Stability of structures
Course project

- Harrsha(20161056)
- Varun(2019210017)

- **Buckling Analysis of Frames:**

Buckling load in frame has to be determined using commercial structural analysis software like SAP2000 and compare it with the manual calculations in compliance with AISC Alignment Charts (American Institute for Steel Construction) and IS 800: 2007 . The frame considered is two bay single storey frame with fixed support condition case and hinged support condition. The buckling capacity of the frame is determined using the Sigma P concept. This is an approximate method for determining the critical buckling load of frames. It considers two modes of frame buckling, i.e. Non-Sway Mode for each individual column and a Sway Mode for the whole frame. Non-Sway Modes consider buckling of each individual column about their corresponding weak axis. While the Sway modes consider the simultaneous buckling of all columns in each storey. In Sway mode, the contribution of leaning columns (hinged-hinged) towards sway stiffness is considered to be negligible and all load on the frame is considered to be carried by the non-leaning columns. After examining each case separately, the smallest load gives the critical buckling load of the frame. Portal frames generally buckle in Sway modes unless laterally braced. The critical buckling load of frames is more in Non-Sway modes compared to the one needed for SideSway buckling regardless of the stiffness of the members.

  - Software used: sap2000
  - Assigned Frame(2 storey, 2 bays)
  - Length of each bay = 5 m

- Height of each storey = 3 meters

GUI for beam columns with different loads and corresponding amplification factors and include an option to compare with beam columns without axial loads:

- Develop a GUI for beam columns with different loads and corresponding amplification factors and include an option to compare with beam columns without axial loads
  - Develop a program for carrying out the stability analysis of the beam columns . For different support and loading combinations
  - Build a gui for users where stability functions are plotted based on the given input conditions.
  - Django will be used the development of the application
  - The api for the application is discussed below

Initially a common class is developed for the defined the **beam-column** with the mentioned conditions and loadings.

```
class Beam_column:
    def __init__(self,I,E,l):
       self.I = I
       self.E = E
       self.l = l
   ## loads->>> axial_load,unform_loading,point_load(self,P):
    def axial_load(self,Pu,w,p):
       self.Pu = Pu
       self.w = w
       self.P = P
    def left_connection(self,lc):
       if(lc=='Fixed'):
             dof = 0
       elif(lc=='hinged'):
             dof = 1
       elif(lc=='roller'):
             dof = 2
       elif(lc=='free'):
             dof = 3
    def right_connection(self,rc):
```

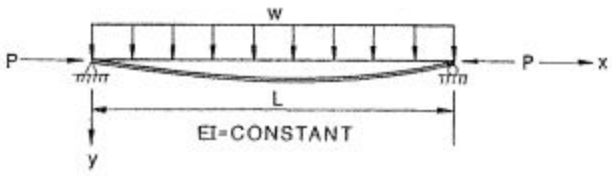
```
      if(lc=='Fixed'):
            dof = 0
      elif(lc=='hinged'):
            dof = 1
      elif(lc=='roller'):
            dof = 2
      elif(lc=='free'):
            dof = 3z

### Attributes of the Api
>> C1.BMD()
>> C1.Critical_load()
>> C1.stability_function()
>> C1.max_deflection()
>> C1.delfection_profile()
```

A solver file will be created to solve the different cases(solver.py)

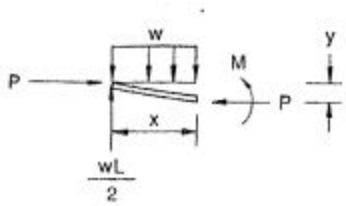
- **Note** :
  - The idea is to implement a solver file which solves the differential equations given the initial conditions and loading conditions.

Stability functions implementation for Beam_columns of different loading cases :

Case 1: Beam-column with uniformly distributed Lateral Load



Loading diagram



FBD

$$M_{ext} = Py - \frac{w}{2}x^2 + \frac{wL}{2}x$$

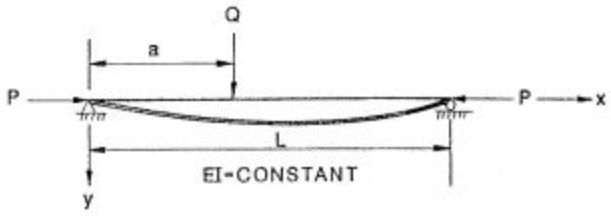
Governing differential Equation

$$y(0) = 0, \qquad y'\left(\frac{L}{2}\right) = 0$$

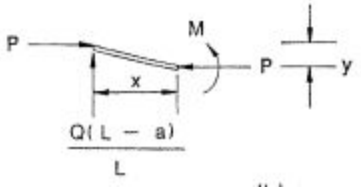
Boundary Conditions of the system

$$M_{int} = -EIy''$$

Case 2 : Beam-column with a concentrated lateral load



Loading diagram



FBD

$$-EIy'' = \frac{Q(L-a)}{L}x + Py \quad \text{for} \quad 0 \leq x \leq a$$

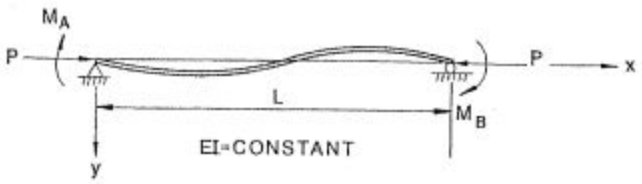
$$-EIy'' = Qa\left(\frac{L-x}{L}\right) + Py \quad \text{for} \quad a \leq x \leq L$$

