in MATLAB, the plot of the maximum vertical displacements vs. *P* can be printed to the screen using MATLAB's `plot` function. After being created, it can be formatted with appropriate titles and axis labels, as shown below.

```
%%Plot the maximum vertical displacements vs. P
P=[-20000,-15000,-10000,-5000,-2000,-1000,-500];
y_max=[-0.199924,-0.182167,-0.148229,-0.088935,-0.037109,-0.018674,-0.009352];
ymax_theo=[-0.380449,-0.285337,-0.190225,-0.095112,-0.038045,-0.019022,-0.009511];
figure(3)
plot(P,y_max);
hold on
plot(P,ymax_theo);
hold off
xlabel('force (N)');
ylabel('maximum displacement (m)');
legend('implicit simulation','beam theory','Location','Best');
title('the Maximum Vertical Displacement vs. Force');
```

When the program is executed, the following output is printed to the screen.



It can be seen that when the absolute value of force becomes larger, the maximum displacement obtained from implicit simulation is non-linear while that obtained from

beam theory is linear. Then it can be concluded that the beam theory is not valid when big deformation happens while implicit simulation is capable of handling large deformation.

Since the error between implicit simulation and the beam theory is quite small when the force is -2000 N, it can be treated as that the two solutions begin to diverge when the force has absolute value larger than 2000.

Conclusion

In this project, four motion problems are solved using implicit numerical simulation. In the former two problems, explicit numerical simulation is also used and after comparing it with the implicit simulation, it can be concluded that the implicit simulation can achieve more accurate results even with larger timestep size. Then although the computing time of explicit simulation is smaller than that of implicit simulation with the same timestep size, the implicit simulation still does not cost too much time with larger timestep size. In the last problem, the implicit numerical simulation of a beam is compared with the beam theory. After comparison, it can be concluded that the implicit simulation works better because it can be applied to larger conditions while the beam theory only works when the deformation is small.

In general, implicit numerical simulation is an effective method to solve equations of motion of type $m_i\ddot{x}_i = f_i(\mathbf{x})$. It uses reasonable computing time with larger timestep size and is more accurate at the same time. In addition, it can generate more reasonable results compared with certain theory.