Extending Continuum Mechanics Module

Implementing Classes for Solving Arches
Extending functionality for Cable Class

Shishir Kushwaha

Google Summer of Code 2024 Proposal

Sympy

Abstract

I would like to extend the continuum mechanics module and complete the following tasks as part of the project this summer:

- Improving methods for **Cable** class by implementing the draw method.
- Extending structures by introducing an **Arch** class:
    1. Defining Arch Class
    2. Implement a solver for Arch under Concentrated Load
    3. Implement a solver for Arch under uniformly distributed load.
    4. Adding a draw method and plot for forces generated in the arch.
- Extended goal: Methods to plot Influence Line diagrams for the classes in the module

Table of Content

About Me

Personal Information

Name	Shishir Kushwaha
E-mail	shishir.kushwaha.mat22@iitbhu.ac.in **(**Work) kushwahashishir1112@gmail.com (Personal)
Github	shishir-11
University	Indian Institute of Technology (BHU), Varanasi
Degree	Integrated Dual Degree (B.Tech + M.Tech)
Time Zone	IST (UTC +5:30)

Academic Background

I am a second-year undergraduate student pursuing a Dual Degree (B.Tech + M.Tech) in Mathematics and Computing under the Department of Mathematical Sciences at the Indian Institute of Technology (BHU), Varanasi. I started my programming journey two years ago and have experience working with several programming languages like Python, C, C++, Java, Javascript, Rust and Golang.
As a student of the Mathematical Sciences Department and due to my interests, I have undertaken the following subjects:

- Differential Equations
- Abstract Algebra
- Linear Algebra
- Graph Theory
- Probability, statistics and Mathematical Modelling
- Discrete Mathematics
- Numerical techniques (involving topics such as the regula-falsi method, Newton-Rhapson method etc.)
- Mathematical methods (involving various transformation techniques including Laplace, Fourier, Hankel etc.)
- Complex Analysis
- Number theory
- Digital Image Processing

Additionally, I have also undertaken courses on Engineering Mechanics and Thermodynamics

- ## Programming Details

    1. I work on a dual boot system, switching between OS depending on the requirement with VS code as my primary code editor.
    2. I have been programming in C, C++ and Python for over 2 years and experience with various libraries and stacks like Scikit Learn, Numpy, Matplotlib and Pytorch.
    3. Familiar with using Git as version control.
    4. Some of my previous work is available on GitHub. This includes contributions to a Website of the Mathematics Society (NextJs and Javascript), a Web3.0 Website (using Solidity, ReactJS and Javascript), implementation and analysis of various Mathematical techniques used for Image enhancement (using Python), implementation and Analysis of various ML Models (like GPT) on Shakespeare's text (using Python, TensorFlow).
    5. I have been a user of Sympy for about 6 months now for my assignments in Mechanical Engineering and its useful integrals, and have been contributing to it for nearly 2 months.

- ## Motivation

    I was introduced to Sympy when I started my academic coursework on Digital Image Processing and Mathematical Methods since it allowed me to work with many integrals and functions (e.g. Bessel) which take a lot of time to solve manually. Later while exploring the functionalities offered by Sympy, I came across the Continuum Mechanics module which allowed me to solve questions related to Engineering Mechanics.
    The current module allows us to solve problems and plot for structures like Beams, Trusses and Cables.
    I wish to extend this module by making a solver for structures like Arches and implementing plotting and influence line diagrams for them and the Cable Classes.

    I have studied the following material to understand the Continuum Mechanics module:

    - Engineering Mechanics by Irving H. Shames
    - Mechanics of Materials by Dr. B.C. Punmia
    - Structural Analysis by R.C. Hibbeler

Contributions To Sympy

Pull Requests

1. #26152: Elliptic functions use visually distinct symbols
2. #26192: Fix Code Recommended for Commutative Problem
3. #26200: Numpy Array Support for POSform and SOPform added
4. #26225: Modify Config file to exclude modules required
5. #26230: Fix Typing Issues and replace with actual function names
6. #26222: Changed definition to match use of Transverse_magnification function
7. #26327: Use raw python type in wigner_3j

Issues Opened

1. #26205: Slotscheck Failing on master branch
2. #26219: Wigner3j has problems with Half Integers

To view other contributions:- Link

The Project

- **Brief Overview**

  The project aims to extend the Continuum Mechanics module. Continuous developments have been made in the module in the past few years, making it extremely useful in solving many engineering problems related to structural analysis. The current module has well-implemented classes for solving problems related to Beams, Trusses and Cables which can be extended to provide more functionality for existing classes and introduce some new Structural classes.

  After a discussion with the mentors and past GSoC mentees in the organisation, I have decided to implement the following ideas:

  - Improving Cable Classes and implementing the draw method
  - Introducing new Arch class
    - Define the Arch class
    - Implement a solver for Arch subjected to concentrated load
    - Implement a solver for Arch subjected to uniformly distributed load
    - Adding the Draw method and plot for forces generated in each member.
  - Extended Goal: Add a method to plot the Influence Line Diagrams for other classes.

Considering the scope of the project, I would like to propose this as a large project (350 hours).

- **Execution Plan**

  I would like to implement my project with each of the above-mentioned ideas in one phase/part for each:

  1. **Phase 1:**
     This phase would consist of improving the Cable class by implementing the draw method. This would allow the users to draw the diagrams for better visualization of the structure.

