CIV 102 Project Team 106 Design Calculations

Tannaz Chowdhury, Hannah Moore, Katherine Shepherd

HAND CALCULATIONS LOAD CASE 1 DESIGN 0

The train is centred - Maximum Bending Moment

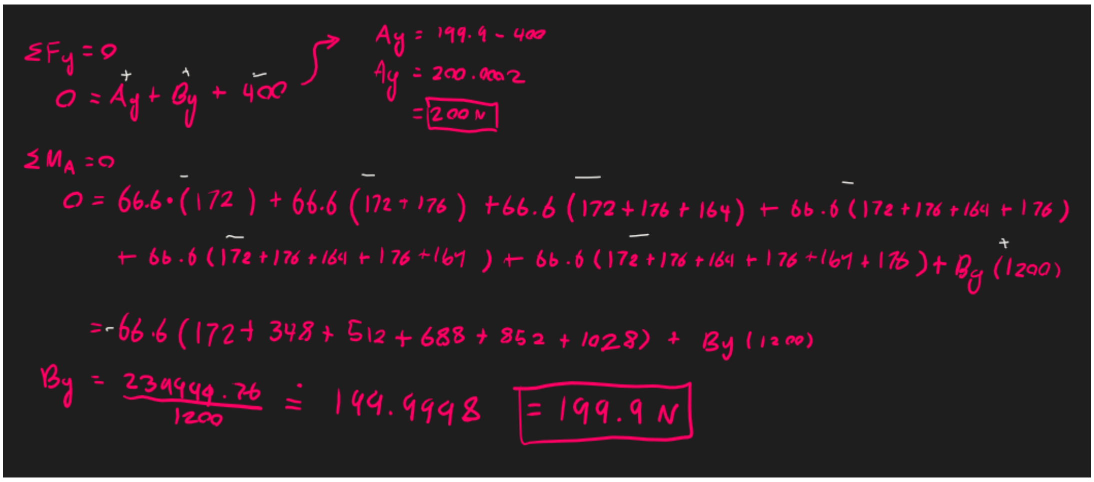


Figure 1: The reaction forces were calculated using the method of moments for improved accuracy, despite the potential application of symmetry to simplify the process.

