



## Python Simulations for Engineering Education: Resultant and Equilibrium in Coplanar Non-Concurrent Forces

Pankaj Dumka<sup>1\*</sup>, Darshana Dave<sup>2</sup>, Chandrakant Sonawane<sup>3</sup>, Arun Bongale<sup>3</sup>, Choon Kit Chan<sup>4</sup>, Ghanshyam Tejani<sup>5,6</sup>

<sup>1</sup> Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Raghuarh 473226, India

<sup>2</sup> Production Engineering Department, Government Engineering College Bhavnagar, Gujarat 364002, India

<sup>3</sup> Symbiosis Institute of Technology, Symbiosis International Deemed University, Pune 412115, India

<sup>4</sup> Faculty of Engineering and Quantity Surveying, INTI International University, Nilai 71800, Malaysia

<sup>5</sup> Applied Science Research Center, Applied Science Private University, Amman 11937, Jordan

<sup>6</sup> Jadara Research Center, Jadara University, Irbid 21110, Jordan

Corresponding Author Email: [p.dumka.ipec@gmail.com](mailto:p.dumka.ipec@gmail.com)

Copyright: ©2025 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/mmep.120205>

### ABSTRACT

**Received:** 28 September 2024

**Revised:** 9 November 2024

**Accepted:** 15 November 2024

**Available online:** 28 February 2025

#### Keywords:

*static equilibrium, Python programming, coplanar forces system, non-coplanar forces, inclusive innovation*

This article presents a detailed simulation approach for analyzing the equilibrium of coplanar and non-concurrent force systems by using Python programming. Coplanar forces are the forces that lie within the same plane, while non-concurrent forces do not intersect at a common point, creating complex systems which are often encountered in engineering fields such as structural analysis and statics. This study develops a set of Python functions to compute resultant forces, evaluate support reactions, and simulate equilibrium conditions in such force systems. The methodology involves defining force magnitudes, directions, and positions in the code, thus enabling automated calculation of equilibrium parameters through matrix operations (linear algebra) and iterative methods. Ten standard engineering problems have been selected to examine and test the functions, with results showing high accuracy when compared with the established solutions in the existing literature. This work highlights Python's effectiveness as a computational tool for both educational and practical applications in equilibrium studies.

## 1. INTRODUCTION

Studying the forces and their effects on the physical systems is a basis of engineering and physics. In particular, the analysis of coplanar and non-concurrent forces is important for understanding the behaviour of objects subjected to multiple forces which do not intersect at a common point [1, 2]. This introduction sets the stage for a comprehensive search of the theoretical and practical aspects of simulating the resultant and equilibrium of coplanar and non-concurrent forces using Python, which is a versatile and widely-used programming language.

For centuries, scientists and engineers have faced with the fundamental question of how forces interact with the objects and structures. This question forms the basis of classical mechanics, a branch of physics that deals with the motion and equilibrium of physical bodies. In this quest, the concept of a "force" emerged as a fundamental object that causes a body to change its state of motion or rest. Forces can result from interactions such as gravity, contact, tension, compression, and more, making them a central concept in understanding the behaviour of physical systems [1-3].

When several forces act on an object, the net effect of these forces becomes a critical consideration. Coplanar forces are

the subset of forces that all lie within the same plane. However, the term "non-concurrent" means that these forces do not share a common point of intersection. This property separates them from concurrent forces, where lines of action meet at a single point. The study of coplanar and non-concurrent forces becomes particularly relevant when studying the stability and equilibrium of structures, determining the net effect of various loads, and designing components that can survive multiple forces [3-5].

Knowing and modelling the resultant and equilibrium of coplanar and non-concurrent forces have great effects in various scientific and engineering fields. The significance of this study can be summarised as follows [6]:

- **Structural analysis:** In the civil and mechanical engineering, the stability and safety of structures, such as buildings, bridges, and machinery, depends on the correct analysis of the forces [7]. Determining the resultant and equilibrium of non-concurrent forces is central for ensuring that these structures remain stable and do not depart to the stresses imposed on them. Engineers rely on these principles to design and evaluate the strength and reliability of a wide range of structural systems.
- **Mechanical design:** Mechanical engineers normally

face systems subjected to multiple forces. Whether designing automotive components, machinery, or consumer products, engineers must confirm that the components can bear various forces while remaining in equilibrium [8, 9]. The knowledge of how to simulate and analyse these forces using Python will provide a valuable tool for optimizing the mechanical designs.

- **Physics and education:** This study has educational significance, as it forms the basis for the teaching of fundamental principles (statics) in physics and engineering courses. Using Python to simulate force systems and their behaviour can help the students to understand the abstract concepts more concretely. Furthermore, educators can use these simulations to make the subject matter engaging and handy [10, 11].
- **Computational tools:** Python, a versatile and widely adopted programming language, is noteworthy for simulating coplanar non-concurrent forces. Python's rich ecosystem of libraries for numerical calculations and data visualisation makes it an ideal platform for engineers, physicists, and students to perform complex force simulations efficiently. Developing Python-based tools for this purpose can enhance efficiency and accuracy in force analysis. In this article, NumPy, a module dedicated to numerical computations, has been used [12-20].

Conventionally, simulating the coplanar and non-concurrent forces has been depend on the analytical techniques, graphical methods, or specialized software, each with its own advantages and limitations [21, 22]. Analytical methods, i.e., the resolution of forces into components and the use of equilibrium equations, gives exact solutions but they can be time-consuming and challenging to use on the complex systems with several forces. Graphical methods, like the force polygon or funicular polygon, offer a more visual approach but they are less precise, particularly for systems with many forces or when they are used in educational settings [23, 24]. While specialized software packages in structural engineering and mechanics, such as MATLAB or finite element analysis tools, offer powerful solutions, they often require advanced knowledge and are not universally accessible [25, 26]. The proposed Python-based approach aims to bridge these gaps by offering a cost-effective, accessible, and versatile alternative that simplifies the simulation process. Python's ease of use (also no license is required to use it), coupled with its powerful numerical libraries like NumPy, enables engineers, educators, and students to efficiently calculate force resultants, simulate equilibrium states, and visualize complex force interactions. This approach enhances both educational engagement and practical applications, particularly in environments where traditional software may not be readily available.

The primary objectives of this study are to provide an in-depth understanding of the theoretical underpinnings of coplanar non-concurrent forces, including the calculation of the resultant force and the principles of equilibrium, demonstrate how Python, a popular and versatile programming language, can be used to simulate and calculate the resultant and equilibrium of coplanar non-concurrent forces, to highlight the practical applications of this knowledge in fields such as structural engineering and mechanical design.

While this article aims to provide a comprehensive overview of the subject, it is essential to acknowledge its limitations. The scope of this article primarily covers the analysis of coplanar non-concurrent forces in a two-

dimensional plane. Also, the beams analysed are determinate and indeterminate beams have not been considered. More complex three-dimensional force systems, dynamics, and considerations such as friction and material properties are beyond the scope of this article.

## 2. THEORETICAL BACKGROUND

Learning forces and their effects on physical systems is a fundamental thing in the field of engineering and physics. Coplanar and non-concurrent forces (which are the specific class of forces) are the focus of this article. These forces share the property of acting in the same plane but do not intersect at a common point [2]. Understanding the calculation principles behind the resultant forces and their equilibrium is central in numerous engineering applications, such as structural analysis and mechanical design. In the context of structural engineering and the analysis of beams, the principles related to coplanar and non-concurrent forces take on particular significance. Beams, whether cantilevered or simply supported, are critical components of various structures, and understanding how forces act on them is necessary for confirming structural stability and performance [27]. This section will elaborate on the theoretical concepts of coplanar non-concurrent forces, providing a deeper insight into the mathematical principles underpinning their analysis.

