MAE 263F Homework 1

Feng Xu

I. Task

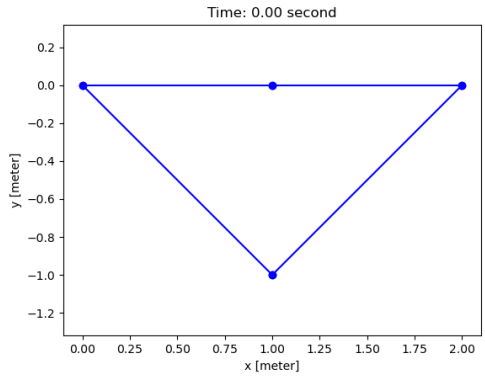

Fig. 1. Shape of the spring network at t=0.

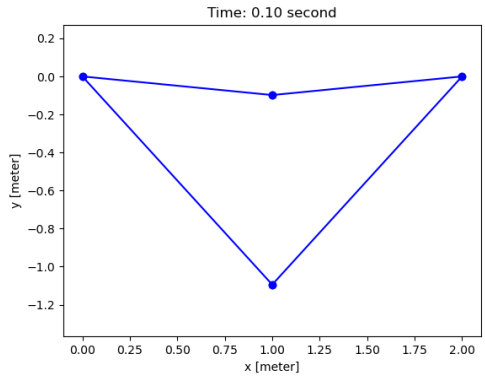

Fig. 2. Shape of the spring network at t=0.1.

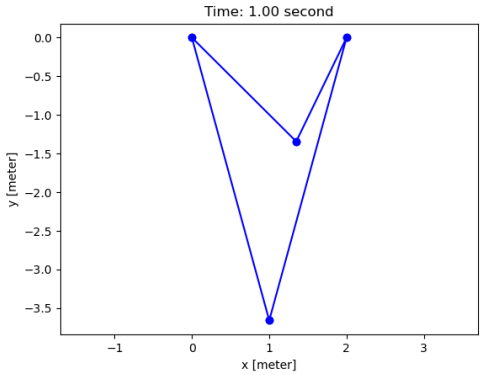

Fig. 3. Shape of the spring network at t=1.

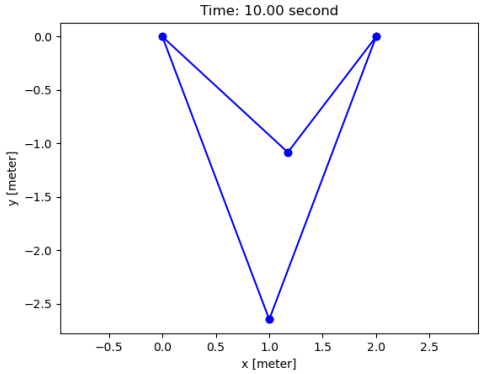

Fig. 4. Shape of the spring network at t=10.

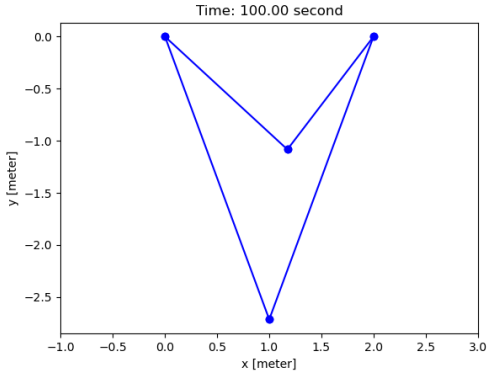

Fig. 5. Shape of the spring network at t=100.

II. Pseudocode and code structure

A. *Functions description*

1. loadnodes: load node (x,y) position from txt file. Input: Nodes.txt file path. Output: Node xy position matrix(Nnode*2)

2. loadsprings: load spring infomation(connection and k) from txt file. Input: Springs.txt file path. Output: Node xy position index matrix of which each spring connects(Nspring*4) and spring constant list(Nspring*1).

3. gradEs: Calculate $\frac{\partial E^{Elastic}}{\partial q}$. Input: - xk (float): x coordinate of the current point - yk (float): y coordinate of the current point - xkp1 (float): x coordinate of the next point - ykp1 (float): y coordinate of the next point - lk (float): reference length - k (float): elastic modulus. Output: F (4x1 array): Gradient array.

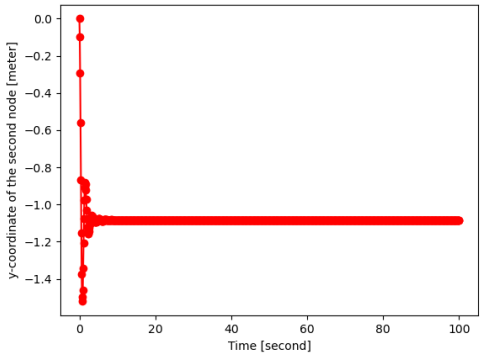

Fig. 6. Time series of y-coordinates of the second node.