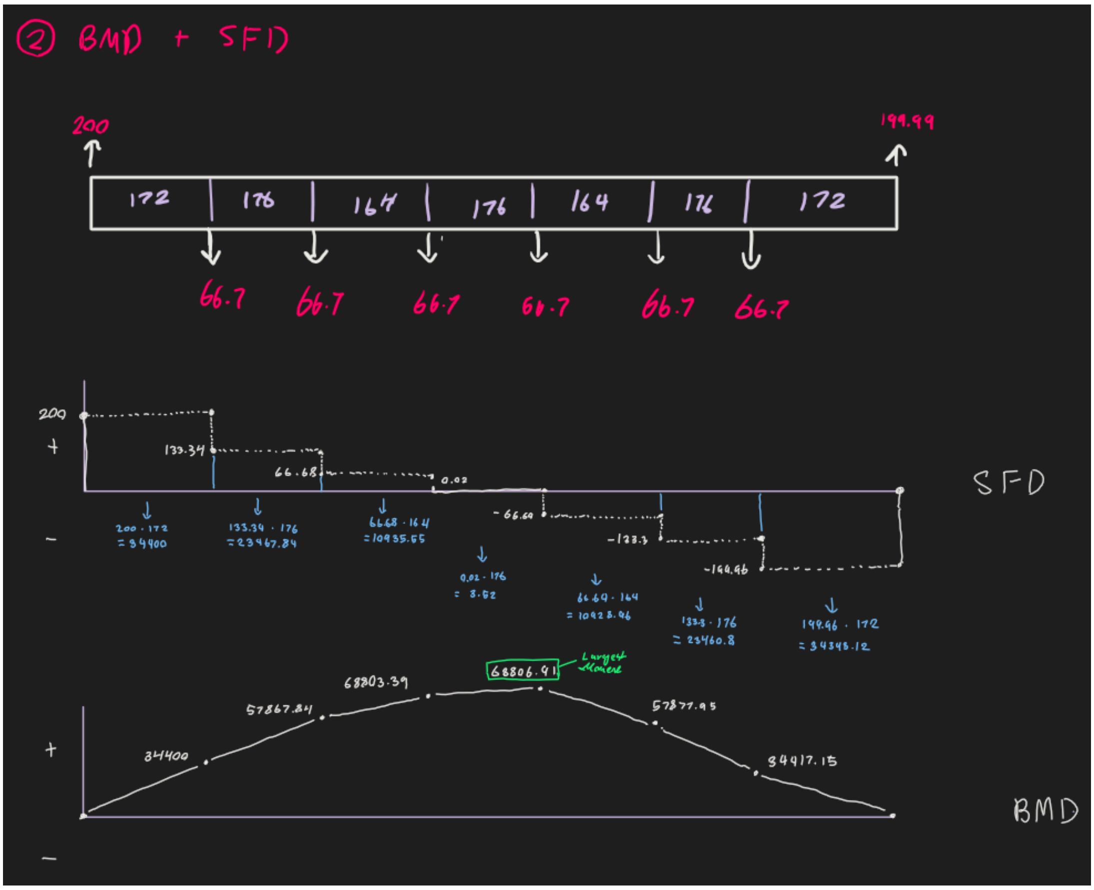


Figure 2: The diagrams illustrate the distribution of shear force and bending moment, with the maximum bending moment occurring at the center section of the beam.

