Continuum Mechanics: Creating a Rich Beam Solver and Extending Continuum Mechanics Module

Prakhar Saxena

Abstract

The continuum mechanics of Sympy is in its early stage of development. During this summer, I plan on adding the following features:

- Adding plotting functions in Beam 3D class for
    - Shear Force
    - Bending Moment
    - Slope
    - Deflection
    - All loading results
- Adding support for calculating equations of Influence Line Diagrams for
    - Reactions
    - Shear Force
    - Bending Moment
- Adding functions to plot the above Influence Line Diagrams
- Adding functions for max deflection, bending moment, and shear in Beam 3D class

Contents

1. About me

1.1 Basic Information

NAME - Prakhar Saxena
EMAIL - prakharsaxena.civ18@iitbhu.ac.in
GITHUB - Psycho-Pirate
UNIVERSITY - Indian Institute of Technology (BHU), Varanasi

1.2 Personal Background

I am a third-year undergraduate student pursuing a dual degree (B.tech + M.tech) in ***Civil Engineering*** at the **Indian Institute of Technology (BHU), Varanasi**. I was introduced to programming about four years ago. I have studied linear and abstract algebra, mathematical methods, and probability and statistics.
I have also studied Engineering Mechanics, Mechanics of Solids, and Structural Mechanics as part of my course structure.

1.3 Programming Details

- I work on Linux Mint operating system with sublime text as my primary text editor because of its features and UI.
- I have been programming in C/C++ for about 4 years. I have been using python for over two years.
- I have added my previous projects to my Github profile. These include a gun shop management system in C++ which uses file handling and object-oriented programming. There is also a sudoku solver in python which solves a given sudoku puzzle.
- I am familiar with using git as a version control system. I have been using and contributing to Sympy for more than a year now and was fascinated by the vast array of functionalities provided by Sympy. I have used the calculus

and matrices modules a lot for my academic purposes as well. Finding Fourier transformation of a function using Sympy is my favorite feature.

```
>>> from sympy import fourier_transform, exp
>>> from sympy.abc import x, k
>>> fourier_transform(exp(-x**2), x, k)
sqrt(pi)*exp(-pi**2*k**2)

>>> from sympy import inverse_fourier_transform, exp, sqrt, pi
>>> inverse_fourier_transform(sqrt(pi)*exp(-(pi*k)**2), k, x)
exp(-x**2)
```

1.4 My motivation

This project intrigued me as I have studied three courses on continuum mechanics as part of *Civil Engineering.* I have also done practical experiments on load testing. I have also read books like:

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

Therefore I started exploring the continuum mechanics module of Sympy. I think this module is underdeveloped as compared to other modules. The support for beams is really impressive but other structures like truss, cables, arches can be implemented as well. The possibilities are endless.
Last year, due to the covid pandemic, the GSoC slots were reduced. Due to this the Continuum Mechanics module was skipped and math-based modules were prioritized. I hope this module gets the attention it deserves this year.

2. Contributions to Sympy

I have been contributing to Sympy for more than a year now and have made various contributions to different modules. However, I have majorly contributed to the Beam Class of Continuum Mechanics Module. Here are some of my contributions. Almost all of the following PRs are merged.

2.1 PRs:

- 18043 - Improved sorting of unordered arguments in subs (Merged)
- 18836 - Added test cases in the ode module (Merged)
- 18900 - Added functions to calculate shear and axial stress in Beam Class and Beam3D class along with Documentation (Merged)
- 18980 - Fixed examples regarding ramp loads (Merged)
- 19015 - Fixed inconsistencies with draw function (Merged)
- 19060 - Fixed error with Pyglot Plotting (Merged)
- 18886 - Fixed errors in examples and added a note regarding sign convention
- 19126 - Function for I.L.D. for reactions added
- 19203 - Function for plotting shear force in Beam3D added along with Documentation (Merged)

2.2 Issues raised:

- 18964 - Inconsistencies in draw function of beam class
- 18979 - The solve_for_reaction_loads function produces wrong results for triangular loads
- 19059 - Pyglot Plotting returns an error

2.3 Discussions:

- 19069 - Influence Line Diagrams for beams
- 19202 - Plotting functions for Beam3D

2.3 Reviewed/Co-Authored PRs:

- 20431 - Fix equations sign in Beam Module

3. The Project

3.1 Brief Overview

The project aims at ***Creating a rich beam solver and extending the continuum mechanics module.*** The continuum mechanics has a working beam module, but more functionality can be added.
After discussion with Jason Moore(@moorepants) and the past GSoCers of the Beam module and considering the short coding period, I have decided to implement the features we are currently missing first instead of working on a new class(truss, cables, etc.).
In this project I plan on implementing the following:

- Adding plotting functions in Beam 3D class for
    - Shear Force
    - Bending Moment
    - Slope
    - Deflection
    - All loading results
- Adding support for calculating equations of Influence Line Diagrams for
    - Reactions
    - Shear Force
    - Bending Moment

- Adding functions to plot the above Influence Line Diagrams
- Adding functions for max deflection, bending moment, and shear in Beam 3D class

3.2 Execution Plan

Here I will explain how I plan to implement these tasks. I have divided them into four phases which include the community bonding period, and the three coding phases.
In ***Phase-1***, I plan on implementing important functions that we currently lack. These will include Plotting Functions for Beam3D Class.
In ***Phase-2***, I plan on implementing new functions which are important from a civil engineering perspective and will further expand the versatility of the module. These will include Influence Line Diagrams for Reactions, Shear, and Moment.
In ***Phase-3***, I plan on implementing minor functions which are available in the Beam class but haven't been implemented for Beam3D Class.

