General Bending Tool

Symbolic & Numeric

Abdelrahman A.Kotb

$\mathcal{AAK}$

Contents

1. Introduction **6**

2. Theoretical Methodology **6**

2.1. Inverse Relationship and Stiffness Coefficients . . . 6
2.2. Calculation of Bending Moments for Various Loads . . . 7
2.3. Comparison: Conventional vs. Singularity Function Methods . . . 7
2.4. Load Corrections for the Singularity Function Method . . . 8
2.4.1. The Need for Load Correction . . . 8
2.4.2. Examples . . . 8
2.4.3. Comparison Table . . . 8
2.5. Singularity Function Implementation . . . 9
2.6. Boundary Conditions and Equilibrium Equations . . . 9
2.7. Sign Convention . . . 10

3. Code Structure Overview **11**

4. Detailed Module Descriptions **11**

4.1. Graphical Utilities . . . 11
4.2. Load Representations . . . 11
4.3. Support Representations . . . 11
4.4. Post-Processing and Visualization . . . 12
4.5. Main Analysis Engine . . . 12

5. How to Use the GUI **15**

6. Code Architecture and Design Considerations **18**

7. Potential Enhancements **19**

8. Conclusion **19**

A. Full MATLAB Code Listings **20**

A.1. `sing.m` . . . 20
A.2. `concentrated_force.m` . . . 20

A.3. concentrated_moment.m 20
A.4. s_rec.m 21
A.5. s_tri.m 21
A.6. fixed.m 21
A.7. roller.m 23
A.8. simple.m 24
A.9. con_f.m 26
A.10. con_m.m 27
A.11. rec.m 28
A.12. n_tri.m 31
A.13. rev_tri.m 33
A.14. trap.m 36
A.15. Main_Solver.m 38

List of Figures

2.1 Sign Convention 10
4.2 Flow Diagram for the Code 14
5.1 Mode Selection 15
5.2 Beam Length input 15
5.3 Supports and Loads declaration 15
5.4 Full Plot of the Problem 16
5.5 Solve Button 16
5.6 Shear and Moment diagrams 16
5.7 Deflection Plot 17
5.8 Solved Unknowns 17
5.9 Moment and Deflection equations 18
5.10 Custom Deflection Calculator 18

List of Tables

2.1 Bending Moment Expressions for Various Load Types 7

2.2 Load Correction Concept for Distributed Loads 9

2.3 Boundary Conditions for Different Support Types 10

1. Introduction

This project is a comprehensive MATLAB-based tool for structural beam analysis under various loads and support conditions. It employs symbolic mathematics (using MATLAB's symbolic toolbox) to derive expressions for bending moments, shear forces, and deflections. Object-oriented programming is used to model different load and support types, and graphical outputs (plots and LaTeX-formatted equations) are generated for visualization and verification.

---

2. Theoretical Methodology

In beam bending theory, the deflection of a beam under load is intimately related to the internal bending moments acting along the beam. For a beam that is subject to bending in two principal directions (e.g., bending about the horizontal and vertical axes), the curvature components are defined by the second derivatives of the deflections. Let $u(z)$ and $v(z)$ be the deflections in the X-Z and Y-Z planes, respectively.

For an unsymmetric beam, the bending moments $M_x(z)$ and $M_y(z)$ about the corresponding axes are related to these curvatures by the flexural rigidity (or bending stiffness) matrix, which incorporates the second moments of area I_{xx}, I_{yy} and the product moment of inertia I_{xy}. This relationship can be written in matrix form as:

$$\begin{bmatrix} M_x \\ M_y \end{bmatrix} = -E \begin{bmatrix} I_{xy} & I_{xx} \\ I_{yy} & I_{xy} \end{bmatrix} \begin{bmatrix} u'' \\ v'' \end{bmatrix},$$

where E is the Young's modulus of the beam's material.

For convenience, we denote the flexural rigidity matrix by:

$$\mathbf{D} = \begin{bmatrix} I_{xy} & I_{xx} \\ I_{yy} & I_{xy} \end{bmatrix}.$$

2.1. Inverse Relationship and Stiffness Coefficients

To obtain the curvatures (and, subsequently, the deflections) from the bending moments, we invert the matrix $\mathbf{D}$. The inverse of a 2×2 matrix is given by:

$$\mathbf{D}^{-1} = \frac{1}{\det(\mathbf{D})} \begin{bmatrix} -I_{xy} & I_{xx} \\ I_{yy} & -I_{xy} \end{bmatrix}$$

with

$$\det(\mathbf{D}) = I_{xx}\, I_{yy} - (I_{xy})^2.$$

Defining the stiffness coefficients as:

$$K_{xx} = \frac{I_{xx}}{I_{xx}\,I_{yy} - (I_{xy})^2} \quad , \quad K_{yy} = \frac{I_{yy}}{I_{xx}\,I_{yy} - (I_{xy})^2} \quad , \quad K_{xy} = \frac{I_{xy}}{I_{xx}\,I_{yy} - (I_{xy})^2}$$

the curvatures can now be written as:

$$\begin{bmatrix} u'' \\ v'' \end{bmatrix} = \frac{1}{E} \begin{bmatrix} K_{xy} & -K_{xx} \\ -K_{yy} & K_{xy} \end{bmatrix} \begin{bmatrix} M_x \\ M_y \end{bmatrix}.$$

Finally, integrating the curvature expressions twice (and applying the appropriate boundary conditions) gives the deflection functions $u(z)$ and $v(z)$.

2.2. Calculation of Bending Moments for Various Loads

In a beam, different types of loads produce different moment distributions:

Table 2.1: Bending Moment Expressions for Various Load Types

Load Type	**Moment (Without Singularity)**	**Moment (With Singularity)**
Concentrated Moment	$M(z) = \begin{cases} 0, & z < a, \\ M_0, & z \geq a, \end{cases}$	$M(z) = M_0\,[z-a]^0$
Concentrated Force	$M(z) = \begin{cases} 0, & z < a, \\ P\,(z-a), & z \geq a, \end{cases}$	$M(z) = P\,[z-a]^1$
Uniformly Distributed Load	$M(z) = \begin{cases} 0, & z < a, \\ \frac{w}{2}\,(z-a)^2, & z \geq a, \end{cases}$	$M(z) = \frac{w}{2}\,[z-a]^2$
Triangular Distributed Load	$M(z) = \begin{cases} 0, & z < a, \\ \frac{w}{6(L-a)}\,(z-a)^3, & z \geq a, \end{cases}$	$M(z) = \frac{w}{6(L-a)}\,[z-a]^3$

2.3. Comparison: Conventional vs. Singularity Function Methods

Without singularity functions, one must derive the moment expressions piecewise for each segment of the beam and then enforce continuity conditions at the load application points. This approach can be laborious when multiple loads are present.

In contrast, the singularity function method allows one to write a single, unified expression for $M(z)$ that automatically accounts for discontinuities. This results in a more elegant formulation, particularly when the beam is subject to complex or multiple loading conditions.

Once the bending moment $M(z)$ is known, the inverse relationship discussed above is used to compute the curvatures, and double integration yields the deflections. The inverse stiffness matrix, containing the coefficients K_{xx}, K_{yy}, and K_{xy}, encapsulates the beam's geometric and material properties and serves as a key component in transitioning from moments to deflections.

2.4. Load Corrections for the Singularity Function Method

Singularity functions offer a compact way to represent distributed loads on a beam. However, the standard formulation assumes that the distributed load is defined over the entire beam length (i.e., from $z = 0$ to $z = L$). When a load is applied only over a subinterval $[a, b]$ (with $b < L$), a correction must be introduced so that the singularity function formulation remains valid without altering the overall resultant load or moment.

2.4.1. The Need for Load Correction

For a distributed load applied from $z = a$ to $z = b$, the singularity function representation requires that the load effectively "ends" at $z = L$. In order to reconcile this difference, a correction term is added to the load representation. This correction is designed to:

1. Extend the load definition to the beam end.
2. Cancel out any additional moment or force that would otherwise be introduced by this extension.

2.4.2. Examples

Consider a Uniformly Distributed Load (UDL) of intensity w applied from $z = a$ to $z = b$ (with $b < L$). The conventional (piecewise) bending moment for $z \geq a$ can be written as:

$$M(z) = \begin{cases} 0, & z < a, \\ -\frac{w}{2}(z-a)^2 + C, & z \geq a, \end{cases}$$

where C is chosen such that the moment is continuous at $z = b$.

Using singularity functions, the same load is represented as:

$$M(z) = -\frac{w}{2}[z-a]^2 + \frac{w}{2}[z-b]^2.$$

Here, the term $\frac{w}{2}[z-b]^2$ acts as a load correction. It "turns off" the effect of the UDL beyond $z = b$ so that the net distributed load remains

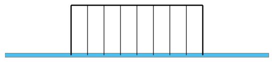
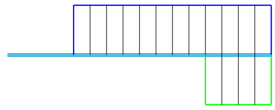
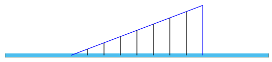
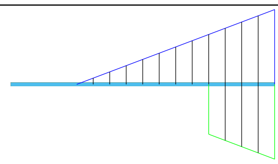
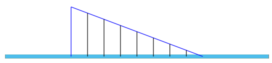
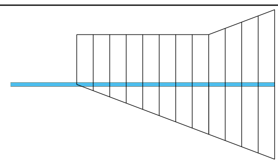
$$R = w(b-a),$$

and the moment at any section is calculated correctly.

2.4.3. Comparison Table

Table 2.2 shows different types of loads and their corrections. Many corrections are valid for the same load; the table shows the methods used in this project.

