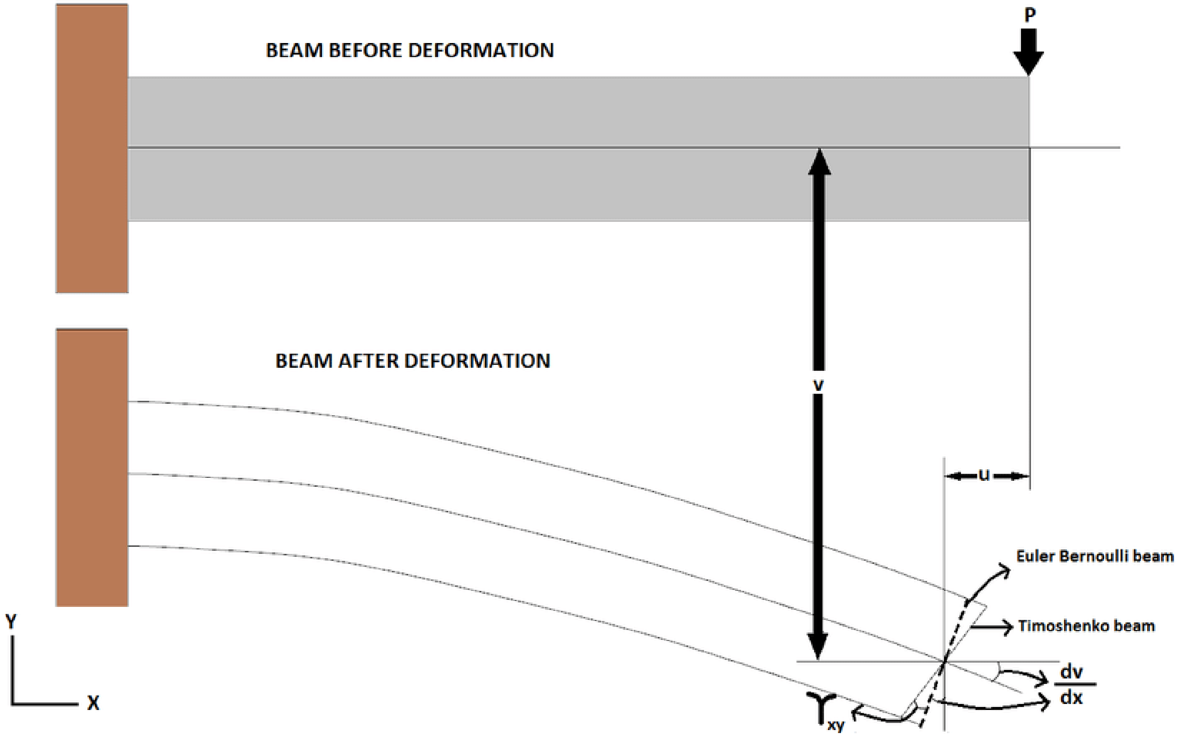
Tutorial 2: Beam Theory

Spring 2022

Instructors: Saravana Prashanth Murali Babu

Euler-Bernoulli Beam Theory

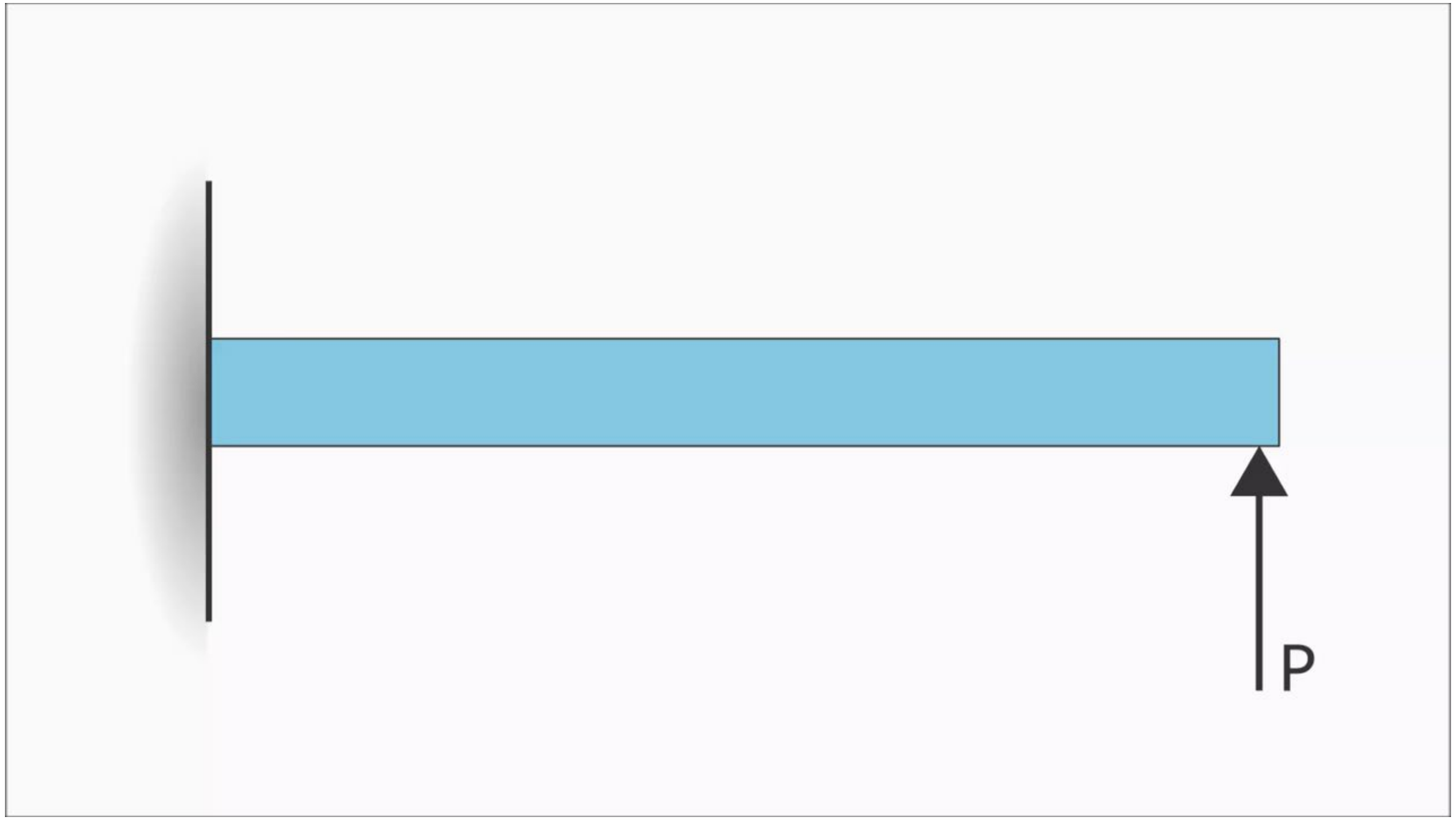


Euler-Bernoulli Beam Theory: $u - y(dv/dx)$

Timoshenko Beam Theory: $u + y(\frac{dv}{dx} + Yx_y)$

Timoshenko Beam Theory

Euler-Bernoulli vs Timoshenko Beam Theory

