Google Summer of Code 2022 Application: Extending the Continuum Mechanics Module

-Advait Pote

Table Of Contents:

About me:

Personal Information:

- **Name**: Advait Pote
- **University**: IIT Bombay
- **Major**: Mechanical Engineering
- **Minor**: Computer Science and Engineering
- **Email**: apote2050@gmail.com
- **GitHub**: AdvaitPote
- **Timezone**: IST (UTC +5:30)

Basic Introduction:

My name is Advait Pote. I am currently in my 2nd year of undergraduate studies at IIT Bombay, India. I am pursuing a major in Mechanical Engineering and a minor in Computer Science and Engineering. Regarded as one of the best engineering institutions in India, IIT Bombay has always been a premier institute holding a ranking of 3 in the country.

I have always been interested in Mathematics since middle school. As time has progressed, my inclination toward Mathematics has only increased. Some of the relevant courses taken by me in my college include

- Calculus I
- Calculus II
- Computer Programming and Utilization
- Linear Algebra
- Differential Equations
- Logic for Computer Science
- Data Structures and Algorithms (The course ends at the end of April 2022)
- Introduction to Numerical Analysis (The course ends at the end of April 2022)

Programming details:

I work on the Windows Operating System with Visual Studio Code as my primary editor. It is easy to use and I have been using it since my early programming days. Coming with many features like Intellisense, VS Code is my first choice editor.

About my programming experience, I was introduced to programming in Middle School where we learned some features of the C Language. However, my first serious

programming venture started towards the end of High School when I started learning C again and introduced myself to python. In my first semester in college, we had a mandatory Introductory Programming Course where we were taught C++. The second semester had a course covering Data Analysis and Interpretation which required knowledge of Python, which enabled me to brush up my Python.

I have been working on git and GitHub for quite a few months now since I started contributing to SymPy.

SymPy has impressed me a lot since I was introduced to it in Autumn 2021. The vast amount of mathematical features offered to the user enables him to solve more and more complex problems with ease. I have used SymPy to solve complex problems and equations in the core courses related to my major as well.

My favorite feature in SymPy has to be the 'integrate' operation. I have had to solve many integrals from my high school to my college and I am very aware of how challenging it is. There are a variety of methods of integration already integrated into the feature which make solving an integral given to us extremely easy.

```
>>> from sympy import *
>>> x, y, z = symbols('x y z')
>>> integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo))
π
```

Contributions to SymPy:

After being introduced to SymPy in October 2021, I made my first contribution (closed PR) in the same month and my first successful contribution (merged PR) was in December. I have started contributing to SymPy by working on fixing issues and this has really helped me get a little experience with the softwares involved and the overall skills required. These are the contributions I have made to SymPy.

- **(Merged)** Cheatsheet: Printing expressions updated to Python 3 #23189
- **(Merged)** physics/control: label for Frequency axis in bode_plot modified #23045
- **(Merged)** printing/pycode: Min and Max added to _known_functions #22914

- **(Merged)** booleans documentation updated #22683
- **(Merged)** calculus tutorial updated #22641
- **(Closed)** top-level guides reordered #22632
- **(Closed)** Files in diophantine modified #22315
- **(Open)** New method to calculate nearest distance between two lines #22776

The Project:

Overview:

With this project, I intend to extend the Continuum Mechanics module in SymPy. The current version of the module which one can use to solve problems in the area of continuum mechanics has been improving at a really impressive rate with the beam module being able to do a variety of useful functions returning the bending moment, shear force as well deflection and slope of the beam under force.

I would first like to extend the geometry module with a few utilities which would in turn help play with the beam module even more explorable as they would help integrate numerous cross-sections. One of the utilities is having a group of functions that would help the user make composite functions using boolean operations on basic shapes. A classic example is the following where the union of two shapes is taken such that a newer and more composite shape is formed.

Next is implementing torsion in the beam module. Being a very important process in the field of Solid Mechanics, Torsion is defined as the twisting of an object due to an applied torque. It has varying applications in the Engineering world and is used in a wide range of fields.