  2. **Phase 2:**
     In this phase, I will be introducing the **Arch** class, and implementing a solver for concentrated and distributed loads on a three-hinged arch.

  3. **Phase 3:**
     A draw method will be implemented in this phase to ensure ease of plotting the diagram as well as forces developed in the members for the users.

  4. **Extended Phase:**
     This phase would focus towards achieving the extended goal. The time left(if available) would be utilised to develop methods to plot influence line diagrams for Trusses, Cable and Arch classes.

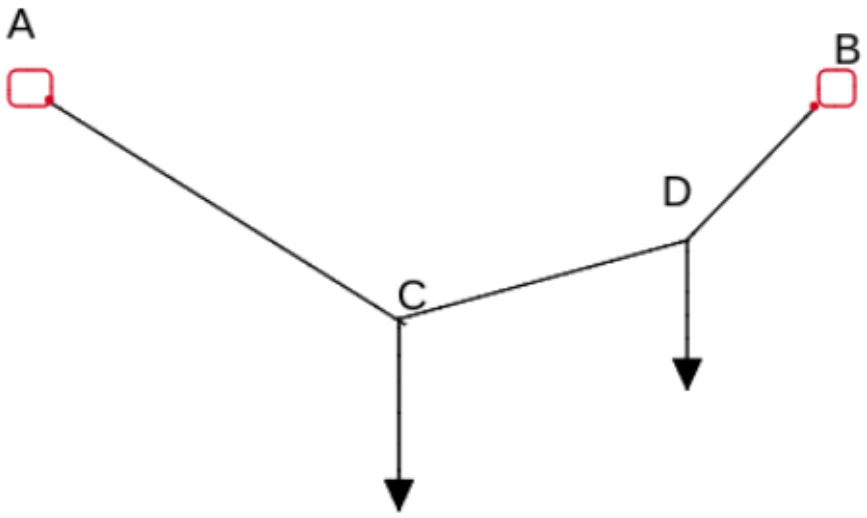
Community Bonding Period

During this period, I will start discussing the project details with my mentors. Since I have been actively contributing to Sympy, I have had the opportunity to interact and share ideas with many great developers. If the discussions with the mentors conclude early, I would like to start **Phase 1** during the Community Bonding period.

Phase 1

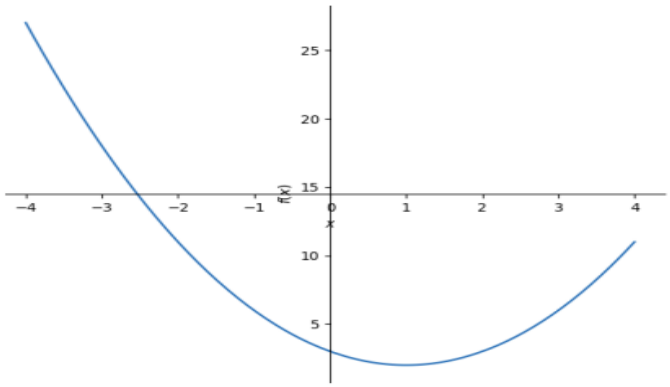
I will be starting this project by working on improvements to the existing classes. Cables are important structures from the point of view of engineering. Last year's work on GSoC defined and implemented the solver for the Cable class along with adding extensive tests to ensure robustness and making some important changes to improve user experience in other classes as well. I would like to extend their work to implement a draw method that would allow users to easily plot the diagrams related to the class and plot tension generated in various components.

This would be implemented by taking inspiration based on the previous work of the mentors. This method would return a plot object that will show the state of the cable, the position of the loads, the position of supports and some other information.



Under the application of concentrated load, the cable takes shapes similar to the above image depending on the position and magnitude of the load it is subjected to.

Under distributed load, the Cable takes a parabolic shape whose function can be found by approximating a quadratic polynomial via the matrix method. A plot similar to the one below will be obtained.



Implementation

```
def draw(self):

    # Since the cable shape is determined by the load applied, we will break the
    # Cable shape into a piecewise function and try to plot it
    # supports= []
    annotations = []

    markers = self._draw_supports()
    # for drawing reaction loads
    annotations += self._draw_reaction_loads()
    # for concentrated loads
    if self.load_type == 'point_load':
        annotations += self._draw_conc_loads()
    # for distributed loads
    else:
        annotations += self._draw_dist_loads()

    # make a function of the cable shape to be plotted of
    # function will be a parabolic shape if distributed loads are applied

    if self.load_type == 'distributed':
        cable_shape = self.to_parabola(self.supports, self.length, self._lowest_x_global,)

    # function will be piecewise if concentrated loads are applied
    else:
        cable_shape = self.to_piecewise(self.supports, self.length , self.loads)

    # plot the cable shape
    cable_plot = plot(cable_shape, (x, self._left_support[0],self.right_support[0]),
        markers=markers, show=False,
                              annotations=annotations,
                              xlim=(self._left_support[0]-1, self._right_support[0]+1),
                              ylim=(min(self._left_support[1], self._right_support[1])-1,
                              max(self._left_support[1], self._right_support[1])+1),
                              axis=False)

    return cable_plot
```

Given above, is a rough implementation of how I would like to approach the issue of plotting the diagram of the cable class.

