Spring-Mass Network Simulation

Jiajin Cui

***Abstract*— This report represents a python simulation of a 2D mass-spring network using both the implicit and explicit Euler integration method. The network consists of four masses (nodes) connected by two linear springs. The simulation computes spring forces, gradients and other metrices in a discrete manner utilizing customized time-steps.**

I. INTRODUCTION

The modeling and simulation of elastic networks are important for the analysis of soft robotic structures. Collections of point masses linked by springs are helpful representations for such systems. For Homework 1, a 2D spring-mass network is simulated using implicit Euler method to reproduce its motion of free nodes under the combined effects of the gravity and spring forces. The simulation starts with reading information including positions and spring properties, the initialization of different matrices, and the determination of time step and total time interval.

II. CONTENT

A. Pseudocode and Code Structure

```
write node.txt, spring.txt
initialize nodes, masses, springs

for each spring:
   compute rest length l_k

set time_step, total_time
initialize position, velocity
set DOF_free (the constrained nodes)

for t = 0 : time_step : total_time:
   compute spring strain ε_k and energies E_k
   external force vector F(x)
   mass matrix M
   myIntegrator(t_new, x_old, u_old…)
   get x_new, u_new
   update x_old, u_old
end
plot
```

Overall, this simulates a 2-D mass-spring network in discrete time using implicit Euler integration.

Main functions:

gradEs:

This function calculates the stretching energy of a spring given the coordinates of two adjacent nodes. It takes the following inputs:
(xk,yk): the coordinates of the current point
(xkp1, ykp1): the coordinates of the next point
l_k: reference length of the spring
EA: elastic modulus
Output: a 4x1 gradient array

hessEs:

This function calculates the hessian of stretching energy of a spring using the same inputs as gradEs, and returns a 4x4 matrix J.

myInt:

This function starts with a copy of x_old. While error is greater than a defined value eps, it solves the tiny movement deltaX for free nodes and return the new position and the new velocity.

B. Choosing an Appropriate Time Step

To ensure numerical stability, the time step must be small to capture the oscillation. For spring-mass pair, we have $T_n = 2\pi\sqrt{\frac{m}{k_{\max}}}$, which, in our case, is roughly 1s. Normally, we want $\Delta t \leq T_n/50$. Given implicit Euler is stable for linear systems, a time step of 0.1s can lead to

convergence. Moreover, considering the tradeoff between computation accuracy and computational cost, a time step of 0.1 should lead to relatively short computation and an accurate result

C. *Implicit v.s. Explicit Method*

As the Explicit method being used, the system becomes unstable even at relatively small time steps. The implicit method is preferred because the downside of the explicit method is that it requires smaller time step, leading to more computation time. Implicit method allows larger time step thus requires less computation time.

D. *Artificial Damping*

As implicit Euler evaluates forces at the future step, it adds numeric viscosity to the system which leads to numeric damping. In Newmark-beta family, parameters beta and gamma control the stability and numerical damping of the time integration. A beta of 0.25 and a gamma of 0.5 or slightly larger can eliminate the artificial damping by preventing the update matrix from shrinking.

III. Conclusion

This work implemented an implicit Euler for a mass-spring network with two fixed nodes and two free nodes. The mathematics formulas have been converted into python functions, and a practical choice of time step has been determined based on the system's natural frequency. A comparison has also been made between implicit Euler and explicit Euler. Finally, the Newmark-beta family has been discussed to introduce effective ways to eliminate the artificial damping in implicit Euler.