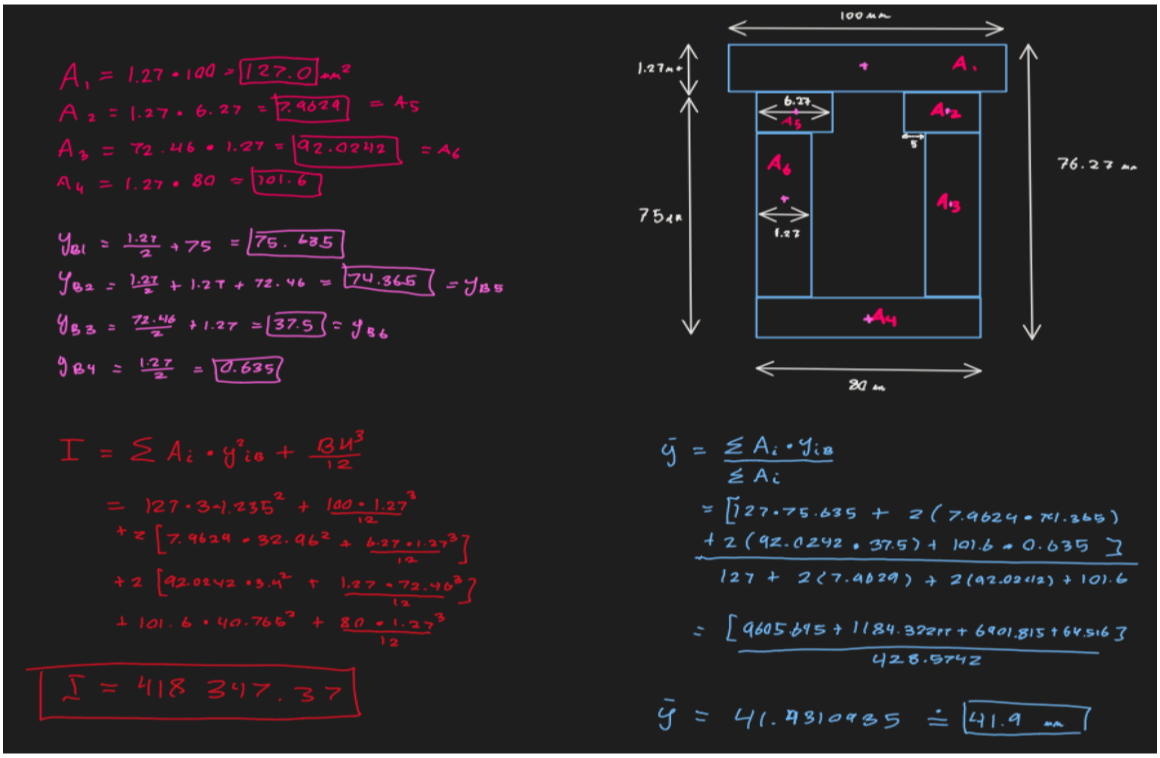


Figure 3: Cross-sectional properties, including moment (M), centroidal height (Y), and second moment of area (I), were calculated using first principles. Maximum decimal precision was maintained to enhance the accuracy of computational results.

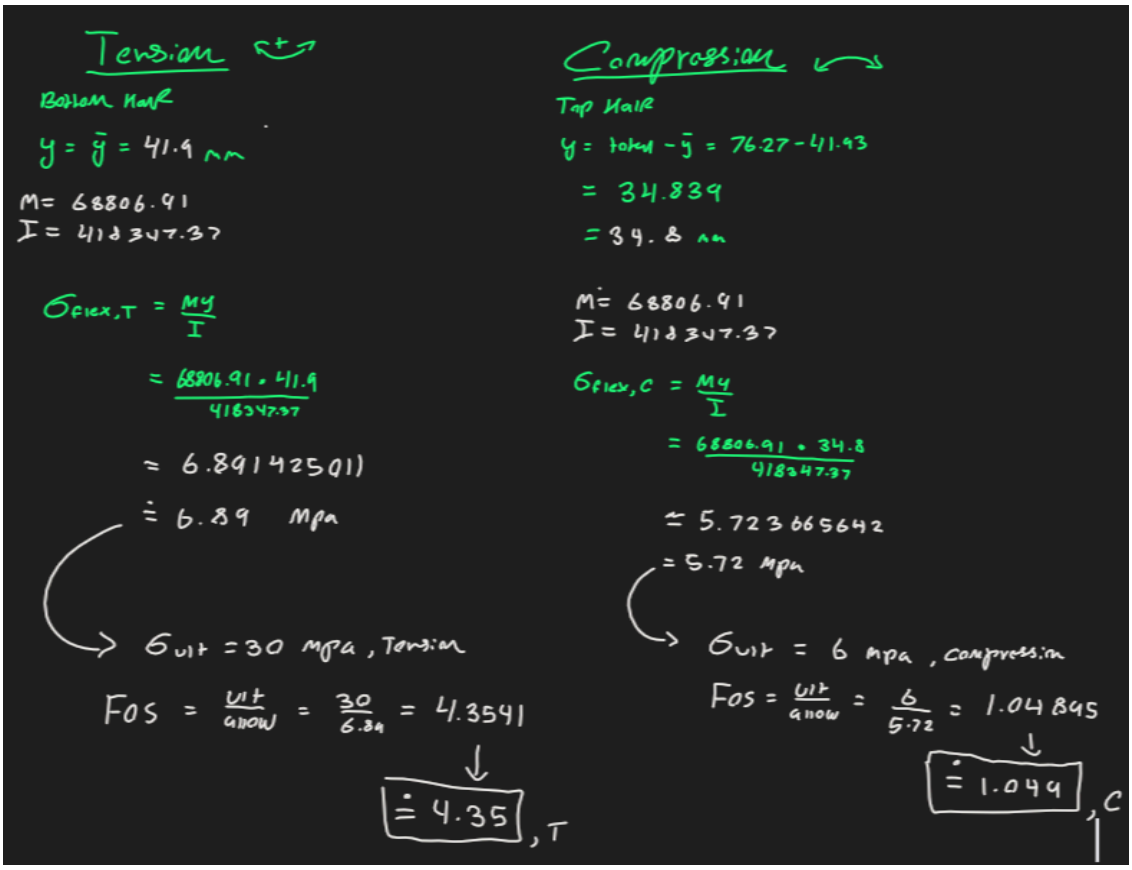


Figure 4: Flexural stress and the Factor of Safety (FOS) were determined for the first failure mode using the provided ultimate stress values, ensuring an accurate assessment of structural performance under maximum load conditions.