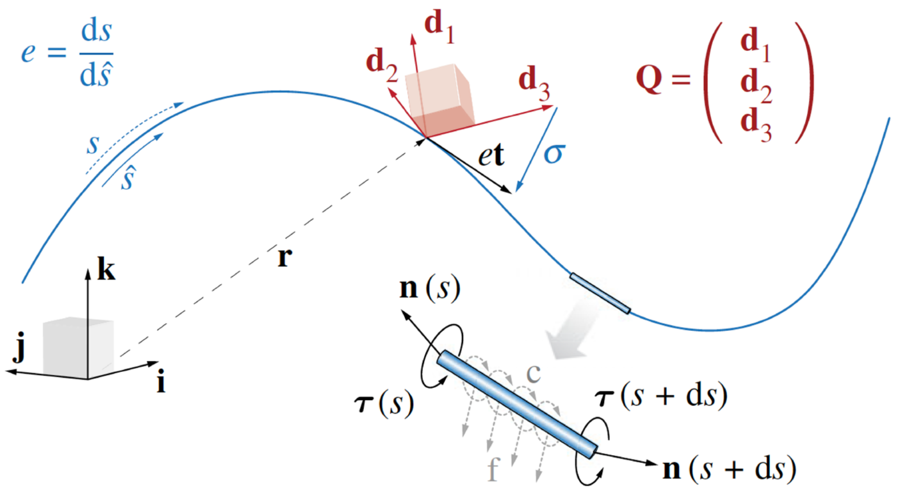


Video link

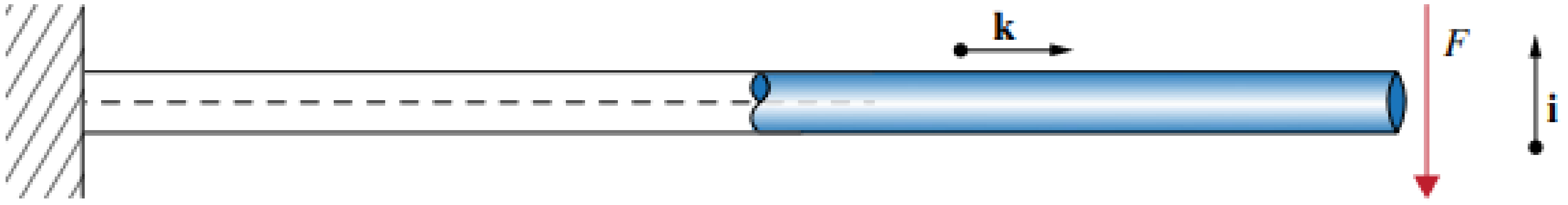
What is Cosserat rod?

The theory of Cosserat rods is a method of modeling 1D, slender rods accounting for **bend, twist, stretch, and shear**; allowing all possible modes of deformation to be considered under a wide range of boundary conditions.

