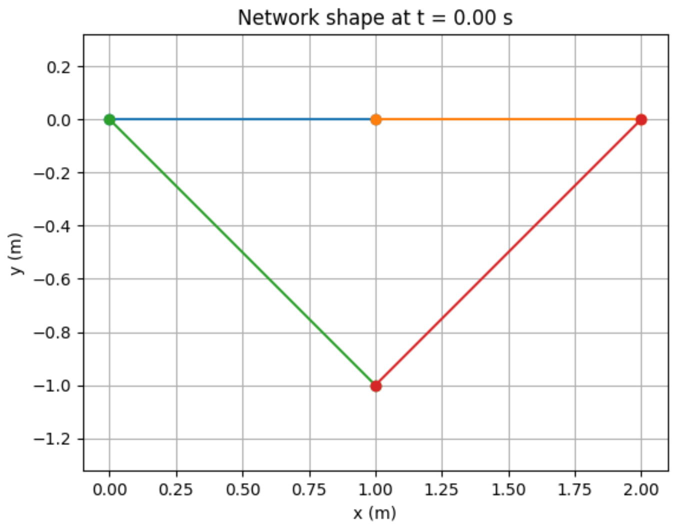
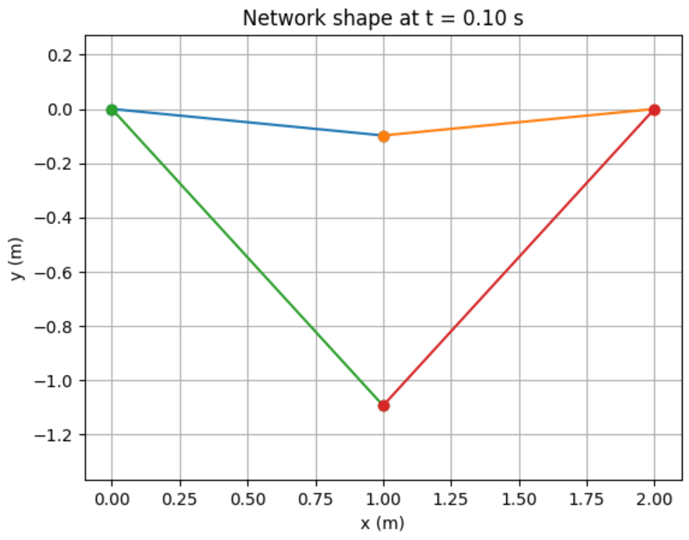
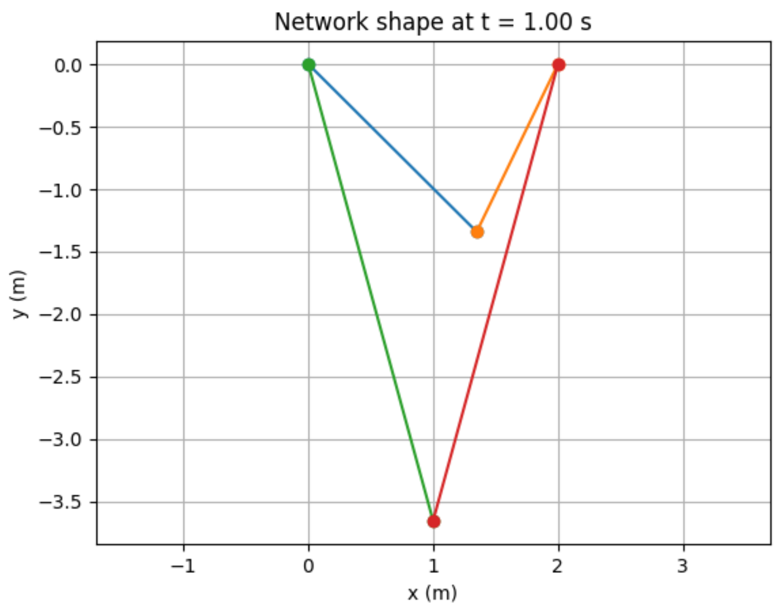
Homework 1 MAE-263F