Governing differential Equations

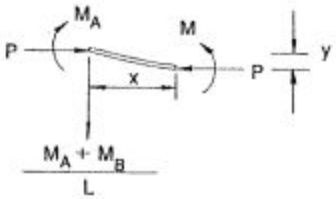
$$y(0) = 0$$
$$y(L) = 0$$

Boundary Conditions of the system

Case 3 : Beam-column subjected to End_moments



Loading diagram



FBD

$$EIy'' + Py = \frac{M_A + M_B}{L}x - M_A$$

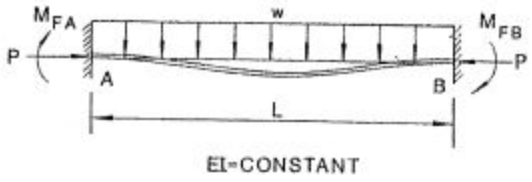
Governing differential Equations

$$y(0) = 0, \qquad y(L) = 0$$

Boundary Conditions of the system

Case 4: Fixed Ended Beam-columns

- **Uniform loaded**



Loading diagram

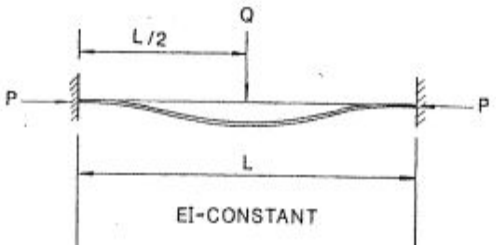
$$\frac{d^4y}{dx^4} + k^2\frac{d^2y}{dx^2} = \frac{w}{EI}$$