Shear glue

V = max shear = 200 N
$I = 418347.37\ mm^4$
$b = 12.54\ mm$ – glue line width
$Q_{glue} = \Sigma A \cdot d_i$ glue – neutral axis
$= 127(34.265) = 4.352 \times 10^3$

$$\tau_{glue} = \frac{VQ}{Ib} = \frac{200(4.35 \times 10^3)}{418347.37(12.54)} = 0.1659$$

$$FOS = \frac{\tau_{max\ glue}}{\tau_{glue}} = \frac{2}{0.1659}$$

$$\boxed{= 12.05}$$

Shear Strength (@ centroid)

$V = 200\ N$
$I = 418347$
$b = 2.54$

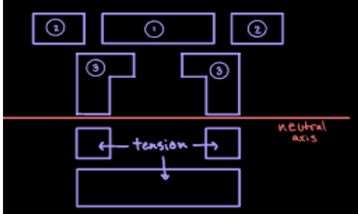
$$\rightarrow Q_{cent} = 2(1.27)(38.7)\left(\frac{28.2}{2}\right) + (80 \cdot 1.27)\left(41.4 - \frac{1.27}{2}\right) = 5552.3$$

$$\tau_{shear} = \frac{VQ}{Ib} = \frac{200(5552.3)}{418347(2.54)} = 1.045$$

$$FOS = \frac{\tau_s}{\tau_{shear}} = \frac{4}{1.045}$$

$$\boxed{= 3.828}$$

Buckling



① Case 1 – $t = 1.27\ mm$, $b = 78.73\ mm$, $k = 4$

$$\sigma_{buck_1} = \frac{k\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = \frac{4\pi^2(4000)}{12(1-0.2^2)}\left(\frac{1.27}{78.73}\right)^2 = 3.57\ MPa$$

② Case 2 – $t = 1.27\ mm$, $b = 10.64\ mm$, $k = 0.425$

$$\sigma_{buck_2} = \frac{k\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = \frac{0.425\pi^2(4000)}{12(1-0.2^2)}\left(\frac{1.27}{10.64}\right)^2 = 20.75\ MPa$$

③ Case 3 – $t = 1.27\ mm$, $b = 33.8\ mm$, $k = 6$

$$\sigma_{buck_3} = \frac{k\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = \frac{6\pi^2(4000)}{12(1-0.2^2)}\left(\frac{1.27}{33.8}\right)^2 = 29.0\ MPa$$

④ Case 4 – $t = 1.27\ mm$, $h = 73.73\ mm$, $a = 400\ mm$

$$\tau_{buck} = \frac{5\pi^2 E}{12(1-0.2^2)}\left[\left(\frac{t}{a}\right)^2 + \left(\frac{t}{h}\right)^2\right] = \frac{5\pi^2(4000)}{12(1-0.2^2)}\left[\left(\frac{1.27}{400}\right)^2 + \left(\frac{1.27}{73.73}\right)^2\right] = 5.26\ MPa$$