The concept of the resultant force is important to understand the net effect of multiple forces acting on a system. For a system the resultant force is the single force that produces the same effect as the original forces combined when applied at the appropriate point [28]. To determine the resultant force, vector addition is employed, a mathematical method that considers both the magnitude and direction of forces.

In this article (in Python formulation), forces are treated as vectors, which are mathematical entities that have both magnitude and direction. An arrow usually represents a vector and can be expressed in the form:  $F=[HV]$ , where,  $H$  represents the horizontal component and  $V$  represents the vertical component of the force  $F$ .

For a system of non-concurrent forces, each force is represented as a vector, and all vectors are considered in the same plane, simplifying the analysis. The vector addition allows us to find the resultant force vector, which, when clubbed with the following steps, will lead to support reactions and moments [1, 3, 6]:

- Resolve external forces into components, i.e., each force in the system is resolved into its horizontal and vertical components. This step involves breaking down each force vector into its constituent parts based on the system's geometry.
- Sum up all the horizontal components and equate them to zero for the equilibrium condition.
- Sum up all the vertical components and equate them to zero for the equilibrium condition.
- If the end is fixed, then the moment will be the sum of all the moments. Whereas, if it is hinge, the sum of the moments will be equal to zero.

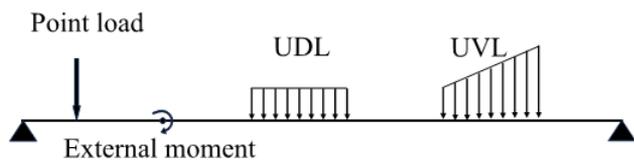
### 2.1 Common type of loadings

In structural engineering, various types of loads come upon, each of which exerts different forces on a structure. Understanding and analysing these loads is necessary for

designing safe and efficient structures. Here are some commonly seen loadings [4, 6, 27]:

- **Concentrated loads:** Concentrated loads are the point loads that act at specific location/point on a structure. These loads are usually represented by forces applied at a single point or distributed over a small area. Examples of concentrated loads include the weight of a person standing on a floor or the force applied by a vehicle's tire on a bridge.
- **Uniformly distributed loads (UDL):** UDL, often called as uniform loads, are the one which are distributed evenly along a specific span of a structure. These loads are usually represented by a force per unit length (e.g., N/m). Common examples of UDLs include the self-weight of beams, uniform snow loads on a roof, or wind pressures acting along the building height.
- **Uniformly varying loads (UVL):** Uniformly varying loads are distributed loads that change linearly along the length of a structural element. A triangular distribution of force often represents these types of loads. An example of a uniformly varying load is the hydrostatic pressure on the side of a dam, which increases with depth.
- **External moments:** External moments, also known as applied moments, are twisting forces applied to a structure. These moments can occur at specific points or along the length of a beam or other structural element. For instance, a door on hinges applies an external moment to its frame.

Figure 1 shows these loads from the point of view of a simply supported beam (the same holds for the cantilever as well). Engineers must consider the effects of these various loadings during the design and analysis of buildings, bridges, and other structures to ensure they can safely support the forces they encounter. By understanding and appropriately accounting for these loads, engineers can create safe, efficient and durable structures.



**Figure 1.** Different types of loads acting on a simply supported beam

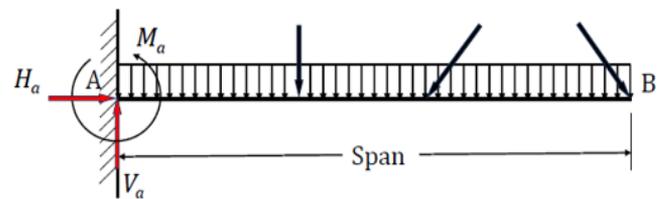
## 2.2 Commonly used beams

- **Cantilever beams:** A cantilever beam is a structural element supported at one end while the other remains free. This type of beam is common in constructions like balconies and diving boards. Understanding how to calculate the resultant of coplanar non-concurrent forces is essential for assessing the stability and strength of cantilever beams. When multiple forces are applied to a cantilever beam, they may not intersect at a single point. Each force must be broken down into horizontal and vertical components to find the resultant support forces. The summation of these components provides the resultant horizontal and vertical reaction forces. The net moment of all the vertical force at the support has to be evaluated for the support moment. In practical terms, calculating the resultant force is

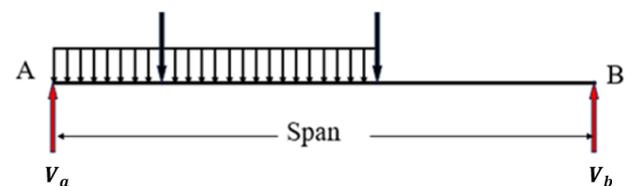
essential to assess a cantilever beam's bending moment, shear forces, and deflections. This analysis ensures that the beam can withstand the loads applied to it, safeguarding against structural failure and ensuring the safety of occupants. In this beam, there are three reaction components at the fixed end, and at the free end, no reaction is there. A typical cantilever beam is shown in Figure 2.

- **Simply supported beams:** A simply supported beam is another common structural element. It is supported at both ends (both ends hinged or one end hinged and the other having a roller), allowing it to rest on external supports, such as walls or columns. This type of beam can be found in floor systems, bridges, and roof trusses. Analysing the resultant of coplanar non-concurrent forces is essential for designing and evaluating the performance of simply supported beams. Similar to cantilever beams, multiple forces can act on simply supported beams. The process of finding the support reactions remains the same, but as the supports are either hinged or have rollers, the moment will be zero there. Assessing the beam's reaction forces at its supports is vital for structural design and ensuring equilibrium. Engineers must determine whether the beam is under tension or compression and evaluate its deformation under the applied loads. A typical simply supported beam having both ends hinged, and one hinged and the other on rollers is shown in Figures 3(a) and (b), respectively.
- **Overhanging beams:** This type of beam is projected beyond the supports. The overhang can be on either side of the supports, as shown in Figure 4.

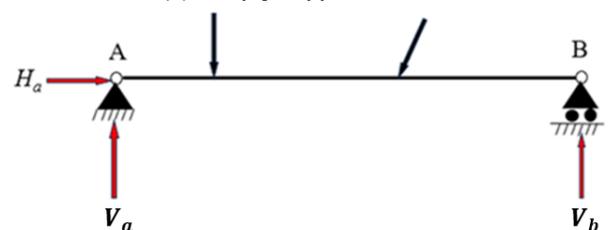
In this article, the statically determinate beams are considered as these beams are the ones in which all reaction components can be obtained with the help of above-mentioned equilibrium conditions.