3.2.1 Community bonding period

During this period I will start discussing details with the mentor regarding the implementation of my project. Since I have been contributing to sympy for more than a year now, I am already familiar with most of the developers. Hence community bonding won't take much time. If all discussions with mentors are completed, I would like to commence to code during the community bonding period itself.

3.2.2 Phase-1

Plotting Functions for Beam3D Class

While going through the Beam module I observed that the Beam3D Class has no plotting functions.
Plotting is really important for the Beam module as it helps us visualize the various distributions across the beam lemgth.
I would like to develop plotting functions for the Beam3D class similar to the ones present in the Beam class.
These would include functions for plotting load, bending moment, slope, and deflection along any given direction.
This is what the Functions would look like

```
    # Function overview
    def plot_shear_force(self, dir="all", subs=None):
        # Plots load applied on the beam along given direction

    def plot_bending_moment(self, dir="all", subs=None):
        # Plots Bending Moment Diagram along given direction

    def plot_slope(self, dir="all", subs=None):
        # Plots slope of elastic curve of deflected beam along given direction

    def plot_deflection(self, dir="all", subs=None):
        # Plots the curvature of deflected beam along given direction

    def plot_loading_results(self, dir="all", subs=None):
        # Plots shear_force", "bending_moment' and deflection due to them
        # on a single graph along given direction
```

Implementation

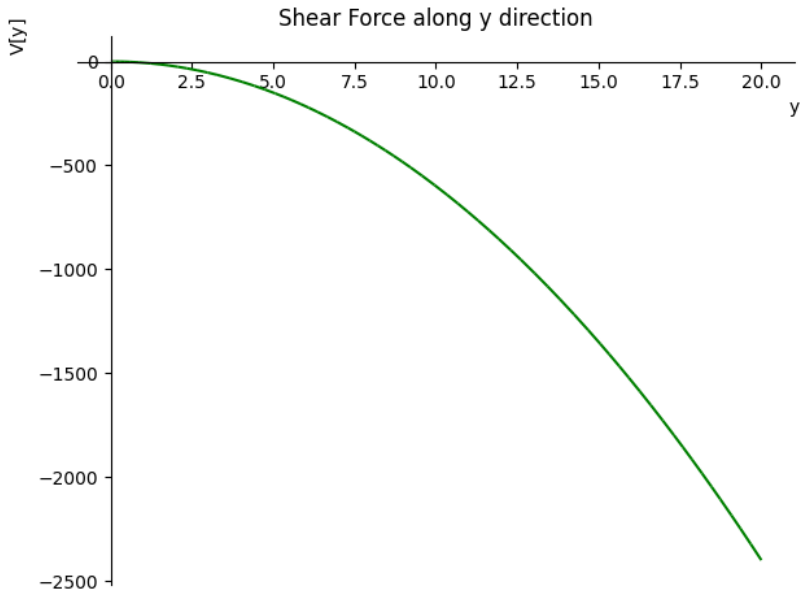
Since we already have plotting functions in the Beam class, the implementation for the Beam3D class isn't difficult.
I have opened a discussion regarding this here.
I have already implemented the Plotting function for Shear Force(S.F.D) in Beam3D in PR#19237 and after discussion with past GSoCers of the beam module, it was successfully merged in the codebase.
The following is a snippet from the **working code** I used to implement this.

```
def plot_shear_force(self, dir="all", subs=None):

    Px = self._plot_shear_force('x')
    Py = self._plot_shear_force('y')
    Pz = self._plot_shear_force('z')
    # For shear force along x direction
    if dir == "x":
        return Px.show()
    # For shear force along y direction
    elif dir == "y":
        return Py.show()
    # For shear force along z direction
    elif dir == "z":
        return Pz.show()
    # For shear force along all direction
    else:
        return PlotGrid(3, 1, Px, Py, Pz)
```