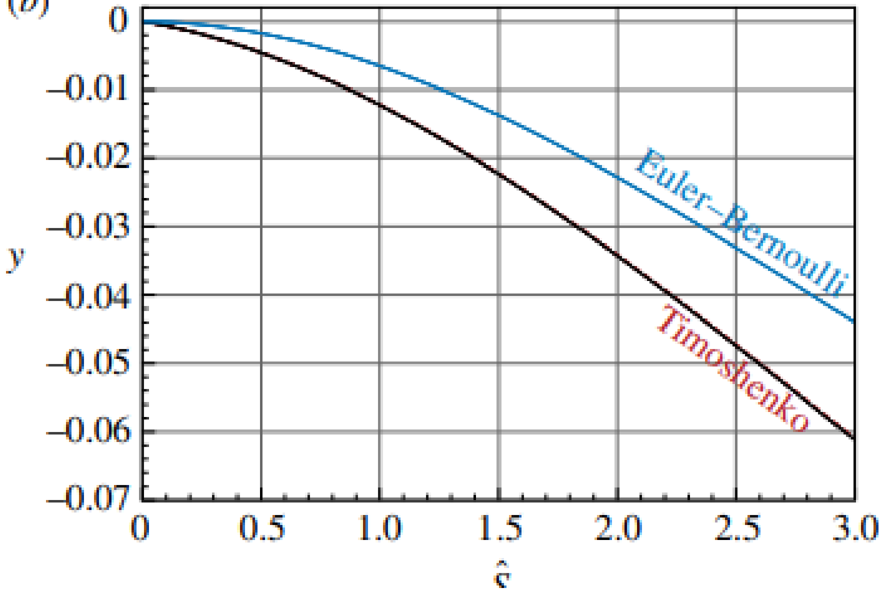
- Cosserat rods are described by a centerline **r(s,t)** and local reference frame **Q(s,t)={d1,d2,d3},** which consists of a triad of orthonormal basis vectors (using the right-hand rule convention).
-
- The dynamics of the rod are then described by the equations for conservation of linear and angular momentum throughout the rod.