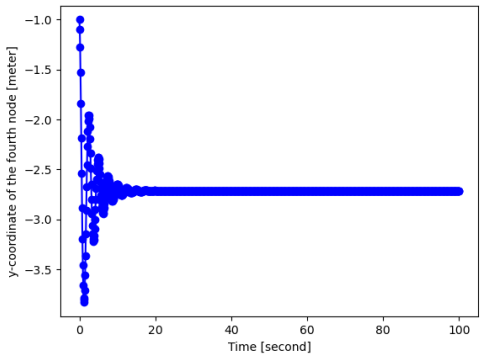

Fig. 7. Time series of y-coordinates of the fourth node.

4. hessEs: Calculate the 4x4 Hessian of the stretching energy with respect to x_k, y_k, x_{k+1}, and y_{k+1}. Input: The same as the input of gradEs. Output: the (4x4 array) Hessian.

5. getExternalForce: Calculate gravity. Input: the mass vector. Output: the gravity vector(Ndof*1).

6. getForceJacobian: Calculate $f(x)$ and $J(x)$. Input: xnew: new node position, xold: old node position, uold: old node velocity, stiffnessmatrix: spring stiffness, indexmatrix: spring node connetion index, m: mass vector, dt: time step size, lk: rest spring length. Output: $f(x)$ and $J(x)$.

7. myInt: Calculate new node position and velocity with Newton's method. Input: xold: old node position, uold: old node velocity, freeDOF: the DOFs which are not fixed, stiffnessmatrix: spring stiffness, indexmatrix: spring node connetion index, m: mass vector, dt: time step size, lk: rest spring length. Output: new node position and velocity.

8. plot: Draw the spring mass graph given the node position and spring information.

9. main: Initialization and simulation loop, iterate over time steps.

III. Choose an Appropriate Time Step Size

Implicit Euler method guarantees a stable simulation for any time step size, but a large time step size will introduce significant numerical damping causing less accuracy, also if the time step size is too large, Newton's method could be harder and slower to converge. If we use a too small time step size, the total simulation time can become to long(take significantly more number of steps to get a specific time). So we need to select a time step that balances accuracy and computational efficiency.

Algorithm 1 Simulation of spring node system

procedure INITIALIZATION
  Load the nodes and springs info from txt file
  Initialize node position and mass and spring length
  Define simulation time range
  Define free DOFs
end procedure
procedure SIMUALTION LOOP
  for k in *steps* **do**
    solve $x \equiv q(t_{k+1})$ using Newton's Method
    while $err > \epsilon$ **do**
      $J(x)$,$f(x)$ $\leftarrow$ Inertia, $\frac{\partial E^{Elastic}}{\partial q}$, External Force and their Jacobian or Hessian
      $\Delta x = J(x) \backslash f(x)$
      $x = x - \Delta x$ ▷ Only update free DOF
      err =norm$(f(x_{free}))$
    end while
    $\dot{q}(t_{k+1}) = \frac{q(t_{k+1}) - q(t_k)}{\Delta t}$
  end for
end procedure

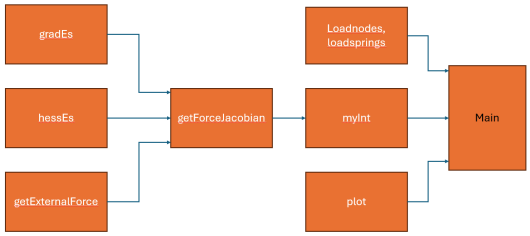

Fig. 8. This is the caption for my awesome figure.

IV. Explicit Euler Simulation

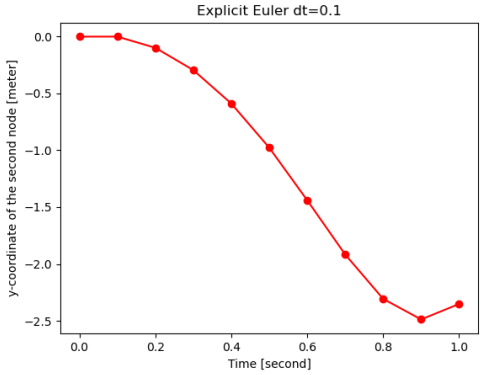

Fig. 9. Time series(t=[0,1]) of y-coordinates of the second node using Explicit Euler with dt=0.1.

