Simulation of Elastic Beam in Viscous Flow and External Load via Implicit and Explicit Numerical Methods

Philip Ng

***Abstract*— Beams are used in various applications of engineering, from infrastructure to vehicles. As rigid as they may seem, all beams are elastic; they deform under the loads they are given. The elastic properties of a beam are integral for understanding its various properties, such as its yield point and fracture point. Knowing how a beam will deform in various conditions helps inform engineers whether a structure is safe for use. In this article, we will explore numerical methods based on governing differential equations to generate a simulation of elastic beams in viscous flow, and under a point load. A larger version of such a simulation can be utilized to predict the elastic behavior of beams of various geometries under any external condition.**

I. INTRODUCTION AND THEORY

Let us consider a N degree-of-freedom system acted on by conservative forces only. This is a valid assumption because most elastic structures, including beams, are primarily affected by such forces. In the case of the beam in viscous flow, we also have damping and weight forces acting on the system. Thus the general equation of motion for the i-th DOF would be

$$f = m_i\ddot{q}_i + \frac{\partial E_{potential}}{\partial q_i} + c_i\dot{q}_i + W = 0 \tag{1}$$

where $c_i\dot{q}_i + W$ are the viscous damping and weight components, respectively. Let us also assume that the beam is a network of connected springs and spheres, like so:

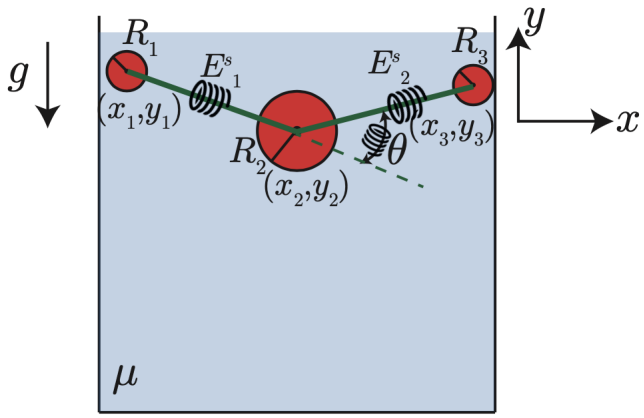

Fig. 1. Beam simplified as network of springs and spheres[1]

We will call each of the spheres *nodes* and the spring between each node *edges*. The angle θ is called the *turn angle* of the system. There are $a = N/2$ nodes and $b = N/2 - 1$ edges. Fig. 1 depicts a 3-node system with gravity oriented in the -y direction. The symbol E_i^s represents the elastic potential energy due to *stretching*, which occur at each edge. E_i^s represents the elastic potential energy due to *bending*, which occurs at each node except for the end nodes. The total elastic potential energy is summarized by

$$E_{potential} = \sum_{a=1}^{n} E_a^s + \sum_{b=1}^{n} E_b^b \tag{3}$$

where a is the number of nodes and b is the number of edges.

To prepare for simulation, we must discretize (1). Below is the discretized equations for the i-th DOF using the implicit and explicit method, respectively. We will be using both in our simulation and comparing the results. (3) is an implicit equation, while (4) is an explicit equation. Implicit equations are harder to program but converge at larger time-steps and thus require less time to compute.

$$\frac{m_i}{\Delta t^2}[\frac{q_i(t_{k+1} - q_i(t_k)}{\Delta t} - \dot{q}_i(t_k)] +$$

$$\frac{\partial E_{potential}}{\partial q_i} + c_i[\frac{q_i(t_{k+1} - q_i(t_k)}{\Delta t}] + W = 0 \tag{3}$$

$$\frac{m_i}{\Delta t^2}[\frac{q_i(t_{k+1} - q_i(t_k)}{\Delta t} - \dot{q}_i(t_k)] +$$

$$\frac{\partial E_{potential}}{\partial q_i} + c_i\dot{q}_i(t_k) + W = 0 \tag{4}$$

Note that W is only in effect for degrees of freedom in the y-direction. We can now apply the discretized equations to vectors and matrices. Since the current problem is in 2-D, we can set up the following N-DOF vector, **q**:

$$\mathbf{q} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_a \\ y_a \end{bmatrix}$$