**Figure 2.** A typical cantilever beam



(a) Simply supported beam



(b) One end fixed and the other on roller

**Figure 3.** Typical simply supported beams

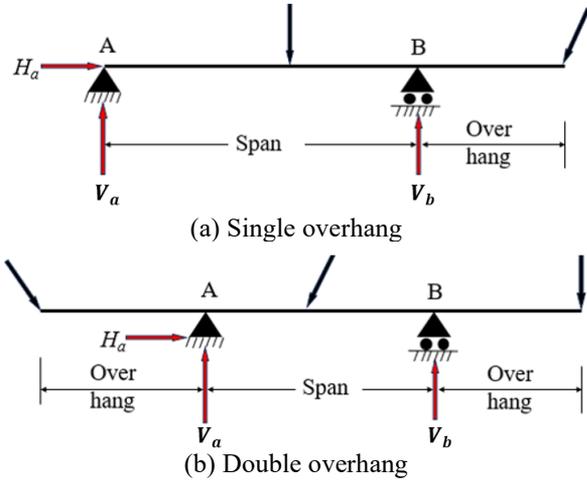


Figure 4. Typical schematic of overhanging beams

### 2.3 Reaction calculations

#### Cantilever beams

(a) Point loads: For the point loading shown in Figure 5, the reactions and moments at the fixed end ( $H_a$ ,  $V_a$ , and  $M_a$ ) are evaluated as:

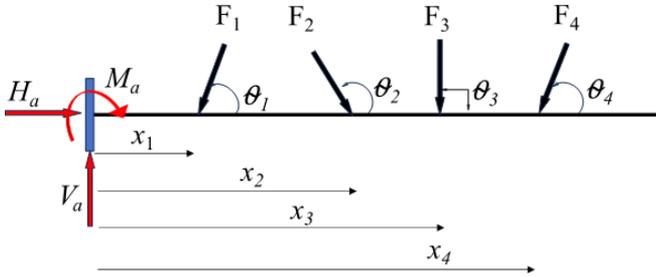


Figure 5. Point loads acting on the cantilever

$$V_a = \sum_{i=1}^{i=n} F_i \sin(\theta_i) \quad (1)$$

$$H_a = \sum_{i=1}^{i=n} F_i \cos(\theta_i) \quad (2)$$

$$M_a = \sum_{i=1}^{i=n} (F_i \sin(\theta_i) \times x_i) \quad (3)$$

(b) UDL: For UDL's shown in Figure 6, the reactions and moments at the fixed end are evaluated as:

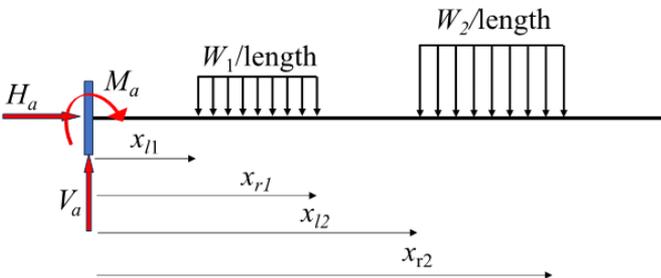


Figure 6. UDL acting on cantilever

$$V_a = \sum_{i=1}^{i=n} W_i (x_{ri} - x_{li}) \quad (4)$$

$$H_a = 0 \quad (5)$$

$$M_a = \sum_{i=1}^{i=n} W_i (x_{ri} - x_{li}) (x_{ri} + x_{li})/2 \quad (6)$$

(c) UVL: For UVL, as shown in Figure 7, the reactions are evaluated as shown in Eqs. (7) and (8), but the lever arm for the moment at the fixed end are evaluated considering two cases viz. case-I and case-II.

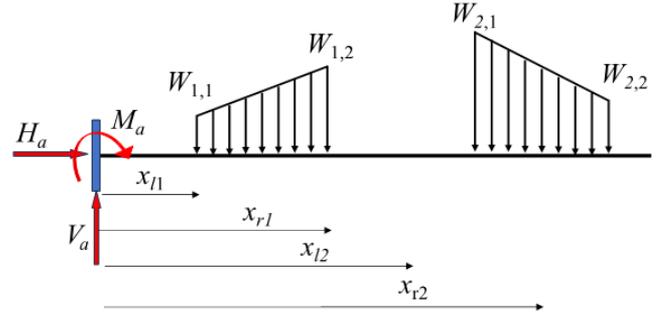


Figure 7. UVL acting on the cantilever

$$V_a = \sum_{i=1}^{i=n} \left(\frac{1}{2}\right) (W_{i,1} + W_{i,2}) (x_{ri} - x_{li}) \quad (7)$$

$$H_a = 0 \quad (8)$$

$$M_a = \sum_{i=1}^{i=n} \left(\frac{1}{2}\right) (W_{i,1} + W_{i,2}) (x_{ri} - x_{li}) \times \ell_i \quad (9)$$

Case-I: The left end load of UVL is lower than the right load

$$\ell_i = \left( x_{li} + (x_{ri} - x_{li}) \left( 1 - \frac{W_{i,2} + 2W_{i,1}}{W_{i,1} + W_{i,2}} \times \frac{1}{3} \right) \right) \quad (10)$$

Case-II: The right end load of UVL is lower than left load

$$\ell_i = \left( x_{li} + \frac{(x_{ri} - x_{li})}{3} \left( \frac{W_{i,1} + 2W_{i,2}}{W_{i,1} + W_{i,2}} \right) \right) \quad (11)$$

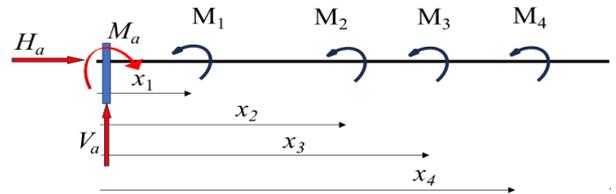


Figure 8. External moments acting on cantilever

(d) External moments: For external moments shown in Figure 8, the reactions at the support are evaluated as:

$$V_a = 0 \quad (12)$$

$$H_a = 0 \quad (13)$$

$$M_a = \sum_{i=1}^{i=n} M_i \quad (14)$$

### Simply supported beam

In case of simply supported beam as the moment at the reactions is zero so in the calculations they are not been taken into account.

(a) Point loads: For the point loading shown in Figure 9 the reactions at the supports ( $H_a$ ,  $V_a$ ,  $V_b$ , and  $M_a$ ) are evaluated as:

$$V_b = \frac{1}{L} \sum_{i=1}^{i=n} (F_i \sin(\theta_i) \times x_i) \quad (15)$$

$$H_a = \sum_{i=1}^{i=n} F_i \cos(\theta_i) \quad (16)$$

$$V_a = \sum_{i=1}^{i=n} F_i \sin(\theta_i) - V_b \quad (17)$$

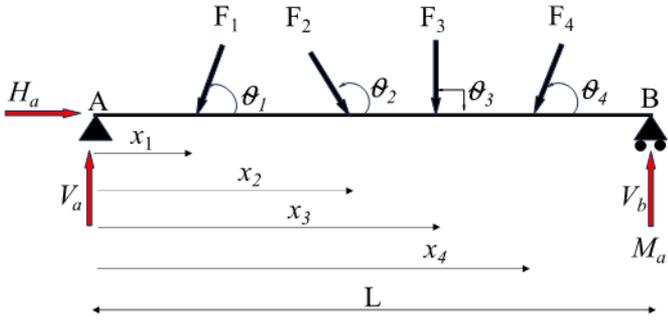


Figure 9. Point loads acting on simply supported beam

(b) UVL: For UVL's shown in Figure 10, the reactions at the supports are evaluated as:

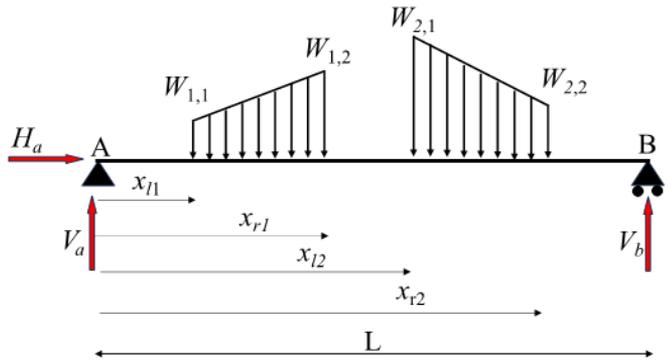


Figure 10. UVL acting on simply supported beam

$$V_b = \left(\frac{1}{2L}\right) \sum_{i=1}^{i=n} (W_{i,1} + W_{i,2}) (x_{ri} - x_{li}) \times \ell_i \quad (18)$$

$$H_a = 0 \quad (19)$$

$$V_a = \left(\frac{1}{2}\right) \sum_{i=1}^{i=n} (W_{i,1} + W_{i,2}) (x_{ri} - x_{li}) - V_b \quad (20)$$

For  $\ell_i$ , Eqs. (10) and (11) will be used.