Case 1 lowest σ_{buck} ∴

$$FOS_{flex\ buckling} = \frac{\sigma_{buck\ 1}}{\sigma_{top}} = \boxed{0.624}$$

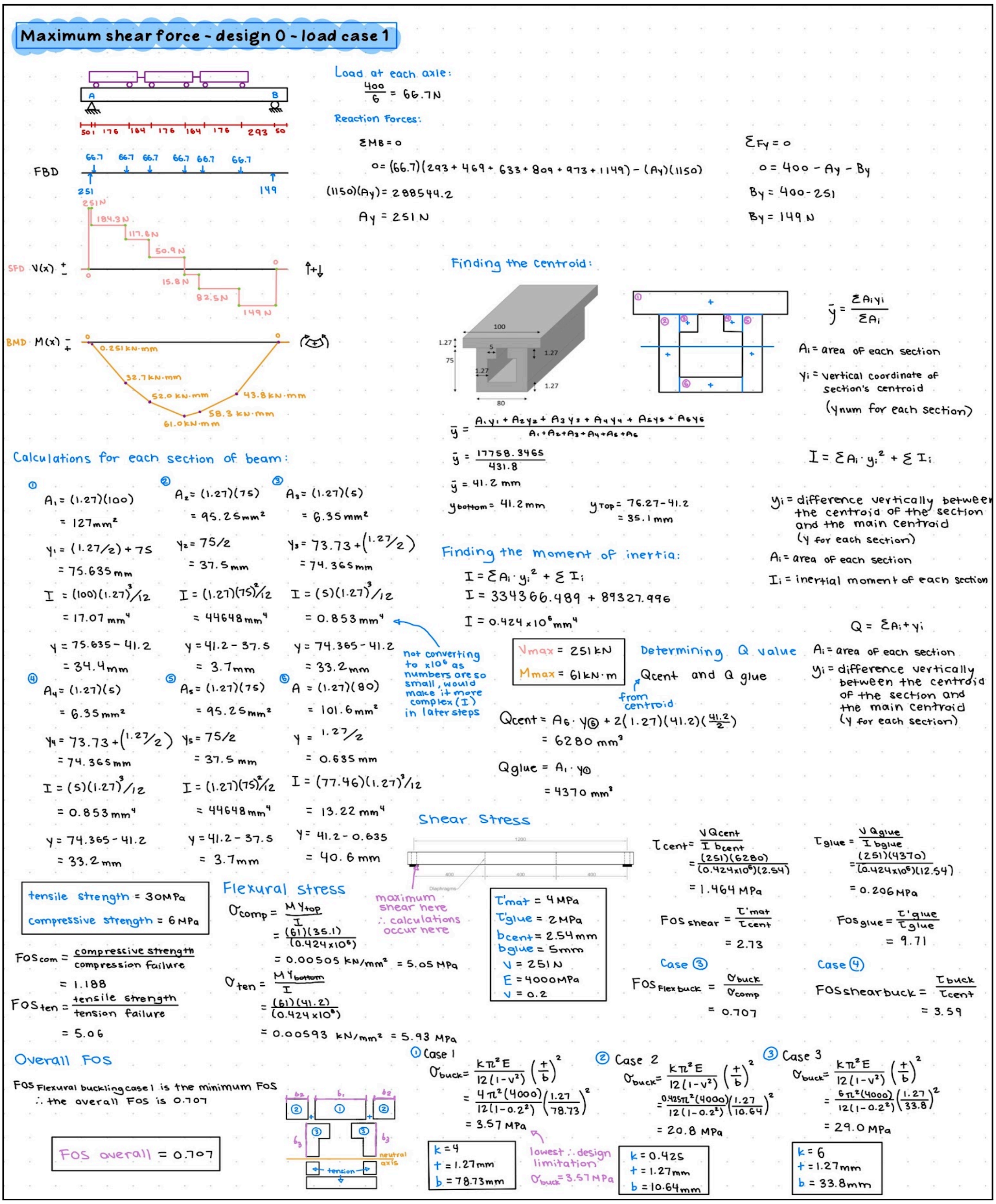
$$FOS_{Shear\ buckling} = \frac{\tau_{buck}}{\tau_{top}} = \boxed{5.03}$$

Minimum FOS = 0.624 from flexural buckling

Figure 5: The FOS for shear and four buckling cases were calculated using the provided ultimate values. The lowest FOS, determined to be 0.624, occurred in flexural buckling, indicating the critical failure mode

The bending moment reaches its maximum when the train's centre of mass (Load Case 1) is positioned at the centre, providing the greatest area under the curve [Figure 2]. Initial calculations with the train-centred confirmed this alignment, eliminating the need for recalculations at the location of the maximum bending moment.