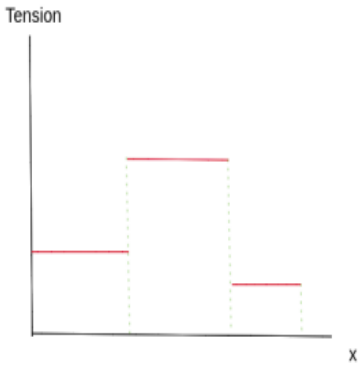
The piecewise functions are already available in the Sympy library and we would just need to get the coordinates of the points of application of concentrated loads, find the equation of line segments and convert them to a piecewise function.

```
def to_piecewise(self, supports, length, loads):
    # Define symbols
    x = symbols('x')
    # Define the piecewise function
    # Assuming the x coordinates of the nodes are in sorted order
    f = Piecewise(
        (self._left_support[1], x == self._left_support[0]),
        (self._right_support[1], x == self._right_support[0])
    )
    x_existing = self._left_support[0]
    y_existing = self._left_support[1]
    for x1, y1, _, _ in loads["point_load"]:
        f = Piecewise(
            (f, x <= x_existing),
            ((y1-y_existing)/(x1-x_existing)*x, x > x_existing & x <= x1),
        )

    return f
```

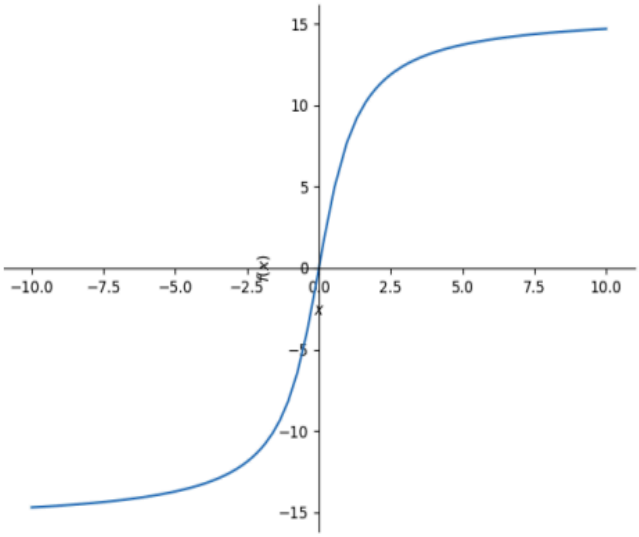
For the distributed loads, I will be implementing a function that through Vandermond's Matrix Method, finds the equation of the cable and plots it using sympy.plotting module.

```
def to_parabola(self, supports, length, lowest_x,lowest_y):
    # get the y coordinate of lowest point of the cable which needs to be
specified in solve
    points = [(self._left_support[0], self._left_support[1]),
(self._right_support[0], self._right_support[1]),(lowest_x, lowest_y)]
    A = Matrix([[point[0]**2, point[0], 1] for point in points])
    B = Matrix([point[1] for point in points])

    # solve for the coefficients of the parabola
    a, b, c = A.inv()*B
    x = symbols('x')

    quadratic = a*x**2 + b*x + c

    return quadratic
```

For plotting the diagrams for tension, I would like to use _tension['distributed'] to get the tension in various points for distributed loads and for point_loads _tension would give the list of segments, which could be used to plot the tension diagram.



The diagram given above shows the type of tension generated expected when Point_load is applied to the Cable. This can be achieved with the help of the Piecewise function and Sympy plotting



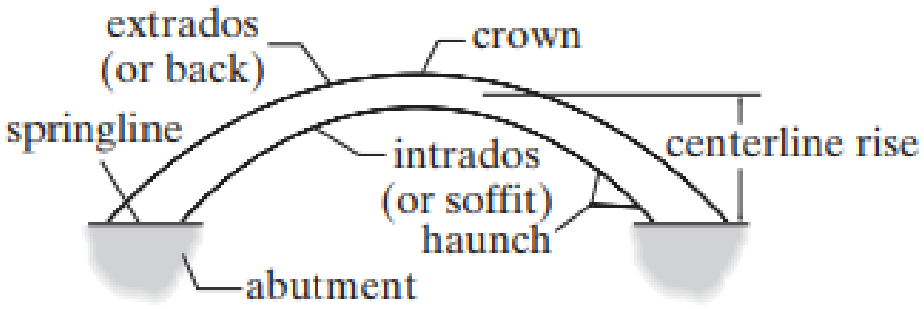
The above diagram shows the expected shape of the tension generated when a distributed load is applied to the Cable. We find the function of tension with respect to its coordinate and plot it using Sympy's plotting module similar to what is used in the draw method.

Phase 2

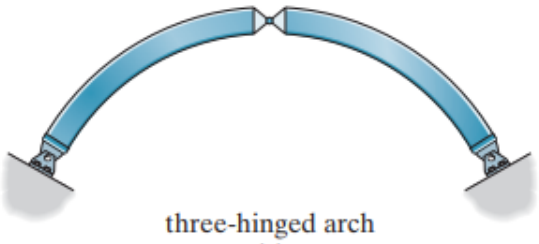
Overview

In this Phase, the focus lies on defining the Arch Class and implementing a solver that would allow us to solve for the reaction forces generated at the supports and the forces(radial shear and normal stress) generated in the arch due to the loads. I would like to focus on the implementation of the three-hinged arch structure .