Finally, another structure planned to be implemented is a Truss. A truss is a rigid structure that is an assembly of members connected by nodes that can be joints. The main motive of the Truss class would be to initialize a truss of any kind we want with the appropriate members, nodes, external forces, moments, etc, and subsequently, using

the Method Of Joints, have an Equilibrium Analysis on each node to calculate the reactions and internal forces for the appropriate nodes and members respectively.

Implementation Plan:

I have divided this project into different phases, particularly four which include the Community Bonding Period and the three Coding Phases. Each phase has an aim and a particular set of objectives to fulfill:

In phase I, I plan on extending the geometry module. This will be done with the functions mentioned above.

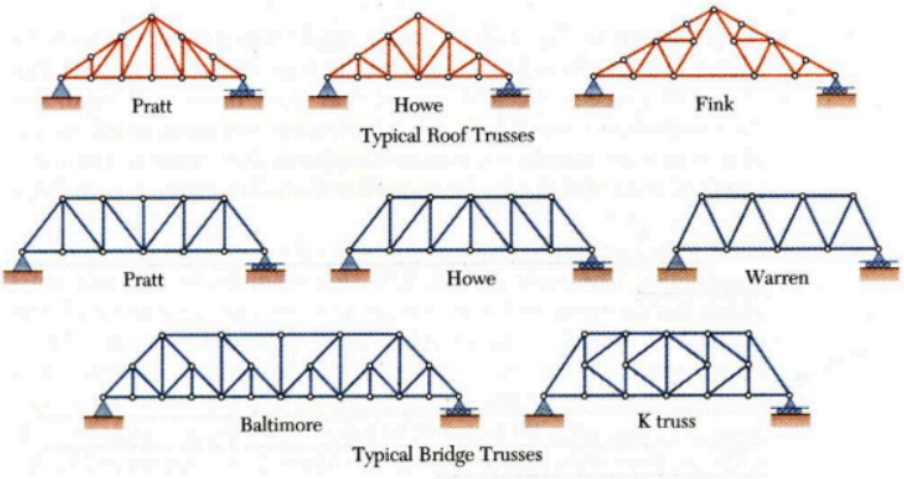
In phase II, implementing torsion in the beam is what is planned. The appropriate functions are tentatively going to be placed in the ‘beam’ class at first and then the ‘beam3d’ class.

In phase III, finally, I plan to implement the Truss class.

The details for each of the mentioned phases are mentioned in the latter part of this proposal.

Phase I:

A Truss is an assembly of members such as beams, connected by nodes, that create a rigid structure. In engineering, a truss is a structure that consists of two-force members only. The members act such that the entire structure behaves as a single object. Some examples of trusses are



Trusses are extremely important in engineering applications and can be seen in numerous real-world applications like bridges.

There are two methods for the analysis of Trusses:

- Method of Joints
- Method of Sections

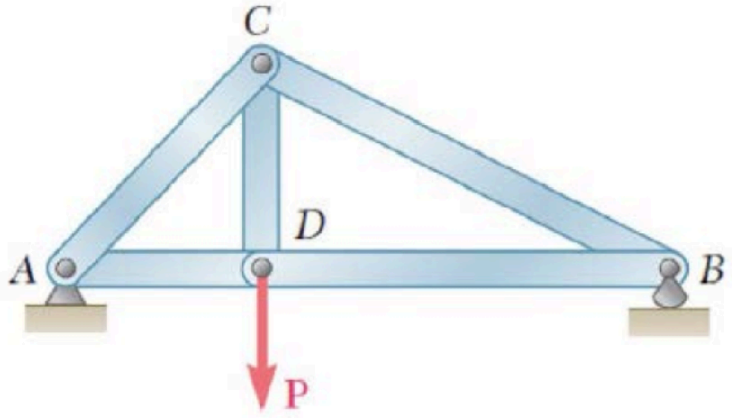
We would be using the former.