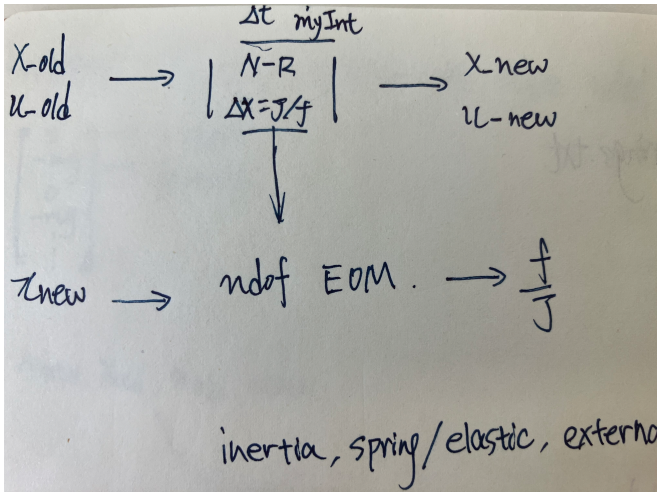
References

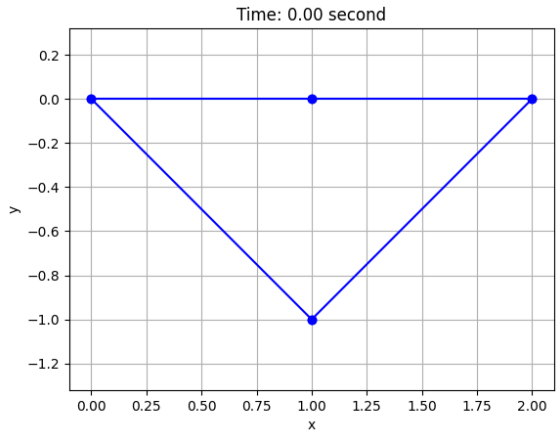
[1] Professor M. Khalid Jawed, "Spring Network Simulation (Colab notebook)," *Google Colab*, 2025. [Online]. Available: https://colab.research.google.com/drive/1CkzR6aSyDcSsIKdKCEQFDIJKMcavuzhI. Accessed: Oct. 15, 2025.

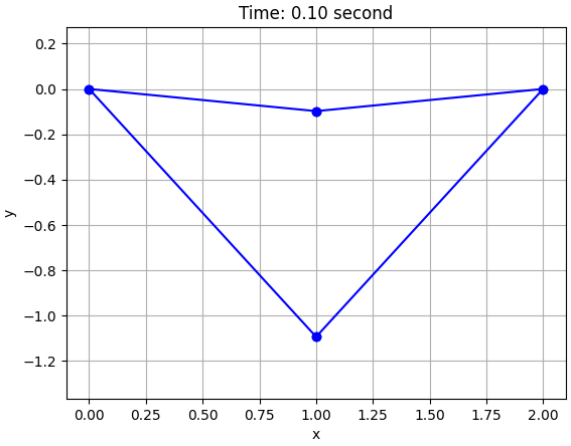
[2] MAE-263F Course Staff, "Homework 1 notes/instructions," *UCLA BruinLearn*, 2025. [Online]. Available: https://bruinlearn.ucla.edu/courses/216502/files/21999662?module_item_id=7665460. Accessed: Oct. 15, 2025.

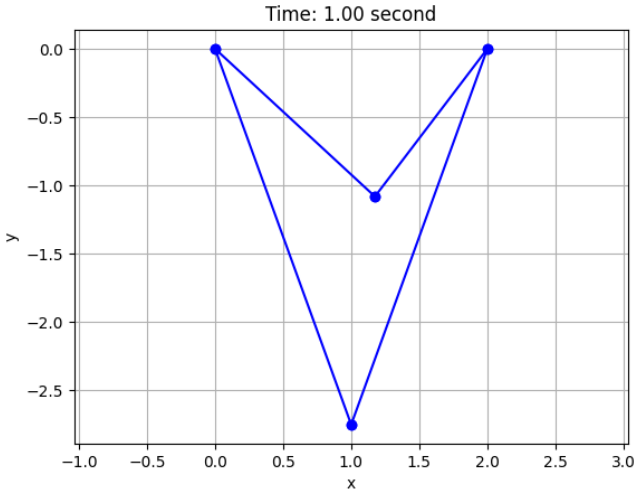
[3] Idaho National Laboratory, "NewmarkBeta — MOOSE Framework Documentation," *MOOSE Framework*, [Online]. Available: https://mooseframework.inl.gov/source/timeintegrators/NewmarkBeta.html. Accessed: Oct. 15, 202

