Nathan Ge
905597306

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.

Inputs: nodes.txt → initial (x, y) coordinates for each node springs.txt → connectivity (i, j) and stiffness k for each spring dt → time step size T → total simulation time	Output: Time evolution of node positions (x, y) Plots of spring network at t = {0, 0.1, 1, 10, 100} s Plots of y-coords of free nodes 1 and 3 versus time

Process

1. Read node and spring data
   - Read (x, y) coordinates into node_matrix
   - Read spring connections and stiffnesses into index_matrix and stiffness_matrix
   - Compute rest lengths l_k for all springs at t = 0

2. Preprocessing
   - Assemble x_old (initial positions) from node_matrix
   - Initialize velocity u_old = 0 and mass vector m = 1 kg for all DOFs
   - Define free degrees of freedom (DOFs): nodes 1 and 3
   - Define gravity vector W = **getFexternal**(m)

3. **Main Loop**: Time Integrator Simulation Loop (implicit Euler)
   For each time step t → t + Δt:
   a. Call **myInt**() to compute x_new and u_new
      i. Use Newton–Raphson iterations:
         - Evaluate total force f(x_new) and Jacobian J(x_new) using **getForceJacobian**
            - getForceJacobian uses **gradEs** for force and **hessEs** for spring calculations, and **getFexternal** for external force
         - Solve reduced system for free DOFs: J_free Δx = f_free
         - Update x_new ← x_new – Δx
         - Iterate until ||f_free|| < tolerance
   b. Update velocity u_new = (x_new – x_old) / Δt

4. Results
   - Plot shape of spring network at specified time steps
   - Plot y(t) for nodes 1 and 3

Nathan Ge
905597306

Helper Functions

myInt

Uses Newton-Raphson to solve the implicit Euler and find the predicted position and velocity at the next time step within a certain tolerance. Uses getForceJacobian to compute the total summed Jacobian and force at a given time step to compute deltaX for convergence on each iteration of Newton-Raphson.
Input: new time step, hold position and velocity, new time, DOFs that are free to move, nodes and springs matrix
Output: new position and velocity

getFexternal

Computes external forces in the x and y directions for each node. In this problem, there's only weight in the y-direction.
Input: mass (m) is a vector of size ndof (= 2 times the number of nodes)
Output: weight (W) is a vector of the same size

gradEs

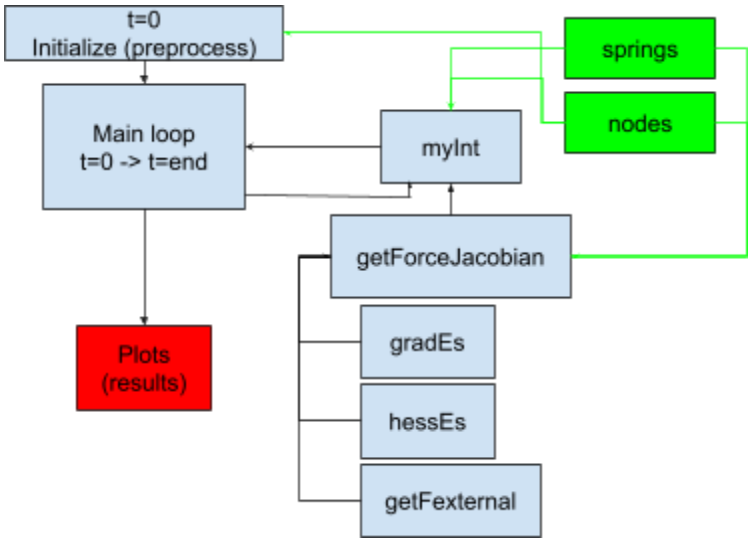
Calculate the gradient of the stretching energy with respect to the coordinates. Computed using elastic energy equations.
Input: Coordinates of nodes i and j, rest length, stiffness
Output: gradient vector

hessEs

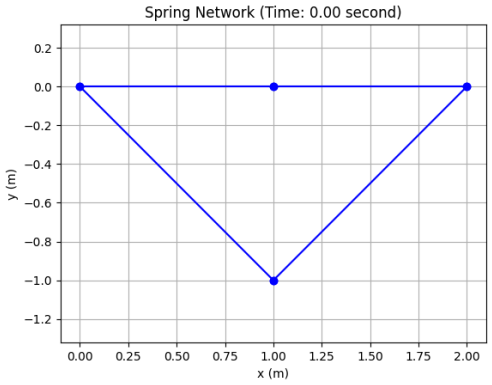
Computes Jacobian, or Hessian (second derivative) of spring energy. This is used in Newton-Raphson.
Input: Coordinates of nodes i and j, rest length, stiffness
Output: 4x4 Hessian of the stretching energy E_k^s with respect to