Table 2.2: Load Correction Concept for Distributed Loads

Distributed Load Type	Load	Correction
Uniform		
Triangular		
Reversed Triangular		

2.5. Singularity Function Implementation

The function sing.m implements the singularity function—a compact representation used to model discontinuities in beam loading. In structural analysis, the singularity function is typically defined as

$$\langle z - a \rangle^n = \begin{cases} (z-a)^n, & \text{if } z \geq a, \\ 0, & \text{if } z < a. \end{cases}$$

In our implementation (sing.m), the function calculates the difference

$$m = z - x,$$

and then multiplies $(z - x)^n$ by the factor

$$\frac{\text{sign}(m)}{2} + \frac{1}{2},$$

which acts as a switch:

- When $z \geq x$, $\text{sign}(m) = 1$, so the factor is 1, and the function returns $(z - x)^n$.
- When $z < x$, $\text{sign}(m) = -1$, making the factor 0, and the function returns 0.

2.6. Boundary Conditions and Equilibrium Equations

The boundary conditions for a beam are essential to determine the unknown reaction forces and integration constants. In this project, the following support types are considered:

- **Fixed Support:** The support constrains both the vertical displacement and the rotation (slope) at the point of support.

- **Simple Support:** The support restrains vertical displacement, but it allows rotation.
- **Roller Support:** The support prevents vertical displacement while permitting horizontal movement and rotation.

Table 2.3 summarizes the imposed constraints for each support type.

Table 2.3: Boundary Conditions for Different Support Types

Support Type	Constraints	Imposed Equations
Fixed Support	$v = 0, \quad v' = 0$	$v(a) = 0, \quad v'(a) = 0$
Simple Support	$v = 0$	$v(a) = 0$
Roller Support	$v = 0$	$v(a) = 0$

In addition to the support boundary conditions, the overall equilibrium of the beam must satisfy the following conditions in each plane:

$$\sum F_y = 0, \qquad \text{(Vertical force equilibrium)}$$
$$\sum F_x = 0, \qquad \text{(Horizontal force equilibrium)}$$
$$\sum M = 0, \qquad \text{(Moment equilibrium)}$$

These equilibrium equations, together with the boundary conditions from the supports, form the basis for solving the beam's bending problem.

2.7. Sign Convention

The used sign convention is the same as [1]. Figure 2.1 shows the convention.

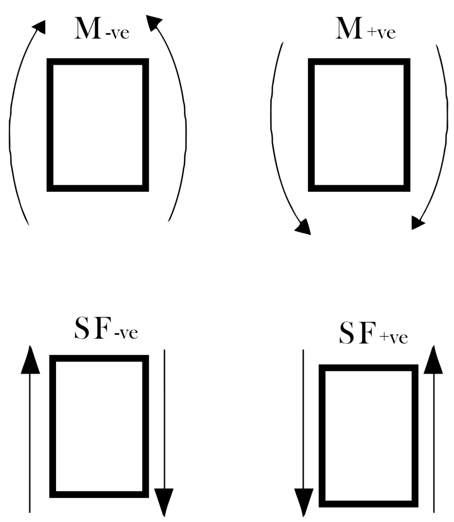


Figure 2.1: Sign Convention

3. Code Structure Overview

The project consists of several MATLAB files and classes, each dedicated to a specific aspect of the analysis:

- **Graphical Utilities:** (`circular_arrow.m`, `initial_plot.m`) for custom graphics.
- **Load Representations:** (`con_f.m`, `con_m.m`, `n_tri.m`, `rev_tri.m`, `rec.m`,`trap.m`) for different load types.
- **Support Representations:** (`fixed.m`, `roller.m`, `simple.m`) for various supports.
- **Post-Processing and Visualization:** (`final_plots.m`, `momentdisp.m`, `solplot.m`, `sub_plot.m`, `preview_reset.m`).
- **Main Analysis Engine:** (`Main_Solver.m`) which ties everything together.
- **Singularity Functions:** (`sing.m`, `s_rec.m`, `s_tri.m`) for load calculations.

---

4. Detailed Module Descriptions

Below is a brief description of each module. See the Appendix for the complete code listings.

4.1. Graphical Utilities

- `circular_arrow.m`: Draws a circular arrow representing applied moments.
- `initial_plot.m`: Sets up the initial plotting axes and draws the beam's neutral axis.

4.2. Load Representations

- `con_f.m`: Class for concentrated forces.
- `con_m.m`: Class for concentrated moments.
- `rec.m`: Class for Uniformaly distributed loads.
- `n_tri.m` **and** `rev_tri.m`: Classes for triangular distributed loads.
- `trap.m`: Class for trapezoidal distributed loads.
- **Singularity Functions:** `sing.m`, `s_rec.m`, `s_tri.m` implement the singularity functions.

4.3. Support Representations

- `fixed.m`: Represents fixed supports.
- `roller.m`: Models roller supports.
- `simple.m`: Represents simple supports.

4.4. Post-Processing and Visualization

- `final_plots.m`: Plots bending moments, shear forces, and deflections.
- `momentdisp.m` **and** `solplot.m`: Display solved equations.
- `sub_plot.m`: Converts symbolic expressions into numeric data for plotting.
- `preview_reset.m`: Resets the preview axes.

4.5. Main Analysis Engine

`Main_Solver.m` is responsible for assembling the complete beam response by aggregating the contributions from both loads and supports and then solving for the deflections and slopes. In our approach, both loads and supports contribute their full share to the overall bending moment expressions $M_x(z)$ and $M_y(z)$ as well as to the initial moments M_{01} and M_{02} (moments at $z = 0$ in the y–z and x–z planes, respectively). The workflow is as follows:

1. **Initialization and Setup:**
   The solver begins by declaring all necessary global and symbolic variables and by setting assumptions (such as $L > 0$ and $p > 0$). It also verifies that beam parameters (like moment of inertia or stiffness coefficients) are properly defined.

2. **Aggregation of Load and Support Contributions:**
   The Solver_GUI passes two cell arrays—one containing load objects and one containing support objects—to the main solver. Each load and support computes its moment contribution using internal functions. These values are stored (typically in a property named `final_moment`) and summed to form the global moment expressions. Notably, both loads and supports contribute to the overall moments $M_x(z)$ and $M_y(z)$, as well as the moments M_{01} and M_{02} at $z = 0$.

3. **Formulation of Equilibrium Equations:**
   The solver sets up the global equilibrium conditions:

   $$\sum F_y = 0, \quad \sum F_x = 0, \quad \sum M = 0,$$

   where the moment equilibrium includes the contributions from both the moments M_{01} and M_{02}.

4. **Integration to Obtain Deflection and Slope:**
   The relationship between the bending moment and the beam's curvature is given by

   $$\frac{d^2 w}{dz^2} = \frac{M(z)}{EI}.$$

   The solver integrates this equation twice:

   - The first integration produces the slope $w'(z)$.
   - The second integration yields the deflection $w(z)$.

   These integrations introduce unknown constants $c_1, c_2, c_3, and\ c_4$, which are later determined.

5. **Application of Boundary Conditions and Solving:**
   The solver applies the support boundary conditions from the supports along with the equilibrium equations to create a system of equations. It then solves this system symbolically to determine the unknown constants and reaction forces.

6. **Substitution and Final Output Generation:**
   The solved constants are substituted back into the integrated expressions, resulting in the final equations for slope and deflection along the beam. Additional quantities, such as shear forces, are computed as needed.

7. **Returning Results to the GUI:**
   The processed outputs —deflection curves, moment diagrams, final equations, and reaction forces— are then returned to the GUI for visualization and further user interaction.

Figure 4.2 shows the connections between all functions and classes that build up the GUI