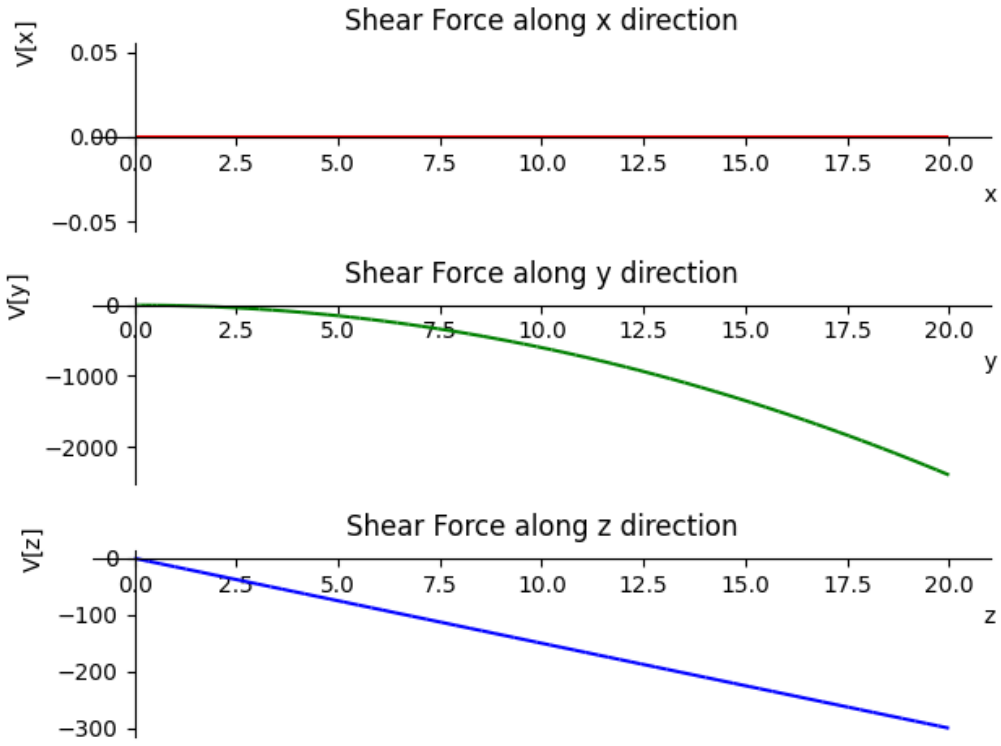
```
# Helper Function for plot_shear_force
def _plot_shear_force(self, dir, subs=None):

    shear_force = self.shear_force()

    if dir == 'x':
        dir_num = 0
        color = 'r'

    elif dir == 'y':
        dir_num = 1
        color = 'g'

    elif dir == 'z':
        dir_num = 2
        color = 'b'

    if subs is None:
        subs = {}

    for sym in shear_force[dir_num].atoms(Symbol):
        if sym == self.variable:
            continue
        if sym not in subs:
            raise ValueError('Value of %s was not passed.' %sym)
    if self.length in subs:
        length = subs[self.length]
    else:
        length = self.length

    return plot(shear_force[dir_num].subs(subs),
        (self.variable, 0, length), show = False,
        title='Shear Force along %c direction'%dir,
        xlabel=r'$\mathrm{%c}$'%dir,
        ylabel=r'$\mathrm{V[%c]}$'%dir, line_color=color)
```

These functions will output the following example plot:

- If direction is specified:



- If no direction is specified:



Since the above implementation (for Shear Force) has already been approved and merged by the developers, the implementation for other functions (Bending Moment, Slope, Deflection) will be similar to this.

3.2.3 Phase 2

Influence Line Diagrams

An **influence line** represents the variation of either the **reaction**, **shear,** or **moment** at a specific point in a member as a concentrated force moves over the member.

- Once this line is constructed, one can tell at a glance where the moving load should be placed on the structure so that it creates the greatest influence at the specified point.
- Furthermore, the magnitude of the associated reaction, shear, moment, or deflection at the point can then be calculated from the ordinates of the influence-line diagram.

For these reasons, influence lines play an important part in the design of bridges, industrial crane rails, conveyors, and other structures where loads move across their span.
We currently have functions to plot S.F.D. and B.M.D. I would like to add influence line diagrams for every reaction force, shear, and moment.

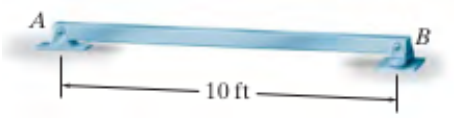
The difference between constructing an influence line and constructing a shear or moment diagram is that influence lines represent the effect of a **moving load** only at a specified point on a member, whereas shear and moment diagrams represent the effect of **fixed loads** at all points along the axis of the member.

I have already opened a discussion stating the importance of Influence Line Diagrams in further detail. I am also attaching some documents containing the theory as well as some examples.
Theory
Examples

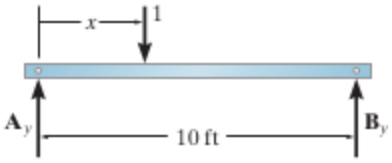
I.L.D. for Reactions

Let's consider this example:

Construct the influence lines for the vertical reaction at A and B of the given beam.



In order to construct I.L.D. for reaction at A, a unit load is placed on the beam at a distance x and the value of $\boldsymbol{A_y}$ is calculated by summing moments about ***B***.



When the unit load is placed at a variable distance x from ***A***, the reaction $\boldsymbol{A_y}$ as a function of x can be determined from

$$\circlearrowleft +\Sigma M_B = 0; \qquad -A_y(10) + (10 - x)(1) = 0$$

$$A_y = 1 - \tfrac{1}{10}x$$

Similarly for $\boldsymbol{B_y}$

$$B_y = \frac{1}{10}x$$

<u>Implementation</u>

This can be implemented by the following function.

- It generates shear and moment equations using the pre-existing shear force and bending moment functions and then considers the effect of adding a unit load at distance ***X*** and puts them in the equations.