Like cables, arches can be used to reduce the bending moments in long-span structures. Essentially, an arch acts as an inverted cable, so it receives its load mainly in compression although, because of its rigidity, it must also resist some bending and shear depending upon how it is loaded and shaped. In particular, if the arch has a parabolic shape and it is subjected to a uniform horizontally distributed vertical load, then from the analysis of cables it follows that only compressive forces will be resisted by the arch. Under these conditions, the arch shape is called a funicular arch because no bending or shear forces occur within the arch. Given below is a diagram of a general Arch.



A three-hinged Arch is statically determinate and is not affected by settlement or temperature changes.



The diagram above shows the general structure of a three-hinged arch.
I would try to tackle the problem by following the following steps:

Defining the Arch class:
The new class will be introduced here. Users would be allowed to initialize an Arch object and define its properties like the type of arch (‘hinged’, ’tied’), and the position of hinges. Examples and explanations related to both Arch types will be given later.

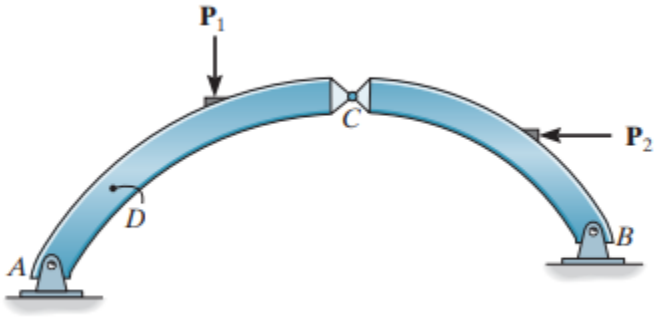
```
class Arch:
    def __init__(self,**kwargs):
        self._arch_type = kwargs.get('arch_type','hinged') # format (hinged, tied)
        self._left_support = kwargs.get('left_support') # format (x,y)
        self._right_support = kwargs.get('right_support') # format (x,y)
        self._hinge_position = kwargs.get('hinge_position') # format (x,y)
        self._rope_position_left = {} # for tied support
        self._rope_position_right = {} # for tied support
        self._conc_loads = [] #format [(label,x,y,magnitude,angle)]
        self._dist_loads = [] #format [(label,x1,x2,magnitude)]
        self._shape = None
        self._angle = None
        self._loads = []

    def add_load(self,*args):
        # Similar to the implementation in the Beam class we can use order to
define the type of load
        # or moment applied to the structure which might be more suitable for
further development
        # A remove load function can be added which can be
        # used to remove the loads based on their labels
        pass

    def remove_load(self,label):
        # remove the load based on the label
        pass

    def add_support(self,support_type,position):
        pass

    def remove_support(self,support_type):
        pass
```

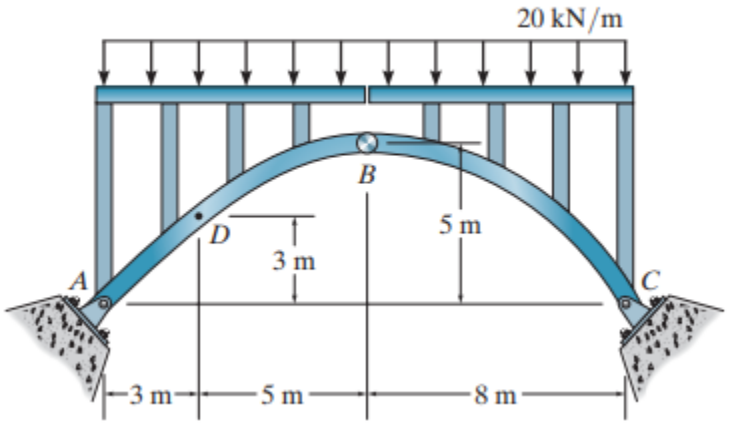
The module would initially solve for the statically determinate arches which are three-hinged arch or tied arches. For three hinged arches, we would need the position of the three hinges, i.e. the left support, right support and the top hinge. For a tied arch, one of the supports can be made into a rolling support instead of a pin-hinged support.

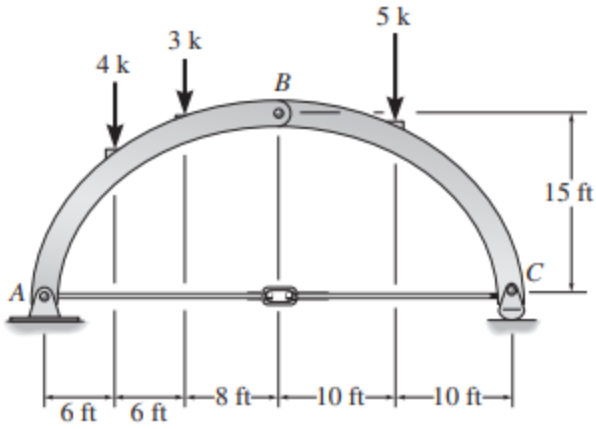
The loads would be divided into two ‘concentrated’(or point loads) and distributed loads that will be stored along with their direction, magnitude and label in a list.