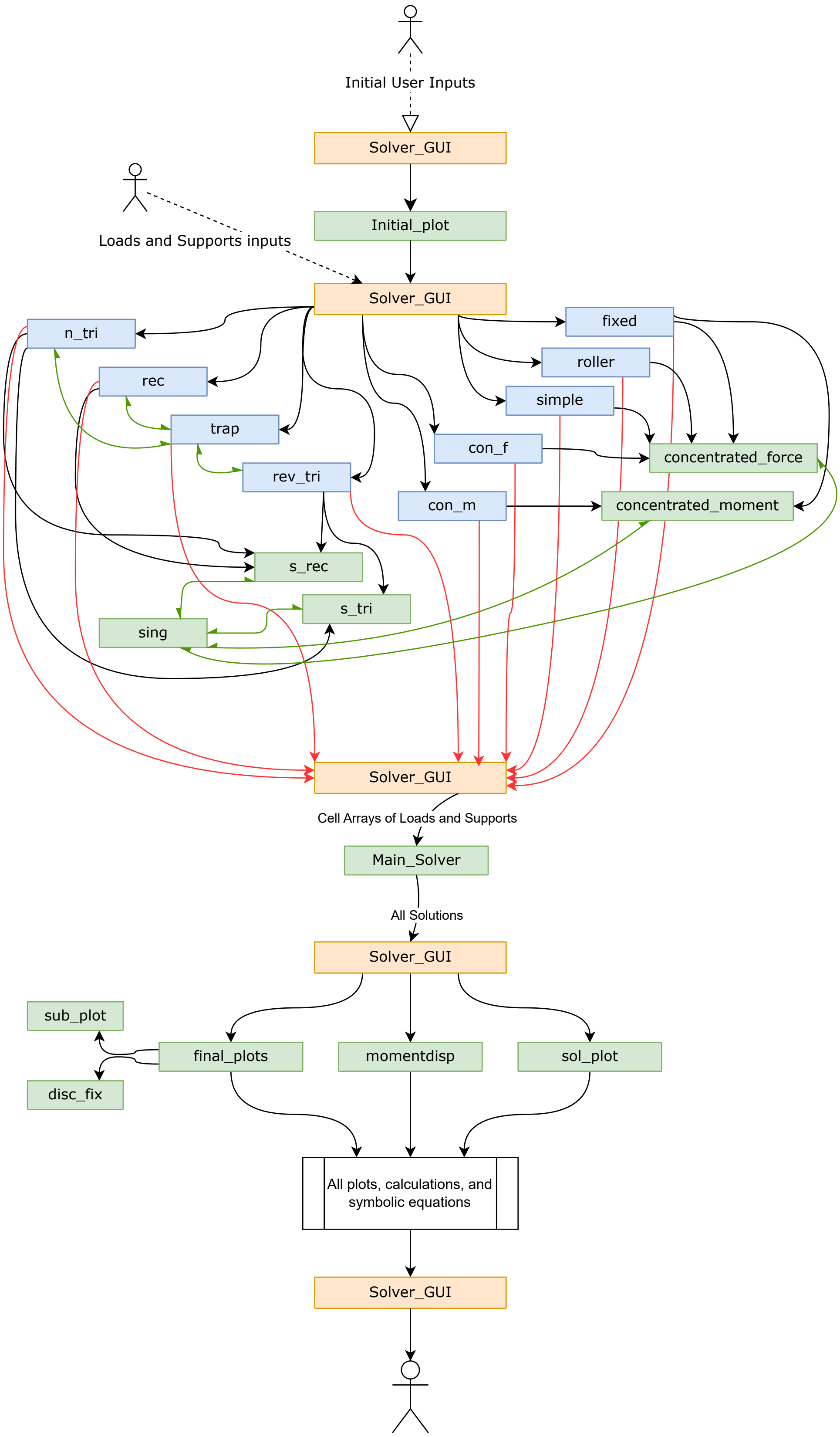


Figure 4.2: Flow Diagram for the Code

5. How to Use the GUI

The GUI provides a step-by-step interface:

1. **Choose Analysis Mode:** As shown in Figure 5.1 select one of the following:
   - Symbolic Mode: Solve symbolically in terms of k_{xx}, k_{yy}, k_{xy}.
   - I-Coefficients Mode: Input coefficients for I_{xx}, I_{yy}, I_{xy}.
   - k-Coefficients Mode: Input coefficients for k_{xx}, k_{yy}, k_{xy}.

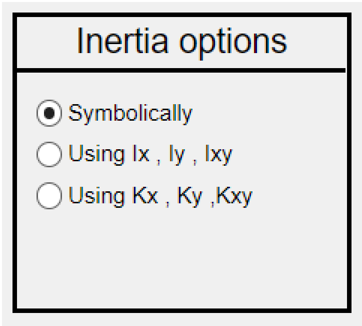


Figure 5.1: Mode Selection

2. **Input Beam Length:** Enter the beam length -Figure 5.2.
3. **Click "Begin":** This initializes the analysis.

Figure 5.2: Beam Length input

4. **Declare Supports and Loads:** Use the GUI to add/delete supports and loads -Figure 5.3. For loads, an extra Preview option displays the load and its singularity correction.

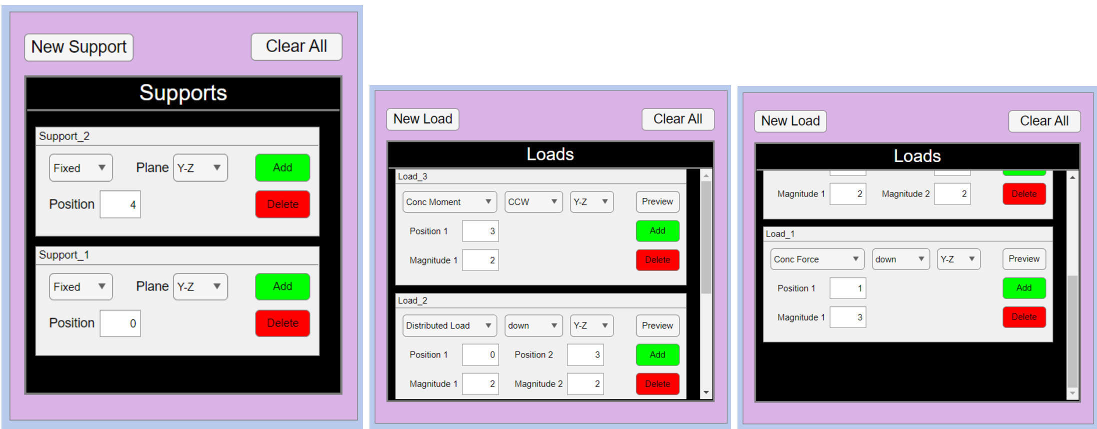


Figure 5.3: Supports and Loads declaration

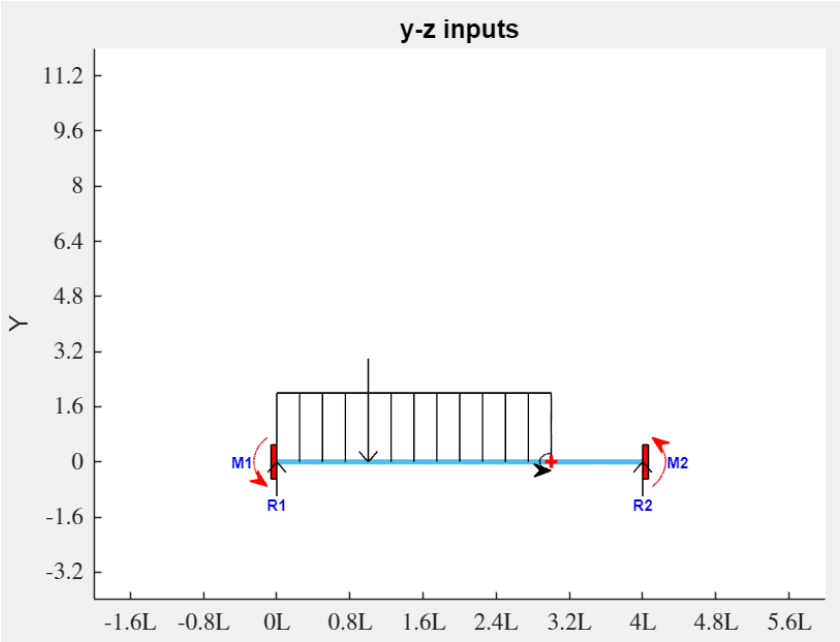


Figure 5.4: Full Plot of the Problem

5. **Generate Graphs:** In the Graphs tab, click Solve -Figure 5.5- to produce bending moment, shear -Figure 5.6-, and deflection -Figure 5.7- diagrams.

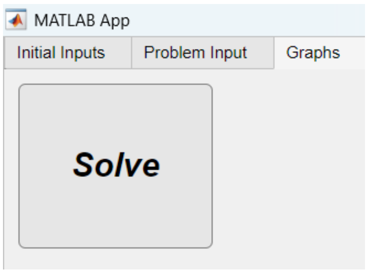


Figure 5.5: Solve Button

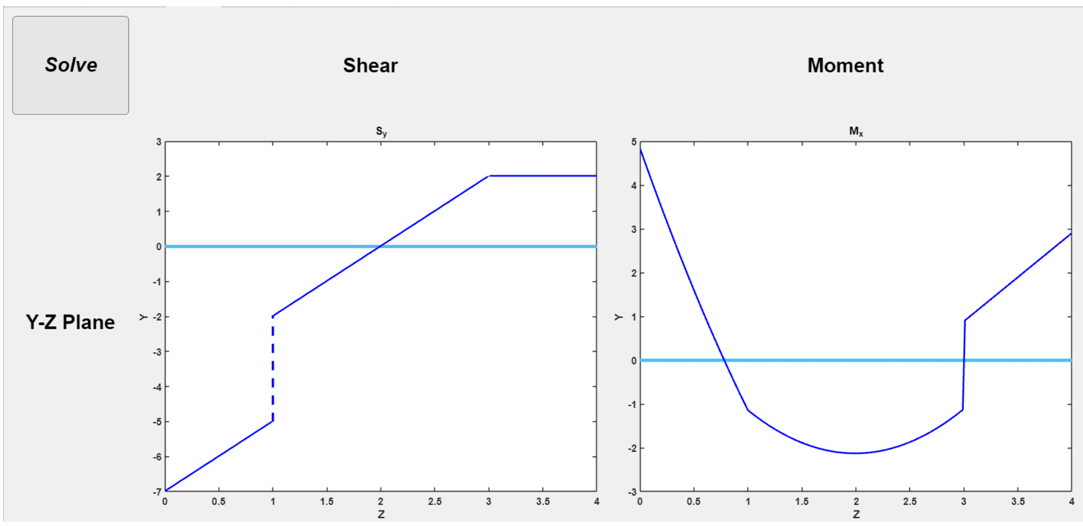


Figure 5.6: Shear and Moment diagrams

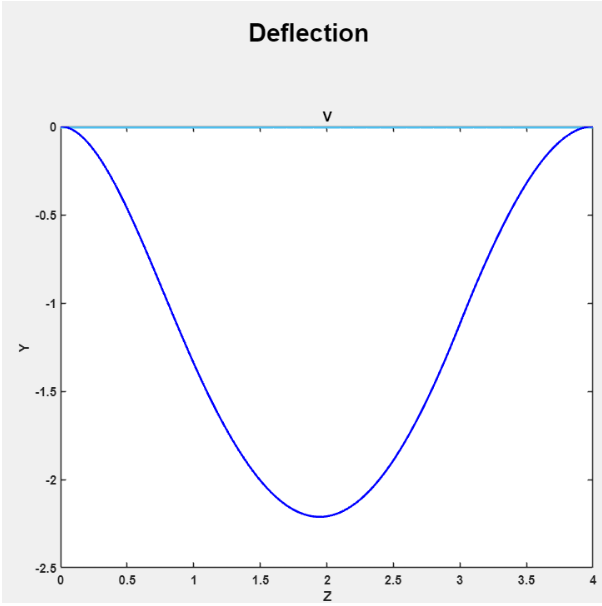


Figure 5.7: Deflection Plot

6. **View Equations:** In the Equations tab, view all solved unknowns -Figure 5.8- and equations -Figure 5.9- formatted in LaTeX.