For the mass component, we will use an N X N diagonal lumped mass matrix **M**. The components m_{ii} represent the mass of node *i* We will also create a N X N diagonal damping matrix **C**, where c_{ii} represents the damping coefficient on node *i*. Note a represents the number of nodes.

$$\mathbf{M} = \begin{bmatrix} m_{11} & 0 & 0 & \dots & 0 \\ 0 & m_{11} & 0 & \dots & 0 \\ 0 & 0 & m_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & m_{aa} \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} c_{11} & 0 & 0 & \dots & 0 \\ 0 & c_{11} & 0 & \dots & 0 \\ 0 & 0 & c_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & c_{aa} \end{bmatrix}$$

The weight matrix **W** only acts on y-oriented DOFs, so it is zero for all odd indices.

$$\mathbf{W} = \begin{bmatrix} 0 \\ W_1 \\ 0 \\ W_2 \\ \vdots \\ 0 \\ W_a \end{bmatrix}$$

With these matrices defined, the governing equation (1) becomes

$$\boldsymbol{f} = \boldsymbol{M}\ddot{\boldsymbol{q}} + \frac{\partial E_{potential}}{\partial \boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{W} = 0 \tag{5}$$

To solve (5) explicitly, simple algebraic manipulation can be used. The implicit method will require Newton Rhapson Iteration, which further requires calculation of the Jacobian. The Jacobian is the gradient of the governing equation, which yields the discretized equation

$$\boldsymbol{J} = \frac{\boldsymbol{M}}{\Delta t^2} + \frac{\boldsymbol{C}}{\Delta t} + \frac{\partial^2 E_{potential}}{\partial^2 \boldsymbol{q}} \tag{6}$$

where

$$\frac{\partial^2 E_{potential}}{\partial^2 \boldsymbol{q}} = \sum_i^n \frac{\partial^2 E_i^s}{\partial^2 \boldsymbol{q}} + \sum_i^n \frac{\partial^2 E_j^b}{\partial^2 \boldsymbol{q}} \tag{7}$$

The gradients and Hessians of the potential energies can be calculated using the functions `gradEs`, `gradEb`, `hessEs`, `hessEb` in the Appendix. Using the vectorized equations (5) and (6), we can generate a simulation of a beam with N nodes with viscous and/or external forces acting open the system.

II. GENERAL IMPLEMENTATION

A. Initial Parameters and Matrices

For proof of concept, this simulation involves a cylindrical beam with metal spheres as nodes. We first set the simulation size to N = 3. Because this is a 2-D problem, there will be a total of 2N = 6 DOFs. Next, we define the necessary physical parameters: viscosity, the density of the spheres and viscous fluid, and gravity. The beam's physical and geometrical parameters should also be defined: Young's Modulus, cross sectional area, and aerial moment of inertia. We also set the simulation parameters, such as time-step dt and the end-time of the simulation. For the specified problem, this is the equation for sphere mass ans weight, respectively, for node a:

$$m_{aa} = \frac{4}{3}\pi R_a^3 \rho_{metal} \tag{8}$$

$$W_a = \frac{4}{3}\pi R_a^3 (\rho_{metal} - \rho_{fluid}) \tag{9}$$

where ρ_{metal} and ρ_{fluid} is the density of the metal sphere and viscous fluid, respectively, and R_a is the radius of the sphere. Below is the equation for the damping coefficient C for node *i*:

$$C_a = 6\pi\mu R_a \tag{10}$$

where μ is the viscosity in Pascal-seconds (Pa-s). With the initial parameters initialized, we can generate the radius vector **R**, **M**, **C**, **W**, **P**, and **q**.

R is a vector of size N X 1, and can be populated using a `for` loop. In this case, the radius of all nodes except the middle node are 0.005m. The radius of the middle node is 0.025m.

M is a matrix of size N X N with every two diagonal components equal, since both elements correspond to the same node. We generate this by doing a `for` loop from DOF i to N, and setting both `m(2*i-1,2*i-1)` and `m(2*i,2*i)` to the same mass using (8).

C is a damping matrix of size N X N with every two diagonal components equal, since both elements correspond to the same node. The same `for` loop can be used as in **M**, except each element is equated to (10).