Method of Joints simply applies force equilibrium equations in the x-direction and in the y-direction (for 2 dimensions) for each node. Hence, we simply get 2*n equations where j is the number of nodes in the structure. These 2*n equations are solved and the reaction forces on the nodes, which are supported by pinned (both horizontal and vertical components) and roller (only horizontal component exists) joints, and the internal forces in each beam (two-force members) are obtained. Sometimes moment equilibrium, about z, equations about pinned joints may be needed as well. This would add i equations to 2*n equations where i is the number of pinned joints. Now for the system to be solvable,

$$m + r \leq 2j+i$$

where m is the number of members and r is the number of pinned/roller joints in the structure.

The methodology and the subsequent implementation that is planned to be used can be shown by an example as follows:



In the above truss, A is a pinned joint, B is a roller joint and P is applied as an external load at D.

The number of members, and hence internal forces, is 5 and the number of reaction components is 2 at A and 1 at B. Therefore, we have a total of 8 variables. Now, the number of equations that are available is 2*4, that is, 8. Hence, the given system is solvable.

The equations are as follows:

A: $R_{A,x} + F_{AD} + F_{AC}\cos\theta = 0$
A: $R_{A,y} + F_{AC}\sin\theta = 0$

B: $-F_{BD} - F_{BC}\cos\phi = 0$
B: $F_{BC}\sin\phi + R_{B,y} = 0$

C: $-F_{CA}\cos\theta + F_{CB}\cos\phi = 0$
C: $-F_{CA}\sin\theta - F_{CD} - F_{CB}\sin\phi = 0$

D: $-F_{DA} + F_{DB} = 0$
D: $-P + F_{DC} = 0$

The given system of equations, from our previous knowledge of Linear Algebra, can be reduced to a Matrix multiplication form:

$$C \cdot F = R$$
$$\Rightarrow F = C^{-1} \cdot R$$

Where C is,

$$\begin{bmatrix} 1 & 0 & 0 & \cos\theta & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \sin\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\cos\phi & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \sin\phi & 0 \\ 0 & 0 & 0 & -\cos\theta & 0 & \cos\phi & 0 & 0 \\ 0 & 0 & 0 & -\sin\theta & 0 & -\sin\phi & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

F is the transpose of

$$\begin{bmatrix} R_{A,x} & R_{A,y} & R_{B,y} & F_{AC} & F_{AD} & F_{BC} & F_{BD} & F_{CD} \end{bmatrix}$$

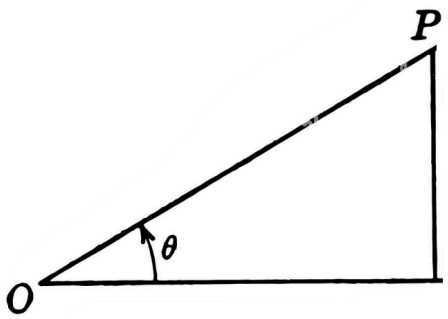
And finally, R is the transpose of,

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & P \end{bmatrix}$$

The subsequent solution for the column matrix F is our solution.

The important component for finding F is the C i.e. the Coefficient matrix as we would call it. Below is the procedure to calculate the same.

Assume a member k in between nodes i and j (O and P respectively in the diagram) at an angle θ with the horizontal.



Hence, the member k would be having the internal force F_{ij} out of node i and F_{ji} out of node j. This internal force would exist for nodes i and j only and hence would be involved in only 4 equations. This would mean that in C, for column k, $C_{n,k}$ is zero for all n except for

