UNIVERSITY OF CALIFORNIA, LOS ANGELES

MAE 263F: Mechanics of Flexible Structures and Soft Robot

Professor: Khalid Jawed

Homework 1 Report

Lovleen Kaur

Oct 15, 2025

1. Implicit Euler Pseudocode and Code Structure

Code Structure:

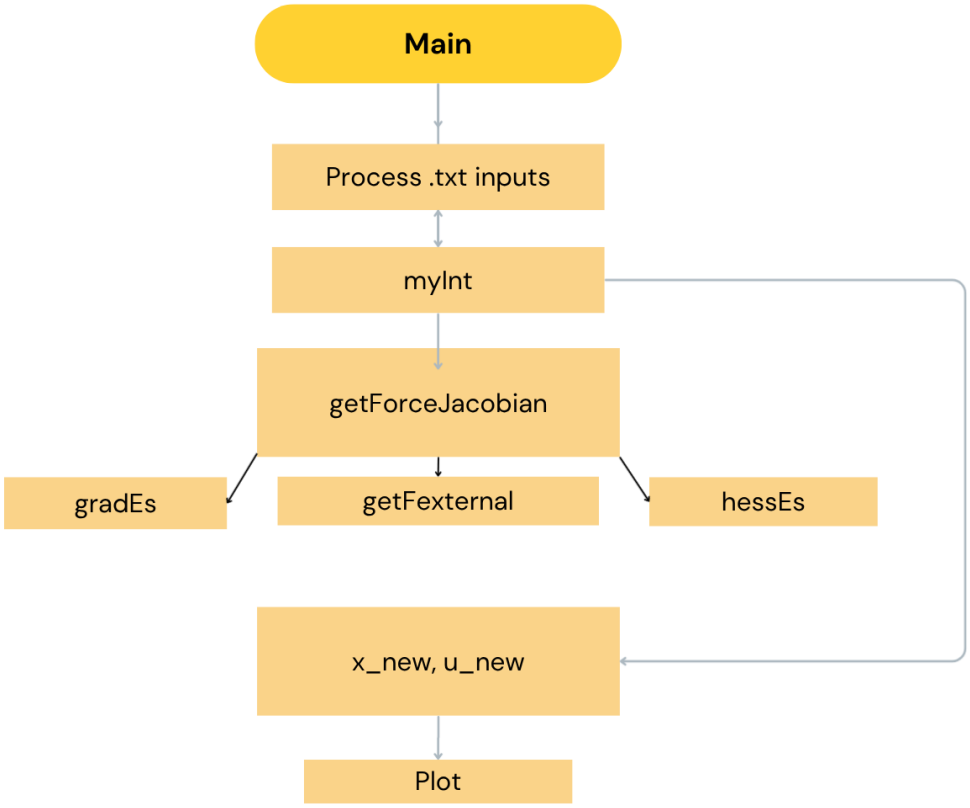


Figure 1: Code Structure

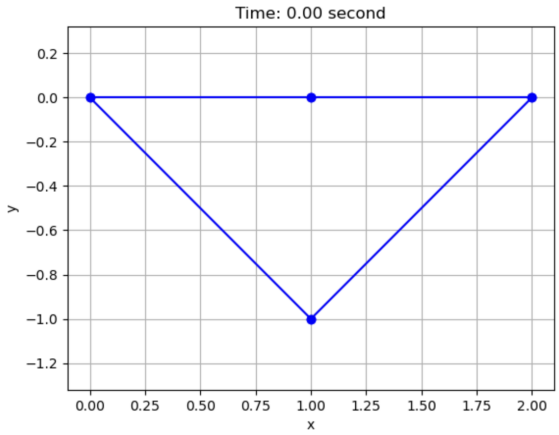
The main function starts with processing the nodes and the springs information. Then functions that will be called later are created such as gradES, getFexternal, hessEs, getForceJacobian, and myInt. The myInt function is the main solver that results in the new positions and velocities for the nodes. This function calls getForceJacobian which then calls gradES, getFexternal, and hessEs. The following are the descriptions for each of the functions:

Function	Description
`myint`	Starts with a guess and iterates using the Newton–Raphson method to converge to a solution for a given tolerance, resulting in new position and velocity of the nodes.
`gradES`	Calculates the gradient of the stretching energy with respect to the coordinates.
`hessES`	Calculates the 4x4 Hessian of the stretching energy with respect to the coordinates.
`getFexternal`	Calculates the external force, which is gravity in this case.
`getJacobian`	Returns the node forces and the jacobian matrix for the system. This function calls `getFexternal`, `hessES`, and `gradES` to compute everything.