Governing Differential Equation

$$y(0) = 0, \qquad y'(0) = 0, \qquad y(L) = 0, \qquad y'(L) = 0$$

Boundary Conditions

- **Concentrated loaded**



Loading diagram

$$\frac{d^3y}{dx^3} + k^2\frac{dy}{dx} = -\frac{Q}{2EI}$$

Governing Differential Equation

$$y(0) = 0, \qquad y'(0) = 0, \qquad y'\left(\frac{L}{2}\right) = 0$$

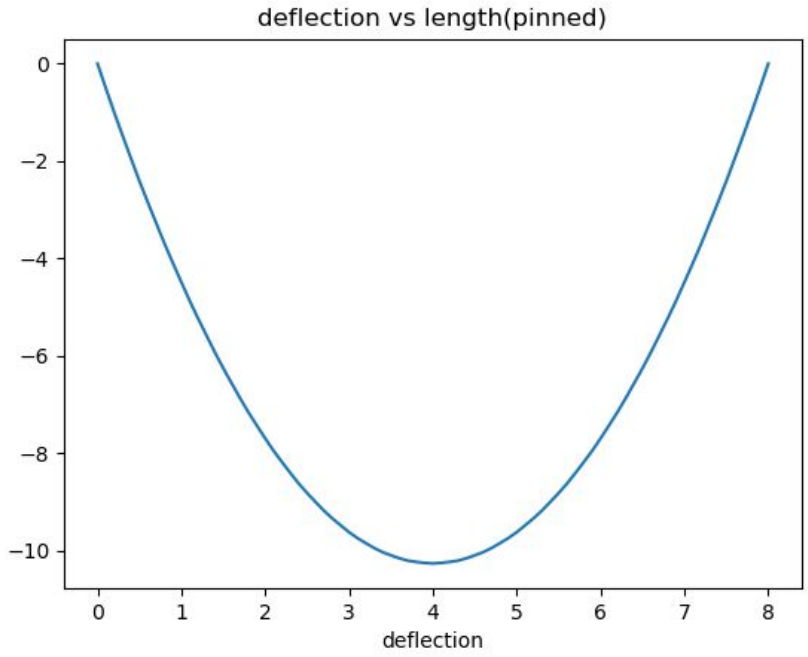
Boundary Conditions

The differential equation will be solve using ODE_solver

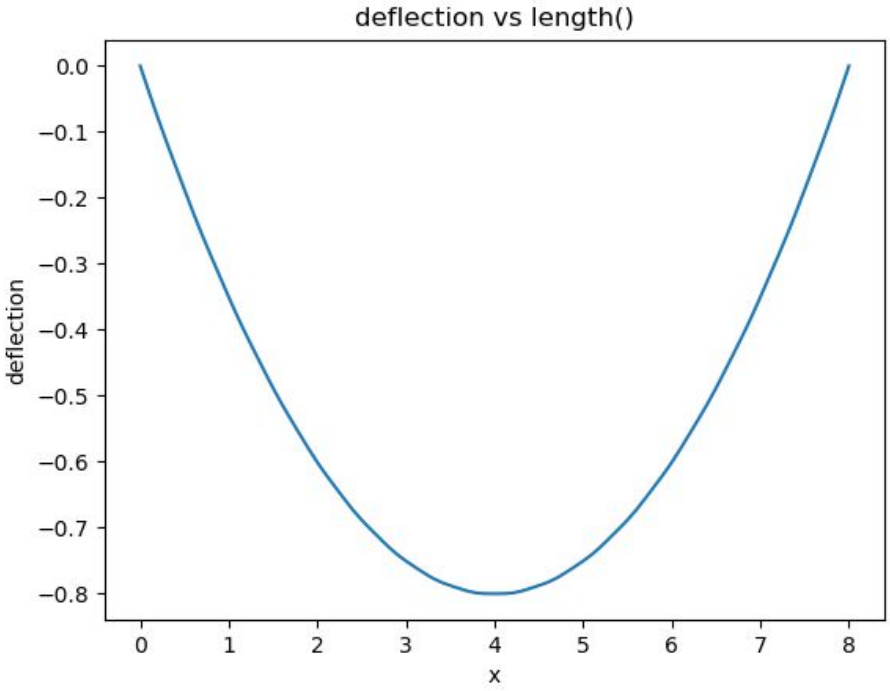
Api for Beam-Column Implemented:

```
>>> from Beam_column import *
>>> E, I, P = symbols('E, I, P', positive=True)
>>> c1 = Beam_column(10, E, I,
15000,{'uniform':8000,'point':[10000,5]},[0,0],top="hinged",
bottom="hinged")
>>> c1.boundary_conditions
{'deflection': [(0, 0), (10, 0)], 'slope': [(0, 0)]}
>>> c1.solve_slope_deflection()
>>> c1.deflection()
sqrt(6)*sqrt(E)*sqrt(I)*(-Ma - Mb +
50*w)*sin(50*sqrt(6)*x/(sqrt(E)*sqrt(I)))/45000000 - E*I*w/225000000 +
Ma*x/150000 + Ma/15000 + Mb*x/150000 - Q/30000 + w*x**2/30000 - w*x/3000 +
(E*I*w - 15000*Ma + 7500*Q)*cos(50*sqrt(6)*x/(sqrt(E)*sqrt(I)))/225000000
```

p = 78000
w = 10e4
l = 8
E = 2e5
I = 10**-3



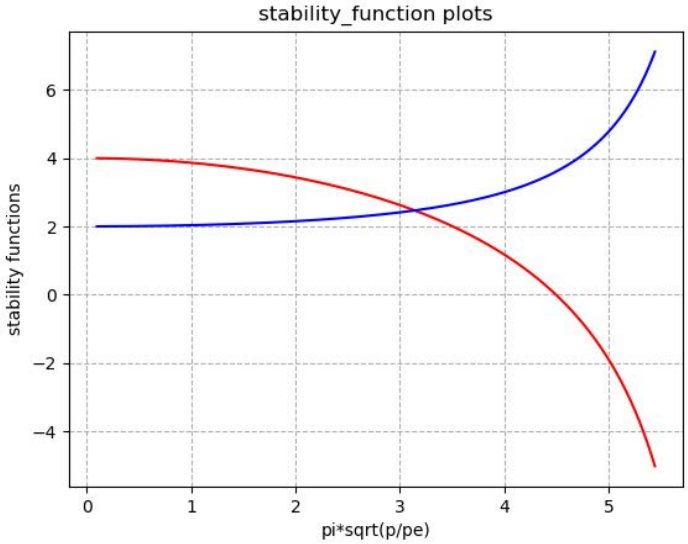
p = 4000
w = 10e4
l = 8
E = 2e5
I = 10**-3



Deflection in MM

plot of the stability function :

- For Beam column with End-moments(and without relative displacements)




```
## Example pseudo_code_for_the above graph
def stability_function():
    for i in range(1,3000):
       kl = np.pi*np.sqrt(i/1000)
       c11 = (kl*np.sin(kl)-((kl)**2)*np.cos(kl))/
                     (2-2*np.cos(kl)-(kl)*np.sin(kl))
       c12 = (((kl)**2) - kl*np.sin(kl))/
                     (2-2*np.cos(kl)-(kl)*np.sin(kl))
```