Marwan Fayed

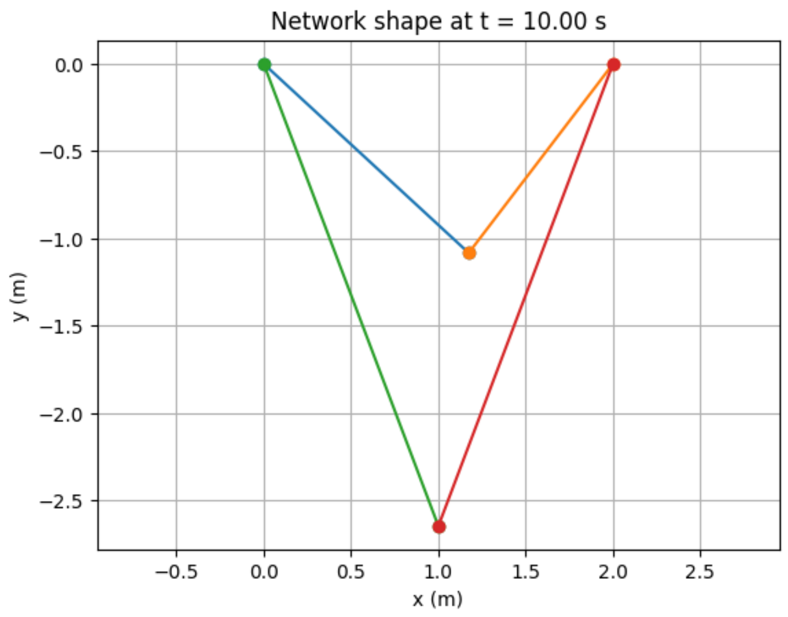
1. **Pseudocode and code structure: Write pseudocode for your simulator describing its main logic and workflow. Briefly describe the main functions and scripts in your implementation (2–3 sentences each), including their inputs and outputs. Create a simple block diagram showing how the functions or scripts interact (i.e., which one calls which). There is no strict format requirement—use your best judgment to make it understandable to someone with an undergraduate-level engineering background.**