The Class would take inspiration from classes already implemented, i.e. Cable, Truss and Beam.

After the initial implementation of the Class, we would be able to achieve an object whose physical significance would be similar to the following images.







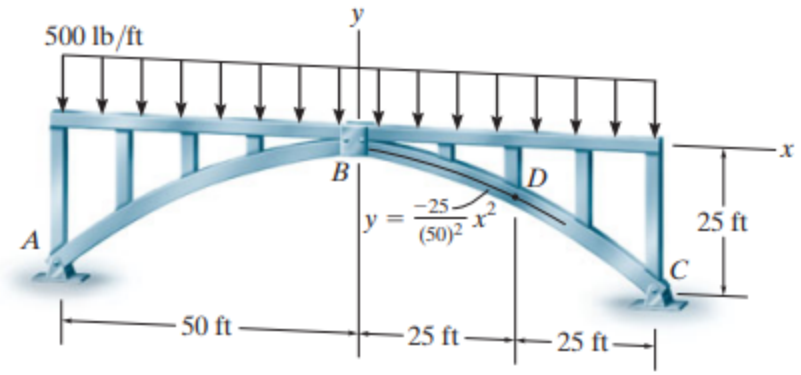
Implementing Solver for the Arches

A general scheme for solving the problems related to arches under loads is given:

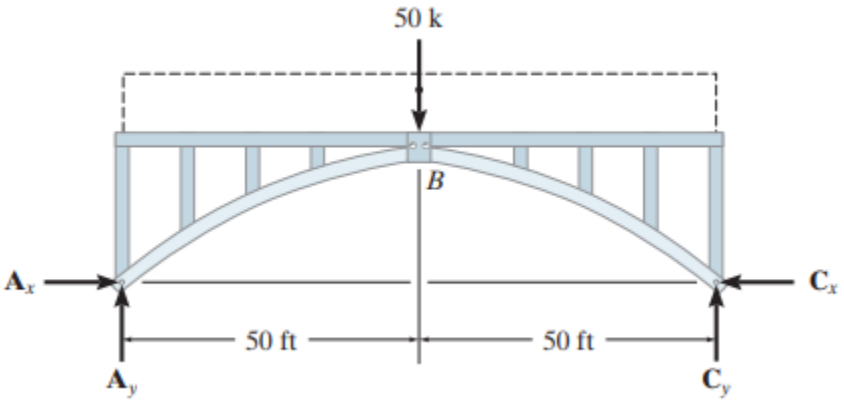
a. The sum of all external forces is zero and the sum of moments at left support, right support and the topmost hinge is also zero. Used to find out the reaction forces in supports and tension in the rope(in case of tied arches).
b. Find the equation of the parabolic shape (height as a function of distance)
c. Use the equation generated to find the angle at the point where we need to find our results or make a general equation in terms of $tan(\theta)$, x.
d. Calculate the bending moment at any point by adding up all the moments on the left side of the point.
e. We resolve the forces at any point into axial force(normal thrust), and shear force(radial).
f. We write these two forces in terms of shear force and horizontal force at supports.
g. Shear force can be calculated by summing up the forces in the vertical direction on the left side of the point.

For this section of the project, I would focus on making a solver that is able to tackle problems where distributed and point loads are applied to various places of the structure. Suitable errors would be produced in case of wrong inputs (for example, giving a net inward force in the case of a roller joint with tied rope). The aim would be to compute the reaction forces at joints, tension developed in rope (if used), bending moment, axial forces and shear forces developed in the structure. A rough implementation along with an example is given below.

We will try to find the Reaction forces at supports, bending moment and shear force developed in the arch.



Consider the FBD of the entire arch, and apply equations of equilibrium.

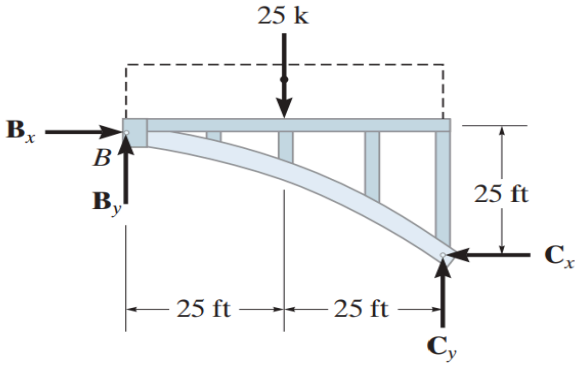


Entire arch:

$$\circlearrowleft + \Sigma M_A = 0; \qquad C_y(100\text{ ft}) - 50\text{ k}(50\text{ ft}) = 0$$

$$C_y = 25\text{ k}$$

Now consider the segment BC and apply equilibrium equations.



$$\circlearrowleft + \Sigma M_B = 0; \qquad -25\,\text{k}(25\,\text{ft}) + 25\,\text{k}(50\,\text{ft}) - C_x(25\,\text{ft}) = 0$$

$$C_x = 25\,\text{k}$$

$$\overset{+}{\rightarrow} \Sigma F_x = 0; \qquad B_x = 25\,\text{k}$$

$$+\uparrow \Sigma F_y = 0; \qquad B_y - 25\,\text{k} + 25\,\text{k} = 0$$

$$B_y = 0$$

Now we need to find the equation of the shape of the arch in the xy plane.

