MAE 263F HW1

Spring Network

Nathan Ge[1]

I. PROBLEM STATEMENT

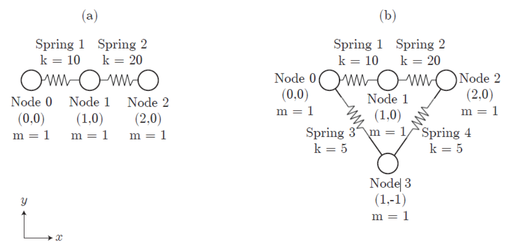
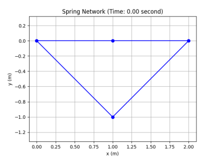


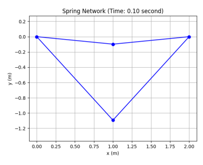
Fig. 1: Network of Springs

Task. Write a simulator using implicit Euler to simulate network (b) from t = 0 to t = 100 s.

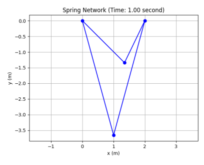
- Plot the shape of the spring network at t = 0, 0.1, 1, 10, 100 s. Include axis labels with units, legends if needed, and a title indicating the time.
- As a function of time, plot the y-coordinates of all free nodes (i.e., nodes 1 and 3). Include labels with units.



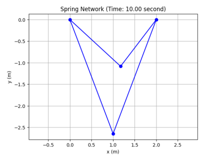
(a) $t = 0$ s



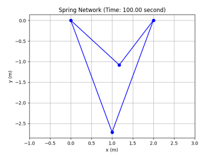
(b) $t = 0.1$



(c) $t = 1$ s



(d) $t = 10$ s



(e) $t = 100$ s

Fig. 2: Time evolution of the spring network at selected time steps.

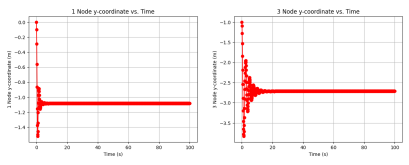


Fig. 3: Time evolution of y position of nodes 1 and 3

*This work was not supported by any organization

[1]Nathan Ge is a student in the Department of Mechanical and Aerospace Engineering, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90095 `nzge@g.ucla.edu`

II. PSEUDOCODE AND CODE STRUCTURE

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

Outputs:

- Time evolution of node positions (x, y)
- Plots of spring network at $t = \{0, 0.1, 1, 10, 100\}$ s
- Plots of y-coordinates of free nodes 1 and 3 versus time

Process

- 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 $t = 0$
- Preprocessing
  - Assemble x_{old} (initial positions) from `node_matrix`
  - Initialize velocity $u_{\text{old}} = 0$ and mass vector $m = 1$ kg for all DOFs
  - Define free degrees of freedom (DOFs): nodes 1 and 3
  - Define gravity vector $W = \text{getFexternal}(m)$
- Main Loop: Time Integrator Simulation (implicit Euler)
  - For each time step $t \rightarrow t + \Delta t$:
    1) Call `myInt()` to compute x_{new} and u_{new}
    2) Newton–Raphson iterations:
       * Evaluate total force $f(x_{\text{new}})$ and Jacobian $J(x_{\text{new}})$ using `getForceJacobian`
       * Solve reduced system for free DOFs: $J_{\text{free}} \Delta x = f_{\text{free}}$
       * Update $x_{\text{new}} \leftarrow x_{\text{new}} - \Delta x$
       * Iterate until $\|f_{\text{free}}\| <$ tolerance
    3) Update velocity $u_{\text{new}} = (x_{\text{new}} - x_{\text{old}})/\Delta t$
- Results
  - Plot shape of spring network at specified time steps
  - Plot $y(t)$ for nodes 1 and 3

Helper Functions

Function	Description (Inputs / Outputs)
`myInt`	Uses Newton–Raphson to solve implicit Euler for predicted position and velocity. **Input:** new time step, old position/velocity, free DOFs, node and spring matrices. **Output:** new position and velocity.
`getFexternal`	Computes external forces (gravity). **Input:** mass vector m **Output:** weight vector W
`gradEs`	Computes gradient of spring stretching energy. **Input:** coordinates of nodes i, j, rest length, stiffness **Output:** gradient vector
`hessEs`	Computes Hessian (Jacobian) of spring energy. **Input:** coordinates of nodes i, j, rest length, stiffness **Output:** 4×4 Hessian
`getForceJacobian`	Assembles total force and Jacobian for Newton–Raphson iterations. **Input:** stiffness, rest lengths, time step, positions, velocities **Output:** total force and Jacobian

TABLE I: Summary of helper functions used in the implicit Euler simulator.