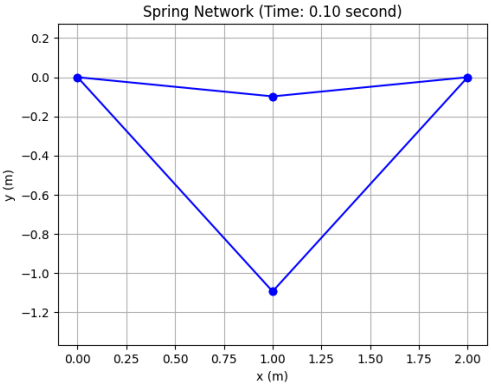
getForceJacobian

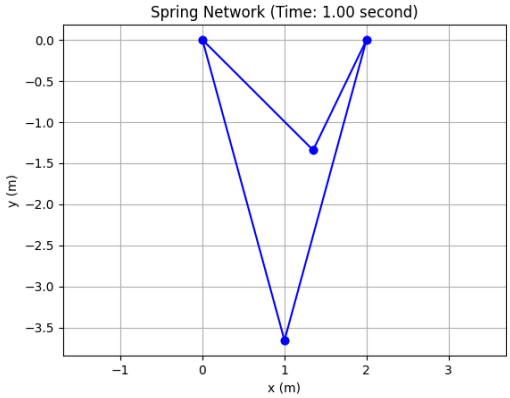
Assembles total internal/external force and Jacobian using gradEs (spring force), gradHess (spring energy Jacobian), and getForceExternal (external force calculations, such as weight) for Newton–Raphson integrator at a given time step.
Input: stiffness, rest lengths, time step, hold position and velocity, and new position (x_new, which is iterated upon with Newton-Raphson).
Output: total force and Jacobian

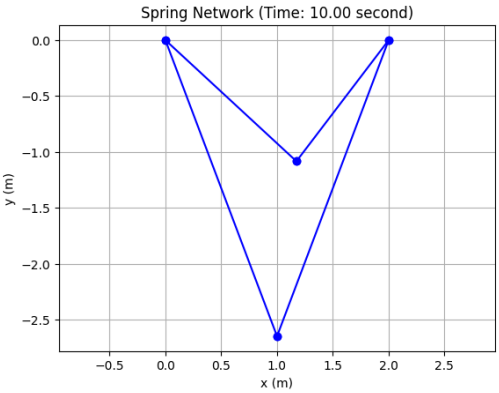


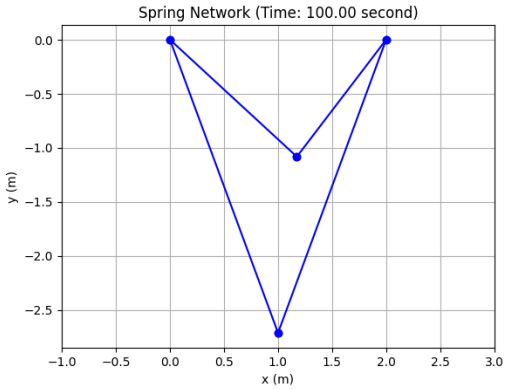
Nathan Ge
905597306

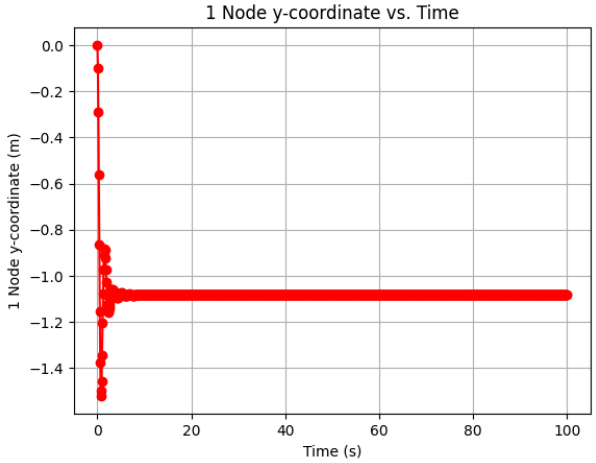


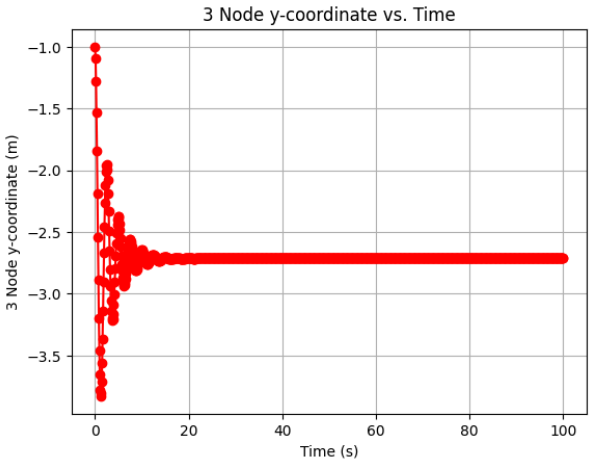












Nathan Ge
905597306

2. How do you choose an appropriate time step size Δt?
The goal is to find the largest stable Δt that still captures the physical oscillations accurately.

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

My implementation of explicit Euler encountered numerical errors even at $\Delta t = 10^{-6}$ s.

The explicit method only works with extremely small time steps, since the linear approximation of the function that forms the basis of the explicit method functions more accurately when zoomed in closer to the function. But at the point you choose a time step granular enough to make function approximations to advance the explicit method, it becomes too computationally inefficient and is sometimes still not good enough.

The implicit method is preferable in this case due to the nonlinearity of the equations of motion. Although computationally more intensive in design, the iterative steps that numerically converge onto a reasonable estimate eliminate a lot of the potential problems of instability and nonconvergence that the explicit method experiences.

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.

The Newmark–β family of numerical integrators uses parameters to tune the numerical damping to an oscillatory behavior that persists with realistic amplitude. The Newmark-β numerical integration family balances physical realism and numerical stability. Some variations use parameters like β (controls position integration accuracy) and γ (controls velocity integration accuracy), for instance, or trapezoidal parameters to conserve energy in linear elastic systems.

Nathan Ge
905597306

5. (Ungraded) Newmark–β Simulation: Simulate the spring networks in (a) and (b) using the Newmark–β method and demonstrate that the artificial damping observed in the implicit Euler method is eliminated.

With beta = ¼ and gamma = ½ we were able to reduce the damping observed in the implicit Euler method. Tweaking the parameters could yield better results.