W is a weight vector of size N X 1, and can be populated using a `for` loop. Only the even-index components are populated by $W_a = m_a g.$, where a is the node index. Thus we can loop from DOF i to N, setting `W(2*i)` to weight in (9) and all other components to 0.

P is an external force vector of size N X 1, whose components are all 0 except at the node the force is applied, where its value is equal to P_y. We can find the node of action using the formula `nodeOfAction = round(c / l * (N-1))` where c is the distance along the beam at which P_y is applied, and l is the length of the beam.

Finally, the initial DOF vector **q0** is a N X 1 vector, where each even-index element q_i represents the y-position of a node, and each odd-index element the x-position of a node. In this problem, all y-components are set to 0 initially, while the x-compoennts are set to $l/(N-1)$, where l is the length of the beam. Thus the nodes are equally spaced along the beam. Thus a loop can be made from DOF i to n setting `q0(2*i-1) = l / (N-1)` and `q0(2*i) = 0`. We

will also initialize a velocity vector **u0** for later analysis.

B. Implicit Method

To begin the implicit method, we set the timestep to $dt = 10^{-2}$ seconds, an ending time `maxTime`, and a variable measuring the number of time steps `steps = round(maxTime / dt)`. To measure the velocity of the middle node we create a variable `midNodeVelocity` with length `steps`. Finally, we set a force-dependent tolerance `eps = E*I/l2^ * 1e-3` for the Newton-Rhapson method and an initial error `err = 10 * eps`. Now we can proceed with the time-marching scheme.

The pseudocode for the Newton-Rhapson iteration is as follows:

Algorithm 1 Implicit Iteration via Newton-Rhapson

for every timestep from 0 to endtime **do**
  Update old DOFs and velocities
  Redefine error
  while error $>$ tolerance **do**
    Define function f without elastic energy
    Define Jacobian J without elastic energy
    Update gradient and Jacobian of linear springs
    for all DOFs **do**
      get gradients using gradEs and hessEs
      Add results of gradEs to f
      Add results of hessEs to J
    end for
    Update gradient and Jacobian of bending springs
    for all DOFs **do**
      get gradients using gradEb and hessEb
      Add results of gradEb to f
      Add results of hessEb to J
    end for
    Update DOF vector and error
    Update q: $q = q - J^{-}1f$
    Update error: err = absolute sum of f
    Update middle node velocity
  end while
  Store new DOF vector and velocities
  plot the beam positions in real time
end for

Because the results of `hessES` and `gradES` are 4 X 4 matrices, they can only take four DOF components as parameters. To ensure proper matrix addition, we use the following indices:

```
  f(2*i-1:2*i+2) = f(2*i-1:2*i+2)
+ dF; J(2*i-1:2*i+2,2*i-1:2*i+2) =
J(2*i-1:2*i+2,2*i-1:2*i+2) + dJ;
```

For `hessES` and `gradES`, which are 6 X 6 matrices, we choose these six DOF components as parameters:

```
  f(2*i-3:2*i+2) = f(2*i-3:2*i+2)
+ dF; J(2*i-3:2*i+2,2*i-3:2*i+2) =
J(2*i-3:2*i+2,2*i-3:2*i+2) + dJ;
```

Because the implicit method is computationally efficient, we can plot the beam position as it changes.

C. Explicit Method

In the explicit method, the future state of a system is computed without consideration of future states, but only on the current state. Therefore there is no need for Newton Rhapson iteration, but only algebraic manipulation to solve for $q(t_{k+1})$. Thus we manipulate and vectorize the governing discretized equation (4), yielding the following:

$$\boldsymbol{q} = \boldsymbol{q} + \Delta t[\dot{\boldsymbol{q}} - \Delta t(\boldsymbol{M}^{-1}(\boldsymbol{W} + \boldsymbol{C}\dot{\boldsymbol{q}} + \frac{\partial E_{potential}}{\partial \boldsymbol{q}}))] \quad (11)$$

However, this method is computationally heavy and requires a small dt to converge. In this case we choose $dt = 10^{-5}$ seconds. Below is the pseudocode for the explicit process:

Algorithm 2 Explicit Iteration via Newton-Rhapson

for every timestep from 0 to endtime **do**
  Update old DOFs and velocities
  Redefine error
  Initialize Gradient of Elastic Energy Vector
  Update gradient of linear spring energy
  for all DOFs **do**
    get gradients using gradEs and hessEs
    Add results of gradEs
  end for
  Update gradient and Jacobian of bending spring energy
  for all DOFs **do**
    get gradients using gradEb and hessEb
    Add results of gradEb
  end for
  Update and store DOF and velocity vector
  Update middle node position and velocity
end for

The same indexing for the gradient functions applies for the explicit method and the implicit method. Although explicit methods are easier to program, they are computationally expensive and volatile at large time steps.