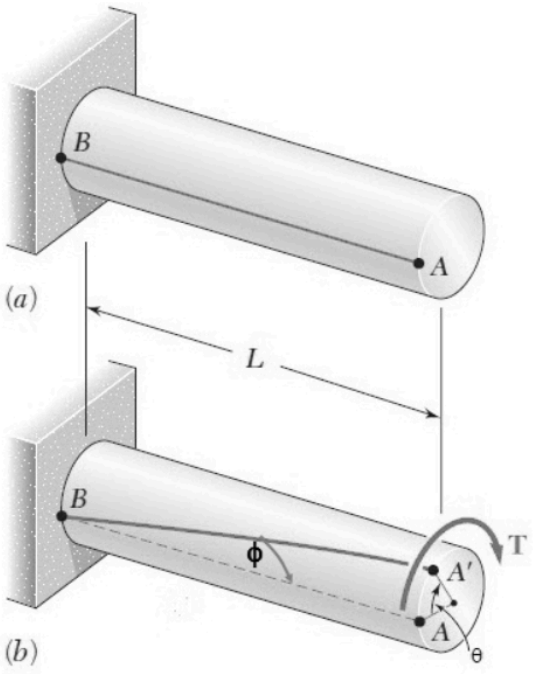
$$C_{2i-1,\,k} = cos\,\theta$$
$$C_{2i,\,k} = sin\,\theta$$
$$C_{2j-1,\,k} = -\,cos\,\theta$$
$$C_{2j,\,k} = -\,sin\,\theta$$

Iterating the procedure for all of its members, C is obtained. Now the external load matrix L is simply the collection of the external loads on its nodes. Thus if an external load P acts at an angle ϕ with the horizontal on a node j, then for L, $L_{2j-1} = -\,Pcos\,\phi$ and $L_{2j} = -\,Psin\,\phi$. The negative sign is because the external load terms are moved from the LHS to the RHS.

```
>>> from sympy import *
>>> init_printing
>>> P = symbols('P')
>>> t = Truss()
>>> t.add_node("node_1", 0, 0) # node 1 is assumed to be at the origin
>>> t.add_node("node_2", 3, 0)
>>> t.add_node("node_3", 1, 1)
>>> t.add_node("node_4", 1, 0)
>>> t.add_member("member_1", start_node="node_1", end_node= "node_3") #
starting and ending nodes to be provided
>>> t.add_member("member_2", "node_1", "node_4")
>>> t.add_member("member_3", "node_2", "node_3")
>>> t.add_member("member_4", "node_2", "node_4")
>>> t.add_member("member_5", "node_3", "node_4")
>>> t.add_support("node_1", type="fixed")
>>> t.add_support("node_2", "roller")
>>> t.add_load("node_4", magnitude=P, angle=-90)
>>> t.solve() # solves for internal forces and reactions at appropriate nodes
>>> t.reactions("node_1")
2⋅P
───
 3
>>> t.internal_forces("member_1")
-2⋅√2⋅P
──────
   3
```

Phase II:

In the field of Solid Mechanics, Torsion is defined as the process of twisting of an object due to an applied torque. When a Torque is applied on a shaft parallel to the shaft or has a non-zero component in the direction parallel to the shaft, the cross-section of the shaft is rotated by a particular angle. This angle of rotation is different for each cross-section and it depends on the distance of the cross-section from the point of application of the torque.



Torsion is usually discussed on shafts with a circular shaft. However, the same can be done for non-circular cross-sections as well. The plan is to start with a circular cross-section first. In a circular cross-section, the analysis is quite easy as the relation between the angle of rotation of a cross-section and the distance of the cross-section from the wall, as shown above, is linear.

$$\theta(z) = \alpha z$$

where θ is the rotation of the cross-section (radians) and α is the unit angle of twist per unit length (radians/m).

The same relation is not linear and hence, not as simple for non-circular shafts and involves something called a Prantdl-Stress Function (φ).

Integrating this into SymPy is indeed a very challenging task and hence, instead of having the whole Prandtl-Stress function, the plan is to perform the analysis on some commonly used non-circular cross-sections, like a square, rectangle, etc, beforehand and then use the derived results and formulae directly.