- When we apply $\lim_{x \to l}$ to the shear and moment functions, equations are generated at point $x = l$, where l is the length of the beam. These equations are nothing but the static equations at point $x = l$ considering the beam is be placed at $x = 0$.
- These equations can be modified by considering a unit load to be placed at a distance ***X*** from the origin and adding its effect to the shear and moment equations.
- Here the static equations are implemented by simply considering the force and moment due to the unit load and the resulting equations are plotted.
- It solves the equations using linsolve and gives the desired reaction equations. Then it plots the resulting I.L.D. using Sympy's plotting function.

Note : The variable '***x***' and '***X***' are different variables. '***x***' is the variable in terms of which singularity functions are written and limit $\lim_{x \to l}$ is applied. The variable '***X***' represents the distance of unit load applied on the beam. The final equations are derived in terms of '***X***'.

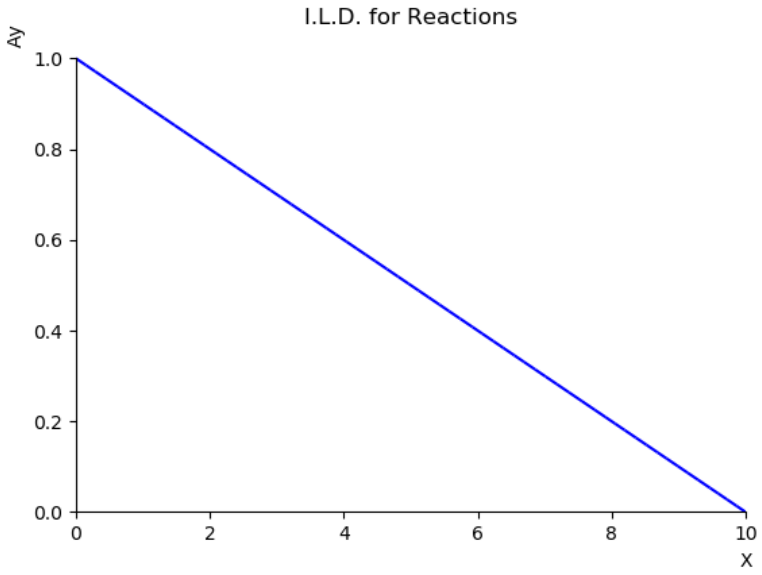
This **working code** is a rough implementation of the above idea. Here ild_reactions_eq() function determines the equations discussed above and plot_ild_reactions() function plots them.

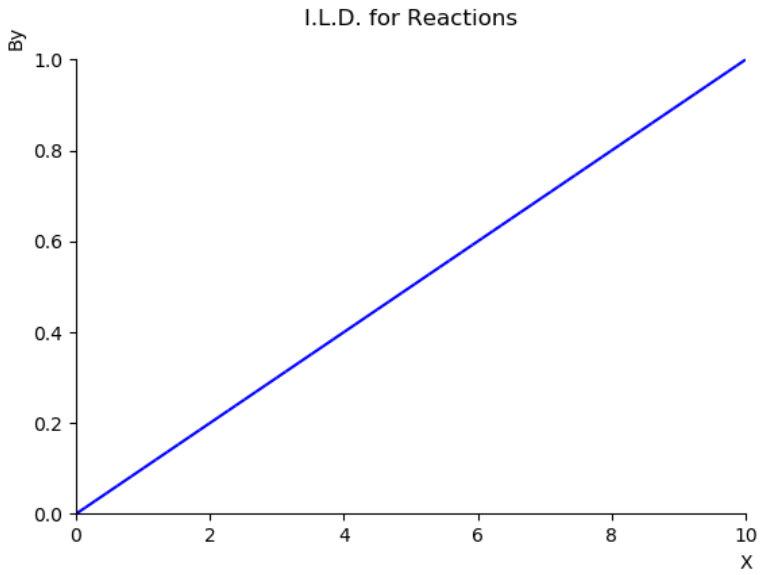
```
def ild_reactions_eq(self, val, *reactions):
    """

    Determines the equations of reaction forces under the effect
    of a moving load for Influence Line Diagram."


    Parameters
    ==========
    val : Integer
        Magnitude of moving load
    reactions :
        The reaction forces applied on the beam.

    """
    x = self.variable
    l = self.length
    C3 = Symbol('C3')
    C4 = Symbol('C4')
    X = symbols('X')

    # Adding effect of load of magnitude val on shear
    shear_curve = limit(self.shear_force(), x, l) + val
    # Adding effect of load of magnitude val on moment
    moment_curve = limit(self.bending_moment(), x, l) + val*(l-X)

    slope_eqs = []
    deflection_eqs = []

    slope_curve = integrate(self.bending_moment(), x) + C3
    for position, value in self._boundary_conditions['slope']:
        eqs = slope_curve.subs(x, position) - value
        slope_eqs.append(eqs)

    deflection_curve = integrate(slope_curve, x) + C4
    for position, value in self._boundary_conditions['deflection']:
        eqs = deflection_curve.subs(x, position) - value
        deflection_eqs.append(eqs)

    solution = list((linsolve([shear_curve, moment_curve] + slope_eqs
                        + deflection_eqs, (C3, C4) + reactions).args)[0])
    solution = solution[2:]

    # Determining the equations and solving them.
    self._reaction_loads = dict(zip(reactions, solution))
    return self._reaction_loads
```

```
def plot_ild_reactions(self):
    """

    Plots the Influence Line Diagram of Reaction Forces
    under the effect of a moving load.

    """
    X = symbols('X')
    for i in self._reaction_loads:
            plot(self._reaction_loads[i], (X, 0, self._length),
            title='I.L.D. for Reactions',
            xlabel=X, ylabel=i, line_color='blue')
```

