Homework 1 Report

John Meshinsky

***Abstract*— This homework covers the simulation of Spring network systems and looking into the difference of Implicit and Explicit Euler Methods of calculations.**

I. Question 1

A. Pesudocode

The Implicit Method Functions: {

gradEs(xk, yk, xkp1, ykp1, l_k, k):

This function calculates the spring energy gradient of the node's current position to its next. It takes the current and next positions of the nodes and the spring and length between them to return a force array. This function was used in lecture 3 Sim Code.

INPUT:
xk, yk: coordinates of current node point
xkp1, ykp1: coordinates of next node point
l_k: reference (rest) length of spring
k: spring stiffness constant

OUTPUT:
F: 4-element array [dE/dxk, dE/dyk, dE/dxkp1, dE/dykp1]

hessEs(xk, yk, xkp1, ykp1, l_k, k):

This function calculates the Hessian of the spring energy node's current position to its next. This function is used for the spring network Jacobian function in the calculations. This function was used in lecture 3 Sim Code.

INPUT:
xk, yk: coordinates of current node point
xkp1, ykp1: coordinates of next node point
l_k: reference (rest) length of spring
k: spring stiffness constant

OUTPUT:
F: 4x4 symmetric Hessian matrix

getFexternal(m):

This function calculates the force of gravity on the nodes. It calculates gravity in the negative y direction.

INPUT:
m: mass array (size ndof = 2*number_of_nodes)

OUTPUT:
W: weight/force array (size ndof)

getForceJacobian(x_new, x_old, u_old, stiffness_matrix, index_matrix, m, dt, l_k)"

This function calculates the total force vector and Jacobian matrix for a given current position of a node and this next position. This function calls on gradEs(), hessEs(), and getFexternal() functions to combine them for the resulting change. This function is used for the Implicit Method Simulation. This function was used in lecture 3 Sim Code.

INPUT:
x_new: current position estimate
x_old: position at previous time step
u_old: velocity at previous time step
stiffness_matrix: array of spring stiffnesses
index_matrix: array mapping springs to node DOFs
m: mass array
dt: time step size
l_k: rest lengths of springs

OUTPUT:
f: residual force vector
J: Jacobian matrix

myInt(t_new, x_old, u_old, free_DOF, stiffness_matrix, index_matrix, m, dt):

This function calculates the positions of the free Nodes by using Implicit Euler integration with Newton-Raphson. This function calls getForceJacobian() function to use the Jacobian and total forces to find the delta of the positions. This function only changes it for the specific indexed positions of the free Nodes.

INPUT:
t_new: new time value
x_old: position at previous time
u_old: velocity at previous time
free_DOF: indices of degrees of freedom that can move
stiffness_matrix, index_matrix, m, dt: system parameters

OUTPUT:
x_new: position at new time

plot(x, index_matrix, t):

This function makes the figures of the Mass-Spring system in the given time, connecting the nodes with springs. The nodes are represented with blue circles with blue lines

connecting them representing the springs that connect them.This function was used in lecture 3 Sim Code.

INPUT:
x: position vector (all DOFs)
index_matrix: spring connectivity
t: current time

OUTPUT:
Creates a new 2D grid figure.

Main function (Implicit) (most code is modified from the lecture 3 Sim code):

First two .txt files are created:
nodes.txt: a matrix of all the node x, y, positions in the system
Springs.txt: a matrix of the indices of the springs and their stiffnesses

The .txt files are converted to a matrix of all the node x, y, positions, matrix of the indices of the springs, and an array of the spring stiffnesses.

The masses and length of the spring values are also set up.
The time array is also set up using the max time, being 100 seconds, and the chosen time step, dt = 0.1.

An array of the indexed positions of the free nodes, Node 1 and Node 3, are recorded to call and update their positions over time

Two empty arrays are made to record the y-positions of the free Nodes to graph later.

The figure of the initial positions of the Nodes is then made.