For a simple shaft of circular cross-section and of length L, shear modulus G, and polar moment of inertia J, if a torque T is applied at the end i.e at z = L, then θ varies as

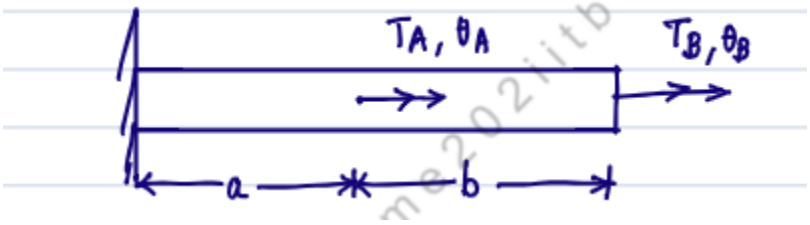
$$\theta(z) = \frac{T}{GJ}z$$

The quantities that I would want to obtain from the torsion module are given an arrangement of a shaft(s) with their geometrical and mechanical properties, torques at different positions and the appropriate boundary conditions:

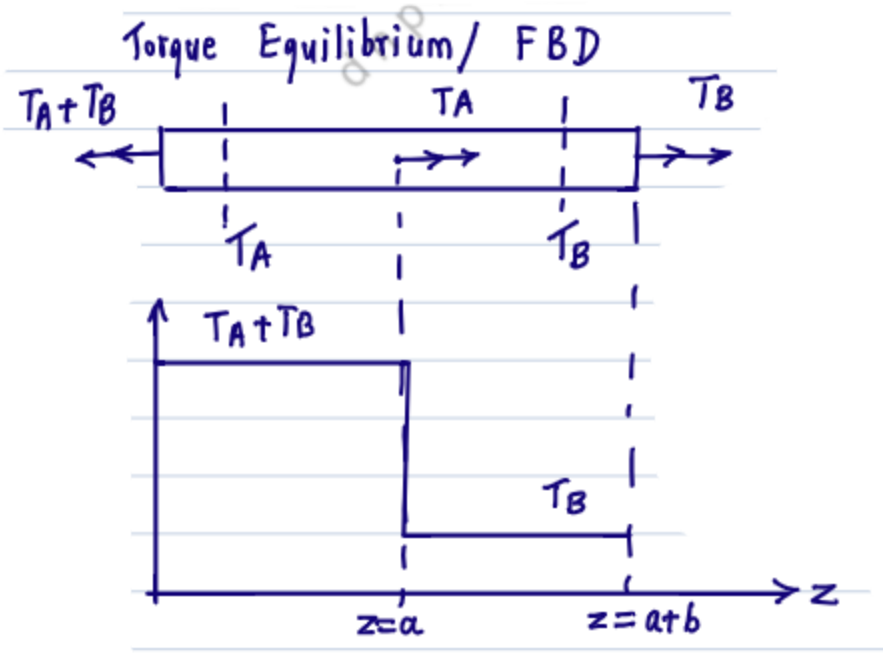
- θ(z) for a distance z from the support.
- Maximum angle of twist in the shaft.
- Maximum Shear Stress in the shaft

and maybe a few more as well.

Another thing is that the analysis being done is for multiple torques along the shaft. An example is shown below.



For this shaft, a Torque diagram can be drawn appropriately,



As per this torque diagram, the angle of twist at the length z=a+b is,

$$\theta(z = a + b) = \frac{(T_A + T_B)a}{GJ} + \frac{T_B b}{GJ}$$

Implementation:

When we start with a given problem, we need a shaft to work with. There are two choices that we have: Either implement the same functions in the same 'beam' class or initialize a new class called 'shaft'. The reason why the second choice can be considered is that I thought the beam module should strictly be kept for bending. However, keeping both torsion and bending in the same beam module would be good for simplicity and also can be promising for many future ventures like combined loading.