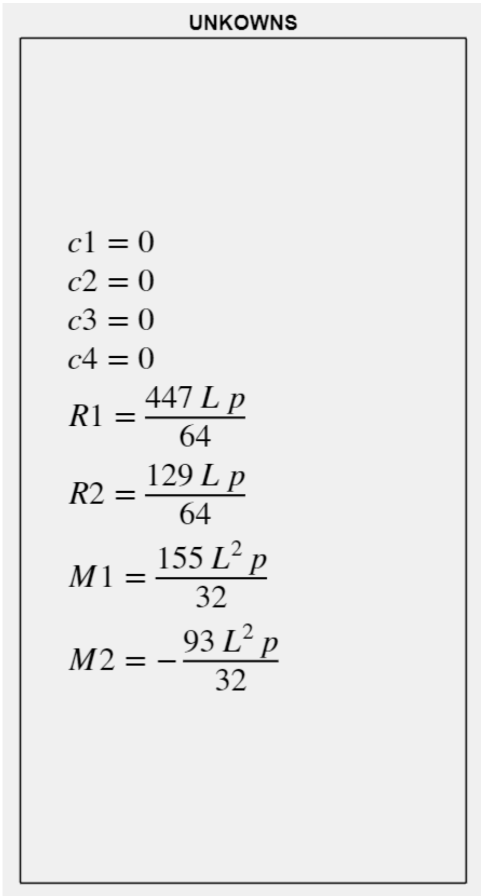


Figure 5.8: Solved Unknowns

Equations

$$M_x = 4.84375\,L^2\,p + Z^2\,p - 2\,L^2\,p\,(-[Z-3L]^0) - 6.984375\,L\,Z\,p + p\,(3\,L - Z)^2\,(-[Z-3L]^0) + 3\,L\,p\,(L-Z)\,(-[Z-L]^0)$$

$$M_y = 0$$

$$v = -\frac{K_y}{E}\left(\frac{Z^4\,p}{12} + 2.421875\,L^2\,Z^2\,p + \frac{p\,(-2[Z-3L]^0)\,(3\,L-Z)^4}{24} - 1.164062\,L\,Z^3\,p + \frac{L\,p\,(-2[Z-L]^0)\,(L-Z)^3}{4} - \frac{L^2\,p\,(-2[Z-3L]^0)\,(3\,L-Z)^2}{2}\right)$$

$$u = \frac{K_{xy}}{E}\left(\frac{Z^4\,p}{12} + 2.421875\,L^2\,Z^2\,p + \frac{p\,(-2[Z-3L]^0)\,(3\,L-Z)^4}{24} - 1.164062\,L\,Z^3\,p + \frac{L\,p\,(-2[Z-L]^0)\,(L-Z)^3}{4} - \frac{L^2\,p\,(-2[Z-3L]^0)\,(3\,L-Z)^2}{2}\right)$$

Figure 5.9: Moment and Deflection equations

7. **Custom Calculator Tab:** View a table of unitless deflection values along the beam- Figure 5.10.

Z/L	v	v'	u	u'
0	0	0	0	0
0.1000	-0.0231	-0.4498	0.0231	0.4498
0.2000	-0.0877	-0.8317	0.0877	0.8317
0.3000	-0.1872	-1.1478	0.1872	1.1478
0.4000	-0.3151	-1.4001	0.3151	1.4001
0.5000	-0.4652	-1.5905	0.4652	1.5905
0.6000	-0.6312	-1.7211	0.6312	1.7211
0.7000	-0.8075	-1.7938	0.8075	1.7938
0.8000	-0.9881	-1.8107	0.9881	1.8107
0.9000	-1.1678	-1.7737	1.1678	1.7737
1.0000	-1.3411	-1.6849	1.3411	1.6849
1.1000	-1.5036	-1.5612	1.5036	1.5612

Z/L	v	v'	u	u'
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3.0000	-1.1172	1.8984	1.1172	-1.8984
3.1000	-0.9321	1.7993	0.9321	-1.7993
3.2000	-0.7580	1.6800	0.7580	-1.6800
3.3000	-0.5968	1.5405	0.5968	-1.5405
3.4000	-0.4506	1.3809	0.4506	-1.3809
3.5000	-0.3213	1.2012	0.3213	-1.2012
3.6000	-0.2110	1.0012	0.2110	-1.0012
3.7000	-0.1217	0.7812	0.1217	-0.7812
3.8000	-0.0554	0.5409	0.0554	-0.5409
3.9000	-0.0142	0.2805	0.0142	-0.2805
4.0000	-0.0000	-0.0000	0.0000	0.0000

Figure 5.10: Custom Deflection Calculator

For more details, a walkthrough video can be found here.

6. Code Architecture and Design Considerations

- **Modularity and OOP:** Each load and support is implemented as a separate class.
- **Symbolic vs. Numerical Computation:** Analytical expressions are derived symbolically and then evaluated numerically.
- **Graphical Feedback:** Custom plots and equation displays provide immediate visual feedback.
- **Global Variables:** Global variables manage axes and beam parameters.

7. Potential Enhancements

- Expanding error handling.
- Adding more support classes -like hinges.
- Adding symmetry and anti-symmetry options.

---

8. Conclusion

This report presents the development of a MATLAB-based tool for structural beam analysis under various loading and support conditions. By integrating symbolic mathematics with object-oriented programming, the tool efficiently computes bending moments, shear forces, and deflections while providing clear graphical representations. The comparison between conventional and singularity function methods highlights the advantages of the latter in simplifying complex beam loading scenarios. Additionally, the tool's graphical interface enhances user interaction and result interpretation. Future enhancements could include expanded support options, improved error handling, and additional load configurations, making this tool even more robust for engineering applications.

References

[1] T.H.G. Megson, *Aircraft Structures for Engineering Students*, Elsevier, 2016.

[2] MATLAB File Exchange, "Circular Arrow Function."

A. Full MATLAB Code Listings

Below are the most important MATLAB codes used in the project.

A.1. sing.m

```
% sing.m
% Singularity Function
function s = sing(z, x, n)
    syms L;
    assume(L > 0)
    assume(z > 0)
    m = z - x;
    s = (m).^n .* (sign(m)/2 + 1/2);
end
```

Listing 1: sing.m

A.2. concentrated_force.m

```
% Moment due to concentrated force using singularity
% R is the value force
% P is the force position
function x = concentrated_force(p,R)
    syms Z;
    if p == 0
        x = R.*Z;
    else
        x = R.*sing(Z,p,1);
    end
end
```

Listing 2: concentrated_force.m

A.3. concentrated_moment.m

```
% Moment due to concentrated moment using singularity
% M is the value moment
% P is the moment position
function x = concentrated_moment(p,M)
    syms Z;
    if p == 0
        x = M;
    else
        x = M.*sing(Z,p,0);
    end
end
```

Listing 3: concentrated_moment.m

A.4. s_rec.m

```
% Moment due to rectangle using singularity
% pi is the intial position
% Value is the load value
function x = s_rec(p_i,value)
    syms Z ;
    x = (value./2).*sing(Z,p_i,2);
end
```

Listing 4: s_rec.m

A.5. s_tri.m

```
% Moment due a normal triangle using singularity ---> after corrections
% (all triangles must reach the end of the beam)
% pi is the intial position
% Value is the load value
% eb is the Beam length
function x = s_tri(p_i,value,eb)
    syms Z p L;
    if p_i==eb
        x=0;
    else
        x = (value./(eb-p_i)).*sing(Z,p_i,3)./6;
    end
end
```

Listing 5: s_tri.m

A.6. fixed.m

```
% Class for Fixed supports
% syntax:
%fixed(position,plane,index)
% - Position: Coeff of L ; Example: 2 for position 2*L
% - plane: input (1) for yz plane ; input (2) for xz plane
% - index: a number to indicate the symbolic representation of the
% reaction force and moment
classdef fixed
    properties
        p; % Position
        R; % Reaction Force
        M; % Reaction Moment
        n; % Load index
        u; % u condition
        v; % v condition
        u_dash; % u' condition
        v_dash; % v' condition
        plane;% Index of the plane
        final_moment;% Final moment due to supports reaction
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
```

```
        f;%temp figure handle
    end
    methods
        function obj = fixed(pos,plane,o)
            global yz_axes xz_axes;
            if plane == 1
                axes1 = yz_axes;
            elseif plane == 2
                axes1 = xz_axes;
            end
            axes1.NextPlot = "add";
            syms L p;
            s = "R" + o;
            ss = "M" + o;
            pos = subs(pos,L,1)*L;
            assume(L>0)
            assume(p>0)
            obj.plane = plane; % It acts in both planes but each has its own unknowns
            obj.p = pos;
            obj.R = sym(s);
            obj.M = sym(ss);
            obj.n = o;
            if plane == 1
               obj.v = 0;
               obj.v_dash = 0;
               obj.u = nan;
               obj.u_dash = nan;
            elseif plane == 2
               obj.u = 0;
               obj.u_dash = 0;
               obj.v = nan;
               obj.v_dash = nan;
            else
                error("wrong plane index")
            end
            x1 = concentrated_force(pos,-obj.R);
            x2 = concentrated_moment(pos,obj.M);
            obj.final_moment = x1+x2;
            obj.fy = obj.R;
            obj.m0 = obj.fy*pos+obj.M;
            % plotting
            pos = double(pos./L);
            if pos == 0
                x = [0 -0.125 -0.125 0];
                y = [1 1 -1 -1].*0.5;
                pl1 = fill(axes1,x,y,'r');
                [pl2, obj.f] = circular_arrow(axes1, 1 , [1-0.5 0], 180 , 90 , -1 , 'r');
                pl3 = text(axes1,(pos-0.5)*2,0,string(obj.M),"HorizontalAlignment","left",'Color','b','FontWeight','bold');
            else
                x = 2.*[pos pos+0.125/2 pos+0.125/2 pos];
                y = [1 1 -1 -1].*0.5;
                pl1 = fill(axes1,x,y,'r');
                [pl2,obj.f] = circular_arrow(axes1, 1 , [2.*pos-0.5 0], 0 , 90 , -1 , 'r');
                pl3 = text(axes1,(pos+0.5)*2,0,string(obj.M),"HorizontalAlignment","right",'Color','b','FontWeight','bold');
```

```
            end

            x_v = pos.*ones([10 1]);
            y_v = linspace(0,-1,10);
            x_arr = linspace(0,0.1,10);
            y_arr = 3.*x_arr;

            pl4 = plot(axes1,2.*x_v,y_v,'-k',2.*(-x_arr+pos),-y_arr,'-k',2.*(x_arr+pos)
                ,-y_arr,'-k');
            pl5 = text(axes1,2*pos,-1.25,string(obj.R),"HorizontalAlignment","center",'
                Color','b','FontWeight','bold');
            obj.pl = [pl1 pl2 pl3 pl4' pl5];

        end
    end
end
```