(c) UDL: For UDL's shown in Figure 11, the reactions at the supports are evaluated as:

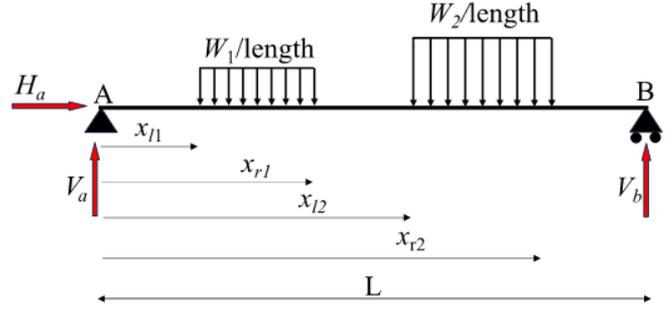


Figure 11. UDL acting on simply supported beam

$$V_b = \left(\frac{1}{2L}\right) \sum_{i=1}^{i=n} W_i (x_{ri} - x_{li}) (x_{ri} + x_{li}) \quad (21)$$

$$H_a = 0 \quad (22)$$

$$V_a = \sum_{i=1}^{i=n} W_i (x_{ri} - x_{li}) - V_b \quad (23)$$

(d) External moments: For external moments shown in Figure 12, the reactions at the support are evaluated as:

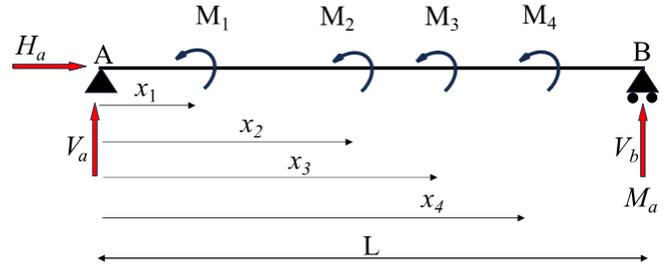


Figure 12. External moments acting on simply supported beam

$$V_b = \left(\frac{1}{L}\right) \sum_{i=1}^{i=n} M_i \quad (24)$$

$$H_a = 0 \quad (25)$$

$$V_a = -V_b \quad (26)$$

Point to be noted in all the above calculations is that downward loading is considered as positive. All the angles are measured counter-clock wise. For cantilever the  $x$  coordinate starts at fixed end whereas, for simply supported beam it will start from left support.

### 3. PYTHON FUNCTIONS TO OBTAIN SUPPORT REACTIONS

To simulate the resultant and equilibrium of coplanar non-concurrent forces in Python the NumPy library/module has been used [29-33]. The solution of the above functions is based on the following assumptions:

- The position of any external force or moment is considered from left to right, i.e., for cantilever all the positions are measured considering fixed end as  $x = 0$  and for simply supported beam the left support is at  $x = 0$ .

- For overhang case which is on the left side of the support of simply supported beam negative  $x$  position is considered.

- All the angles are measured counter clockwise from the beams as shown in Figures 5 and 8.

- All the determinate cases have been taken, i.e., the beam is designed for the determinate cases.

The Python functions presented in this section are designed to simulate the equilibrium conditions of coplanar and the non-concurrent forces in cantilever and simply supported beams, under various loading scenarios. The code was developed to be flexible and modular, allowing for the analysis of point loads, point moments, uniformly distributed loads (UDL), and uniformly varying loads (UVL) applied to both cantilever and simply supported beams. Each function takes in specific arguments that correspond to the forces, positions, and angles as defined in the problem diagrams.

The primary package used in the development of these functions is NumPy, which provides efficient array operations and mathematical functions. Particularly, `array()` is used to handle the input data, and `sum()` is used for summing up the forces and moments. For the angular calculations, the `radians()` function is used to convert the input angles from degrees to radians for accurate trigonometric functions, such as sine and cosine.

## Functions for cantilever beam

---

```
def canti_pl(w,θ,x):
```

```
    """
```

```
    function to evaluate the reaction and moment at the
    fixed end of a cantilever beam with point loads.
    Fixed end is considered as 'A'.
```

```
    Input: list of load (a) magnitudes (b) angle (c) position
    Output: Reaction (Va, Ha) and moment (Ma) at the free end
    """
```

```
    w=array(w)
    θ=array(θ)
    x=array(x)
```

```
    θ=radians(θ)
    Va=sum(w[:]*sin(θ[:]))
    Ha=sum(w[:]*cos(θ[:]))
    Ma=sum(w[:]*sin(θ[:])*x[:])
    return array([Va,Ha,Ma])
```

```
def canti_mom(M,x):
```

```
    """
```

```
    function to evaluate the reaction and moment at the
    fixed end of a cantilever beam with point moments.
    Fixed end is considered as 'A'.
```

```
    Input: list of (a) moments (b) moment position
    Output: Reaction (Va, Ha) and moment (Ma) at the free end
    """
```

```
    M=array(M)
    x=array(x)
```

---

```
Va=0.0
Ha=0.0
Ma=sum(M[:])
return array([Va,Ha,Ma])
```

```
def canti_udl(w,x):
```

```
    """
```

```
    function to evaluate the reaction and moment at the
    fixed end of a cantilever beam with udl.
    Fixed end is considered as 'A'.
```

```
    Input: list of udl (a) magnitudes (b) position
    Output: Reaction (Va, Ha) and moment (Ma) at the free end
    """
```

```
    w=array(w)
    x=array(x)
```

```
    Va=sum(w[:]*(x[:]-x[:0]))
    Ha=0.0
    Ma=sum(w[:]*(x[:]-x[:0])*(x[:]+x[:0])/2.0)
    return array([Va,Ha,Ma])
```

```
def canti_uvl(w,x):
```

```
    """
```

```
    function to evaluate the reaction and moment at the
    fixed end of a cantilever beam with uvl.
    Fixed end is considered as 'A'.
```

```
    Input: list of uvl (a) magnitudes (b) position
    Output: Reaction (Va, Ha) and moment (Ma) at the free end
    """
```

```
    w=array(w)
    x=array(x)
```

```
    Va=sum(0.5*((w[:0]+w[:1])*(x[:1]-x[:0])))
    Ha=0.0
```

```
    m=empty(shape(w)[0])
```

```
    if (w[:0]<w[:1]).all():
        b=w[:0]
        a=w[:1]
        la=x[:0]+(x[:1]-x[:0])*(1-(a+2*b))/(3*(a+b))
        m[:]=0.5*((w[:0]+w[:1])*(x[:1]-x[:0]))*la
```

```
    else:
```

```
        a=w[:0]
        b=w[:1]
        la=x[:0]+(x[:1]-x[:0])*(a+2*b)/(3*(a+b))
        m[:]=0.5*((w[:0]+w[:1])*(x[:1]-x[:0]))*la
```

```
    Ma=sum(m)
    return array([Va,Ha,Ma])
```