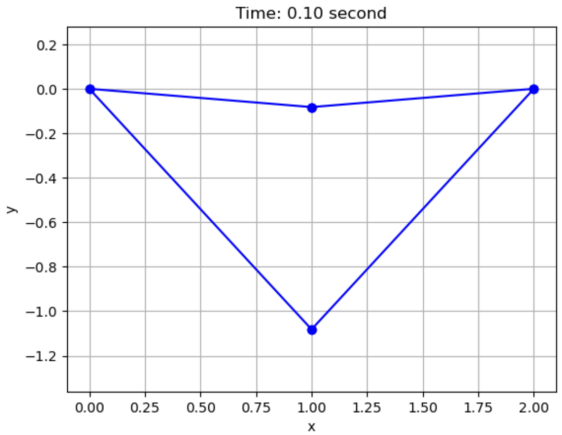
Table 1: Summary of core functions and their roles in the system.

Pseudocode:

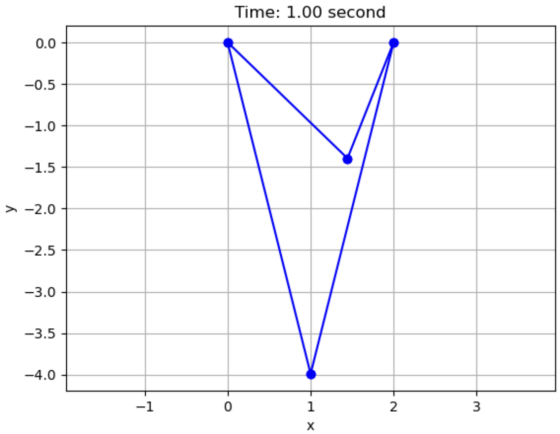
```
 read nodes.txt springs.txt

create inner operation functions:
    gradEs; hessEs; getFexternal

create forceJacobian function takes gradEs
    getForceJacobian(grad, f_ext, hess)

call main integrator function
      myint(getForceJacobian,..)
          while error > tol
              f, J = getForceJacobian(x_new, x_old, u_old,
                     stiffness_matrix, index_matrix, m, dt, l_k)

              deltaX_free = np.linalg.solve(J_free, f_free)
              x_new = x_new - deltaX
              err = np.linalg.norm(f_free)
          end
          u_new = (x_new - x_old) / dt
      return x_new, u_new
      end
plot(x_new, u_new)
```