(We use our own coordinate system for generalised solving and not one mentioned in the diagram)
A user would give the coordinates of both the supports. For this example, let's assume the coordinate of the left support to be (0,0) and the right support to be (100,0)

Take the equation of a parabola,

$$y = a(x - x_{leftsupport})(x - x_{rightsupport})$$

Now we know the coordinates of the crown hinge, left support and right hinge. Substituting them in the above equation.

$$25 = a(50)(50 - 100)$$

$$a = -25/(50 * 50)$$

$$a = -1/100$$

$$y = \frac{100x - x^2}{100}$$

We calculate the angle at any point

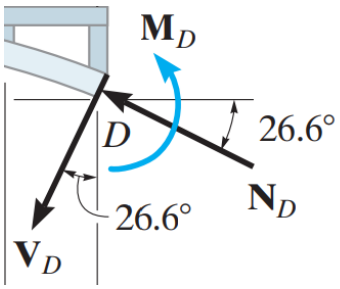
$$tan(\theta) = \frac{dy}{dx} = \frac{100 - 2x}{100}$$

The slope of the segment at point D,

$$tan(\theta)|_{x=75} = -0.5$$

$$\theta = -26.6^\circ$$

Now consider the segment AD. At point D, we try to find the shear and bending moment.



The distributed forces can be converted into a point force and applied to the segment AD to find the moment.

Consider the vertical force to be Fy(vertically up) and the vertical force to be Fx(horizontally right) at point D. The distributed force can be converted to a point force of 37.5k and applied at point 37.5m from support A.

Now write the equations of equilibrium for the structure.

$$F_y + A_y = 37.5k$$

$$F_y = 12.5k$$

$$F_x + A_x = 0$$

$$F_x = -25k$$

$$F_y = V_D \cos(26.6^\circ) - N_D \sin(26.6^\circ)$$

$$F_x = N_D \cos(26.6^\circ) + V_D \sin(26.6^\circ)$$

Through these equations we get.

$$N_D = 28.0\,\mathrm{k}$$

$$V_D = 0$$

Writing the moment balance equation at point D we get

$$h = \frac{100 * 75 - 75^2}{100} = 18.75ft$$

$$\sum M_D = 0$$

$$37.5k * 37.5 - A_y * 75 + A_x * 18.75 + M_D = 0$$

$$M_D = 0$$

Following this method we can solve the problems related to the arches.

Implementation

```
   def solver(self):
        # Step1: we convert the loads into their x and y components
        x_loads , y_loads = self.to_component()

        # Step2: we calculate the reaction forces at the supports
        # for hinged support
        reaction_forces = []

        sum_x , sum_y = self._sum_forces(x_loads,y_loads)
        # using force balance in vertical direction A_y + B_y + sum_y = 0
        # using force balance in horizontal direction A_x + B_x + sum_x = 0

        moment_A_external = self._sum_moments(x_loads,y_loads,self._left_support)
        # using moment balance about left support
        # B_y*(right_support[0]-left_support[0]) + B_x*(right_support[1]-left_support[1])
= moment_A_external

        moment_B_external = self._sum_moments(x_loads,y_loads,self._right_support)
        # using moment balance about right support
        # A_x*(right_support[1]-left_support[1]) - A_y*(right_support[0]-left_support[0])
= moment_hinge_external

        # solving the above 4 equations to get the reaction forces
        A_x,A_y,B_x,B_y = symbols('A_x A_y B_x B_y')
        eq1 = Eq(A_x + B_x+sum_x,0)
        eq2 = Eq(A_y + B_y+sum_y,0)
        eq3 = Eq(B_y*(self._right_support[0]-self._left_support[0]) +\
                B_x*(self._right_support[1]-self._left_support[1]), moment_A_external)
        eq4 = Eq(A_x*(self._right_support[1]-self._left_support[1]) -\
                A_y*(self._right_support[0]-self._left_support[0]), moment_B_external)
        reaction_forces.append(solve((eq1,eq2,eq3,eq4), (A_x,A_y,B_x,B_y)))

        # Step3: we calculate the shape of the arch
        # we have three points on the structure and considering a parabolic shape we can
solve for the coefficients of the parabola
        # using matrix
        y = self.calculate_shape()
        x = symbols('x')
        tangent = diff(y,x)
        self._angle  = atan(tangent)
```