This is what the API and the output looks like:

```
# API
>>> E, I = symbols('E, I')
>>> Ay, By = symbols('Ay, By')
>>> b = Beam(10, E, I)
>>> b.apply_load(Ay, 0, -1)
>>> b.apply_load(By, 10, -1)
>>> b.ild_reactions_eq(-1, Ay, By)
{Ay: 1 - X/10, By: X/10}
>>> b.plot_ild_reactions()
```

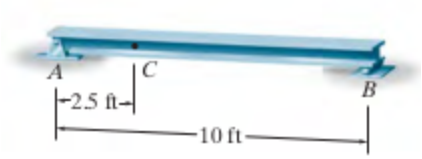




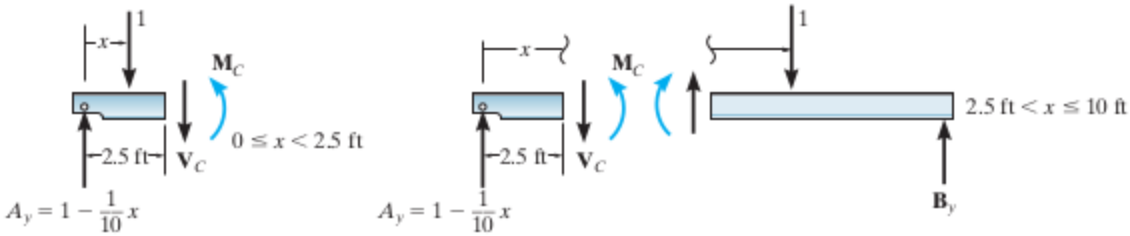
I have already opened #PR19126 regarding this but it needs further discussion with the mentors. I would like to implement this for statically determinate beams first and then extend the functionality to support indeterminate beams as well.

I.L.D. for Shear

Let us consider the following example:
*Construct the influence line for the shear at point **C** of the given beam.*



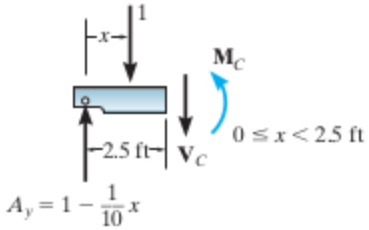
At each selected position $\boldsymbol{x}$ of the unit load,
The method of sections is used to calculate the value of $\boldsymbol{V_C}$. Note in particular that the unit load must be placed just to the left ($X = 2.5^-$) and just to the right ($X = 2.5^+$) of point $\boldsymbol{C}$ since the shear is discontinuous at $\boldsymbol{C}$.



Here two equations have to be determined
since there are two segments for the influence line due to the discontinuity of shear at ***C.***

Implementation

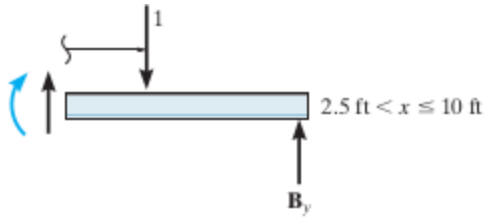
These equations can be determined by creating a function that takes the beam object, the point at which shear is to be determined, and reactions as arguments and splits the given beam into two sections by making two new beams of the segmented lengths. The equations of A_y and B_y can be determined by using the ***ild_reactions()*** function.



For the first section, construct a new beam of segmented length and apply all reactions apart from V_C and unit load, the equation of V_C can be determined by applying ($\lim_{x \to l}$ ***shear_force() - 1)***. In this case, it would simply be equal to

$$V_C = A_y - 1$$

$$V_C = -\frac{x}{10} \qquad 0 \leq X < 2.5$$



Similarly, for the second section, construct a new beam of remaining length and apply all reactions apart from V_C and unit load, the equation of V_C can be determined by applying (***1 -*** $\lim_{x \to l}$ ***shear_force()***). In this case, it would simply be equal to

$$V_C = 1 - B_y$$

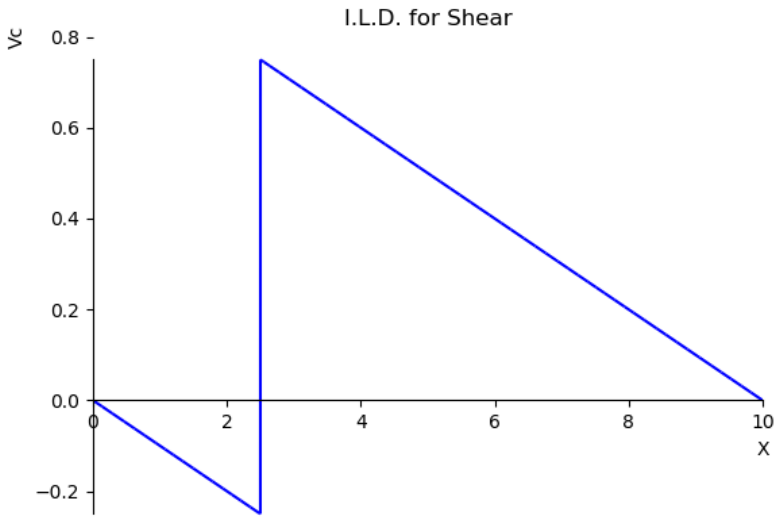
$$V_C = 1 - \frac{X}{10} \qquad 2.5 < X \leq 10$$

Once the equations are determined, V_C is plotted using the plot function.

