**PROJECT**

* **NAME :** K.CHANDRA MOHAN
* **HALL TICKET NO:** 24705A0306
* **BRANCH:** MECHANICAL ENGINEERING
* **YEAR:** 3RD YEAR, B.TECH
* **COLLEGE:** ANNAMACHARYA INSTITUTE OF TECHNOLOGY & SCIENCES, RAJAMPET

**BEAM DEFLECTION**

* Analyzes how beams bend under loads
* **INPUTS:** Shear force diagram
* **OUTPUTS:**numpy,matplotlib,scipy

**SHEAR FORCE DIAGRAM**:

* **source code:**

import matplotlib.pyplot as plt

class PointLoad:

def \_\_init\_\_(self, magnitude, position):

self.magnitude = magnitude # Load in Newtons

self.position = position # Position from left support in meters

class Beam:

def \_\_init\_\_(self, length):

self.length = length

self.loads = []

def add\_point\_load(self, magnitude, position):

self.loads.append(PointLoad(magnitude, position))

def calculate\_reactions(self):

"""Assume simply supported beam with supports at x=0 and x=L"""

total\_load = sum(load.magnitude for load in self.loads)

moment\_about\_A = sum(load.magnitude \* load.position for load in self.loads)

Rb = moment\_about\_A / self.length

Ra = total\_load - Rb

def shear\_force\_distribution(self, step=0.01):

Ra, Rb = self.calculate\_reactions()

x\_vals = []

shear\_vals = []

shear = Ra

current\_load\_index = 0

for x in frange(0, self.length, step):

# Check if we hit a load

while (current\_load\_index < len(self.loads) and

abs(x - self.loads[current\_load\_index].position) < step / 2):

shear -= self.loads[current\_load\_index].magnitude

current\_load\_index += 1

x\_vals.append(x)

shear\_vals.append(shear)

return x\_vals, shear\_vals

def frange(start, stop, step):

"""Range function that supports float stepping"""

while start <= stop:

yield round(start, 5)

start += step

def plot\_shear\_force(x, shear):

plt.figure(figsize=(10, 4))

plt.plot(x, shear, label='Shear Force', color='blue')

plt.axhline(0, color='black', linewidth=0.8)

plt.xlabel("Beam Length (m)")

plt.ylabel("Shear Force (N)")

plt.title("Shear Force Diagram")

plt.grid(True)

plt.legend()

plt.show()

beam = Beam(length=10)

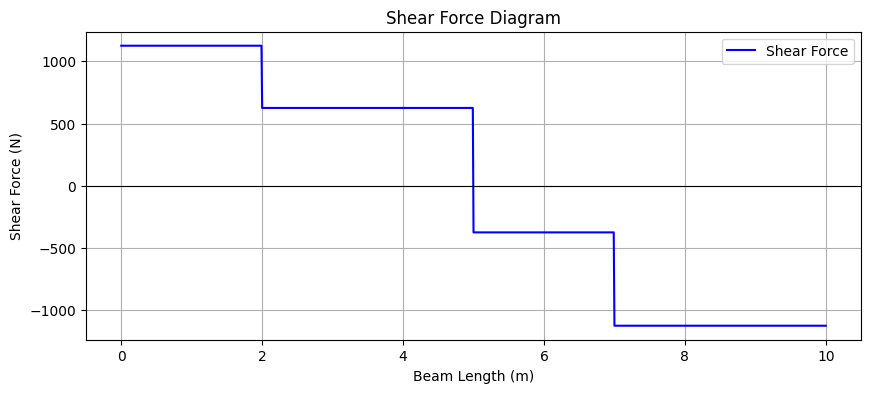
beam.add\_point\_load(500, 2)

beam.add\_point\_load(1000, 5)

beam.add\_point\_load(750, 7)

x\_vals, shear\_vals = beam.shear\_force\_distribution()

plot\_shear\_force(x\_vals, shear\_vals)



**BENDING MOMENT DIAGRAM:**

* Analyzes how beams bend under loads
* **INPUTS:**Bending moment diagram
* **OUTPUTS:**numpy,matplotlib,scipy
* **source code:**

import matplotlib.pyplot as plt

class PointLoad:

def \_\_init\_\_(self, magnitude, position):

self.magnitude = magnitude

self.position = position

class Beam:

def \_\_init\_\_(self, length):

self.length = length

self.loads = []

def add\_point\_load(self, magnitude, position):

self.loads.append(PointLoad(magnitude, position))

def calculate\_reactions(self):

total\_load = sum(load.magnitude for load in self.loads)

moment\_about\_A = sum(load.magnitude \* load.position for load in self.loads)

Rb = moment\_about\_A / self.length

Ra = total\_load - Rb

return Ra, Rb

def bending\_moment\_distribution(self, step=0.01):

Ra, Rb = self.calculate\_reactions()

x\_vals = []

moment\_vals = []

for x in frange(0, self.length, step):

moment = 0

moment += Ra \* x

for load in self.loads:

if x >= load.position:

moment -= load.magnitude \* (x - load.position)

x\_vals.append(x)

moment\_vals.append(moment)

return x\_vals, moment\_vals

def frange(start, stop, step):

while start <= stop:

yield round(start, 5)

start += step

def plot\_bending\_moment(x, moment):

plt.figure(figsize=(10, 4))

plt.plot(x, moment, label='Bending Moment', color='green')

plt.axhline(0, color='black', linewidth=0.8)

plt.xlabel("Beam Length (m)")

plt.ylabel("Bending Moment (Nm)")

plt.title("Bending Moment Diagram")

plt.grid(True)

plt.legend()

plt.show()

beam = Beam(length=10)