To calculate the bending moment and shear force at various points

```
   def get_forces(self,x):
        # now to calculate the shear forces and the bending moments at different points
of the arch we consider
        # only the forces to the left of the point and calculate the shear force and
bending moment at that point
        loads = self.give_loads_left(x)
        x_loads,y_loads = self.to_component(loads)
        sum_x,sum_y = self._sum_forces(x_loads,y_loads)

        # using force balance in vertical direction A_y + F_y + sum_y = 0
        # using force balance in horizontal direction A_x + F_x + sum_x = 0
        theta = self._angle.subs(x,x)

        A_x,A_y,F_x,F_y = symbols('A_x A_y F_x F_y')
        eq1 = Eq(A_x + F_x+sum_x,0)
        eq2 = Eq(A_y + F_y+sum_y,0)

        forces = []
        forces.append(solve((eq1,eq2), (A_x,A_y,F_x,F_y)))

        # using equation F_x = N_D cos(thetha) + V_D sin(theta) and
        # F_y = N_D sin(theta) - V_D cos(theta) we can solve for N_D and V_D
        N_D,V_D = symbols('N_D V_D')
        eq3 = Eq(N_D*cos(theta) + V_D*sin(theta),forces[0][F_x])
        eq4 = Eq(N_D*sin(theta) - V_D*cos(theta),forces[0][F_y])
        forces.append(solve((eq3,eq4), (N_D,V_D)))

        # we can use the moment balance about the point to find out the bending moment
        # using shape we calculate the y coordinate of the point
        h = self._shape.subs(x,x)
        # using moment balance about the point
        # M = -A_y*(self._left_support[0]-x) + A_x*(h-self._left_support[1]) +
moment_external = 0

        moment_external = self._sum_moments(x_loads,y_loads,x)
        eq5 = Eq(-forces[0][A_y]*(self._left_support[0]-x) +
forces[0][A_x]*(h-self._left_support[1])+moment_external,0)
        forces.append(solve(eq5,forces[0][A_y],forces[0][A_x]))
        return forces
```

The sign convention used and the actual implementation of the functionality would be finalised after discussion with the mentors to ensure optimised solving and standardised sign conventions. We can use Singularity functions and integrations in the solver which I would like to implement taking inspiration from its usage in Beam class.

Similar to the to_parabola method shown in Phase 1, the **calculate_shape** method can be implemented to get the equation of a parabolic arch.

give_loads_left is used to get all the loads to the left since calculating the bending moment and shear forces only require the forces to either the left or the right side of the point.

to_components will be used to convert applied loads into loads in the x and y directions to ease the calculations. The method would store the components in x direction and y direction for each force along with the coordinates along with the magnitudes which may be symbolic or numeric and ease use in later calculations.

For calculation purposes, we would need to find the coordinates of centroid of forces which can be found by using formula:-

$$\text{Centroid} = \frac{\int_a^b x \cdot f(x)\,dx}{\int_a^b f(x)\,dx}$$

Which can be used in calculation for moments easily.

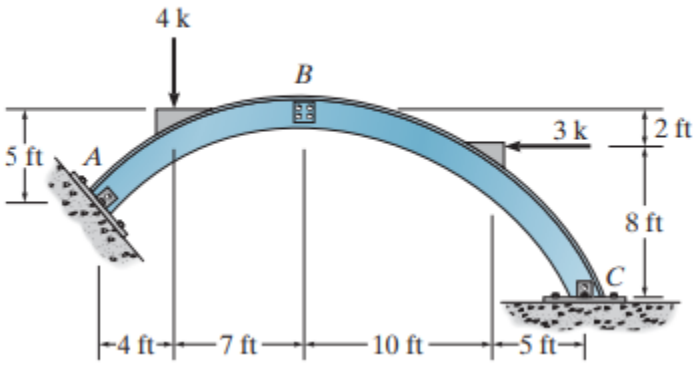
```
numerator = integrate(x*f_x, (x,start_x,end_x))
denominator = integrate(x, (x,start_x,end_x))
centroid = numerator/denominator
```

This centroid value can be made to convert the distributed load into point load for certain calculations.

Phase 3

Overview

This phase would focus upon the implementation of the various plots and figures related to the Arch class. The Arch is a significant structure from an engineer's point of view and hence various plots are needed for a better understanding of load distribution and material that would be required to build the arch. I would like to implement a draw method similar to the one defined in other classes and like the one described in the Phase1 of this proposal will then be implemented for visualisation purposes and intuition. This would allow the users to plot the arch, the defined loads, reactions forces, position of supports etc.



Detailed diagrams like this would be possible with the implementation of the method.

I would also like to plot the bending moment, axial force and shear generated at every x coordinate.

Implementing Draw for the Arches

We would sort the point loads by their x coordinates and at each x coordinate and some other points between them (depending on the accuracy needed) find the value of these forces and plot them or we could convert the forces and moments into a piecewise function and plot it. We could also convert the forces into a function (using Dirac delta functions for point loads and a known function for distributed loads) and use them to obtain the various results (Details would be finalised later).This method would utilise the current sympy plotting module to maintain uniformity with the rest of the module.

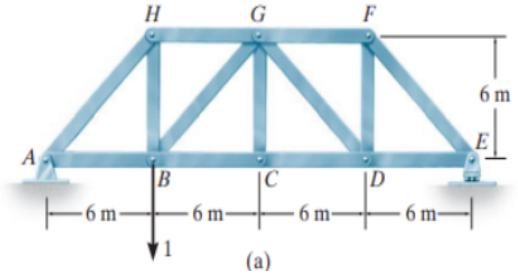
Extended Phase

Overview

Influence line diagrams are graphical representations used in structural engineering to visualize the effect of a moving load on a structure. They are particularly useful for determining the maximum response (such as shear force, bending moment, or deflection) at various points along a structure caused by the movement of a load.

Influence line diagrams typically show the response function as a function of the position of the load along the structure. The diagram illustrates how the response varies as the load moves along the structure, providing engineers with valuable insight into the critical points and maximum values of the response.