Listing 6: fixed.m

A.7. roller.m

```
%Class for roller support
% syntax:
% roller(position,plane,index)
% - Position: Coeff of L ; Example: 2 for position 2*L
% - plane: input (1) for yz plane ; input (2) for xz plane
% - index: a number to indicate the symbolic representation of the
% reaction force and moment
classdef roller
    properties
        p; % Position
        R; % Reaction Force
        u; % u condition
        v; % v condition
        n; % Load index
        u_dash; % u' condition
        v_dash; % v' condition
        plane;% Index of the plane
        final_moment;% Final moment due to supports reaction
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = roller(pos,plane,o)
            global yz_axes xz_axes;
            if plane == 1
                axes1 = yz_axes;
            elseif plane == 2
                axes1 = xz_axes;
            end
            axes1.NextPlot = "add";
            syms L p;
            s = "R" + o;
            pos = subs(pos,L,1)*L;
            assume(L>0)
```

```
            assume(p>0)
            obj.plane = plane;
            obj.p = pos;
            obj.R = sym(s);
            obj.n = o;
            if plane == 1
               obj.v = 0;
               obj.u = nan;
            elseif plane == 2
               obj.u = 0;
               obj.v = nan;
            else
                error("wrong plane index")
            end
            obj.u_dash = nan;
            obj.v_dash = nan;
            x1 = concentrated_force(pos,-obj.R);
            obj.final_moment = x1;
            obj.fy = obj.R;
            obj.m0 = obj.fy*pos;
            pos = double(pos./L);
            r = 0.2;
            theta = linspace(0,2*pi,20);
            x = r.*cos(theta);
            y = r.*sin(theta);

            pl1 = fill(axes1,(x+2.*pos),y-r,'r');
            pl2 = text(axes1,2*pos,-0.75,string(obj.R),"HorizontalAlignment","center",'
                Color','r','FontWeight','bold');
            obj.pl = [pl1 pl2];
        end
    end
end
```

Listing 7: roller.m

A.8. simple.m

```
%Class for Simple supports
% syntax:
% simple(position,plane,index)
% - Position: Coeff of L ; Example: 2 for position 2*L
% - plane: input (1) for yz plane ; input (2) for xz plane
% - index: a number to indicate the symbolic representation of the
% reaction force and moment
classdef simple
    properties
        p; % Position
        R; % Reaction Force
        u; % u condition
        v; % v condition
        n; % Load index
        u_dash; % u' condition
        v_dash; % v' condition
        plane;% Index of the plane
        final_moment;% Final moment due to supports reaction
```

```
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = simple(pos,plane,o)
            global yz_axes xz_axes;
            if plane == 1
                axes1 = yz_axes;
            elseif plane == 2
                axes1 = xz_axes;
            end
            axes1.NextPlot = "add";
            syms L p;
            s = "R" + o;
            pos = subs(pos,L,1)*L;
            assume(L>0)
            assume(p>0)
            obj.plane = plane;
            obj.p = pos;
            obj.R = sym(s);
            if plane == 1
               obj.v = 0;
               obj.u = nan;
            elseif plane == 2
               obj.u = 0;
               obj.v = nan;
            else
                error("wrong plane index")
            end
            obj.n = o;
            obj.u_dash = nan;
            obj.v_dash = nan;
            x1 = concentrated_force(pos,-obj.R);
            obj.final_moment = x1;
            obj.fy = obj.R;
            obj.m0 = obj.fy*pos;
            pos = 2.*double(pos./L);
            x = [pos pos+0.25 pos-0.25];
            y = [0 -0.25 -0.25];

            pl1 = fill(axes1,x,y,'r');
            pl2 = text(axes1,pos,-0.5,string(obj.R),"HorizontalAlignment","center",'
                Color','r','FontWeight','bold');
            obj.pl = [pl1 pl2];
        end
    end
end
```

Listing 8: simple.m

A.9. con_f.m

```
% Class for Concenterated Forces
% syntax:
% con_f(Position,Magnitude,direction,plane,Preview)
% - Position: Coeff of L ; Example: 2 for position 2*L
% - Magnitude: Coeff of p*L ; Example: 2 for a force of 2*p*L
% - direction: is "down" for downward force or "up" for upward force
% - plane: input (1) for yz plane ; input (2) for xz plane
% - Preview: (OPTIONAL): input True for plotting on the preview axes
% or False to plot on the original axis
classdef con_f
    properties
        p; % Position
        R; % Magnitude
        final_moment; % Final moment due to concentrated force
        pl; % All plots of the object
        fy; % Resultant force in the upward direction
        m0; % Moment about z=0
        plane; % Index of the plane
    end
    methods
        function obj = con_f(pos,R,dir,plane,isprev)
            global yz_axes xz_axes prevaxes;
            if nargin<5
                isprev = false;
            end
            if plane == 1 && ~isprev
                axes1 = yz_axes;
            elseif plane == 2 && ~isprev
                axes1 = xz_axes;
            elseif isprev
                axes1 = prevaxes;
            end
            syms p L;
            pos = subs(pos,L,1)*L;
            R = subs(R,p,1);
            R = subs(R,L,1)*p*L;
            axes1.NextPlot = "add";
            obj.plane = plane;
            if strcmpi(dir,"up")
                si = 1;
                obj.fy = R;
                R = -R;
            elseif strcmpi(dir,"down")
                si = -1;
                obj.fy = -R;
            end
            assume(L>0)
            assume(p>0)
            obj.p = pos;
            obj.R = R;
            x1 = concentrated_force(pos,obj.R);
            obj.final_moment = x1;
            obj.m0 = obj.fy*pos;
            % plotting
            position = double(pos./L);
            magnitude = double(R./p./L);
```

```
            x_v = position.*ones([10 1]);
            y_v = linspace(0,magnitude,10);
            x_arr = linspace(0,0.1,10);
            y_arr = 3.*x_arr;
            pl1 = plot(axes1,2.*x_v,y_v,'-k',2.*(-si.*x_arr+position),-si.*y_arr,'-k'
                ,2.*(si.*x_arr+position),-si.*y_arr,'-k');
            obj.pl = pl1';
        end
    end
end
```

Listing 9: con_f.m

A.10. con_m.m

```
% Class for concentrated Moment
% syntax:
% con_m(Position,Magnitude,direction,plane,Preview)
% - Position: Coeff of L ; Example: 2 for position 2*L
% - Magnitude: Coeff of p*L^2 ; Example: 2 for a force of 2*p*L^2
% - direction: is 'cw' for Clockwise Moment or 'ccw' for
% Counterclockwise Moment
% - plane: input (1) for yz plane ; input (2) for xz plane
% - Preview: (OPTIONAL): input True for plotting on the preview axes
% or False to plot on the original axis
classdef con_m
    properties
        p; % Position
        M; % Magnitude
        final_moment; % Final moment due to concentrated moment
        pl; % All plots of the object
        plane; % Index of the plane
        fy; % Resultant force in the upward direction (always = 0)
        m0; % Moment about z=0
        f; %figure handle
    end
    methods
        function obj = con_m(pos,m,dir,plane,isprev)
            global yz_axes xz_axes prevaxes;
            if nargin<5
                isprev = false;
            end
            if plane == 1 && ~isprev
                axes1 = yz_axes;
            elseif plane == 2 && ~isprev
                axes1 = xz_axes;
            elseif isprev
                axes1 = prevaxes;
            end
            axes1.NextPlot = "add";
            obj.plane = plane;
            syms p L;
            pos = subs(pos,L,1)*L;
            m = subs(m,p,1);
            m = subs(m,L,1)*p*L^2;
            assume(L>0)
```

```
            assume(p>0)
            obj.p = pos;
            if strcmpi(dir,"ccw")
                obj.M = m;
                mark = '+';
                angle = 180;
                lw = 2;
                s = 50;
            elseif strcmpi(dir,"cw")
                obj.M = -m;
                mark = '.';
                angle = 0;
                lw = 3;
                s = 100;
            end
            x1 = concentrated_moment(pos,obj.M);
            obj.final_moment = x1;
            obj.fy = 0;
            obj.m0 = obj.M;
            % plotting
            position = double(pos./L);
            magnitude = double(obj.M./p./L^2);
            [obj.pl,obj.f] = circular_arrow(axes1, 0.25 , [2.*position 0], angle , 180
                , -sign(magnitude));
            pl2 = scatter(axes1,2.*position,0,s,'r',mark,'LineWidth',lw);
            obj.pl = [obj.pl pl2];
        end
    end
end
```