The Train Placement for Maximizing Shear Force



(4) Case 4

$$\tau_{buck} = \frac{5\pi^2 E}{12(1-\mu^2)}\left(\left(\frac{t}{a}\right)^2 + \left(\frac{t}{h}\right)^2\right) \quad \boxed{\begin{array}{l} t = 1.27\text{mm} \\ h = 73.73\text{mm} \\ a = 400\text{mm} \end{array}}$$

$$= \frac{5\pi^2(4000)}{12(1-0.2^2)}\left(\left(\frac{1.27}{400}\right)^2 + \left(\frac{1.27}{73.73}\right)^2\right)$$

$$= 5.26 \text{ MPa}$$

Figure 6: A repetition of the calculation methods from Figures 1–5 was applied to determine the maximum shear force, identified 1 mm from the leftmost (or rightmost) pin. This result was verified using the Shear Force Envelope (SFE) output from the code.

The shear force is maximised when the reaction force is at its highest, which occurs when all loads are positioned on one side of the reaction force. This was achieved by placing the train load 1 mm to the right of the leftmost support instead of in the middle, where the maximum bending moment occurs. This is because the maximum shear force is directly influenced by the magnitude of the reaction force at the supports, which is greatest when the load is asymmetrically placed, creating an unbalanced distribution of forces. Conversely, centring the load distributes forces more evenly, reducing the reaction force and, consequently, the shear force.

Programming Evidence

Validation of Design 0 calculations using code

(matches calculated values above by hand)

<u>Buckling Capacities:</u>
Case 1 - Web Buckling: 3.57 MPa
Case 2 - Side Flange Buckling: 20.77 MPa
Case 3 - Middle Flange Buckling: 31.39 MPa
Shear Buckling Capacity: 5.26 MPa

<u>Central Locomotive Location:</u>
Maximum Shear Force: 200.00 N at x = 0.0 mm
Maximum Bending Moment: 68600.00 N·mm at x = 511.0 mm

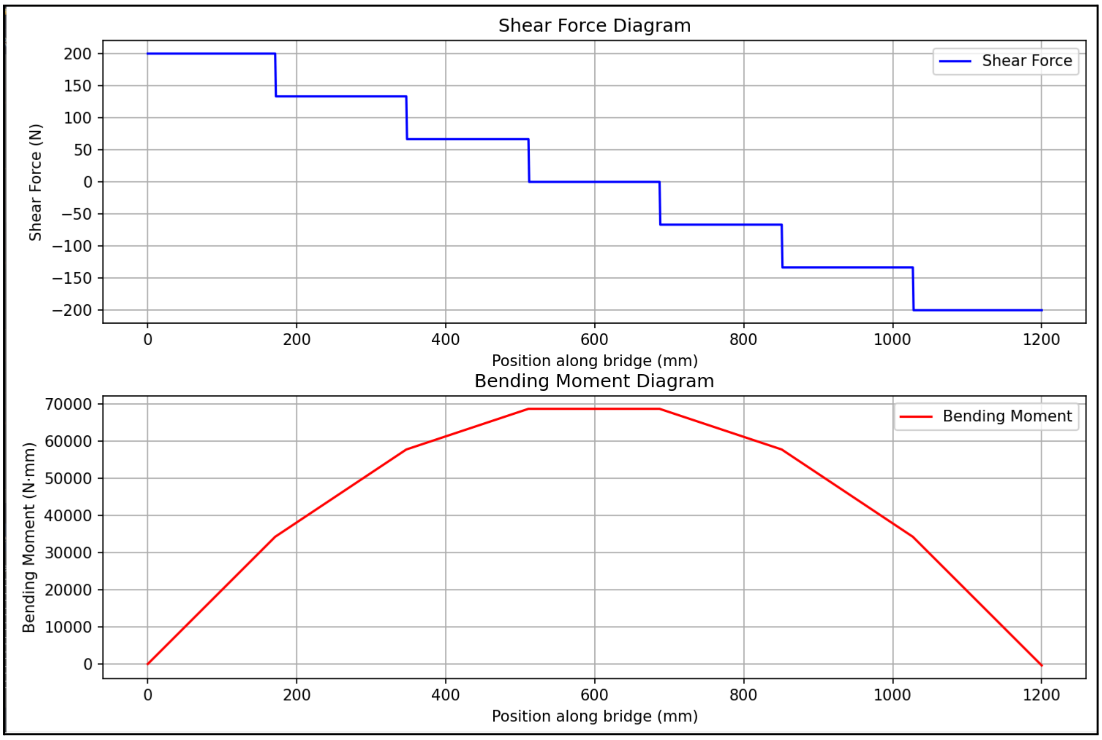
<u>Stresses:</u>
Flexural Stress (Tension): 6.73 MPa
Flexural Stress (Compression): 5.35 MPa
Maximum Shear Stress: 1.16 MPa
Maximum Glue Shear Stress: 0.15 MPa