The Function then enters a loop over the time-array calling the myInt() function to find the new positions and velocity of the free nodes over time, occasionally producing a figure of new node positions if the time is t=0.1, 1.0, 10.0, or 100. This will also append the new y-corrinadates of the free Nodes in the arrays made earlier

After the loop, a plot is made of the y_coordinates of the free nodes and how they changed over time.
}

Explicit Euler function: {

getForceEx(x_new, x_old, u_old, stiffness_matrix, index_matrix, m, dt, l_k)"

This function calculates the total force vector at this next position. This function is a modified version of the getForceJacobian() function. This function calls on gradEs(), hessEs(), and getFexternal() functions to combine them for the resulting change. This function is used for the Explicit Method Simulation. This function was used in lecture 3 Sim Code.

INPUT:
x_new: current position estimate
x_old: position at previous time step
u_old: velocity at previous time step
stiffness_matrix: array of spring stiffnesses
index_matrix: array mapping springs to node DOFs
m: mass array
dt: time step size
l_k: rest lengths of springs

OUTPUT:
f: residual force vector

myIntEx(t_new, x_old, u_old, free_DOF, stiffness_matrix, index_matrix, m, dt):

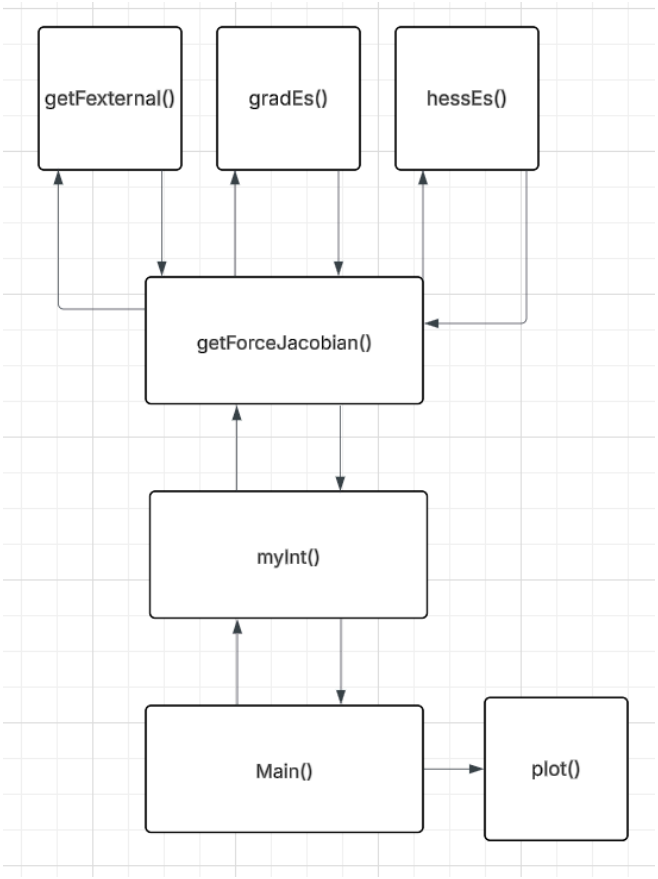
This function calculates the positions of the free Nodes by using Implicit Euler integration with Newton-Raphson. This function is a modified version of the myInt(). function. This function calls getForceEx() function to use total forces to find the delta of the positions. This function only changes it for the specific indexed positions of the free Nodes.

INPUT:
t_new: new time value
x_old: position at previous time
u_old: velocity at previous time
free_DOF: indices of degrees of freedom that can move
stiffness_matrix, index_matrix, m, dt: system parameters