Listing 10: con_m.m

A.11. rec.m

```
% Class for Rectangle distriputed load
% syntax:
% rec(p_i,p_f,v,plane,isprev)
% - p_i: initial position; Coeff of L ; Example: 2 for position 2*L
% - p_f: final position; Coeff of L ; Example: 4 for position 4*L
% - v: value is the Coeff of p ; Example: 2 for load of 2*p
% - plane: input (1) for yz plane ; input (2) for xz plane
% - isprev: (OPTIONAL): True (to plot original load only) or
% False (to plot original & correction load)
classdef rec
    properties
        start; % Starting Position of Rectangle
        ending_beam; % Length of the beam
        value; % Value of the distributed load
        corr_start; % Starting position of the correction
        plane;% Index of the plane
        final_moment;% Final moment due to concentrated moment
        % plotting stuff
        y_value;%numerical value for load
        domain;% z domain for original load
        domain_corr1;% z domain for correction1 load
        domain_corr2;% z domain for correction2 load
```

```
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = rec(p_i,pf,v,plane,isprev,istrap,isrev)
            global eb yz_axes xz_axes prevaxes corraxes;
            axes2 = corraxes;
            if nargin<6
                istrap = false;
            end
            if nargin<7
                isrev = false;
            end
            if nargin<5
                isprev = true;
            end

            if plane == 1 && isprev
                axes1 = yz_axes;
            elseif plane == 2 && isprev
                axes1 = xz_axes;
            elseif ~isprev
                axes1 = prevaxes;
            end

            syms L p;
            p_i = subs(p_i,L,1)*L;
            pf = subs(pf,L,1)*L;
            v = subs(v,p,1)*p;
            assume(L>0)
            assume(p>0)
            obj.start = p_i;
            obj.plane = plane;
            obj.ending_beam = eb;
            obj.corr_start = pf;
            obj.value = v;
            x1 = s_rec(p_i,obj.value);
            x2 = s_rec(pf,obj.value);
            obj.final_moment = x1-x2;
            obj.fy = -(pf-p_i)*v;
            obj.m0 = obj.fy*(p_i+0.5*(pf-p_i));

            % plotting
            pl1=[];
            pl2=[];
            pl3=[];
            pl4=[];
            pl5=[];
            y_plot = linspace(0,double(obj.value./p),100);
            obj.domain = double(obj.start./L):0.1:double(pf/L);
            obj.domain_corr1 = double(obj.start./L):0.1:double(eb/L);
            obj.domain_corr2 = double(pf./L):0.1:double(eb/L);
            obj.y_value = double(obj.value./p).*ones(size(obj.domain));
            y_value1 = double(obj.value./p.*ones(size(obj.domain_corr1)));
            y_value2 = double(obj.value./p.*ones(size(obj.domain_corr2)));
            x_f1 = p_i./L.*ones(size(y_plot));
            x_f2 = pf./L.*ones(size(y_plot));
```

```
            x_f3 = eb./L.*ones(size(y_plot));
            axes1.NextPlot = "add";
            obj.pl = [];
            % plot original load
            if ~istrap
                if ~isrev
                    pl1 = plot(axes1,2.*obj.domain,obj.y_value,'-k', 2.*x_f1,y_plot,'-k'
                        ,2.*x_f2,y_plot,'-k','LineWidth',1);
                    obj.pl = [obj.pl pl1'];
                end
            end
                % plot correction
                if ~isprev
                    axes2.NextPlot = "add";
                    pl2 = plot(axes2,2.*obj.domain_corr1,y_value1,'-b',2.*obj.
                        domain_corr2,-y_value2,'-g',2.*x_f1,y_plot,'-b', 2.*x_f3,y_plot,
                        '-b',2.*x_f2,-y_plot,'-g', 2.*x_f3,-y_plot,'-g','LineWidth',1);
                    obj.pl = [obj.pl pl2'];
                end

            % arrows for original
            if ~istrap
                if ~isrev
                    c=1;
                    for i = min(obj.domain)+0.25:0.25:max(obj.domain)-0.25
                        x_v = i.*ones([10 1]);
                        y_v = linspace(0,obj.y_value(1),10);
                        pl3(c) = plot(axes1,2.*x_v,y_v,'-k');
                        c = c+1;
                    end
                    obj.pl = [obj.pl pl3];
                end
            end
                if ~isprev
                    axes2.NextPlot = "add";
                    c = 1;
                    for i = min(obj.domain)+0.25:0.25:obj.domain_corr1(end)-0.25
                            x_v = i.*ones([10 1]);
                            y_v = linspace(0,obj.y_value(1),10);
                            pl4(c) = plot(axes2,2.*x_v,y_v,'-k');
                        if i > obj.domain_corr2(1)
                            pl5(c) = plot(axes2,2.*x_v,-y_v,'-k');
                            if i == obj.domain_corr1(end)-0.25
                                obj.pl = [obj.pl pl5];
                            end
                        end
                        c = c+1;
                    end
                    obj.pl = [obj.pl pl4];
                end
        end
    end
end
```

Listing 11: rec.m

A.12. n_tri.m

```
% Class for normal triangle
% syntax:
% n_tri(p_i,p_f,v_f,plane,isprev)
% - p_i: initial position; Coeff of L ; Example: 2 for position 2*L
% - p_f: final position; Coeff of L ; Example: 4 for position 4*L
% - v_f: final value is the Coeff of p ; Example: 2 for load of 2*p
% - plane: input (1) for yz plane ; input (2) for xz plane
% - isprev: (OPTIONAL): True (to plot original load only) or
% False (to plot original & correction load)
classdef n_tri
    properties
        start; % Starting Position
        ending_tri; % Ending Position of Original Load
        ending_beam; % Length of the beam
        value_tri; % Value of Correction's large triangle
        value_rec; % Value of Correction's rectangle
        value_tri2; % Value of Correction's small triangle
        corr_start; % Starting Position of Correction's small triangle
        plane;% Index of the plane
        final_moment;% Final moment due to concentrated moment
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = n_tri(p_i,pf,vf,plane,isprev,istrap)
            global eb yz_axes xz_axes corraxes prevaxes;

            if nargin<6
                istrap = false;
            end
            if nargin<5
                isprev = true;
            end

            if plane == 1 && isprev
                axes1 = yz_axes;
            elseif plane == 2 && isprev
                axes1 = xz_axes;
            elseif ~isprev
                axes1 = prevaxes;
            end
            axes1.NextPlot = "add";

            syms p L;
            p_i = subs(p_i,L,1)*L;
            pf = subs(pf,L,1)*L;
            vf = subs(vf,p,1)*p;
            assume(L>0)
            assume(p>0)
            obj.start = p_i;
            obj.plane = plane;
            obj.ending_tri = pf;
            obj.ending_beam = eb;
            obj.value_tri = (vf./(pf-p_i)).*(eb-p_i);
            obj.corr_start = pf;
```

```
            obj.value_rec = -vf;
            obj.value_tri2 = -(obj.value_tri-vf);
            if pf~=eb
                x1 = s_tri(p_i,obj.value_tri,eb);
                x2 = s_tri(obj.corr_start,obj.value_tri2,eb);
                x3 = s_rec(obj.corr_start,obj.value_rec);
                obj.final_moment = x1-x2-x3;
            else
                x1 = s_tri(p_i,vf,eb);
                obj.final_moment = x1;
            end
            obj.fy = -0.5*(pf-p_i)*vf;
            obj.m0 = obj.fy*(p_i+2*(pf-p_i)/3);
            % plotting
            pl1=[];
            pl2=[];
            pl3=[];
            pl4=[];
            pl5=[];
            pl6=[];
            p_i = double(p_i./L);
            pf = double(pf./L);
            eb2 = double(eb./L);
            vf = double(vf./p);
            pol = polyfit([p_i pf],[0 vf],1);
            x = linspace(p_i,pf,10);
            y = polyval(pol,x);
            y_vf = linspace(0,vf,10);
            x_pf = pf.*ones(size(y_vf));
            obj.pl = [];
            if ~istrap
                axes(axes1)
                pl1 = plot(axes1,2.*x,y,'-b',2.*x_pf,y_vf,'-b');
                c=1;
                for i = p_i+0.25:0.25:pf-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,polyval(pol,i),10);
                    pl2(c) = plot(axes1,2.*x_v,y_v,'-k');
                    c = c+1;
                end
                obj.pl = [pl1' pl2];
                if ~isprev
                    % correction plotting
                    corraxes.NextPlot = "add";

                    x_corr1 = linspace(p_i,eb2,10);
                    y_corr1 = polyval(pol,x_corr1);
                    y_vf_corr1 = linspace(0,y_corr1(end),10);
                    x_pf_corr1 = eb2.*ones(size(y_vf_corr1));
                    pl3 = plot(corraxes,2.*x_corr1,y_corr1,'-b',2.*x_pf_corr1,y_vf_corr1
                        ,'-b');
                    c = 1;
                    for i = p_i+0.25:0.25:eb2-0.25
                        x_v = i.*ones([10 1]);
                        y_v = linspace(0,polyval(pol,i),10);
                        pl4(c) = plot(corraxes,2.*x_v,y_v,'-k');
                        c = c+1;
                    end
```

```
                    x_corr2 = linspace(pf,eb2,10);
                    y_corr2 = -polyval(pol,x_corr2);
                    y_vi_corr2 = linspace(0,-vf,10);
                    y_vf_corr2 = linspace(0,y_corr2(end),10);
                    x_pi_corr2 = pf.*ones(size(y_vi_corr2));
                    x_pf_corr2 = eb2.*ones(size(y_vf_corr2));
                    pl5 = plot(corraxes,2.*x_corr2,y_corr2,'-g',2.*x_pi_corr2,y_vi_corr2
                        ,'-g',2.*x_pf_corr2,y_vf_corr2,'-g');
                    c = 1;
                    for i = pf+0.25:0.25:eb2-0.25
                        x_v = i.*ones([10 1]);
                        y_v = linspace(0,-polyval(pol,i),10);
                        pl6(c) = plot(corraxes,2.*x_v,y_v,'-k');
                        c = c+1;
                    end
                    obj.pl = [obj.pl pl3' pl4 pl5' pl6];
                end
            end
        end
    end
end
```