(a)



(b)



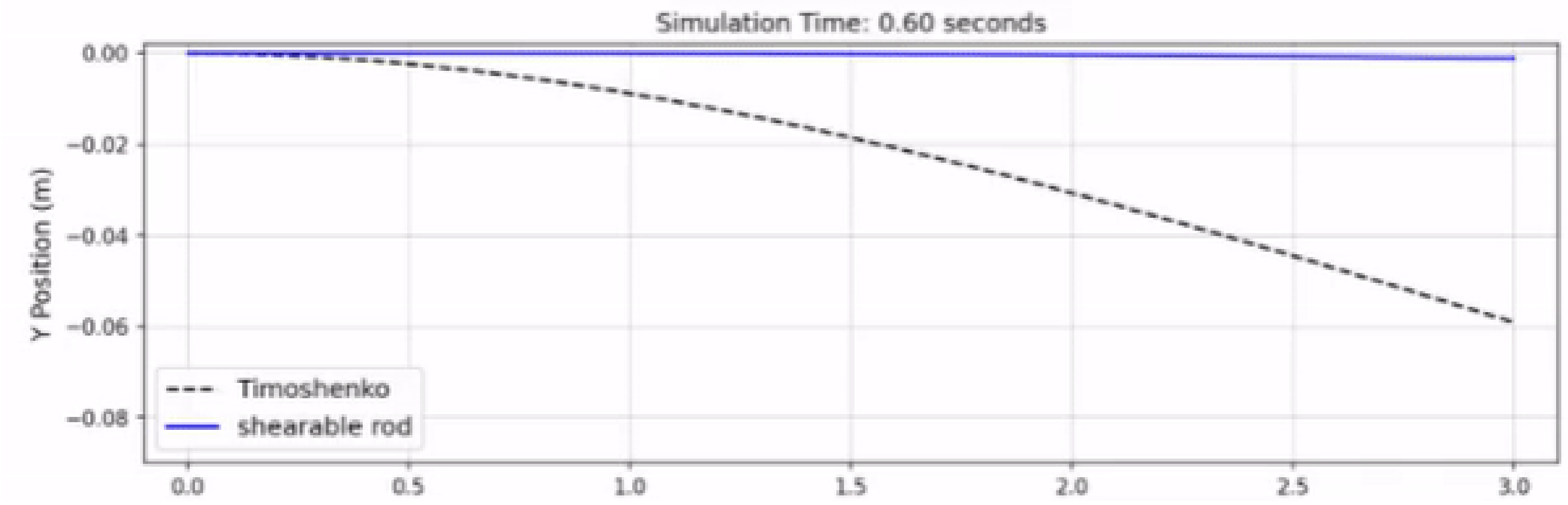
Time–space convergence study for a cantilever beam.

(a) We consider the static solution of a beam clamped at one end s = 0 and subject to the downward force F at the free end s = L.

(b) Comparison between the Timoshenko analytical y (black lines)

- numerical yn (red dashed lines) vertical displacements with respect to the initial rod configuration.
- blue the corresponding Euler–Bernoulli solution.

Dynamic simulation



Step 1: Import necessary modules and create the simulation

In [ ]:
```
import numpy as np

# Import Wrappers
from elastica.wrappers import BaseSystemCollection, Constraints, Forcing

# Import Cosserat Rod Class
from elastica.rod.cosserat_rod import CosseratRod

# Import Boundary Condition Classes
from elastica.boundary_conditions import OneEndFixedRod, FreeRod
from elastica.external_forces import EndpointForces

# Import Timestepping Functions
from elastica.timestepper.symplectic_steppers import PositionVerlet
from elastica.timestepper import integrate
```

```
class TimoshenkoBeamSimulator(BaseSystemCollection, Constraints, Forcing):
    pass

timoshenko_sim = TimoshenkoBeamSimulator()
```

Step 2: Define parameters for each rod and include rod

```python
# setting up test params
n_elem = 100     # number of elements

start = np.zeros((3,))                    # Starting position of first node in rod
direction = np.array([0.0, 0.0, 1.0])   # Direction the rod extends
normal = np.array([0.0, 1.0, 0.0])      # normal vector of rod


base_length = 3.0     ####                # original length of rod (m)
base_radius = 0.25    ####                # original radius of rod (m)
base_area = np.pi * base_radius ** 2   ###

density = 1000  # density of rod (kg/m^3)
# For shear modulus of 1e4, nu is 99!
nu = 0.1  ####  # Energy dissipation of rod
E = 1e6   ####  # Elastic Modulus (Pa)
poisson_ratio = 0.5  ###              # Poisson Ratio

shear_modulus = E / (poisson_ratio + 1.0)
```

```python
shearable_rod = CosseratRod.straight_rod(n_elem, start, direction, normal,
                base_length, base_radius, density, nu, E,
                shear_modulus=shear_modulus,n)
timoshenko_sim.append(shearable_rod)
```