We can take the second choice and create a new class called 'shaft':

```
"""
This module can be used to solve 2D problems related to torsion in a shaft due to
multiple torques along its length.
"""
Class Shaft:
    def __init__(self, length, cross_section, polar_moi, shear_modulus,
variable=Symbol('z')):
        # initializes the class with the geometrical parameters of the
```

```
        # shaft and also the mechanical properties like shear modulus
        # The cross_section accepts a Geometrical Entity as a parameter
        # polar_moi is the polar moment of inertia (J) for the cross section
```

The torque would be applied using the method ‘apply torque’. This would take arguments of the magnitude of the torque and the distance from the axis where the torque is applied, which would be initialized to the length of the shaft itself. Hence, a particular amount of torque when applied would be assumed to be at the end of the shaft itself. This can be reserved for a future discussion as to if this initialization is really necessary.

Needless to say, one can have multiple torques along the shaft at different positions. As mentioned previously, Torque diagrams like the above can be used for solving. From the Torque Diagram, for a point z_0 away from the fixed support, for all z < z_0, the torque diagram tells us the corresponding torque experienced at that point. Hence,

$$\theta(z = z_0) = \frac{1}{GJ}\int_0^{z_0} T(z)dz$$

where T(z) is the Torque Diagram.

```
    def apply_torque(self, value, point):
        # applies a torque in the shaft at the point given. Hence, multiple
        # torques can be applied on a shaft at different positions along the
        # shaft
```

Below is how the user would initialize a circular shaft with length L. The torque T is applied at z = L/2. The angle of twist at z = L/4 and z = L are calculated and displayed.

```
>>> from sympy import *
>>> init_printing
>>> G, J, T = symbols('G, J, T')
>>> s = shaft(L, circle, J, G)
>>> s.apply_torque(T, L/2)
>>> s.twist(L/4)
 T·L
─────
4·G·J
```

```
>>> s.twist(L)
 T.L
─────
2·G·J
```

Subsequently plotting functions can also be added to the same.

Phase III:

In this phase, I plan to add some functions to the geometry module of SymPy. The current beam module excepts different cross-sections making it more flexible for different kinds of beams with different properties. One can have a circular beam and a square beam as well. In order to define varied cross-sections, the geometry module needs to be used. Adding appropriate functions to the geometry module would thus help in making the continuum mechanics module much more flexible.

The functions to be added are:

- Calculating the first moment of area for the ‘Ellipse’ class.
- Making composite shapes using Boolean operations on basic shapes in the geometry module.

The first moment of area for an ellipse is defined as

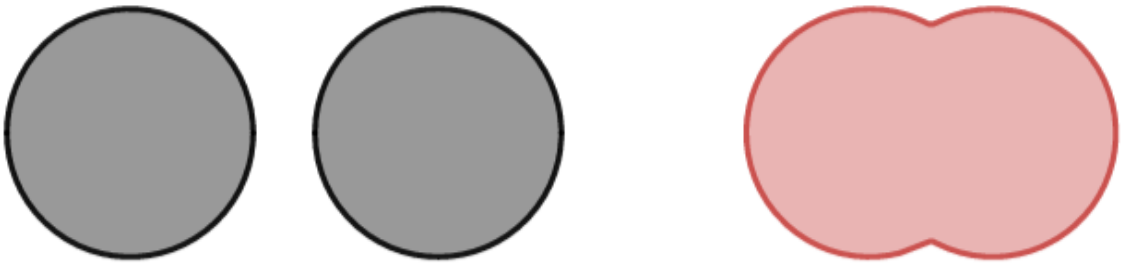
$$Q_x = \int y dA \qquad\qquad Q_y = \int x dA$$

The first moment of area is zero about the centroid. Therefore, just like in polygons, here it is calculated for an area, above or below a certain point of interest, that makes up a smaller portion of the polygon. This area is bounded by the point of interest and the extreme end (top or bottom) of the polygon. The first moment for this area is then determined about the centroidal axis of the initial polygon.