Listing 12: n_tri.m

A.13. rev_tri.m

```
% Reverse triangle class
% syntax:
% rev_tri(p_i,p_f,v_i,plane,isprev)
% - p_i: initial position; Coeff of L ; Example: 2 for position 2*L
% - p_f: final position; Coeff of L ; Example: 4 for position 4*L
% - v_i: initial value is the Coeff of p ; Example: 2 for load of 2*p
% - plane: input (1) for yz plane ; input (2) for xz plane
% - isprev: (OPTIONAL): True (to plot original load only) or
% False (to plot original & correction load)
classdef rev_tri
    properties
        start; % strating position
        ending_tri; % ending position
        ending_beam; % beam length
        value_tri; % value of correction's large triangle
        value_rec; % correction rectangle value
        value_tri2; % value of correction's small triangle
        corr_start; % strating position of the small triangle correction
        plane;% Index of the plane
        final_moment;% Final moment due to concentrated moment
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = rev_tri(p_i,pf,vi,plane,isprev,istrap,vf)
            global eb yz_axes xz_axes prevaxes corraxes;
            axes2 = corraxes;
            if nargin<6
```

```
                istrap = false;
            end
            if nargin<7
                vf = 0;
            end
            if nargin<5
                isprev = true;
            end

            if plane == 1 && isprev
                axes1 = yz_axes;
            elseif plane == 2 && isprev
                axes1 = xz_axes;
            elseif ~isprev
                axes1 = prevaxes;
            end

            axes1.NextPlot = "add";
            syms p L;
            p_i = subs(p_i,L,1)*L;
            pf = subs(pf,L,1)*L;
            vi = subs(vi,p,1)*p;
            assume(L>0)
            assume(p>0)
            obj.start = p_i;
            obj.plane = plane;
            obj.ending_tri = pf;
            obj.ending_beam = eb;
            obj.value_tri = (vi./(-p_i+pf)).*(eb-p_i);
            obj.corr_start = p_i;
            obj.value_rec = vi;
            obj.value_tri2 = obj.value_tri-vi;
            x1 = s_tri(p_i,obj.value_tri,eb);
            x2 = s_tri(pf,obj.value_tri2,eb);
            x3 = s_rec(obj.corr_start,obj.value_rec);
            obj.final_moment = -x1+x2+x3;
            obj.fy = -0.5*(pf-p_i)*vi;
            obj.m0 = obj.fy*(p_i+(pf-p_i)/3);


            % plotting
            obj.pl =[];pl1=[];pl2=[];pl3=[];pl4=[]; pl5=[];pl6=[];pl7=[];pl8=[];pl9=[];
            p_i = double(p_i./L);
            pf = double(pf./L);
            eb2 = double(eb./L);
            vi = double(vi./p);
            pol = polyfit([p_i pf],[vi 0],1);
            x = linspace(p_i,pf,10);
            y = polyval(pol,x);
            y_vf = linspace(0,vi,10);
            x_pi = p_i.*ones(size(y_vf));
            if ~istrap
               pl1= plot(axes1,2.*x,y,'-b',2.*x_pi,y_vf,'-b');
               obj.pl = [obj.pl pl1'];
               c=1;
                for i = p_i+0.25:0.25:pf-0.25
                   x_v = i.*ones([10 1]);
                   y_v = linspace(0,polyval(pol,i),10);
```

```
                pl2(c) = plot(axes1,2.*x_v,y_v,'-k');
                c = c+1;
            end
            obj.pl = [obj.pl pl2];
        end
        if ~isprev
            axes2.NextPlot = "add";
            % correction plotting
            pol = polyfit([p_i pf],[0 vi],1);
            x_corr01 = linspace(p_i,pf,10);
            x_corr02 = linspace(pf,eb2,10);
            y_corr1 = [-polyval(pol,x_corr01) -polyval(pol,x_corr02)-vf];
            y_vf_corr01 = linspace(0,y_corr1(end),10);
            y_vf_corr02 = linspace(0,y_corr1(11),10);
            x_pf_corr01 = eb2.*ones(size(y_vf_corr01));
            x_pf_corr02 = pf.*ones(size(y_vf_corr02));
            pl3 = plot(axes2,2.*[x_corr01 x_corr02],y_corr1,'-k',2.*x_pf_corr01,
                y_vf_corr01,'-k',2.*x_pf_corr02,y_vf_corr02,'-k');
            c=1;
            for i = p_i+0.25:0.25:pf-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,-polyval(pol,i),10);
                    pl4(c) = plot(axes2,2.*x_v,y_v,'-k');
                    c = c+1;
            end
            c=1;
            for i = pf:0.25:eb2-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,-polyval(pol,i)-vf,10);
                    pl5(c) = plot(axes2,2.*x_v,y_v,'-k');
                    c = c+1;
            end

            x_corr2 = linspace(pf,eb2,10);
            y_corr2 = polyval(pol,(x_corr2+p_i-pf))+vf+vi;
            y_vf_corr2 = linspace(vf+vi,y_corr2(end),10);
            x_pf_corr2 = eb2.*ones(size(y_vf_corr2));
            pl6=plot(axes2,2.*x_corr2,y_corr2,'-k',2.*x_pf_corr2,y_vf_corr2,'-k');

            % rec
            y_vf2 = linspace(vf,vi+vf,10);
            y_rec = x_corr01.^0.*vi+vf;
            x_eb = eb2.*ones(size(y_vf2));
            pl7 = plot(axes2,2.*x_corr01,y_rec,'-k',2.*x_pi,y_vf2,'-k',2.*x_eb,
                y_vf2,'-k');
            c=1;
            for i = p_i+0.25:0.25:pf-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,vi+vf,10);
                    pl8(c) = plot(axes2,2.*x_v,y_v,'-k');
                    c = c+1;
            end
            c=1;
            for i = pf:0.25:eb2-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,polyval(pol,(i+p_i-pf))+vf+vi,10);
                    pl9(c) = plot(axes2,2.*x_v,y_v,'-k');
                    c = c+1;
```

```
                end
                obj.pl = [obj.pl pl3' pl4 pl5 pl6' pl7' pl8 pl9];
            end
        end
    end
end
```