---

## Functions for simply supported beam

---

```
def ssb_pl(w,θ,x,L):
```

```
    """
```

```
    function to evaluate the reactions at the
    ends of a simply supported beam with point loads.
    End 'A' is considered as pin joint and 'B' is roller.
```

```
    Input: list of load (a) magnitudes (b) angle (c) position
    & (d) length of beam.
```

---

---

```

Output: Reactions (Va, Ha, Vb)
"""

w=array(w)
θ=array(θ)
x=array(x)

θ=radians(θ)
Vb=sum(w[:] * sin(θ[:]) * x[:]) / L
Va=sum(w[:] * sin(θ[:])) - Vb
Ha=sum(w[:] * cos(θ[:]))

return array([Va, Vb, Ha])
def ssb_udl(w, x, L):
    """
    function to evaluate the reaction at the
    ends of a simply supported beam with udl.
    End 'A' is considered as pin joint and 'B' is roller.
    Input: list of udl (a) magnitudes (b) positions
    & (c) length of beam.
    Output: Reaction (Va, Ha, Vb)
    """
    w=array(w)
    x=array(x)

    Vb=sum(w[:] * (x[:, 1] - x[:, 0]) * (x[:, 1] + x[:, 0]) / 2.0) / L
    Va=sum(w[:] * (x[:, 1] - x[:, 0])) - Vb
    Ha=0.0

    return array([Va, Vb, Ha])
def ssb_uvl(w, x, L):
    """
    function to evaluate the reaction at the
    ends of a simply supported beam with uvl.
    End 'A' is considered as pin joint and 'B' is roller.

    Input: list of uvl (a) magnitudes (b) positions
    & (c) length of beam.
    Output: Reaction (Va, Ha, Vb)
    """
    w=array(w)
    x=array(x)

    m=empty(shape(w)[0])

    if (w[:, 0] < w[:, 1]).all():
        b=w[:, 0]
        a=w[:, 1]
        la=x[:, 0] + (x[:, 1] - x[:, 0]) * (1 - (a + 2 * b) / (3 * (a + b)))
        m[:] = 0.5 * ((w[:, 0] + w[:, 1]) * (x[:, 1] - x[:, 0])) * la
    else:
        a=w[:, 0]
        b=w[:, 1]
        la=x[:, 0] + (x[:, 1] - x[:, 0]) * (a + 2 * b) / (3 * (a + b))
        m[:] = 0.5 * ((w[:, 0] + w[:, 1]) * (x[:, 1] - x[:, 0])) * la

    Vb=sum(m) / L

    Va=sum(0.5 * ((w[:, 0] + w[:, 1]) * (x[:, 1] - x[:, 0]))) - Vb

    Ha=0.0

```

---

```

return array([Va, Vb, Ha])
def ssb_mom(M, x, L):
    """
    function to evaluate the reaction at the
    ends of a simply supported beam with point moments.
    End 'A' is considered as pin joint and 'B' is roller.

    Input: list of moment (a) magnitudes (b) positions
    & (c) length of beam.
    Output: Reactions (Va, Ha, Vb)
    """
    M=array(M)
    x=array(x)

    Vb=sum(M[:]) / L
    Va=-Vb
    Ha=0.0
    return array([Va, Vb, Ha])

```

---

While using the above functions one has to be very carefully pass the arguments. Table 1 will help the user to supply the inputs in a proper format.

**Table 1.** Functions and their arguments

S. No.	Function	Arguments
1	<b>canti_pl</b> (w, θ, x)	With respect to the Figure 5: $w = [F_1, F_2, F_3, F_4]$ $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]$ $x = [x_1, x_2, x_3, x_4]$
2	<b>canti_mom</b> (M, x)	With respect to the Figure 8: $M = [M_1, M_2, M_3, M_4]$ $x = [x_1, x_2, x_3, x_4]$
3	<b>canti_udl</b> (w, x)	With respect to the Figure 6: $w = [W_1, W_2]$ $x = [[x_{l1}, x_{r1}], [x_{l2}, x_{r2}]]$
4	<b>canti_uvl</b> (w, x)	With respect to the Figure 7: $w = [[W_{1,1}, W_{1,2}], [W_{2,1}, W_{2,2}]]$ $x = [[x_{l1}, x_{r1}], [x_{l2}, x_{r2}]]$
5	<b>ssb_pl</b> (w, θ, x, L)	With respect to the Figure 9: $w = [F_1, F_2, F_3, F_4]$ $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]$ $x = [x_1, x_2, x_3, x_4]$ $L = L$
6	<b>ssb_udl</b> (w, x, L)	With respect to the Figure 11: $w = [W_1, W_2]$ $x = [[x_{l1}, x_{r1}], [x_{l2}, x_{r2}]]$ $L = L$
7	<b>ssb_uvl</b> (w, x, L)	With respect to the Figure 10: $w = [[W_{1,1}, W_{1,2}], [W_{2,1}, W_{2,2}]]$ $x = [[x_{l1}, x_{r1}], [x_{l2}, x_{r2}]]$ $L = L$
8	<b>ssb_mom</b> (M, x, L)	With respect to the Figure 12: $M = [M_1, M_2, M_3, M_4]$ $x = [x_1, x_2, x_3, x_4]$ $L = L$

---

To test the developed Python functions, ten typical problems were chosen based on their significance to practical applications in structural and mechanical engineering. These problems include a variety of force systems with different magnitudes, directions, and points of application, demonstrating common situations encountered in structural analysis and mechanical design. The selection was made to

cover a wide range of difficulties, from simple two-force systems to more complex multi-force equilibria. This diversity ensured that the functions could be evaluated across different levels of difficulty and their applicability to real-world engineering problems.

#### 4. IMPLEMENTATION OF PYTHON FUNCTIONS

In this section all the Python functions developed above are tested against different types of loadings on cantilever and simply supported beams.

**Example 1:** For the cantilever beam shown below in Figure 13, evaluate the fixed end reactions:

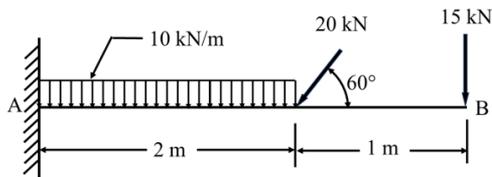


Figure 13. Cantilever beam with UDL and point load

**Solution:** Figure 14 represents the forces, reactions and moments acting on the beam shown in Figure 13.

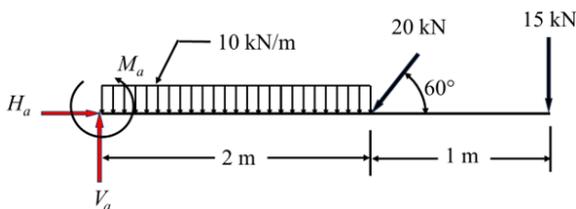


Figure 14. FBD

This problem has two types of loads viz. point load and udl. The functions used are: **canti\_pl** ( $w, \theta, x$ ) and **canti\_udl** ( $w, \theta, x$ ). The program to call the functions and its output is as follows:

**Program**

```
Va,Ha,Ma=canti_pl([15,20],[90,60],[3,2])+canti_udl([10],[0,2])
print(f"Va= {round(Va,3)}, Ha={round(Ha,3)}, Ma = {round(Ma,3)}")
```