III. RESULTS

A. 3 Node Viscous Case

The following figures depict the beam displacement at various times, and the position and velocity of the middle node over time.

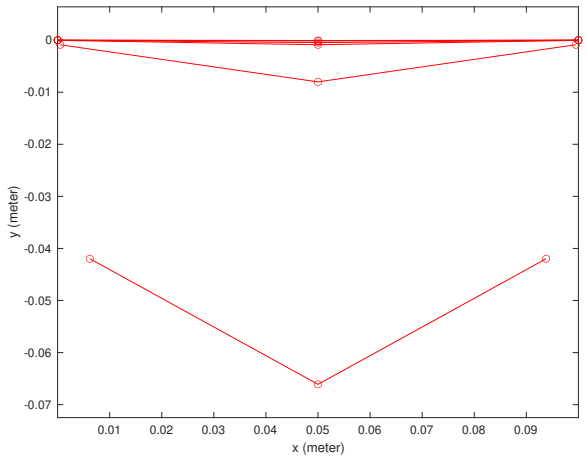

Fig. 2. 3-node beam displacements at time $t = 0, 0.01, 0.05, 0.1, 1, 10$ seconds

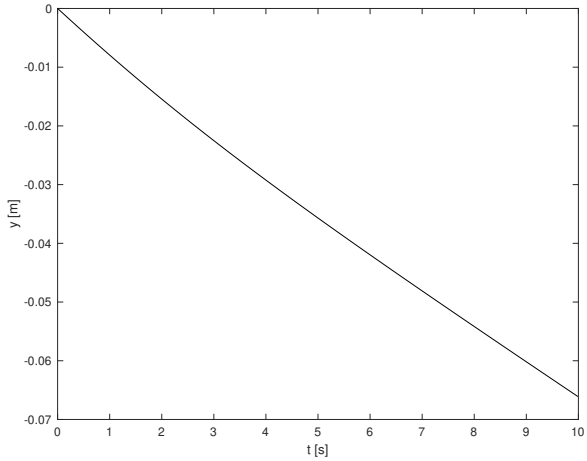

Fig. 3. Middle node (Node 2) position varying with time

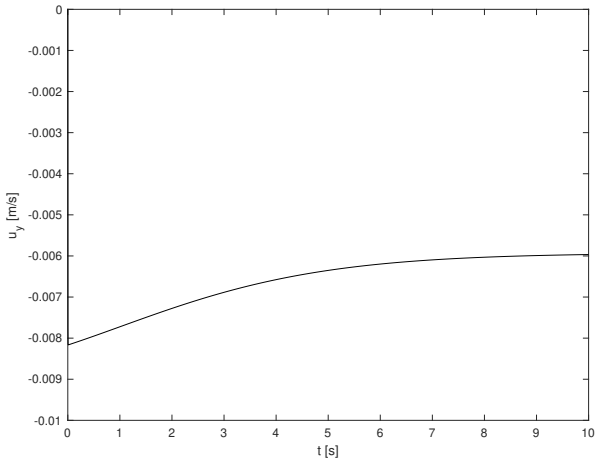

Fig. 4. Middle node (Node 2) velocity varying with time

B. 21 Node Viscous Case

The following figures depict the position and velocity of the center node over time.