Listing 13: rev_tri.m

A.14. trap.m

```
%Class for trapezoid distributed load
% syntax:
% trap(p_i,p_f,v_i,v_f,plane,isprev)
% - p_i: initial position; Coeff of L ; Example: 2 for position 2*L
% - p_f: final position; Coeff of L ; Example: 4 for position 4*L
% - v_i: initial value is the Coeff of p ; Example: 2 for load of 2*p
% - v_f: final value is the Coeff of p ; Example: 3 for load of 3*p
% - plane: input (1) for yz plane ; input (2) for xz plane
% - isprev: (OPTIONAL): True (to plot original load only) or
% False (to plot original & correction load)
classdef trap
    properties
        start; %strating position for distributed load (pi)
        ending_beam; %beam length (eb)
        value1; %trapezoid value from left
        value2; %Trapezoid value from right
        corr_start; %Correction intatiating position
        plane; %Plane of action
        final_moment;% Final moment due to supports reaction
        pl;% All plots of the object
        fy;% Resultant force in the upward direction
        m0;% Moment about z=0
    end
    methods
        function obj = trap(p_i,pf,v1,v2,plane,isprev)
            if nargin<6
                isprev = true;
            end

            global eb yz_axes xz_axes corraxes prevaxes;
            axes2 = corraxes;
            if plane == 1 && isprev
                axes1 = yz_axes;
            elseif plane == 2 && isprev
                axes1 = xz_axes;
            elseif ~isprev
                axes1 = prevaxes;
            end
            axes1.NextPlot = "add";

            syms p L;
            p_i = subs(p_i,L,1)*L;
            pf = subs(pf,L,1)*L;
            v1 = subs(v1,p,1)*p;
            v2 = subs(v2,p,1)*p;
```

```
            assume(L>0)
            assume(p>0)
            obj.start = p_i;
            obj.plane = plane;
            obj.ending_beam = eb;
            obj.corr_start = pf;
            obj.value1 = v1;
            obj.value2 = v2;
            obj.pl=[];
            vi = double(v1./p);
            vf = double(v2./p);
            obj.fy = -0.5*(pf-p_i)*(v1+v2);
            if abs(vf)>abs(vi)
                x1 = rec(p_i,pf,vi,plane,false,true,false);
                x2 = n_tri(p_i,pf,vf-vi,plane,isprev,true);
                obj.final_moment = x1.final_moment+x2.final_moment;
                obj.pl = [obj.pl x1.pl x2.pl];
                m1 = -(pf-p_i)*(v1)*(p_i+0.5*(pf-p_i));
                m2 = -0.5*(pf-p_i)*(v2-v1)*(p_i+2*(pf-p_i)/3);
                obj.m0 = m1+m2;
            else
                x1 = rec(p_i,pf,vf,plane,isprev,false,true);
                x2 = rev_tri(p_i,pf,vi-vf,plane,isprev,true,vf);
                obj.final_moment = x1.final_moment+x2.final_moment;
                obj.pl = [obj.pl x1.pl x2.pl];
                m1 = -(pf-p_i)*(v2)*(p_i+0.5*(pf-p_i));
                m2 = -0.5*(pf-p_i)*(v1-v2)*(p_i+(pf-p_i)/3);
                obj.m0 = m1+m2;
            end
            p_i = double(p_i./L);
            pf = double(pf./L);
            eb2 = double(eb./L);

            pol = polyfit([p_i pf],[vi vf],1);
            x = linspace(p_i,pf,10);
            y = polyval(pol,x);
            y_vi = linspace(0,vi,10);
            y_vf = linspace(0,vf,10);
            x_pi = p_i.*ones(size(y_vi));
            x_pf = pf.*ones(size(y_vf));
            pl1 = plot(axes1,2.*x,y,'-c',2.*x_pi,y_vi,'-c',2.*x_pf,y_vf,'-c');
            c=1;
            pl2=[];pl3=[];pl4=[];pl5=[];pl6=[];
            for i = p_i+0.25:0.25:pf-0.25
                    x_v = i.*ones([10 1]);
                    y_v = linspace(0,polyval(pol,i),10);
                    pl2(c) = plot(axes1,2.*x_v,y_v,'-k');
                    c = c+1;
            end
            obj.pl = [obj.pl pl1' pl2];
            % correction
            if ~isprev
                axes2.NextPlot = "add";
                if abs(vf)>abs(vi)
                    % positive
                    x1 = linspace(p_i,eb2,10);
                    y1 = polyval(pol,x1);
                    y_vi1 = linspace(0,vi,10);
```

```
                    y_vf1 = linspace(0,y1(end),10);
                    x_pi1 = p_i.*ones(size(y_vi1));
                    x_pf1 = eb2.*ones(size(y_vf1));
                    pl3 = plot(axes2,2.*x1,y1,'-b',2.*x_pi1,y_vi1,'-b',2.*x_pf1,y_vf1,'-
                        b');
                    c=1;
                    for i = p_i+0.25:0.25:eb2-0.25
                        x_v = i.*ones([10 1]);
                        y_v = linspace(0,polyval(pol,i),10);
                        pl4(c) = plot(axes2,2.*x_v,y_v,'-k');
                        c = c+1;
                    end
                    % negative
                    x2 = linspace(pf,eb2,10);
                    y2 = -polyval(pol,x2);
                    y_vi2 = -linspace(0,vf,10);
                    y_vf2 = linspace(0,y2(end),10);
                    x_pi2 = pf.*ones(size(y_vi2));
                    x_pf2 = eb2.*ones(size(y_vf2));
                    pl5 = plot(axes2,2.*x2,y2,'-b',2.*x_pi2,y_vi2,'-b',2.*x_pf2,y_vf2,'-
                        b');
                    c=1;
                    for i = pf+0.25:0.25:eb2-0.25
                        x_v = i.*ones([10 1]);
                        y_v = linspace(0,-polyval(pol,i),10);
                        pl6(c) = plot(axes2,2.*x_v,y_v,'-k');
                        c = c+1;
                    end
                    obj.pl = [obj.pl pl3' pl4 pl5' pl6];
                end
            end
        end
    end
end
```

Listing 14: trap.m

A.15. Main_Solver.m

```
% Final Function
% Syntax:
% [Mx,My,Sx,Sy,v,v_dash,u,u_dash,solutions,isyz,isxz] = Structure_Project(supports,
    loads)
% Where:
% Outputs:
% Mx: Moment in Y-Z plane
% My: Moment in X-Z plane
% v: Deflection in Y direction
% v_dash: Slope of deflection in Y
% u: Deflection in X direction
% u_dash: Slope of deflection in X
% solutions: Struct for Values of Unknowns
% isyz: True if Y-Z plane has loads or supports
% isxz: True if X-Z plane has loads or supports
%
% Inputs:
```

```
% supports: A cell array containing all Supports
% loads: A cell array containing all Loads


function [Mx,My,Sx,Sy,v,v_dash,u,u_dash,solutions,isyz,isxz] = Main_Solver(supports,
    loads)
    %% Variables Declaration
    global eb w Is;

    syms L Z c1 c2 c3 c4 p E d t;

    assume(L>0)
    assume(p>0)
    assume(Z>0)

    if ~isempty(w) && ~isempty(Is)
        if strcmpi(w,'I')
            Ix = Is(1)*t*d^3;
            Iy = Is(2)*t*d^3;
            Ixy = Is(3)*t*d^3;
            K_bar = Ix*Iy-Ixy^2;
            kx = Ix/K_bar;
            ky = Iy/K_bar;
            kxy = Ixy/K_bar;
        elseif strcmpi(w,'k')
            kx = Is(1)/t/d^3;
            ky = Is(2)/t/d^3;
            kxy = Is(3)/t/d^3;
        end
    else
        syms kx ky kxy
    end

    %% Moment creation
    MMs=[]; %Symbolic moments
    RRs = []; %Symbolic Reactions

    z_u = [];z_v = [];z_ud = [];z_vd = [];
    u_value = [];v_value = [];ud_value = [];vd_value = [];

    fi = length(supports); % number of unknown reactions
    Mx = 0;My = 0;Fy = 0;Fx = 0;m01 = 0;m02 = 0;
    isyz = false;isxz = false;

    for i = 1:length(supports)
        c_conc = supports{i};
        if c_conc.plane == 1
            if c_conc.p~=eb
                Mx = Mx+ c_conc.final_moment;
            end
            Fy = Fy + c_conc.fy;
            m01 = m01+c_conc.m0;
            isyz = true;
        end
        if c_conc.plane == 2
            if c_conc.p~=eb
                My = My+ c_conc.final_moment;
            end
```

```
            Fx = Fx + c_conc.fy;
            m02 = m02+c_conc.m0;
            isxz = true;
        end
        if ~isnan(c_conc.u)
            z_u = [z_u;c_conc.p];%#ok<AGROW>
            u_value = [u_value;c_conc.u];%#ok<AGROW>
        end
        if ~isnan(c_conc.v)
            z_v = [z_v;c_conc.p];%#ok<AGROW>
            v_value = [v_value;c_conc.v];%#ok<AGROW>
        end
        if ~isnan(c_conc.u_dash)
            z_ud = [z_ud;c_conc.p];%#ok<AGROW>
            ud_value = [ud_value;c_conc.u_dash];%#ok<AGROW>
        end
        if ~isnan(c_conc.v_dash)
            z_vd = [z_vd;c_conc.p];%#ok<AGROW>
            vd_value = [vd_value;c_conc.v_dash];%#ok<AGROW>
        end
        if isa(c_conc,'fixed')
            MMs = [MMs c_conc.M];%#ok<AGROW>
        end
        RRs = [RRs c_conc.R];%#ok<AGROW>
    end

    for i = 1:length(loads)
        c_load = loads{i};
        if c_load.plane == 1
            Mx = Mx+ c_load.final_moment;
            Fy = Fy + c_load.fy;
            m01 = m01+c_load.m0;
            isyz = true;
        end
        if c_load.plane == 2
            My = My+ c_load.final_moment;
            Fx = Fx + c_load.fy;
            m02 = m02+ c_load.m0;
            isxz = true;
        end
    end

    unknowns = [c1 c2 c3 c4 RRs MMs];
    trans = -1./E.*[-kxy,kx;ky,-kxy]*[Mx;My];
    Mx_ph = Mx;
    My_ph = My;

    if Mx~=0
        Mx = children(Mx);
    end
    if My~=0
        My = children(My);
    end

    Mx1i = int(Mx,Z);
    Mx2i = int(Mx1i,Z);
    My1i = int(My,Z);
    My2i = int(My1i,Z);
```

```
    Mx1 = sum([Mx1i(:)]);
    Mx2 = sum([Mx2i(:)]);
    My1 = sum([My1i(:)]);
    My2 = sum([My2i(:)]);

    trans1 = -1./E.*[-kxy,kx;ky,-kxy]*[Mx1;My1];
    trans2 = -1./E.*[-kxy,kx;ky,-kxy]*[Mx2;My2];
    u2_dash = trans(1);
    v2_dash = trans(2);
    u_dash = trans1(1) + c1;
    v_dash = trans1(2) + c3;
    u = trans2(1) + c1*Z + c2;
    v = trans2(2) + c3*Z + c4;

    if isyz && ~isxz
        eqns = [subs(v_dash,Z,z_vd)==vd_value ; subs(v,Z,z_v)==v_value ; Fy == 0; m01 == 0];
    elseif isxz && ~isyz
        eqns = [subs(u_dash,Z,z_ud)==ud_value ; subs(u,Z,z_u)==u_value ; Fx == 0; m02 == 0];
    elseif isyz && isxz
        eqns = [subs(v_dash,Z,z_vd)==vd_value ; subs(v,Z,z_v)==v_value ; Fy == 0; m01 == 0 ;subs(u_dash,Z,z_ud)==ud_value ; subs(u,Z,z_u)==u_value ;Fx == 0; m02 == 0];
    end

    solutions = solve(eqns,unknowns);
    Mx = Mx_ph;
    My = My_ph;
    D = digits(7); %for better display

    for i = 1:length(unknowns)
        unknown.var = char(unknowns(i));
        current = vpa(solutions.(unknown.var));
        Mx = subs(Mx,unknowns(i),current);
        My = subs(My,unknowns(i),current);
        v = subs(v,unknowns(i),current);
        u = subs(u,unknowns(i),current);
        u_dash = subs(u_dash,unknowns(i),current);
        v_dash = subs(v_dash,unknowns(i),current);
    end
    Sx = diff(My,Z);
    Sy = diff(Mx,Z);
    digits(D);
end
```

Listing 15: Main_Solver.m