**Output**

Va= 52.321, Ha =10.0, Ma = 99.641

**Example 2:** For the cantilever beam shown below in Figure 15, evaluate the fixed end reactions:

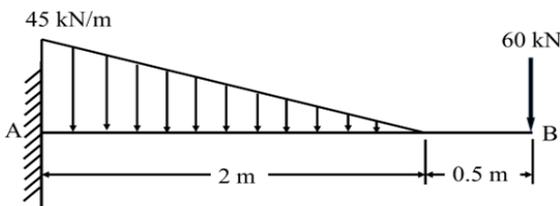


Figure 15. Cantiler beam with UVL and point load

**Solution:** Figure 16 represents the forces, reactions and moments acting on the beam shown in Figure 15.

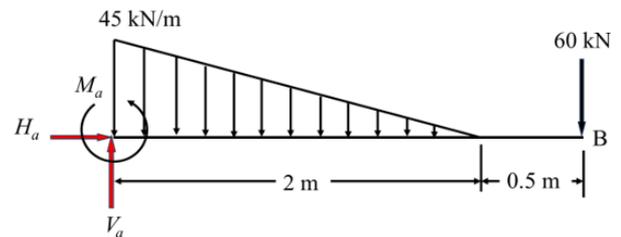


Figure 16. FBD

This problem has two types of loads viz. point load and uvl. The functions used are: **canti\_pl** ( $w, \theta, x$ ) and **canti\_uvl** ( $w, \theta, x$ ). The program to call the functions and its output is as follows:

**Program**

```
Va,Ha,Ma=canti_pl([60],[90],[2.5])+canti_uvl([[45,60]],[0,2])
print(f"Va= {round(Va,3)}, Ha={round(Ha,3)}, Ma = {round(Ma,3)}")
```

**Output**

Va= 105.0, Ha =0.0, Ma = 180.0

**Example 3:** For the simply supported beam shown below in Figure 17, evaluate the reactions at the end:

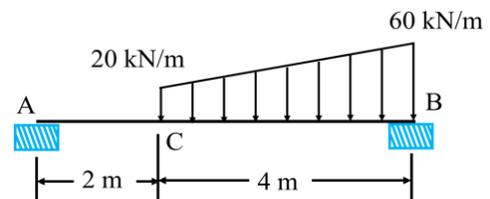


Figure 17. Cantilever beam with UVL

**Solution:** Figure 18 represents the forces and reactions acting on the beam shown in Figure 17.

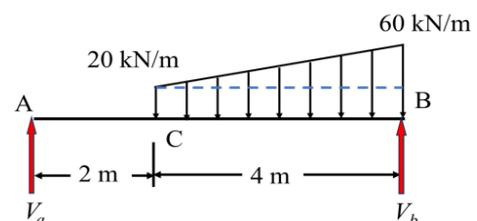


Figure 18. FBD

This problem has only uvl. The functions used is: **ssb\_uvl** ( $w, x, L$ ). The program to call the functions and its output is as follows:

**Program**

```
w=[[20,60]]
x=[[2,6]]
L=6
Va,Vb,Ha=ssb_uvl(w,x,L)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha = {round(Ha,3)}")
```

**Output**

Va= 44.444,Vb= 115.556, Ha =0.0

**Example 4:** For the simply supported beam and its FBD shown below in Figure 19, evaluate the reactions at the end:

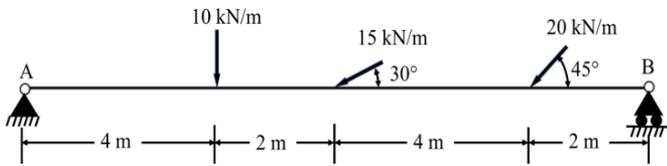


Figure 19. Simply supported beam with point loads

**Solution:** Figure 20 represents the forces and reactions on the beam shown in Figure 19.

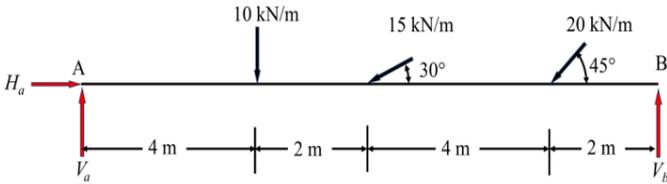


Figure 20. FBD

This problem has only point loads. The functions used is: **ssb\_pl**(w,θ,x,L). The program to call the functions and its output is as follows:

**Program**

```
w=[10,15,20]
θ=[90,30,45]
x=[4,6,10]
L=12
Va,Vb,Ha=ssb_pl(w,θ,x,L)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha
={round(Ha,3)}")
```

**Output**

Va= 12.774, Vb= 18.868, Ha =27.133

**Example 5:** For the simply supported beam shown below in Figure 21, evaluate the reactions at the end:

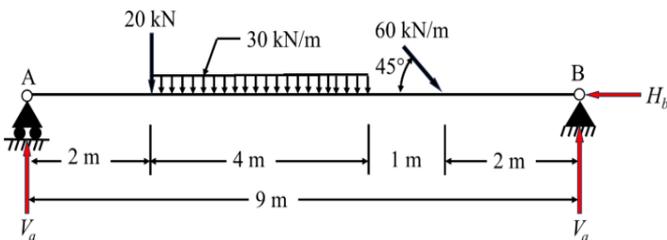


Figure 21. Simply supported beam with UVL and point loads

**Solution:** This problem has point loads and udl. The functions used are: **ssb\_pl**(w,θ,x,L) and **ssb\_udl**(w,x,L). The program to call the functions and its output is as follows:

**Program**

```
Va,Vb,Hb=ssb_pl([20,60],[90,180-45],[2,7],9)+ssb_udl([30],[2,6],9)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha
={round(Ha,3)}")
```

**Output**

Va= 91.65, Vb= 90.776, Hb =-42.426

**Example 6:** For the simply supported beam shown below in Figure 22, evaluate the reactions at the end:

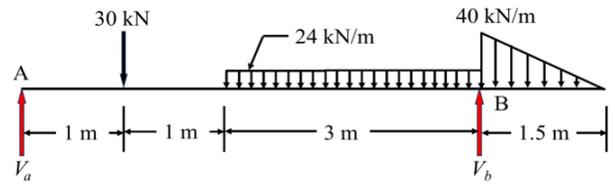


Figure 22. FBD of an over hanging beam

**Solution:** This problem has point load, uvl, and udl. The functions used are: **ssb\_pl**(w,θ,x,L), **ssb\_udl**(w,x,L), and **ssb\_uvl**(w,x,L). The program to call the functions and its output is as follows:

**Program**

```
Va,Vb,Ha=ssb_pl([30],[90],[1],5)+ssb_udl([24],[2,5],5)+ssb_uvl([40,0],[5,6.5],5)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha
={round(Ha,3)}")
```

**Output**

Va= 42.6, Vb= 89.4, Ha =0.0

**Example 7:** For the overhanging beam shown below in Figure 23, evaluate the reactions at the end:

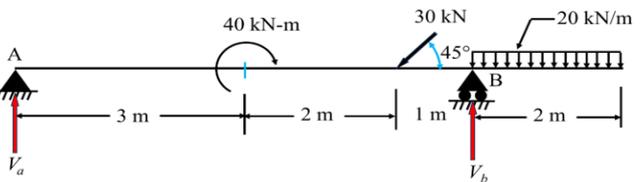


Figure 23. Overhanging beam with moment, point load, and UDL

**Solution:** This problem has point load, external moment, and udl. The functions used are: **ssb\_pl**(w,θ,x,L), **ssb\_udl**(w,x,L), and **ssb\_mom**(w,x,L). The program to call the functions and its output is as follows:

**Program**

```
Va,Vb,Ha=ssb_pl([30],[45],[5],6)+ssb_udl([20],[6,8],6)+ssb_mom([40],[3],6)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha
={round(Ha,3)}")
```