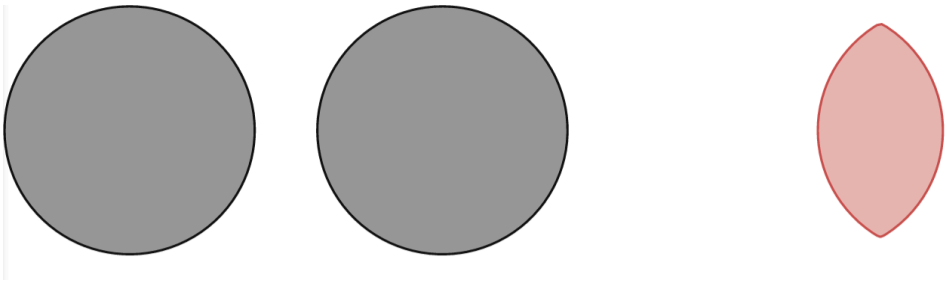
```
def first_moment_of_area(self, point=None):
    # returns the first moment of area of an ellipse about the point
    # given or its centroid itself. It is calculated for an area,
    # above or below a certain point of interest, that makes up a
```

```
    # smaller portion of the polygon. This area is bounded by
    # the point of interest and the extreme end (top or bottom) of
    # the polygon.
```

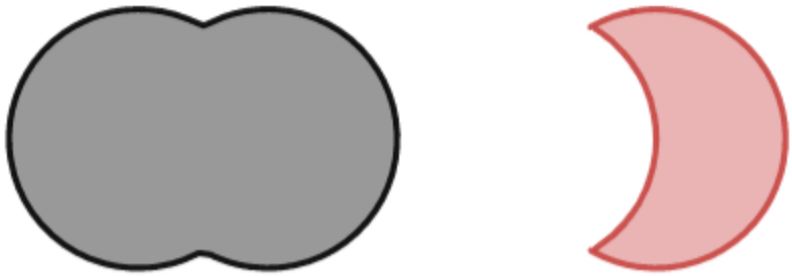
The latter function is a way to make composite shapes as below by simply using boolean operations on two simple shapes. Here, the union of two circles was taken to give an uncommon composite shape.



Similarly, the intersection operation on both the shapes is as follows,



Lastly, the negation operation can be carried out as shown. It is to be noted that the negation is carried out on the left circle and the two circles combined are our entire universe itself.



For both polygons and ellipses, the method for intersection already exists.

Currently, the functions will be added for polygons and the resultant shapes can be demonstrated with ease as the resultant will be a polygon as well. The problems arise when ellipses are involved as the shapes demonstrated above in the circles' example cannot be shown by a class in SymPy and there needs to be a way to define an arbitrary shape as above.

The given operations being called are as shown.

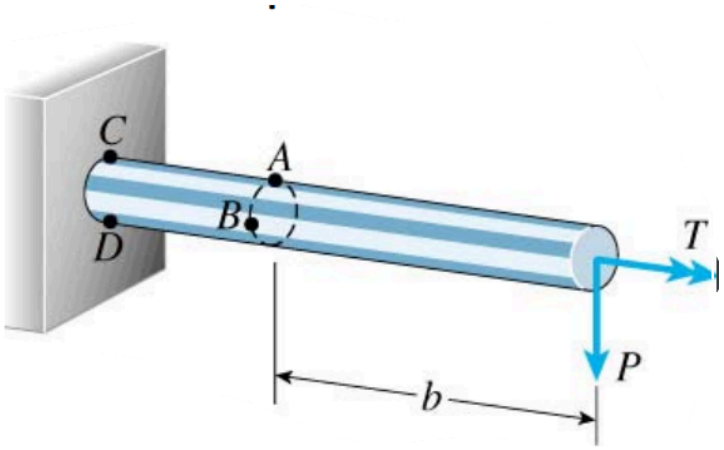
```
def bool_union(self, p):
    # gives the union of the polygon with the geometric shape p. The
    # resultant is a polygon as well.

def bool_negation(self, universe):
    # gives the negation of the polygon from the given universe. The
    # resultant is a polygon as well.
```

```
>>> from sympy import *
>>> p1 = Polygon((0, 0), (0, 2), (2, 2), (2, 0))
>>> p2 = Polygon((1.5, 0), (1.5, 2), (3.5, 2), (3.5, 0))
>>> p3 = p1.bool_union(p=p2)
>>> p3
Polygon(Point2D(0, 0), Point2D(0, 2), Point2D(3.5, 2), Point2D(3.5, 0))
>>> p4 = Polygon((1, 0), (1, 1), (2, 1), (2, 2))
>>> p5 = p4.bool_negation(universe=p1)
>>> p5
Polygon(Point2D(0, 0), Point2D(0, 2), Point2D(2, 2), Point2D(2, 1), Point2D(1,
1), Point2D(1, 0))
```