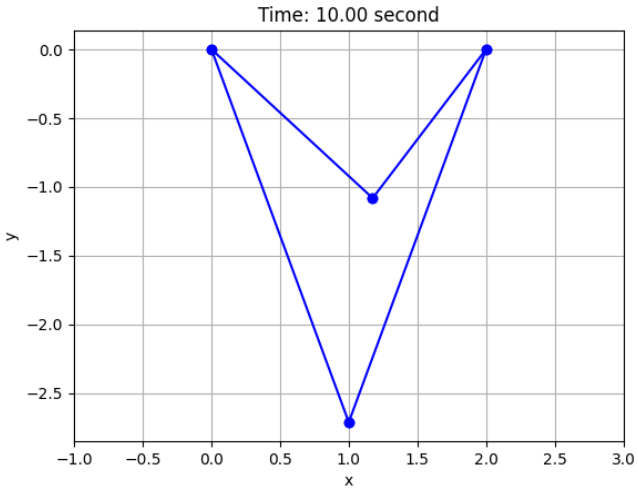
APPENDIX

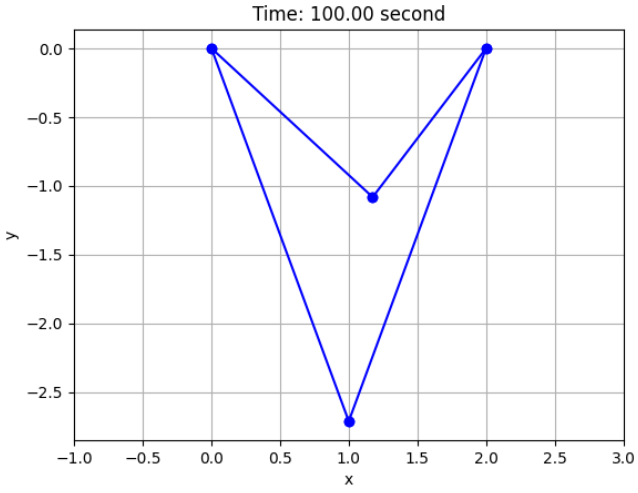

Δt myInt
X-old
u-old
N-R
ΔX=J/f
X-new
u-new
Xnew
ndof EOM.
f
J
inertia, spring/elastic, externa

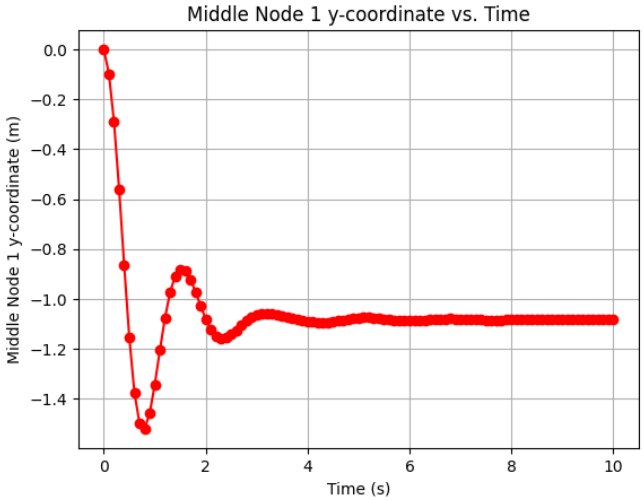

Time: 0.00 second
x
y

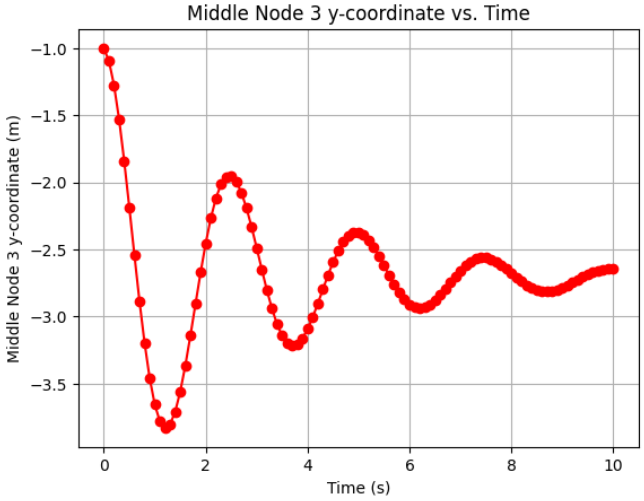

Time: 0.10 second
x
y


Time: 1.00 second
x
y


Time: 10.00 second
x
y


Time: 100.00 second
x
y


Middle Node 1 y-coordinate vs. Time
Middle Node 1 y-coordinate (m)
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
0
2
4
6
8
10
Time (s)


Middle Node 3 y-coordinate vs. Time
Middle Node 3 y-coordinate (m)
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
0
2
4
6
8
10
Time (s)