**Output**

Va= -9.798, Vb= 71.011, Ha =21.213

**Example 8:** For the simply supported beam shown below in Figure 24, evaluate the reactions at the end:

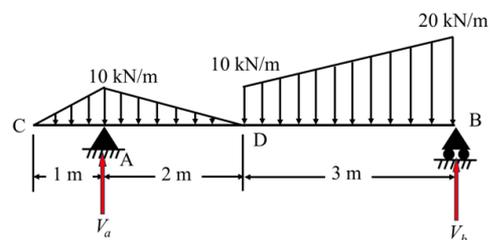


Figure 24. Overhanging beam with UVL

**Solution:** This problem has two uvl's. The function used is: **ssb\_uvl**(w,x,L). The program to call the functions and its output is as follows:

### Program

```
Va,Vb,Ha=ssb_uvl([[0,10],[10,0],[10,20]],[[-1,0],[0,2],[2,5]],5)
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Ha
={round(Ha,3)}")
```

### Output

Va= 26.0, Vb= 34.0, Ha =0.0

**Example 9:** For the simply supported beam shown below in Figure 25, evaluate the reactions at the end:

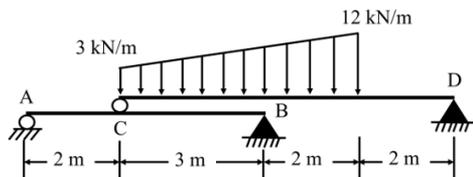


Figure 25. A complex beam

**Solution:** Free body diagrams of beams AB and CD are shown below in Figure 26.

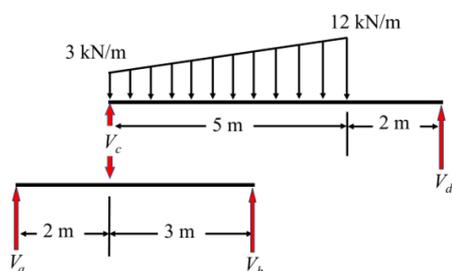


Figure 26. FBD

This problem has one uvl but if looked closely, then the problem can be divided into two. First, the top beam has to be solved with uvl, which will result in a reaction at roller and D. Then the reaction at C will act as a point load for the bottom beam, which now can be solved for the reactions at A and B. The function used is: **ssb\_udl**(w,x,L), and **ssb\_pl**(w,x,L). The program to call the functions and its output is as follows:

### Program

```
Vc,Vd,Hd=ssb_uvl([[3,12]],[[0,5]],7)
Va,Vb,Hb=ssb_pl([Vc],[90],[2],5)
print(f"Vc= {round(Vc,3)},Vd= {round(Vd,3)}, Hd
={round(Hd,3)}")
print(f"Va= {round(Va,3)},Vb= {round(Vb,3)}, Hb
={round(Hb,3)}")
```

### Output

Vc= 21.429, Vd= 16.071, Hd =0.0  
Va= 12.857, Vb= 8.571, Hb =0.0

**Example 10:** For the simply supported beam shown below in Figure 27, evaluate the reactions at the end:

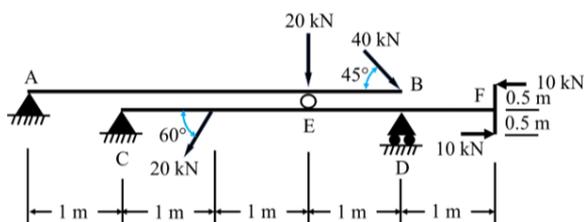


Figure 27. Arrangements of overhanging beams

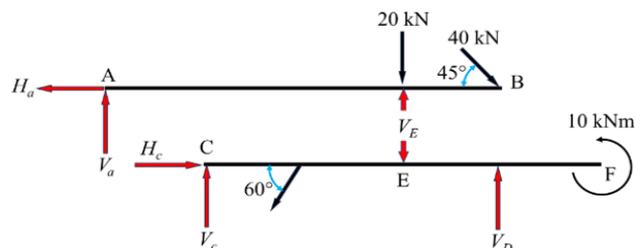


Figure 28. FBD

**Solution:** The free body diagram of beams shown in Figure 26 is shown in Figure 28.

This problem also has two beams to solve. First, the top beam has to be solved with two-point loads, resulting in reaction at E and A. Then, the reaction at E will act as a point load for the bottom beam, which now can be solved for the reactions at C and D. Also, there is one moment acting at F of magnitude 10 N-m. The function used is: **ssb\_pl**(w,θ,x,L), and **ssb\_mom**(w,x,L). The program to call the functions and its output is as follows:

### Program

```
Va,Ve,Ha=ssb_pl([20,40],[90,180-45],[3,4],3)
Vc,Vd,Hc=ssb_pl([Ve,20],[90,60],[2,1],3)+ssb_mom([-
10*1],[4],3)
print(f"Va= {round(Va,3)},Ve= {round(Ve,3)}, Ha
={round(Ha,3)}")
print(f"Vc= {round(Vc,3)},Vd= {round(Vd,3)}, Hc
={round(Hc,3)}")
```

### Output

Va= -9.428, Ve= 57.712, Ha =-28.284  
Vc= 34.118, Vd= 40.915, Hc =10.0

While this study largely focuses on theoretical examples, the developed Python functions can be readily applied to a variety of real-world engineering situations as well. For instance, they can be used in structural analysis for designing safe and stable buildings, bridges, and mechanical systems subjected to multiple forces and moments. These functions can also serve as an educational tool to help the students understand the principles of statics and equilibrium in a more collaborative and practical way.

## 5. CONCLUSIONS

In this article, coplanar non-concurrent forces in beams were modelled using Python Programming. First, the mathematics for different types of loading is being presented. Then, functions were developed for both cantilever and simply supported beams for different types of loadings. The developed functions are also presented in the article, along with the explanation of arguments which these functions will take. Then, these functions were tested against ten typical problems in structural engineering. It has been observed that the results obtained from the functions are in good agreement with the literature. The developed Python functions have successfully simulated the coplanar and non-concurrent forces in beams under various loading conditions, providing accurate results that are in sync with the theoretical expectations.

It has also been observed that the ability of Python programming to model complex-looking algorithms in a few lines of code is immense. Hence, it can be said that the

exploration of these forces through Python is a leap toward engineering excellence and scientific progress. Through this article, the readers will gain a comprehensive understanding of the principles of equilibrium of coplanar non-concurrent forces in beams, and they will also learn to apply them to real-world problems with the help of Python programming. The simplicity and usefulness of Python programming make it a valuable tool for both engineering practitioners and students. Engineers can benefit from efficient structural analysis, while students can gain a hands-on understanding of the principles of equilibrium and force systems.

Regardless of the accuracy and versatility of the developed Python functions, some limitations were viewed in the simulations. The present implementation primarily focuses on two-dimensional coplanar and non-concurrent force systems, which restricts its application to more complex three-dimensional or dynamic force systems. Also, the analysis is limited to the determinate beams, with indeterminate systems not yet considered. Future work could aim to expand the capabilities of the Python functions to handle three-dimensional force systems, dynamic loads, and indeterminate structural analyses. Additionally, incorporating more complex material properties and the effects of friction could enhance the realism of the simulations. As the field progresses, further improvements in the user interface and integration with other computational tools may also be explored to increase the accessibility and applicability of the developed approach.

## REFERENCES

[1] Bhavikatti, S.S., Rajashekarappa, K.G. (1994). *Engineering Mechanics*. New Age International, India.

[2] Meriam, J.L., Kraige, L.G., Bolton, J.N. (2020). *Engineering Mechanics: Dynamics*. John Wiley & Sons, USA.