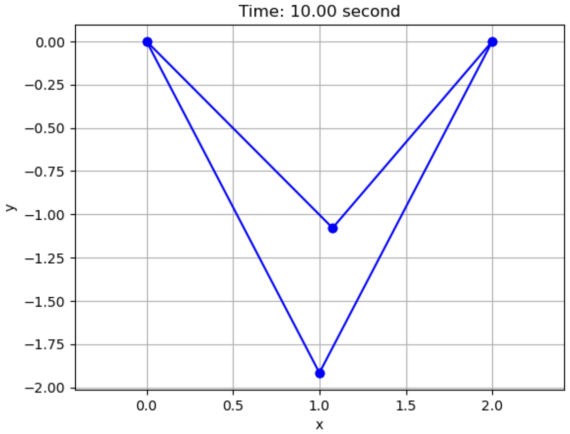
(a) Image 1



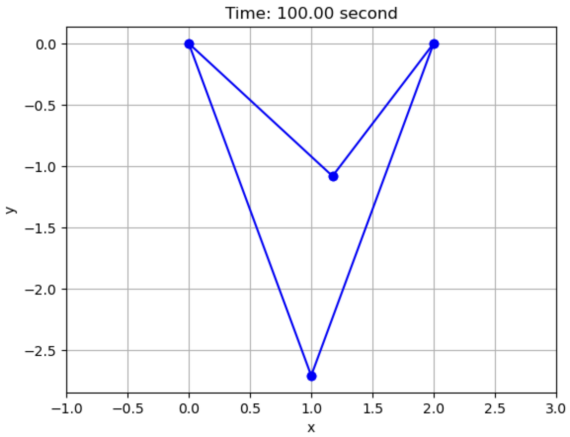
(b) Image 2



(c) Image 3

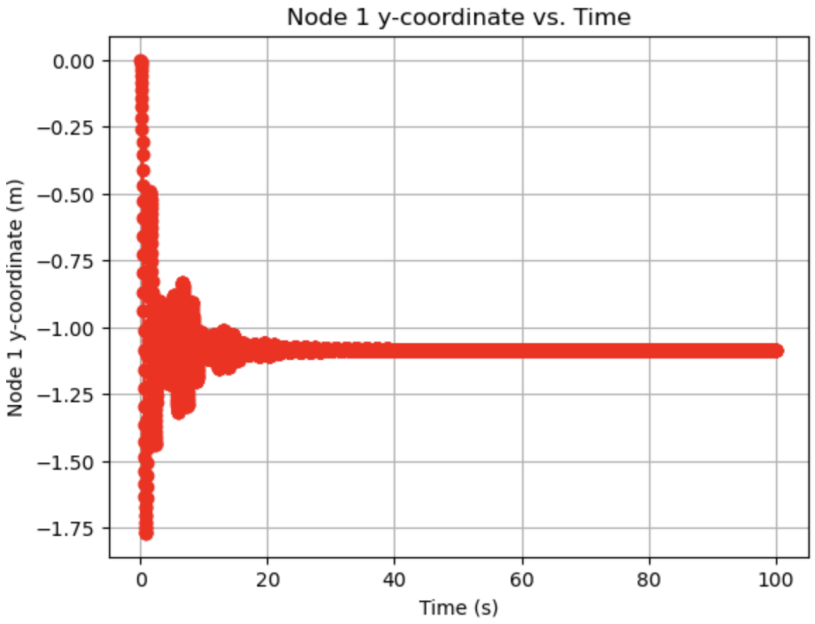


(d) Image 4

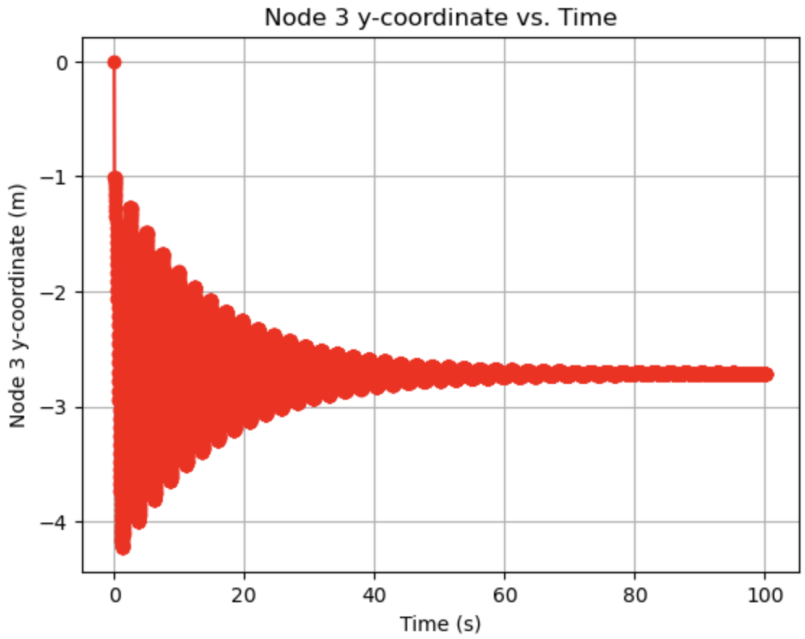


(e) Image 5

Figure 2: Implicit Solution from t = 0 to 100s



(a) Node 1



(b) Node 3

Figure 3: Nodes 1 and 3 Positions Over Time

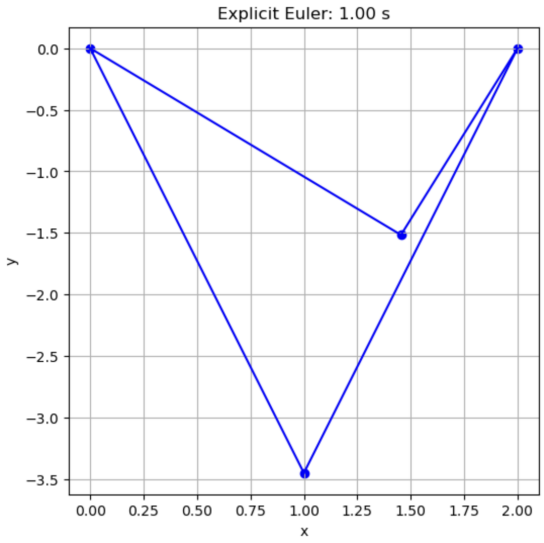
These results show that over time, the free nodes converge and this scheme is stable. The time step use here is 0.02, which is well under the natural time scale of the system $\sqrt{\frac{m}{k_{max}}} = \sqrt{\frac{1}{20}} = 0.22$