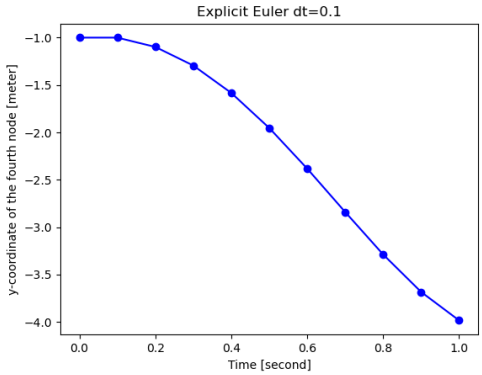

Fig. 10. Time series(t=[0,1]) of y-coordinates of the fourth node using Explicit Euler with dt=0.1.

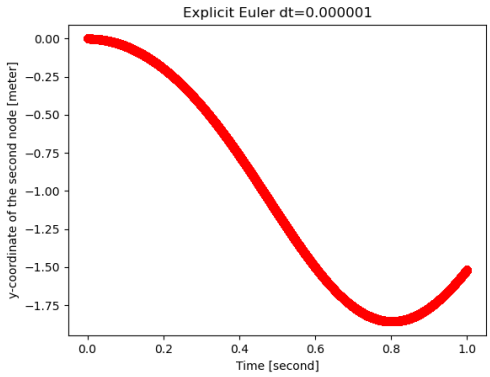

Fig. 11. Time series(t=[0,1]) of y-coordinates of the second node using Explicit Euler with dt=0.000001.

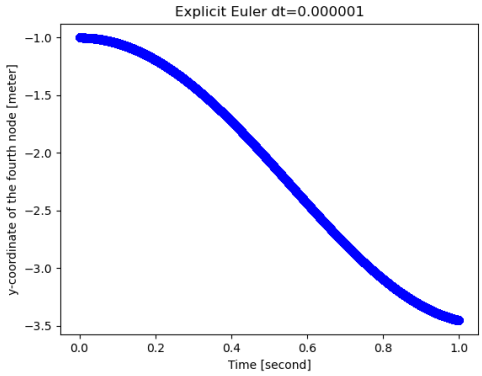

Fig. 12. Time series(t=[0,1]) of y-coordinates of the fourth node using Explicit Euler with dt=0.000001.

As a result, implicit method is more preferable for this spring network because it's stable. For explicit method, even though Δt is as low as 10^{-6}s, it is still unstable and cannot simulate properly. But for implicit method, even though Δt is as high as 0.1s, it simulates well without oscillation too much or exploding.

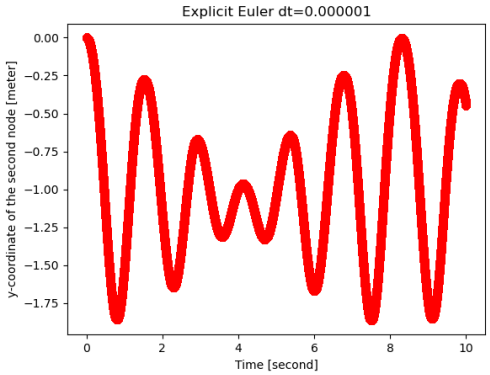

Fig. 13. Time series(t=[0,10]) of y-coordinates of the second node using Explicit Euler with dt=0.000001.

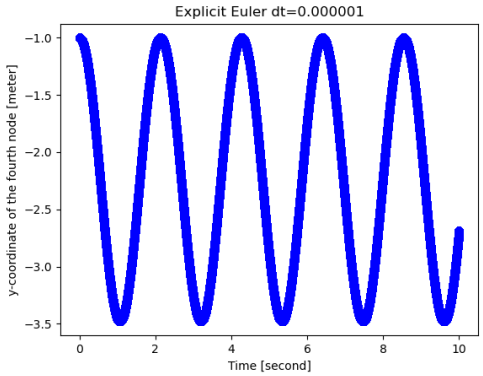

Fig. 14. Time series(t=[0,10]) of y-coordinates of the fourth node using Explicit Euler with dt=0.000001.

V. How Newmark-β Family of Integrators can mitigate artificial damping

$$x_{n+1} = x_n + \dot{x}_n \Delta t + [(\frac{1}{2} - \beta)\ddot{x_n} + \beta \ddot{x_{n+1}}]\Delta t^2$$

$$\dot{x}_{n+1} = \dot{x}_n + [(1-\gamma)\ddot{x_n} + \gamma \ddot{x_{n+1}}]\Delta t$$

Implicit Euler method is first-order accurate, so when calculating the next position, it makes a prediction based on the velocity and forces at the end of the time step without considering acceleration, it is like constantly cuts the corner of the curved trajectory and lose energy through the steps. Introducing acceleration like Newmark-β Family of Integrators is second-order accurate. It conserves energy because it updates the position using the average acceleration from the beginning and the end of the time step. So it is better at following the curved trajectory and helps conserving energy.