[3] Mittelstedt, C. (2023). *Engineering Mechanics 2: Strength of Materials*. Springer Vieweg Berlin, Heidelberg. <https://doi.org/10.1007/978-3-662-66590-9>

[4] Nordin, M.N., Mohd Shah, M.K., Maidin, S.S., Mahmud, Y.H., Ismail, S.S.A. (2023). Outcomes-based approach in engineering education for special education need students: Psychology and rehabilitation elements. *Journal for ReAttach Therapy and Developmental Diversities*, 6(3): 52–58. <https://jrtd.com/index.php/journal/article/view/320>.

[5] Travis, R.C. (1945). An experimental analysis of dynamic and static equilibrium. *Journal of Experimental Psychology*, 35(3): 216-234. <https://psycnet.apa.org/doi/10.1037/h0059788>

[6] Bower, A. F. (2009). *Applied Mechanics of Solids*. CRC Press, USA. <https://doi.org/10.1201/9781439802489>

[7] Powell, G.H. (2010). *Modeling for Structural Analysis: Behavior and Basics*. Berkeley, CA: Computers and Structures.

[8] Overtom, E.M., Horeman, T., Jansen, F.W., Dankelman, J., Schreuder, H.W. (2019). Haptic feedback, force feedback, and force-sensing in simulation training for laparoscopy: A systematic overview. *Journal of Surgical Education*, 76(1): 242-261. <https://doi.org/10.1016/j.jsurg.2018.06.008>

[9] Collins, J., Chand, S., Vanderkop, A., Howard, D. (2021). A review of physics simulators for robotic applications. *IEEE Access*, 9: 51416-51431.

<https://doi.org/10.1109/ACCESS.2021.3068769>

[10] Chernikova, O., Heitzmann, N., Stadler, M., Holzberger, D., Seidel, T., Fischer, F. (2020). Simulation-based learning in higher education: A meta-analysis. *Review of Educational Research*, 90(4): 499-541. <https://doi.org/10.3102/0034654320933544>

[11] McGarr, O. (2021). The use of virtual simulations in teacher education to develop pre-service teachers' behaviour and classroom management skills: Implications for reflective practice. *Journal of Education for Teaching*, 47(2): 274-286.

[12] Varsha, M., Yashashree, S., Ramdas, D.K., Alex, S.A. (2019). A review of existing approaches to increase the computational speed of the python. *International Journal of Research in Engineering, Science and Management*, 2(4): 594-598.

[13] Mishra, P., Tewari, P., Mishra, D.R., Dumka, P. (2023). Numerical modelling of double integration with different data spacing: A Python-based approach. *Mathematics and Computational Sciences*, 4(2): 46-54. <https://doi.org/10.30511/mcs.2023.1990951.1115>

[14] Sanner, M.F. (1999). Python: A programming language for software integration and development. *Journal of Molecular Graphics and Modelling*, 17(1): 57-61.

[15] Fuhrer, C., Solem, J.E., Verdier, O. (2021). *Scientific Computing with Python: High-Performance Scientific Computing with NumPy, SciPy, and Pandas*. Packt Publishing Ltd, UK.

[16] Shein, E. (2015). Python for beginners. *Communications of the ACM*, 58(3): 19-21. <http://doi.org/10.1145/2716560>

[17] Joshi, A.R., Deo, A., Parashar, A., Mishra, D.R., Dumka, P. (2023). Modelling steam power cycle using Python. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 9(1): 152-162. <https://doi.org/10.32628/CSEIT228671>

[18] Gajula, K., Sharma, V., Sharma, B., Mishra, D.R., Dumka, P. (2022). Modelling of energy in transit using Python. *International Journal of Innovative Science and Research Technology*, 7(8): 1152-1156.

[19] Dumka, P., Rana, K., Tomar, S.P.S., Pawar, P.S., Mishra, D.R. (2022). Modelling air standard thermodynamic cycles using Python. *Advances in Engineering Software*, 172: 103186. <https://doi.org/10.1016/j.advengsoft.2022.103186>

[20] Dumka, P., Chauhan, R., Singh, A., Singh, G., Mishra, D. (2022). Implementation of Buckingham's PI theorem using Python. *Advances in Engineering Software*, 173: 103232. <https://doi.org/10.1016/j.advengsoft.2022.103232>

[21] Kiusalaas, J. (2010). *Numerical Methods in Engineering with Python*. Cambridge University Press, UK.

[22] Doerr, H.M. (1997). Experiment, simulation and analysis: An integrated instructional approach to the concept of force. *International Journal of Science Education*, 19(3): 265-282. <https://doi.org/10.1080/0950069970190302>

[23] Reiley, C.E., Akinbiyi, T., Burschka, D., Chang, D.C., Okamura, A.M., Yuh, D.D. (2008). Effects of visual force feedback on robot-assisted surgical task performance. *The Journal of Thoracic and Cardiovascular Surgery*, 135(1): 196-202. <https://doi.org/10.1016/j.jtcvs.2007.08.043>

[24] Williams, L.E., Loftin, R.B., Aldridge, H.A., Leiss, E.L.,

- Bluethmann, W.J. (2002). Kinesthetic and visual force display for telerobotics. In Proceedings 2002 IEEE International Conference on Robotics and Automation, Washington, USA, pp. 1249-1254. <https://doi.org/10.1109/ROBOT.2002.1014714>
- [25] Françoso, M.T., Costa, D.C., Valin, M.M., Amarante, R.R. (2013). Use of open source software for the development of web GIS for accessibility to tourist attractions. *Journal of Civil Engineering and Architecture*, 7(4): 472-486.
- [26] Hu, X., Zhou, Q. (2020). MATLAB software in the numerical calculation of civil engineering. In International Conference on Machine Learning and Big Data Analytics for IoT Security and Privacy, pp. 730-734. [https://doi.org/10.1007/978-3-030-62746-1\\_110](https://doi.org/10.1007/978-3-030-62746-1_110)
- [27] Carrera, E., Giunta, G., Petrolo, M. (2011). *Beam Structures: Classical and Advanced Theories*. John Wiley & Sons, USA.
- [28] Saliklis, E., Saliklis, E. (2019). Non-concurrent forces and the funicular. In *Structures: A Geometric Approach: Graphical Statics and Analysis*. Springer, Cham, USA, pp. 31-56. [https://doi.org/10.1007/978-3-319-98746-0\\_3](https://doi.org/10.1007/978-3-319-98746-0_3)
- [29] Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., et al. (2017). SymPy: Symbolic computing in Python. *PeerJ Computer Science*, 3: e103. <http://doi.org/10.7717/peerj-cs.103>
- [30] Van Der Walt, S., Colbert, S.C., Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2): 22-30. <https://doi.org/10.1109/MCSE.2011.37>
- [31] Bauckhage, C. (2020). NumPy/SciPy Recipes for Data Science: Subset-Constrained Vector Quantization via Mean Discrepancy Minimization, pp. 1-4.
- [32] Johansson, R., John, S. (2019). *Numerical Python: Scientific Computing and Data Science Applications with NumPy, SciPy and Matplotlib*. Apress, USA.
- [33] Ranjani, J., Sheela, A., Meena, K.P. (2019). Combination of NumPy, SciPy and Matplotlib/PyLab—A good alternative methodology to MATLAB—A Comparative analysis. In 2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT), Chennai, India, pp. 1-5. <https://doi.org/10.1109/ICIICT1.2019.8741475>

## NOMENCLATURE

F	external forces
W	per unit length variation of uniformly distributed or uniformly varying loads
V	vertical support reaction
H	horizontal support reaction
$x$	position of external force
$(x_l, x_r)$	span of UDL or UVL
M	moment

## Greek symbols

$\theta$	angle at which the external force will act
----------	--

## Subscripts

$l$	left
$r$	right