Step 3:
Define boundary conditions and applied forces

```
timoshenko_sim.constrain(shearable_rod).using(
    OneEndFixedRod,                          # Displacement BC being applied
    constrained_position_idx=(0,),           # Node number to apply BC
    constrained_director_idx=(0,)            # Element number to apply BC
)
print("One end of the rod is now fixed in place")
```

```
   #Define 1x3 array of the applied forces
origin_force = np.array([0.0, 0.0, 0.0])
end_force = np.array([-20.0, 0.0, 0.0])
ramp_up_time = 5.0

timoshenko_sim.add_forcing_to(shearable_rod).using(
    EndpointForces,              # Traction BC being applied
    origin_force,                # Force vector applied at first node
    end_force,                   # Force vector applied at last node
    ramp_up_time=ramp_up_time    # Ramp up time
)
print("Forces added to the rod")
```

Step 4:
Finalize system, define time stepper and run simulation

```
timoshenko_sim.finalize()
print("System finalized")
```

```
final_time = 10.0
dl = base_length / n_elem
dt = 0.01 * dl
total_steps = int(final_time / dt)
print("Total steps to take", total_steps)

timestepper = PositionVerlet()
```

```
integrate(timestepper, timoshenko_sim, final_time, total_steps)
```

Step 5:
Post Processing Results

Mathematical Model Calculator

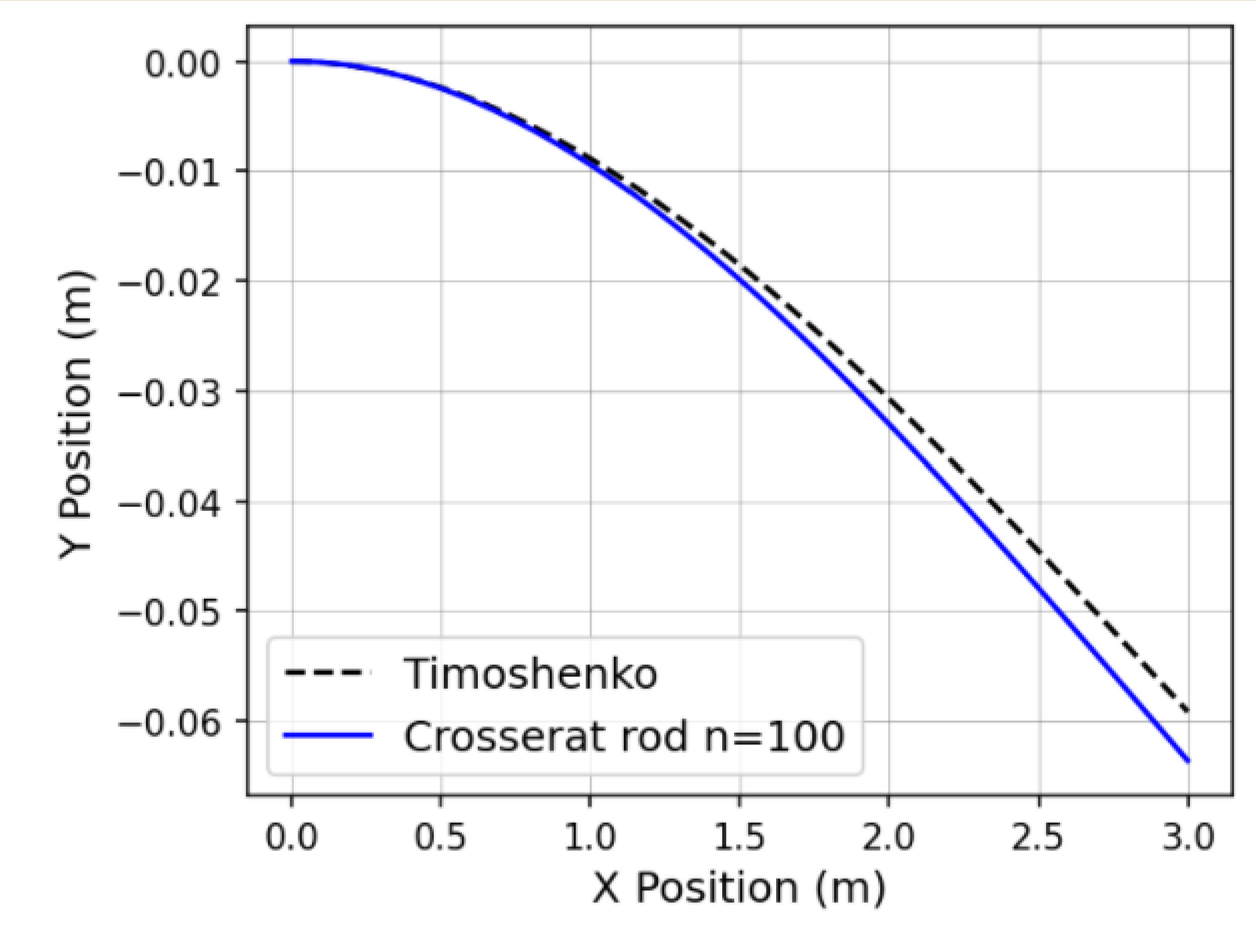
```python
# Compute beam position for sherable and unsherable beams.
def analytical_result(arg_rod, arg_end_force, shearing=True, n_elem=500):
    base_length = np.sum(arg_rod.rest_lengths)
    arg_s = np.linspace(0.0, base_length, n_elem)
    if type(arg_end_force) is np.ndarray:
        acting_force = arg_end_force[np.nonzero(arg_end_force)]
    else:
        acting_force = arg_end_force
    acting_force = np.abs(acting_force)
    linear_prefactor = -acting_force / arg_rod.shear_matrix[0, 0, 0]
    quadratic_prefactor = (
        -acting_force
        / 2.0
        * np.sum(arg_rod.rest_lengths / arg_rod.bend_matrix[0, 0, 0])
    )
    cubic_prefactor = (acting_force / 6.0) / arg_rod.bend_matrix[0, 0, 0]
    if shearing:
        return (
            arg_s,
            arg_s * linear_prefactor
            + arg_s ** 2 * quadratic_prefactor
            + arg_s ** 3 * cubic_prefactor,
        )
    else:
        return arg_s, arg_s ** 2 * quadratic_prefactor + arg_s ** 3 * cubic_prefactor
```