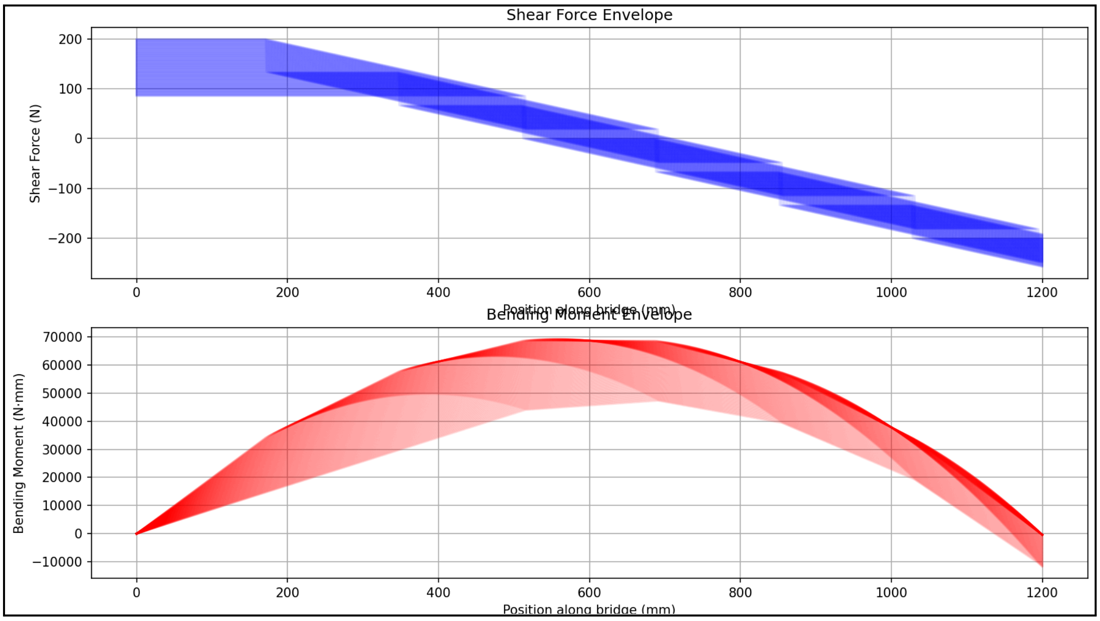
<u>Predicted failure load</u>: 266.9 N

<u>Factors of Safety (</u>CRITICAL ≤ 1.1, OK ≤ 2 , GOOD > 2<u>):</u>
FOS against tension: 4.120 (GOOD)
FOS against compression: 1.037 (CRITICAL) - *The Lowest FOS is 1.0*
FOS against shear: 3.302 (GOOD)
FOS against glue failure: 12.416 (GOOD)