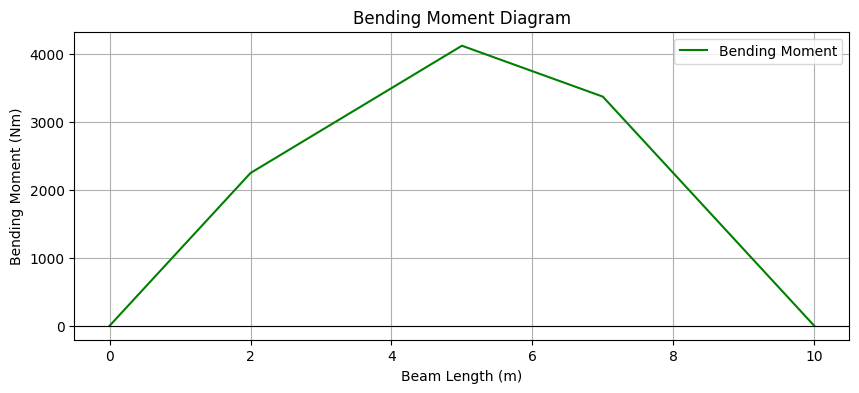
beam.add\_point\_load(500, 2)

beam.add\_point\_load(1000, 5)

beam.add\_point\_load(750, 7)

x\_vals, moment\_vals = beam.bending\_moment\_distribution()

plot\_bending\_moment(x\_vals, moment\_vals)



**DEFLECTION OF CURVE**

* Analyzes how beams bend under loads
* **INPUTS:**Deflection of curve
* **OUTPUTS:**numpy,matplotlib,scipy
* **source code:**

import numpy as np

import matplotlib.pyplot as plt

class PointLoad:

def \_\_init\_\_(self, magnitude, position):

self.magnitude = magnitude

self.position = position

class Beam:

def \_\_init\_\_(self, length, E, I):

self.length = length

self.E = E

self.I = I

self.loads = []

def add\_point\_load(self, magnitude, position):

self.loads.append(PointLoad(magnitude, position))

def calculate\_reactions(self):

total\_load = sum(load.magnitude for load in self.loads)

moment\_about\_A = sum(load.magnitude \* load.position for load in self.loads)

Rb = moment\_about\_A / self.length

Ra = total\_load - Rb

return Ra, Rb

def bending\_moment\_distribution(self, x\_vals):

Ra, Rb = self.calculate\_reactions()

Mx = []

for x in x\_vals:

moment = Ra \* x

for load in self.loads:

if x >= load.position:

moment -= load.magnitude \* (x - load.position)

Mx.append(moment)

return np.array(Mx)

def deflection\_curve(self, num\_points=500):

x\_vals = np.linspace(0, self.length, num\_points)

Mx = self.bending\_moment\_distribution(x\_vals)

EI = self.E \* self.I

y = np.zeros\_like(x\_vals)

dx = x\_vals[1] - x\_vals[0]

for i in range(1, len(x\_vals)):

theta[i] = theta[i-1] + (Mx[i-1] + Mx[i]) \* dx / (2 \* EI)

y[i] = y[i-1] + (theta[i-1] + theta[i]) \* dx / 2

y\_shift = np.interp(self.length, x\_vals, y)

y = y - y\_shift

return x\_vals, y

def plot\_deflection(x\_vals, deflection\_vals):

plt.figure(figsize=(10, 4))

plt.plot(x\_vals, deflection\_vals \* 1000, label='Deflection (mm)', color='purple')

plt.axhline(0, color='black', linewidth=0.8)

plt.xlabel("Beam Length (m)")

plt.ylabel("Deflection (mm)")

plt.title("Deflection Curve of the Beam")

plt.grid(True)

plt.legend()

plt.show()

L = 10

E = 200e9

I = 8.33e-6

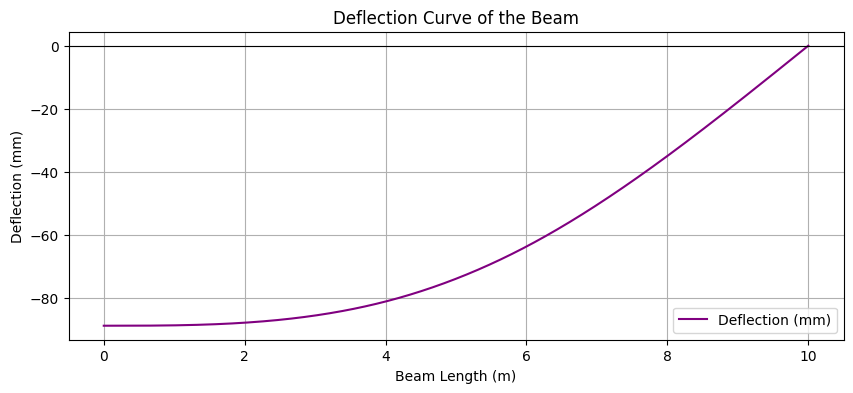
beam = Beam(L, E, I)

beam.add\_point\_load(1000, 4)

beam.add\_point\_load(1500, 6)

x\_vals, y\_vals = beam.deflection\_curve()

plot\_deflection(x\_vals, y\_vals)



**CONCLUSION:**

The provided Python programs enable the analysis of simply supported beams subjected to point loads by calculating and visualizing.