Plot Results Function

```
def plot_timoshenko(shearable_rod, end_force):
    import matplotlib.pyplot as plt
    analytical_shearable_positon = analytical_result(
        shearable_rod, end_force, shearing=True
    )
    fig = plt.figure(figsize=(5, 4), frameon=True, dpi=150)
    ax = fig.add_subplot(111)
    ax.grid(b=True, which="major", color="grey", linestyle="-", linewidth=0.25)
    ax.plot(
        analytical_shearable_positon[0],
        analytical_shearable_positon[1],
        "k--",
        label="Timoshenko",
    )
    ax.plot(
        shearable_rod.position_collection[2, :],
        shearable_rod.position_collection[0, :],
        "b-",
        label="Crosserat rod n=" + str(shearable_rod.n_elems),
    )
    ax.legend(prop={"size": 12})
    ax.set_ylabel("Y Position (m)", fontsize=12)
    ax.set_xlabel("X Position (m)", fontsize=12)
    plt.show()
```

Step 6: Plot Result

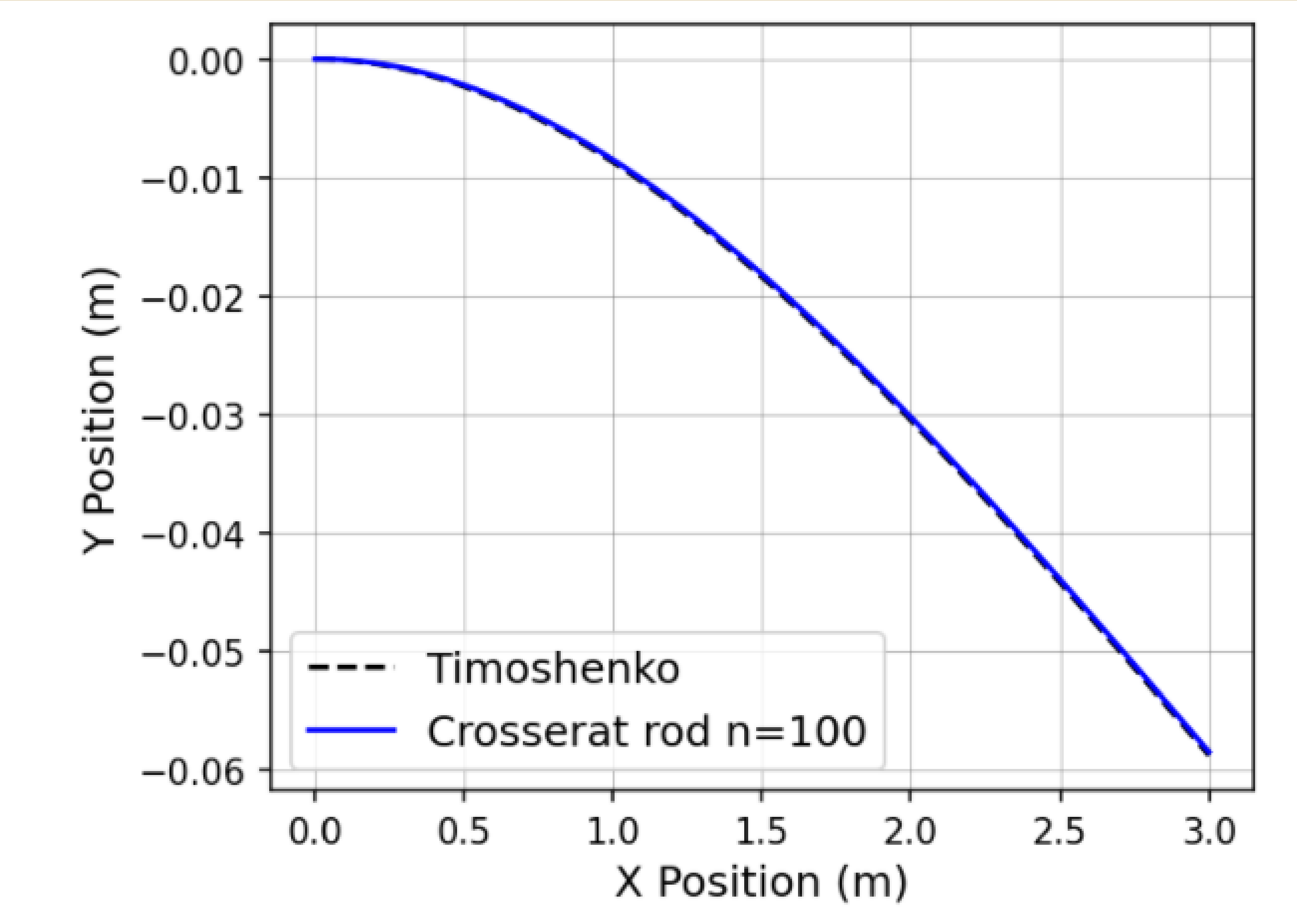
```
plot_timoshenko(shearable_rod, end_force)
```



Task 1:
Unshearable rod

```
density = 1000  # density of rod (kg/m^3)
# For shear modulus of 1e4, nu is 99!
nu = 99  ####  # Energy dissipation of rod
E = 1e6   ####  # Elastic Modulus (Pa)
poisson_ratio = -0.85  ###            # Poisson Ratio

shear_modulus = E / (poisson_ratio + 1.0)
```



Task 2:
Changing dimensions

```
base_length = 3.0      ####              # original length of rod (m)
base_radius = 0.25     ####              # original radius of rod (m)
base_area = np.pi * base_radius ** 2   ###
```

Task 3:
Changing material properties

```
density = 1000  # density of rod (kg/m^3)
# For shear modulus of 1e4, nu is 99!
nu = 0.1   ####  # Energy dissipation of rod
E = 1.694e5   ####  # Elastic Modulus (Pa)
poisson_ratio = 0.5  ###            # Poisson Ratio

shear_modulus = E / (poisson_ratio + 1.0)
```

Reference paper