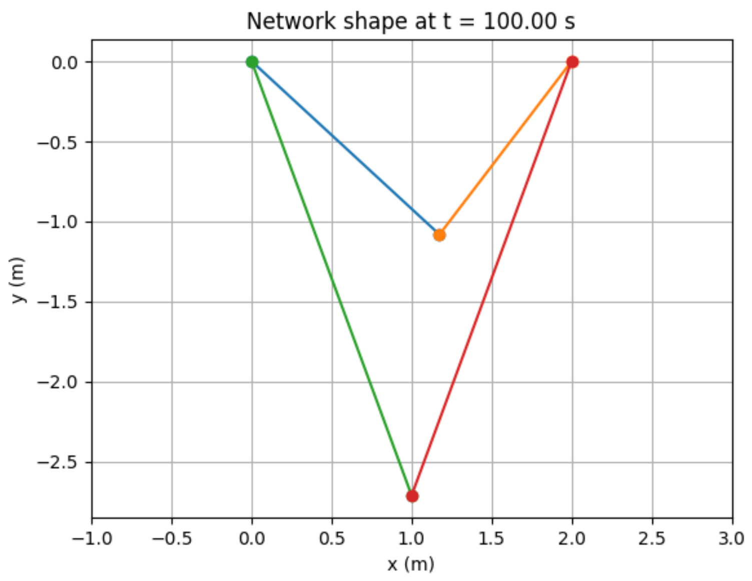


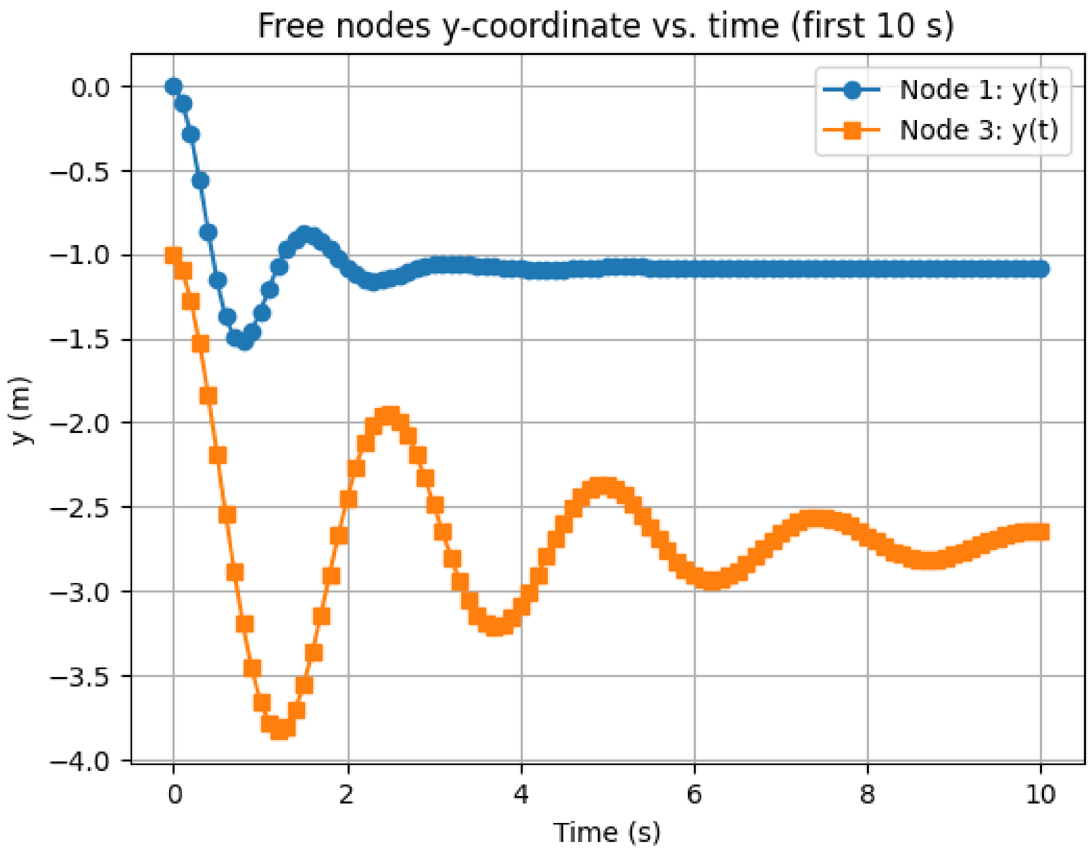


Network shape at t = 1.00 s
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
y (m)
−1
0
1
2
3
x (m)



Network shape at t = 10.00 s
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
y (m)
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x (m)





Pseudocode:

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

Computes the gradient of the spring stretching energy with respect to the two node coordinates.

Inputs: two node coordinates, rest length l_k, stiffness k.

Output: a 4-element vector [∂E/∂xk, ∂E/∂yk, ∂E/∂xkp1, ∂E/∂ykp1].

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

Computes the 4×4 Hessian (second derivatives) of the spring energy.

Inputs: same as gradEs.

Output: symmetric 4×4 stiffness block for Newton iterations.

Input Readers/Writers (nodes.txt, springs.txt)

Handles reading/writing of network data.

Inputs: text files with node coordinates and spring connections.

Outputs: node_matrix, pairs, stiffness_matrix, and index_matrix.

getFexternal(m)

Applies external gravitational load.

Input: mass vector m (length = number of DOFs).

Output: external force vector with −mg on y-DOFs.

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

Assembles the nonlinear residual f and Jacobian J for implicit Euler.

Inputs: current and previous state vectors, spring data, and time step.

Outputs: global force vector f and Jacobian J.

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

Performs a single implicit Euler time step with Newton–Raphson iteration.

Inputs: previous state, mass, stiffness, time step, and free DOFs.

Outputs: updated displacement x_new and velocity u_new.

plot_network(x, index_matrix, tsec)

Visualizes the current network configuration at time tsec.

Inputs: displacement vector and spring connectivity.

Output: plot of the current spring network (for debugging or visualization).

Main Script

Coordinates input loading, initialization, time stepping, and plotting.

Inputs: node and spring data files.

Outputs: time histories of y-displacement at selected nodes and plots.

Block Diagram:

Main Script

↓

myInt (Implicit Integrator)

↓

getForceJacobian (Compute Residual and Jacobian)

↓

gradEs and hessEs (Per-Spring Energy Gradient and Hessian)

↓

getFexternal (Compute External Forces)

↓

Return Updated Forces and Jacobian to myInt

↓

Return Updated Displacements and Velocities to Main Script

↓

plot_network (Visualize Structure)