The pseudo-code looks like this

```
Function ild_shear(beam element, location of point C, reactions)
{
    ild_reactions(reactions)# function is used to calculate reactions in terms of X

    b1=Beam(c, E, I)
    b2=Beam(length-c, E, I)                  #  Two new beam elements are declared of length C and L-C

    b1.apply_load(reactions)
    b2.apply_load(reactions)                 # Respective reaction forces applied except unit load

    # For b1
    Vc1 = limit(b1.shear_force(), x, l) - 1   # This gives Vc = Ay - 1

    # For b2
    Vc2 = 1 - limit(b2.shear_force(), x, l)   # This gives Vc = 1 - By

    plot(Vc1, Vc2)
}
```

The plot would look like this

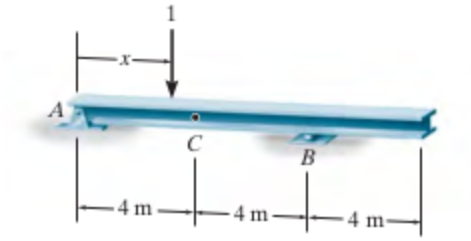


I would like to implement this for statically determinate and indeterminate beams.

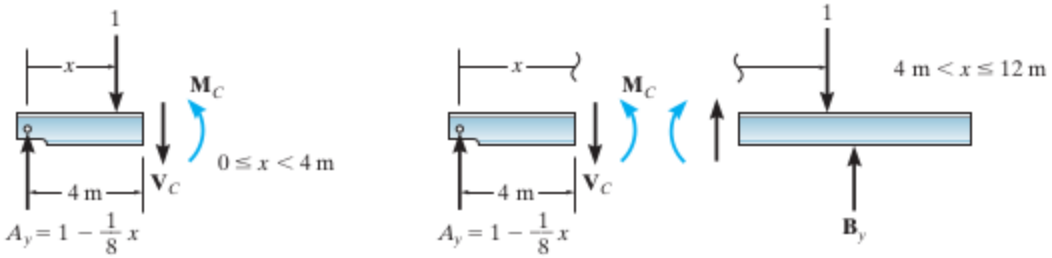
I.L.D. for Moment

Let's consider this example:

Construct the influence line for the moment at point C of the given beam



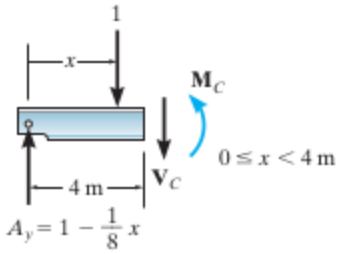
At each selected position of the unit load, the value of M_C is calculated using the method of sections.



The two line segments for the influence line can be determined using $\Sigma M_C = 0$ along with the method of sections. These equations when plotted yield the required influence line.

Implementation

These equations can be determined by creating a function that takes the beam object, the point at which shear is to be determined, and reactions as arguments and splits the given beam into two sections by making two new beams of the segmented lengths. The equations of A_y and B_y can be determined by using the ild_reactions() function.

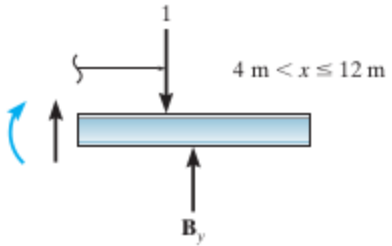


For the first section, construct a new beam of segmented length and apply all reactions and moments apart from unit load and find moment about ***C*** by using ($\lim_{x \to l}$ ***bending_moment() - (l - X))***. In this case, M_C would be equal to

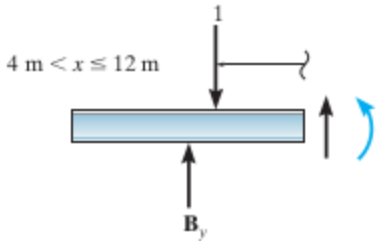
$$M_C = 4*A_y - (l - X)$$

$$M_C = 4*(1 - \frac{X}{8}) - (4 - X)$$

$$M_C = \frac{X}{2} \qquad 0 \leq X < 4$$



Similarly, for the second section, construct a new beam of remaining length and apply all reactions and moments apart unit load, but since our function of bending moment finds moment at the rightmost point, we need to flip this section of the beam (only for calculation purpose). It would look like this



$\boldsymbol{M_C}$ can be determined by applying ($\lim_{x \to l}$ ***bending_moment() - (X - l)***).