Stretch Goals:

In case my work finishes before time, one thing I am looking to is the column class. Pull requests with this class are open and not merged. After they are merged, I would like to implement some features for the ‘column’ class. One is determining the critical load for the pinned-fixed end condition of a column.

Another thing is implementing combined loading. If torsion and bending are implemented in the same beam module, then we can have both bending loads and torsional loads on a beam and the subsequent analysis would be interesting. This can be done in both 2D and 3D.



Timeline:

Community Bonding Period (May 20 - June 12):

In this period, I will be meeting up with my mentors and discussing my project. I will have a good look at the codebase, especially the modules that are under my project’s concern. Having been contributing to SymPy for a long time, this will not take up much time for me. I will still be actively contributing this time by solving existing issues in the continuum mechanics module and other modules of SymPy as well. If my discussion with mentors gets completed before time, I would like to start work on my project in this time itself, if possible.

Phase I:

Week 1 - Week 2 (June 13 - June 26):

- Get started with the implementation of the Truss class.
- Implement the above common functions ‘add_node’ and ‘add_member’ for the class.
- Start work on the ‘solve’ method.

Week 3 - Week 4 (June 27 - July 10):

- Continue and finish the ‘solve’ method along with testing.
- Finish documentation and add test cases.

Phase II:

Week 6 - Week 7 (July 11 - July 24):

- Implement the ‘shaft’ class with the required methods.
- Work on implementing cross-sections for the shaft module (circle, ellipse, polygon, etc.)

Week 8 - Week 10 (July 25 - August 14):

- Start the work for the ‘twist’ method.
- Finish the documentation for the ‘Truss’ class.
- Complete the testing of the above methods and add appropriate test cases.

Phase III:

Week 11 - Week 13 (August 15 - September 4):

- Implement the algorithm for calculating the first moment of area for the ellipses class.
- Implement the algorithm for ‘bool_union’ and ‘bool_negation’ methods in the geometry module.
- Finish the documentation and add relevant test cases.

Final Week (September 5 - September 12):

- Finish any pending work and try to get any open PRs successfully merged.
- Discussion with mentors regarding any future improvements and about the submission.

- Drafting the final report comprising and summarizing work done in the summer.

Commitments during the GSoC period:

I don't have any major commitments this summer and I will be able to devote most of my time to the project that I plan to undertake this summer. After my summer vacations end at the end of July, my fifth-semester classes will start. However, there is never any major load in the month after classes start. Hence, all of August will be the same and so will my devotion to the project. I promise to inform beforehand in case of any problems or inconveniences that might occur for me. If that happens, I promise to make up for it by working extra hours in the later days.

Conclusion:

For the period after the final submission and evaluation, I plan to:

- Work on the continuum mechanics module to ensure all my changes are merged successfully and the required features are implemented.
- Keep discussing with my mentors and all the members & active contributors of SymPy to see how this module can be further improved.
- Keep contributing to SymPy by raising issues and opening & reviewing PRs.

I would like to thank all the members and contributors of this organization who have helped me to this stage, from getting started to contributing to the application process. It has been a rather smooth journey for me and I have learned a lot during the process thanks to this Wonderful Community.

References:

- **Mechanics of Materials** by *R.C. Hibbeler*
- **CE-102 Engineering Mechanics** - *IIT Bombay*
- NPTEL Torsional Load on Shaft
- NPTEL Analysis of Trusses
- Prakhar Saxena's proposal
- Ishan Joshi's proposal