Block Diagram

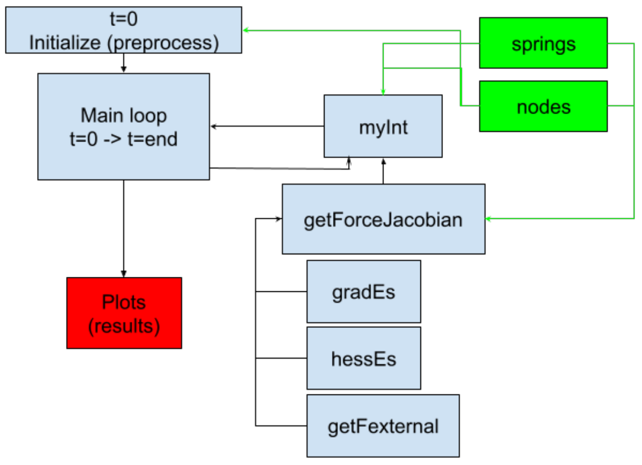


Fig. 4: Network of Springs

III. Questions

Time Step Selection

The goal is to find the largest stable Δt that still accurately captures physical oscillations while minimizing numerical damping.

Explicit Euler Simulation

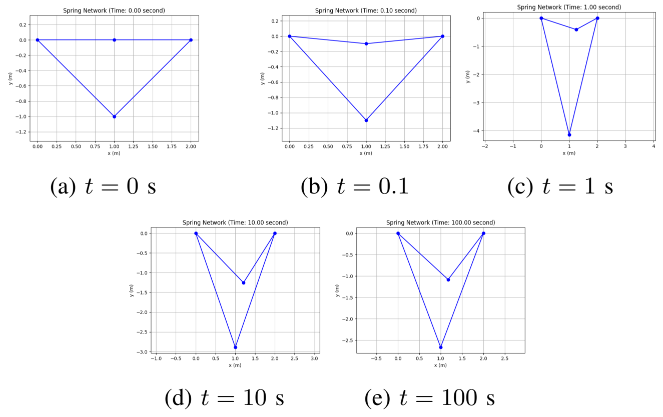
Simulation using explicit Euler for $t \in [0, 1]$ s encountered numerical errors even at $\Delta t = 10^{-6}$ s. The explicit method only works with extremely small time steps, since the linear approximation of the function that forms the basis of the explicit method functions more accurately when zoomed 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.

Artificial Damping and Newmark–β Integrators 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.

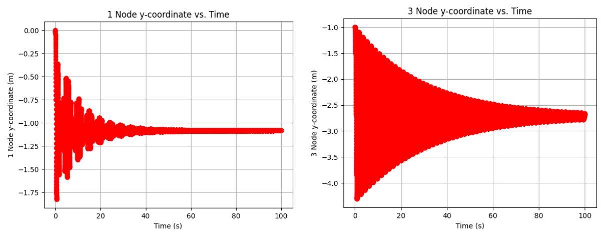
Newmark–β Simulation (Ungraded)

Using $\beta = 1/4$ and $\gamma = 1/2$, the artificial damping observed in implicit Euler is reduced. Tweaking these parameters may further improve results.



(a) $t = 0$ s (b) $t = 0.1$ (c) $t = 1$ s (d) $t = 10$ s (e) $t = 100$ s

Fig. 5: Time evolution of the spring network at selected time steps. ($\beta = 1/4$ and $\gamma = 1/2$)



(a) y-pos vs. time of node 1 (b) y-pos vs. time of node 3

Fig. 6: Time evolution of y position of nodes 1 and 3 ($\beta = 1/4$ and $\gamma = 1/2$)

Acknowledgment

The author thanks Professor Khalid Jawed for guidance and instruction throughout the MAE 263F Soft Robotics course.