↓

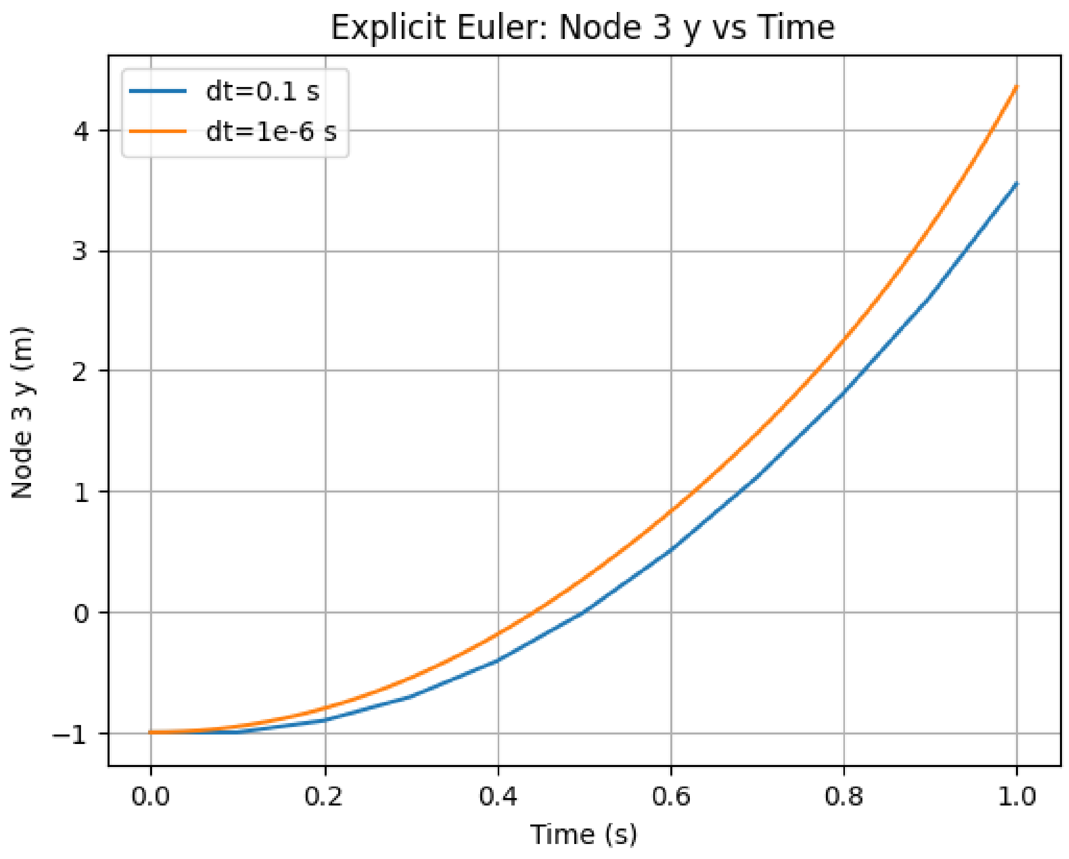
Plot Results (Y-coordinates of Nodes)

2. How do you choose an appropriate time step size Δt?

The time step size $\Delta t$is chosen to balance stability, accuracy, and computational efficiency. For explicit methods, Δtmust be smaller than the critical value $\Delta t_{\text{crit}} = 2/\omega_{\text{max}}$, where $\omega_{\text{max}} = \sqrt{k_{\text{max}}/m_{\text{min}}}$is the highest natural frequency. In practice, $\Delta t$is usually set to about 5–10% of this limit to ensure stability. For implicit schemes, while stability allows larger steps, Δt should still be small enough to capture system dynamics accurately—typically such that nodal displacements change by only a few percent of the smallest spring length per step. A good approach is to start with a small $\Delta t$and increase it gradually until results begin to deviate noticeably.

3. **Simulate the spring network using Explicit Euler for t ∈ [0, 1] s. If it becomes unstable, reduce Δt and try again. If it still fails even at Δt = 10−6 s, state this in the report. Explain which method (implicit vs. explicit) is preferable for this spring network and why.**

Both 0.1s and 10E-6 unstable!



Upon simulating the network with Explicit Euler over $t \in [0,1]$s, successively reducing Δt. The solution became unstable for $\Delta t = 0.1$and remained unstable even after repeated halving down to $\Delta t = 10^{-6}$s. Therefore, Explicit Euler fails to produce a stable solution on this interval even at $\Delta t = 10^{-6}$s. An implicit method is preferable. The network is stiff, so Explicit Euler—being conditionally stable—requires $\Delta t \ll 2/\omega_{\max}$, which is impractically small here and leads to energy blow-up. Implicit schemes are A-stable for linear systems and remain robust for these nonlinear springs via Newton iterations; they allow much larger time steps, reduce spurious high-frequency growth, and reach equilibrium efficiently despite a higher per-step cost. Adding mild damping further improves implicit robustness.

4. The simulation with implicit Euler appears to reach a static state (numerical damping). Read about the Newmark–β family of time integrators and explain how such integrators can mitigate such artificial damping.

When using implicit Euler, the simulation quickly settles into a static state because this method introduces numerical damping, which heavily damps out both high- and low-frequency oscillations. This can make the system appear to "freeze" instead of oscillating naturally.

The Newmark–β family of time integrators provides a more flexible framework that can control this damping behavior. By adjusting its parameters β and γ, one can balance stability, accuracy, and numerical damping. Thus, Newmark–β integrators mitigate artificial damping by allowing selective damping of only nonphysical high-frequency modes while retaining realistic dynamic behavior in the lower-frequency range.