In this case, it would simply be equal to

$$\boldsymbol{M_C} = 4 * \boldsymbol{B_y} - (\boldsymbol{X} - \boldsymbol{l})$$

$$\boldsymbol{M_C} = 4 * \frac{\boldsymbol{X}}{8} - (\boldsymbol{X} - 4)$$

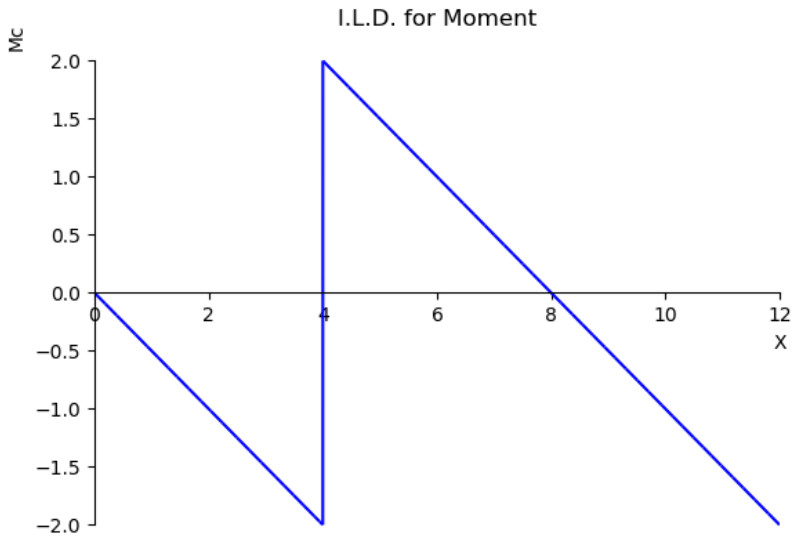
$$\boldsymbol{M_C} = 4 - \frac{\boldsymbol{X}}{2} \qquad\qquad 4 < \boldsymbol{X} \leq 12$$

Once the equations are determined, $\boldsymbol{M_C}$ is plotted using the plot function.

The pseudo-code looks like this

```
Function ild_moment(beam element, location of point C, reactions)
{
    ild_reactions(reactions)                          # function is used to calculate reactions in terms of X

    b1=Beam(c, E, I)
    b2=Beam(length-c, E, I)                           #  Two new beam elements are declared of length C and L-C

    b1.apply_load(reactions)                          # Respective reaction forces applied except unit load

    b2.apply_load(opposite end reactions)             # In order to flip the beam reaction forces from the opposite end

    # For b1
    Mc1 = limit(b1.bending_moment(), x, l) - (l - X)  # This gives Mc = 4*Ay - (4 - X)

    # For b2
    Mc2 = limit(b2.bending_moment(), x, l) - (X - l)  # This gives Mc = 4*By - (X - 4)

    plot(Mc1, Mc2)
}
```

The plot would look like this



I would like to implement this for statically determinate and indeterminate beams.

3.2.4 Phase 3

Function to plot Shear and Axial Stress

In my PR#18900, I added functions to calculate shear and axial stress in the beam class and beam3D class. I would like to make plotting functions for them as well. These functions will use the already created shear_stress() and axial_stress functions and use them to plot shear and axial stress.

This is what the **working code** looks like:

```
def plot_shear_stress(self, subs=None):
    shear_stress = self.shear_stress()  # Function to get value of Shear Stress
    if subs is None:
        subs = {}
    for sym in shear_stress.atoms(Symbol):
        if sym == self.variable:
            continue
        if sym not in subs:
            raise ValueError('Value of %s was not passed.' %sym)
    if self.length in subs:
        length = subs[self.length]
    else:
        length = self.length
    # Returns Plot of Shear Stress
    return plot(shear_stress.subs(subs), (self.variable, 0, length),
                title='Shear Stress', xlabel=r'$\mathrm{x}$', ylabel=r'$\tau$',
                line_color='r')

def plot_axial_stress(self, subs=None):
    axial_stress = self.axial_stress()  # Function to get value of Axial Stress
    if subs is None:
        subs = {}
    for sym in axial_stress.atoms(Symbol):
        if sym == self.variable:
            continue
        if sym not in subs:
            raise ValueError('Value of %s was not passed.' %sym)
    if self.length in subs:
        length = subs[self.length]
    else:
        length = self.length
    # Returns Plot of Axial Stress
    return plot(axial_stress.subs(subs), (self.variable, 0, length),
                title='Axial Stress', xlabel=r'$\mathrm{x}$', ylabel=r'$\sigma$',
                line_color='r')
```

Functions for maximum deflection, bending moment, and shear in Beam3D class

Maximum deflection occurs when the slope is equal to zero. Thus the position of maximum deflection is found out by equating the slope equation to zero.

$$\frac{dv}{dx} = 0$$

while for Cantilever Beam, this occurs at the free support end. In the case of single/double overhanging beams, we can compare both: the point where the slope becomes zero and the overhanging end.

Similarly, the maximum bending moment in each plane can be found out by equating the shear of that plane to 0

$$\frac{dM}{dx} = 0$$

And maximum shear can be found out by equating loading of that plane to 0

$$\frac{dV}{dx} = 0$$

Functions for maximum deflection, bending moment, and shear were added for beams in this PR as part of the 2018 GSoC project. Implementing this for 3D beams would be relatively easier as we already have these functions made for beams.