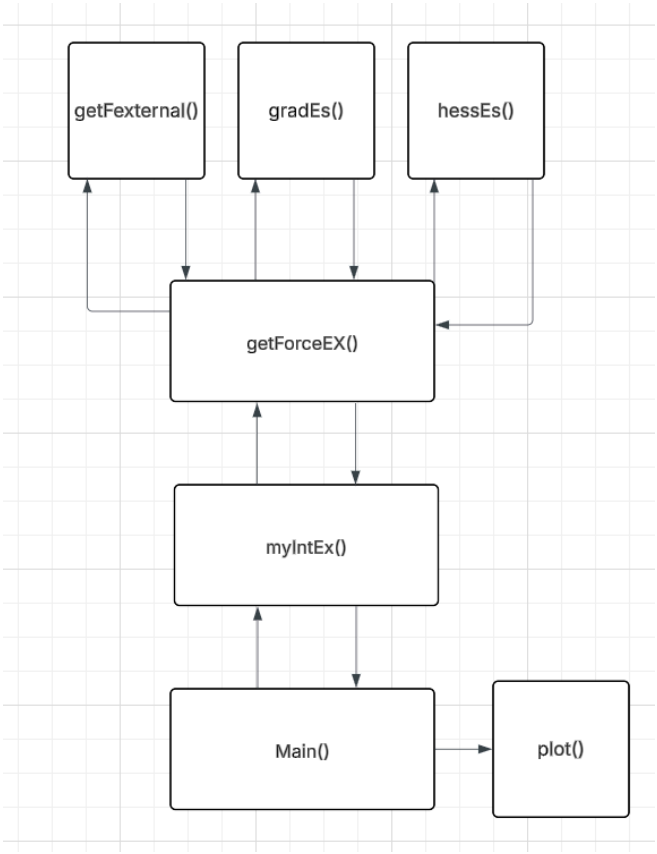
OUTPUT:
x_new: position at new time

The Main Function is mostly the same as the Implicit Method but using myIntEx() function instead of myInt() function to calculate the new positions.
}