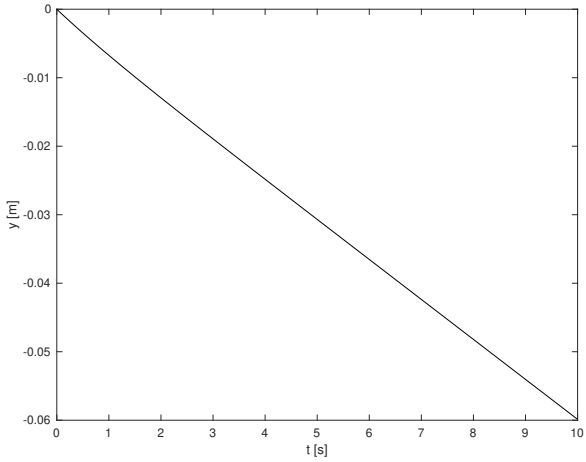

Fig. 5. Position of middle node varying with time

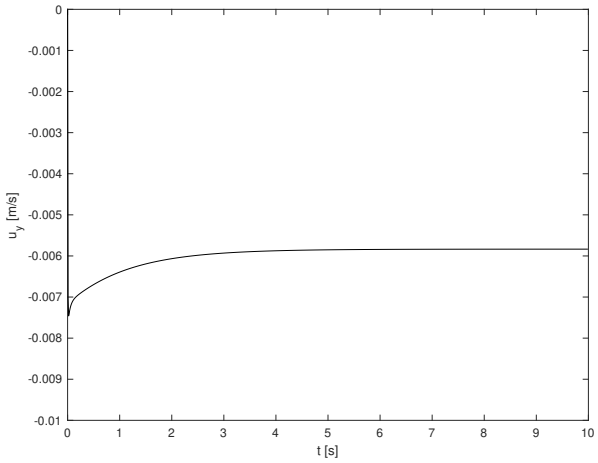

Fig. 6. Velocity of middle node varying with time

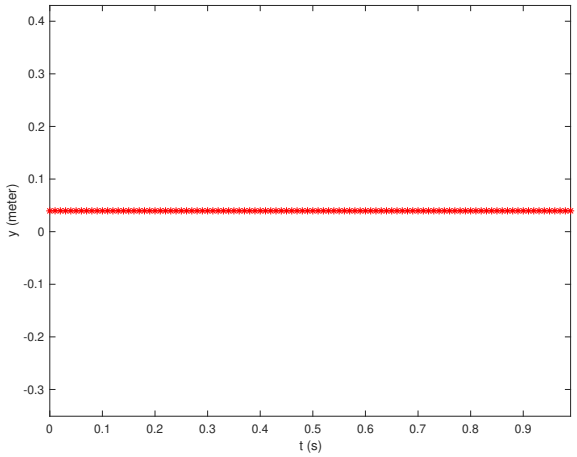

Fig. 8. Maximum displacement of beam y_{max} as a function of time

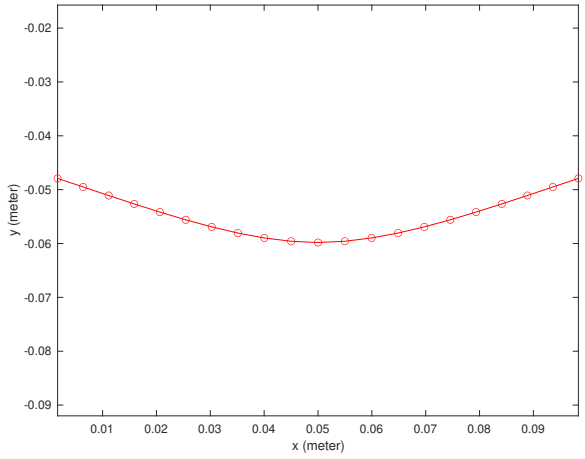

Fig. 7. Final displacement of beam after 10 seconds

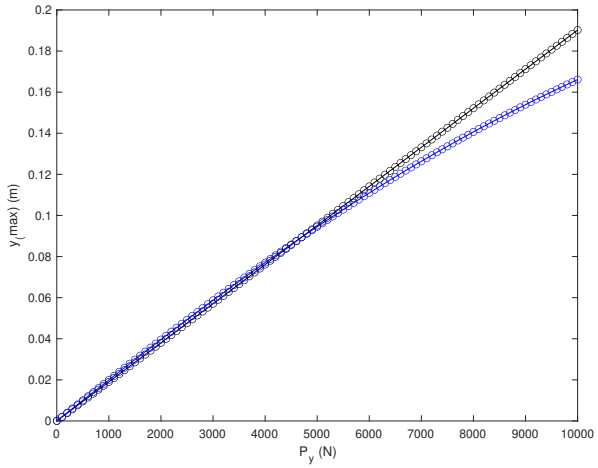

Fig. 9. Relationship between Euler-Bernoulli solution and numerical solution for y_{max} as a function of external force. The blue curve is the numerical solution. P_y