Here's the function overview with the proposed API

```
# Function overview
def max_delfection(self, dir="all", subs=None):
    # Returns the point of maximum deflection with its value
    # along the given direction.

def max_bmoment(self, dir="all", subs=None):
    # Returns points of maximum bending moment with its value
    # along the given direction.

def max_shear force(self, dir="all", subs=None):
    # Returns points of maximum shear force with its value
    # along the given directions.
```

```
# API
>>> b.max_deflection('x')    # along x-direction
>>> b.max_bmoment()          # along all directions
>>> b.shear_force('y')       # along y-direction
```

Stretch Goal

If all my work finishes before the time I would like to start working on the Truss Class.
I did not include it in my coding phases because considering the short coding period it would be very difficult to work on implementing a new class.
Here is something I have in mind:

```
# API
>>> t = Truss()                                          # Initializes truss object
>>> t.add_node (node_number, x_coordinate, y_coordinate) # Adds node
>>> t.add_member(start_node, end_node)                   # Adds member
>>> t.apply_support(node_number, type, direction)        # Applies support(roller/pin)
>>> t.apply_load(magnitude, node_number, direction)      # Applies load
>>> t.solve_for_forces()                                 # Solves for reaction forces
>>> t.reaction_forces()
# Returns a dictionary containing reaction forces and their values
>>> t.internal_forces()
# Returns a dictionary containing internal forces and their values
```

4. The Timeline

4.1 Community Bonding Period (May 4 - June 1)

During this period I will start discussing details with the mentor regarding the implementation of my project. Since I have been contributing to sympy for more than a year now, I am already familiar with most of the developers. Hence community bonding won't take much time. If all discussions with mentors are completed, I would like to commence to code during the community bonding period itself. I will start by solving existing issues in the continuum mechanics module as well as start implementing my ideas for the project.

4.2 Phase-1 (3-Weeks)

Week 1 and 2 (June 7-June 20)

Goal - Add functions for Plotting Bending Moment, Slope, and Deflection

- Implement the above functions.
- Test the above functions and add examples in the documentation

Week 3 (June 21-June 27)

Goal - Add functions for Plotting Loading Results and get the PRs merged

- Implement the remaining functions and test all the functions
- Finish documentation and add test cases

4.3 Phase 2

Week 4 and 5(June 28 -July 11)

Goal - Add functions for Influence line diagrams

- Implement algorithm for I.L.D. for reactions
- Implement algorithm for I.L.D. for shear
- Test the above functions and add examples in the documentation

Evaluation (July 12-July 16)

As per my schedule, I would have completed phase 1 and implemented Influence line diagrams for reactions and shear.

Week 6 and 7 (July 12 - July 25)

Goal - Get any pending PRs merged and implement the remaining functions

- Implement algorithm for I.L.D. for moment
- Finish documentation and add test cases

4.4 Phase 3

Week 8 and 9 (July 26 - August 1)

Goal - Add function for plotting shear and axial stress

- Implement the above function
- Add documentation and test cases
- Get any pending PRs merged

Week 9 and 10 (August 2 - August 16)

Goal - Add functions for Maximum Deflection, Bending Moment, and Shear

- Implement the above functions
- Add documentation and test cases
- Get the PRs merged
- Start working on my stretch goal(if time permits)

Final Evaluation (August 16 - August 23)

- The last week will be used for any leftover tasks and ensuring all PRs get merged
- Discussion with mentor(s) regarding future improvements
- I would submit a detailed report showcasing all the work done during the summer and what more can be done.

Any Plans/Commitment (During GSoC)

My end-semester exams will end in April itself. After that, I will be having summer vacations till July end which will cover most of the coding phase. During this period, I will easily be able to devote 25-30 hours per week.
My next semester will commence in August. However, I won't be having much load at the start of the semester so it will not affect my coding schedule.
In case of any disruptions, I will make sure to inform my mentor in advance and will devote extra time in the upcoming weeks to cover up for any lost time.

I have designed this proposal with a realistic goal, keeping in mind the 175 hour coding period. I have already started implementing some of my work and will make sure it gets completed in the designated time.

5. Post GSoC

- My primary goal would be to complete any unfinished work left and get it merged as soon as possible.
- I would like to further develop the continuum mechanics module by adding support for Truss Class.
- I would also like to add functions for stress-strain analysis on an object.
- I would continue to contribute to Sympy and by fixing bugs and reviewing pull requests

I would like to thank Jason Moore(@moorepants), Jogi Miglani(@jmig5776), Christopher Smith(smichr), Oscar Benjamin(@oscarbenjamin), Aaron Meurer(@asmeurer), and all the other members of the community who helped me a lot while contributing to this wonderful community.

6. References

- **Engineering Mechanics** by *Irving H. Shames*
- **Mechanics of Materials** by *Dr. B.C. Punmia*
- **Structural Analysis** by *R.C. Hibbeler*
- *Sympy Continuum Mechanics Documentation*
- *Truss Analysis using method of joints*
- Jashan Preet Singh's GSoC Proposal
- Ishan Joshi's GSoC Proposal
- Elastic Beams in 3 Dimensions
- NPTEL Influence Lines for Beams
- Structural Analysis 1