B. Block Format:

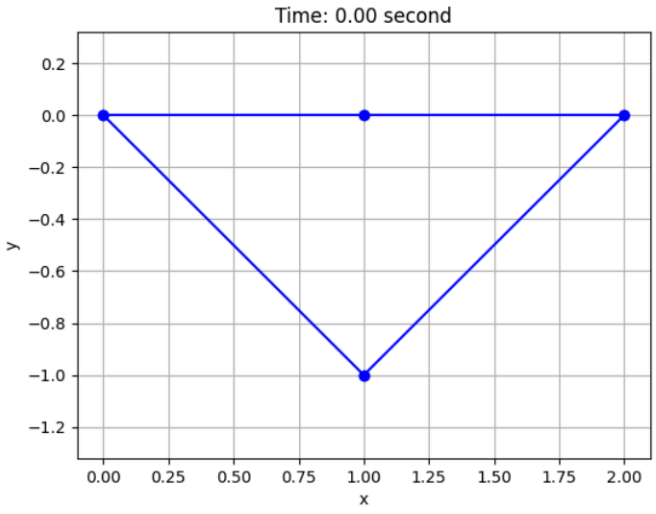


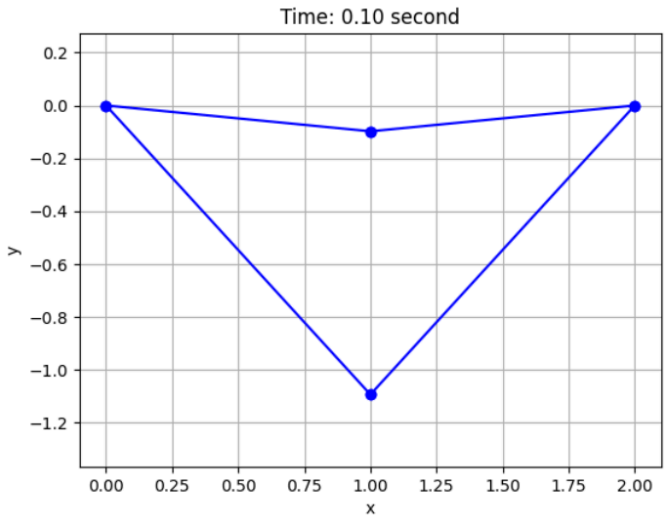
Implicit Method

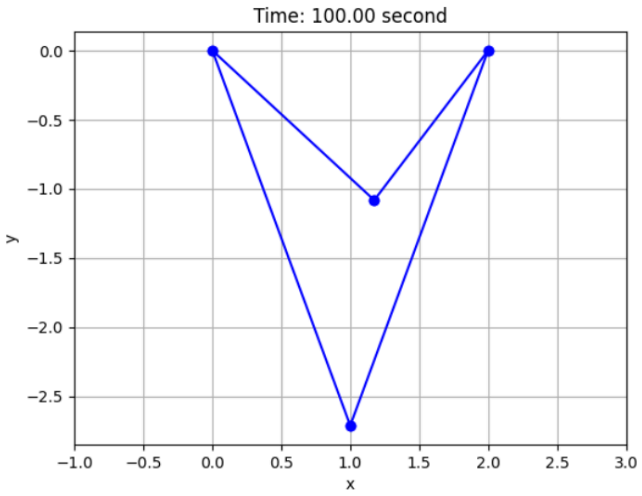


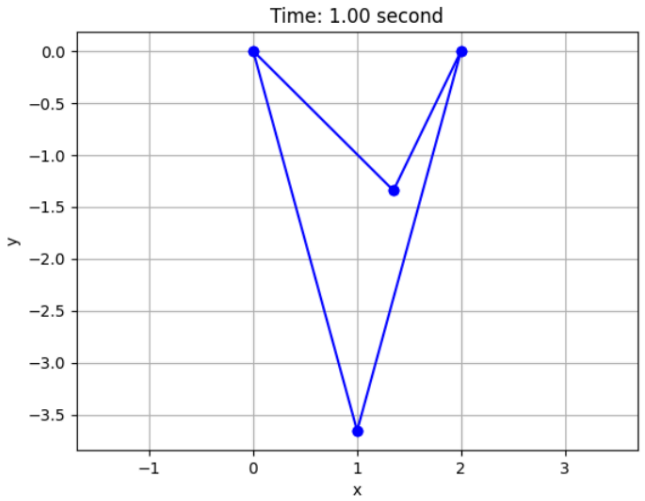
Explicit Method

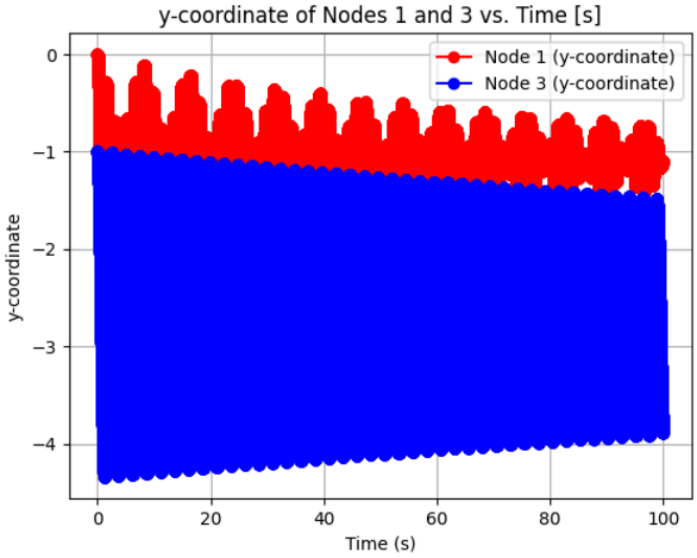
C. Graphs

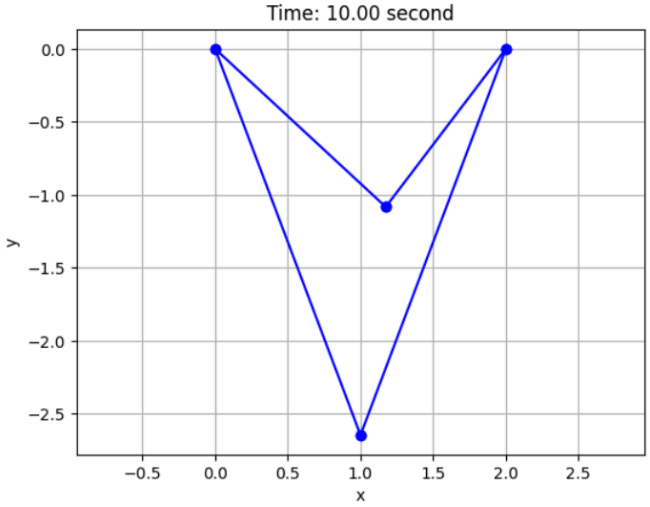










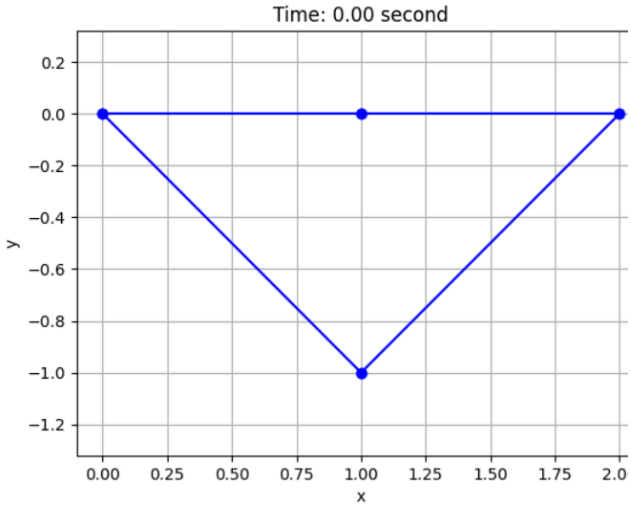


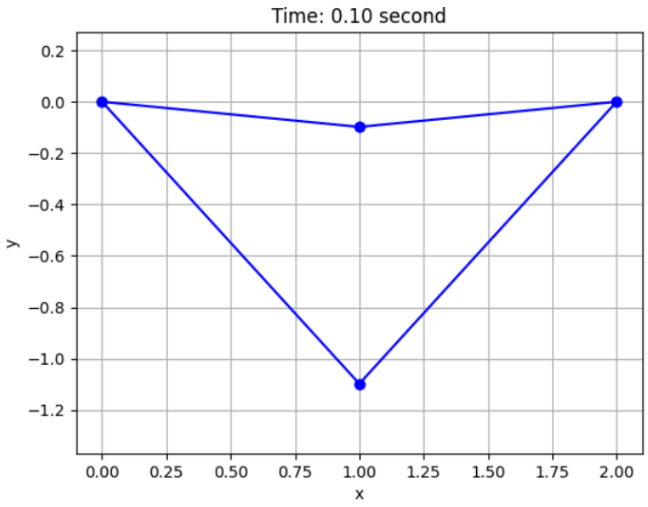