C. External Force Case

The following figures depict the maximum displacement y_{max} of the beam at a force of $P_y = 2000N$, and the discrepancy between the Euler-Bernoulli solution and our numerical solution for y_{max} at larger values of P_y .

IV. DISCUSSION

A. 3 Node Viscous Case

For the 3-Node case, the velocity increases sharply to nearly $8 \times 10^{-3} m/s$ but levels out to a terminal velocity of $5.97 \times 10^{-3} m/s$. This is expected; as the system reaches an equilibrium between viscous and weight forces, the acceleration slows, resulting in a constant, terminal velocity after some time.

The turn angle θ of the system is fully dependent on the vertical and horizontal displacements of each node, and can be calculated programatically using the Law of Cosines. When the radius of each node is the same, the weights become equal, and the turn angle decreases to 0. This is because all nodes are affected by the same weight and viscous forces in the y-direction, therefore reaching the same downward acceleration and velocity. Several runs of

our simulation verify this observation.

As we increase Δt for the explicit and implicit simulations, the simulation will become less stable, diverging earlier than normal. For example, setting $\Delta t = 10^{-4}$ for the explicit simulation causes the solution to diverge after barely 10 seconds, although the simulation runs much faster. For the implicit simulation, a much larger value of $\Delta t = 2$ must be defined to cause divergent behavior. The implicit approach is overall more stable because it can handle much larger time-steps and still converge to a solution, albeit computationally slower.

B. 21 Node Viscous Case

For the 21-Node case, the velocity increases sharply to nearly $7.3 \times 10^{-3} m/s$ but levels out to a terminal velocity of $5.83 \times 10^{-3} m/s$. This is expected; as the system reaches an equilibrium between viscous and weight forces, the acceleration slows, resulting in a constant, terminal velocity after some time.

C. External Force Case

In this case, the beam is assumed to be a hollow cylinder with a point force acting on it. The mass of the beam is represented using the following equation:

$$m = pi * (R^2 - r^2) l\rho/(N-1) \tag{12}$$

where $\rho = 2700 kg/m^3$ is the density of aluminum. P is applied at a point 0.75m away from the left side of the beam as shown in the following diagram:

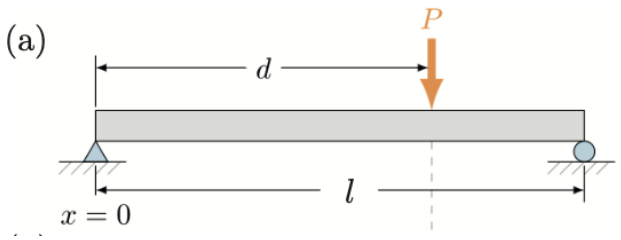

Fig. 10. Beam with external force P_y applied at a location $c = 0.75m$ from the left side. [1]

The maximum vertical displacement of the beam is given by Euler-Bernoulli Beam Theory:

$$y_{max} = \frac{Pc(l^2 - c^2)^{1.5}}{9\sqrt{3}EIl}$$

where

$$c = min(c, l - c)$$

Our simulation yielded a fairly steady displacement value of $y_{max} = 0.0396m$ for an external force of $P_y = 2000N$, compared to the Euler-Bernoulli theoretical value of $y_{max} = 0.0380m$. Although the maximum displacement calculated by Euler Bernoulli Beam Theory is fairly accurate at this specific value of P_y, further simulation showed deteriorating accuracy for larger external forces, as observed in Fig 9. Around $P_y = 6000N$, the solutions begin to diverge, reaching a percent difference of nearly 10% by $P_y = 9000N$. Our simulation is more robust for higher external forces.

V. REFERENCES

REFERENCES

[1] K. Jawed, Discretized Structures Notes.Structures Computer Interaction Lab. 2023.