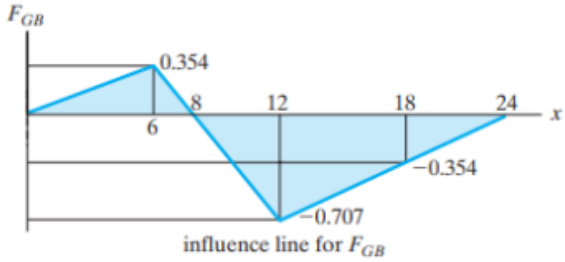
I would like to develop functions to calculate and plot the influence line diagrams for the Truss class. Consider the example of the given truss:



(a)

Consider the influence line for the force in member GB. The results are tabulated below after computation. The plot for the IFL is also given below.

x	F_{GB}
0	0
6	0.354
12	−0.707
18	−0.354
24	0



influence line for F_{GB}

The implementation of this feature would take inspiration from previous works in the module in the Beam class. For this purpose the following sequence can be followed:

1. The user first creates a truss.
2. Nodes at the bottom joints identified.
3. Unit load is applied to these joints and forces caclulated for the required member.
4. These calculatted values can later be used to plot the diagram.

Implementation

```
def _ild_plot(self):

    base_nodes = []
    force_member = []

    for node in self._nodes:
        if(node is node_with_min_x):
            base_nodes.append(node)

    for node in base_nodes:
        self.apply_load(node,1,270)
        t.solve()
        # We store the values
        force_member.append(force_developed)
        #then remove the load
        self.remove_load(node,1,270)
```

We will use the potting module in Sympy to plot the diagram. Final implementation details will be decided after discussion with the mentors.

Timeline

- ## Community Bonding Period(May 1 -May 26)

  During this period, I will be discussing and finalising the details related to the project after discussion with the mentors and making the suggested changes, I will also be exploring the community, and obtaining more knowledge on the tools and documentation. If the discussions related to the project are completed before the end of the community bonding period ends, I would like to start the implementation during this time as suggested by the mentors.

- ## Phase 1(May 27 - Jun 13, About 2 weeks)

  The first phase of the project would be implemented during this period. I plan to achieve the goal of implementing the draw method and plot for tension in different segments of cable. The time is required to implement the draw method and make some changes to the solver to obtain some additional information required for the plots.

- ## Phase 2(Jun 14 - July 22, About 5 weeks)

  1. **Defining the class (June 14 - June 30, About 2 weeks)**
     The Arch class will start the initial defining and implementation period. Methods will be added to allow users to easily add supports, loads and define the type. Previous work in the Beam, Truss and Cable class would be referenced to implement this class as well.

  2. **Implementing the Solver (July 1 - July 22, 3 weeks)**
     In this period, I would be working on developing the solver for the arch class for both the tied arches and the hinged arches. Extensive testing would also be done to ensure the correctness of the solver. This part of the project would consume most of the time.
     I will be working on the implementation of the solver by the time of midterm evaluation and have completed phase 1 of my project.

- ## Phase 3(July 23 - Aug 10, About 2 weeks)

  1. **Implementing the ‘draw method’ (July 23 - Aug 10, About 2 weeks)**
     During this time, I would focus on implementing the ‘draw’ method for the Arch class, similar to the one in the Beam, Truss and Cable class. This would require 2 weeks since it would require the use of plotting, geometry and manual checking.

- ## Final Week(Aug 19 - Aug 26)

  This period would largely focus on making minor changes, finalizing the work and getting it merged.

  One week is left in the timeline(Aug 11 - Aug 18) which would act as a buffer if work in any Phase (1,2 or 3) takes longer than expected or utilised for implementing the extended goal if work in other phases is complete.

Other Commitments During GSoC

During the summer, I have no other obligations apart from my End-semester exams scheduled for the first to second week of May, coinciding with the Community Bonding Period. I plan to compensate for any lost time during my summer vacation weeks by dedicating extra hours to the project. Following the conclusion of my vacation, there will be a relatively light workload for about a month after classes resume, allowing me to commit approximately 25-30 hours per week to the project on average.

Should any circumstances arise that impede the project's progress, I will promptly inform my mentors and make up for it by increasing my workload. In the event of unforeseen obstacles, I am prepared to allocate additional time to the project in the subsequent weeks.

Post GSoC

After GSoC, I would try to complete the extended goal of the project if it remains and keep contributing to Sympy by addressing issues that may arise in Continuum mechanics and other modules, raising PRs and doing code reviews. I would also like to guide the new contributors who wish to contribute to Sympy and the Open Source Community as I received a lot of help from members and contributors in the organization. I would also like to extend the functionality of previous classes by adding methods to plot diagrams related to bending moment and shear forces.

I would like to thank Oscar Benjamin(@oscarbenjamin), Christopher Smith(@smichr), Jason Moore (@moorepants), Aaron Meurer(@asmeurer), Advait Pote(@Advait Pote), Ishan Pandhare(@ishanned), S.Y. Lee (@sylee957) and other members and contributors in the community for their continuous support, allowing me to contribute to Sympy and learn a lot in the process. I am grateful to receive help from them in times of need.

References

1. **Engineering Mechanics** by Irving H. Shames
2. **Mechanics of Materia**l by Dr. B.C. Punmia
3. **Structural Analysis** by R.C. Hibbeler
4. Advait Pote's GSoC proposal
5. Ishan Pandhare's GSoC proposal
6. Sympy Documentation for Continuum_mechanics