II. Question 2

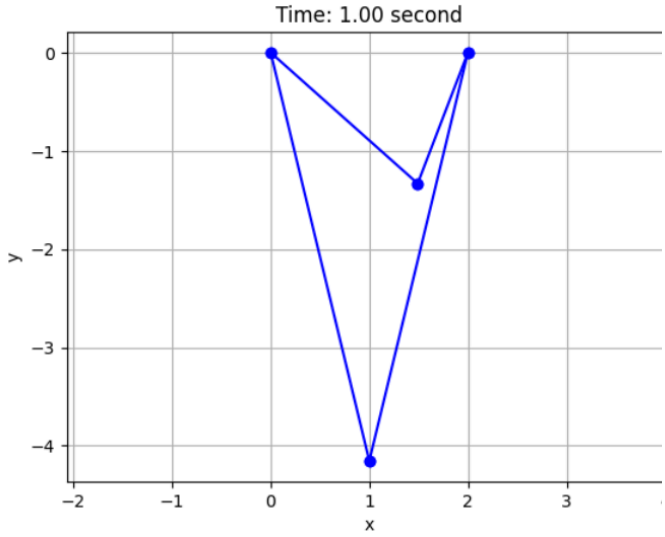
An appropriate time step size needs to be small enough to be more accurate and reduce the artificial damping introduced in the Implicit model functions in this undamped mass-spring network. It also cannot be too small or it takes longer to load. I chose dt=0.001s as it was quickest to load while also minimizing the damping.