2. Selecting an appropriate time step

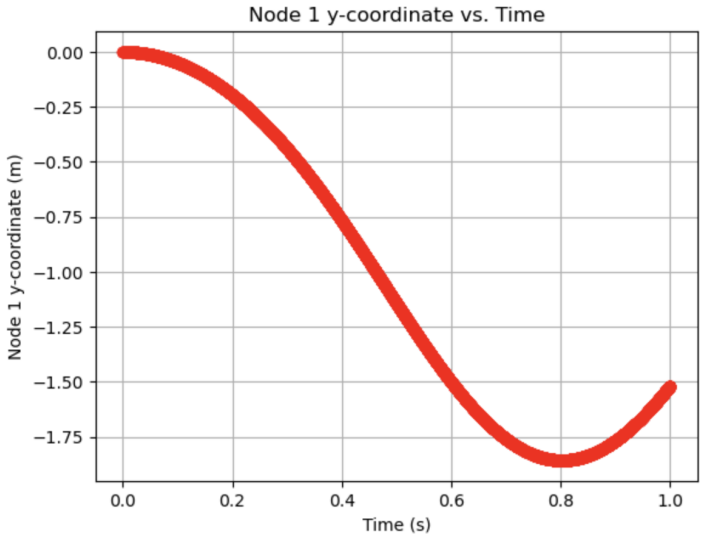
Selecting a time step size depends on the method, the system being analyzed, computational cost, and the speed needed. For schemes that are stable, we can select larger time steps and scheme that are less stable or unstable, the time step must be small. Initially for a simulation, a trial and error approach can also be taken within the stability range to start the simulation and then update the time step depending on convergence needs.

In this case, the time step for the implicit scheme does not have a stability concern but rather its a matter of damping in the solution. With smaller steps there is more oscillation, with larger steps there is more damping. The system converges towards the same place eventually.

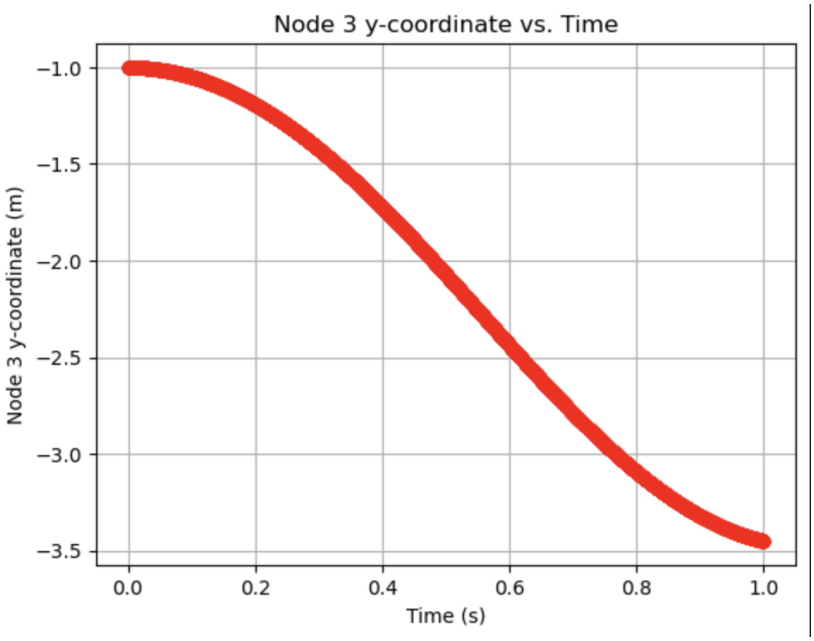
3. Explicit Euler Simulation Analysis



(a) Image 1



(b) Image 2



(c) Image 3

Figure 4: Explicit Euler Results with dt = 1e-6

These results are unstable even with very small dt of 1e-6. Beyond this 1s simulation time, the instabilities grow much more and do not oscillate down to one configuration like the implicit scheme. This scheme is unstable and should not be used for this system. Implicit scheme is stable, with proper parameters for the damping that scheme can produce desired results.

2. Newmark-β family of time integrators

The Newmark-β family is a set of implicit time integration methods for second-order systems. By adjusting β parameter we can artificially control numerical damping. The average acceleration method $\beta = 1/4$ is unconditionally stable and has no numerical damping, preserving energy even for high-frequency modes.

Increasing values introduces controlled high-frequency damping. Overall, the Newmark-β methods mitigate artificial damping by letting us tune or eliminate unwanted numerical dissipation, unlike the implicit scheme covered which applied heavy damping for large time steps.