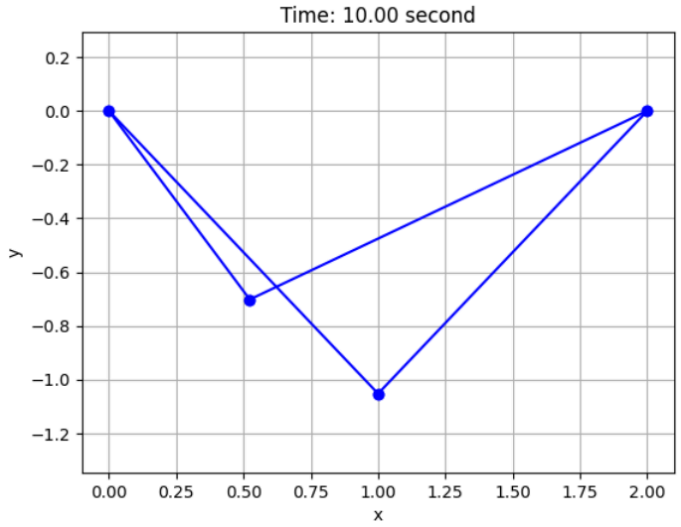
III. Question 3

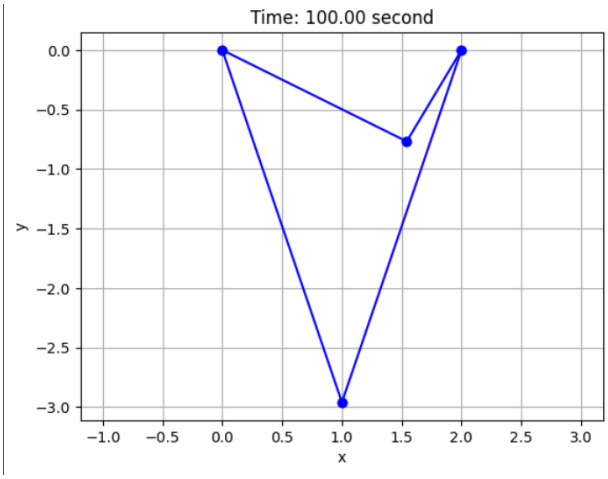
A. Graphs (at dt = 0.1)

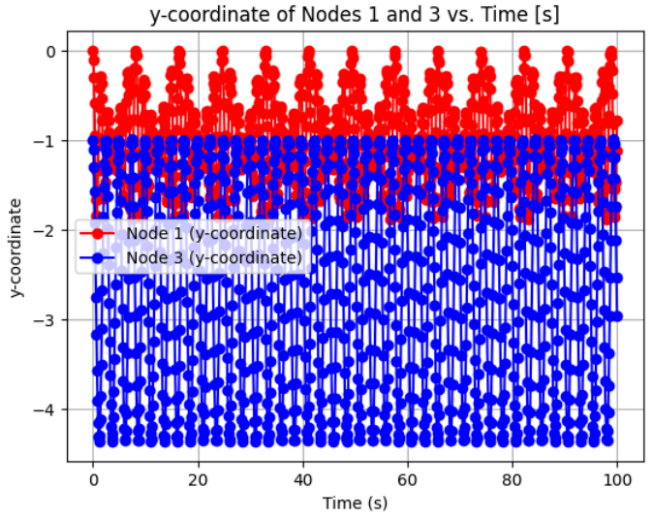












B. Comparison

The Explicit Euler Method is better to use for this spring network because it does not have artificial damping at larger time-steps, like the Implicit method, providing more accurate information. At larger steps, however, the explicit Euler Method does experience instability.

IV. Question 4

The Newark-β Simulations uses the integration of different parameters, β and γ, to adjust the integration for acceleration and damping assumptions to stabilize the function as well as reduce artificial damping from occurring in the simulation.

V. Question 5

Simulation not completed on time.

References

[1] Khalid, J. M. (2025). *Lecture 3: Spring Network Simulation* [Python Code].

[2] Khalid, J. M. (2025). *Module 6: Implicit vs Explicit Methods* [PowerPoint slides].