FOS against buckling in middle flange: 0.616 (CRITICAL)
FOS against buckling in the side flanges: 3.589 (GOOD)
FOS against buckling in webs: 5.424 (GOOD)
FOS against shear buckling: 4.339 (GOOD)

<u>Moving Load Analysis Results:</u>
Maximum Shear Force (Moving): 257.33 N (leftmost wheel at 172.0 mm)
Maximum Bending Moment (Moving): 69260.00 N·mm (leftmost wheel at 44.0 mm)





Code Calculations of Load Case 1, Final Design

Buckling Capacities:
Case 1 - Web Buckling: 14.27 MPa
Case 2 - Side Flange Buckling: 10.01 MPa
Case 3 - Middle Flange Buckling: 126.00 MPa
Shear Buckling Capacity: 5.31 MPa

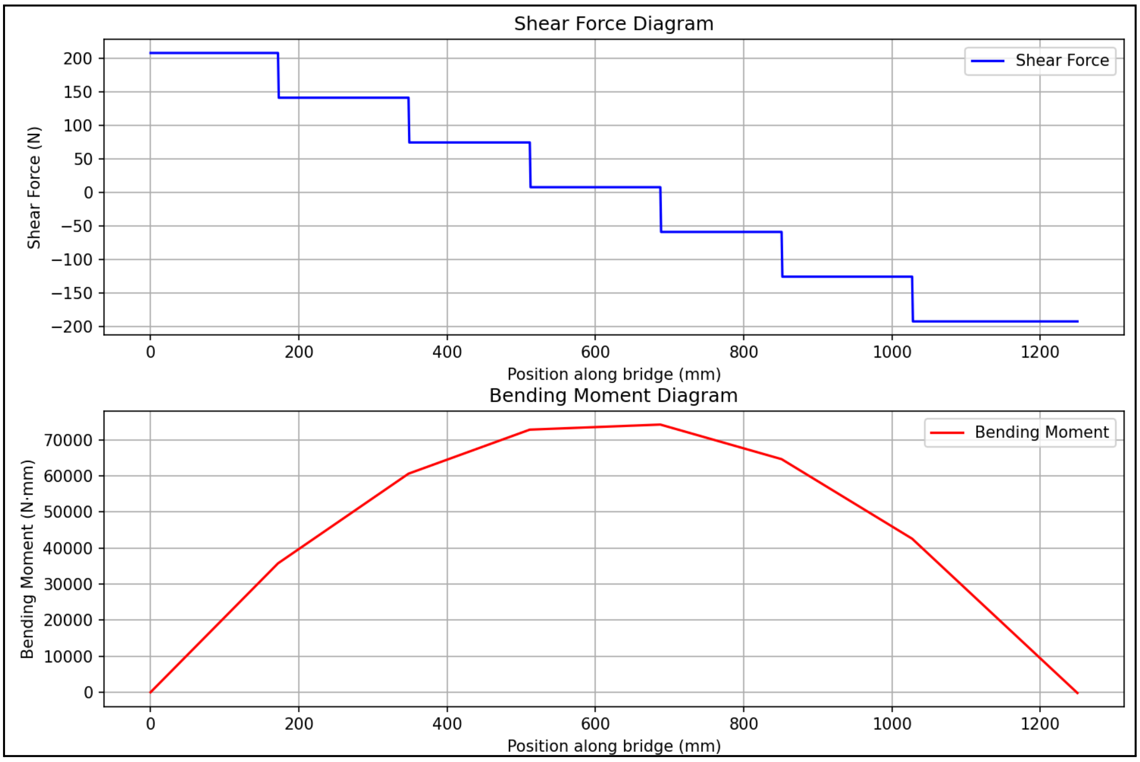
Central Locomotive Location:
Maximum Shear Force: 208.00 N at x = 0.0 mm
Maximum Bending Moment: 74250.00 N·mm at x = 687.5 mm

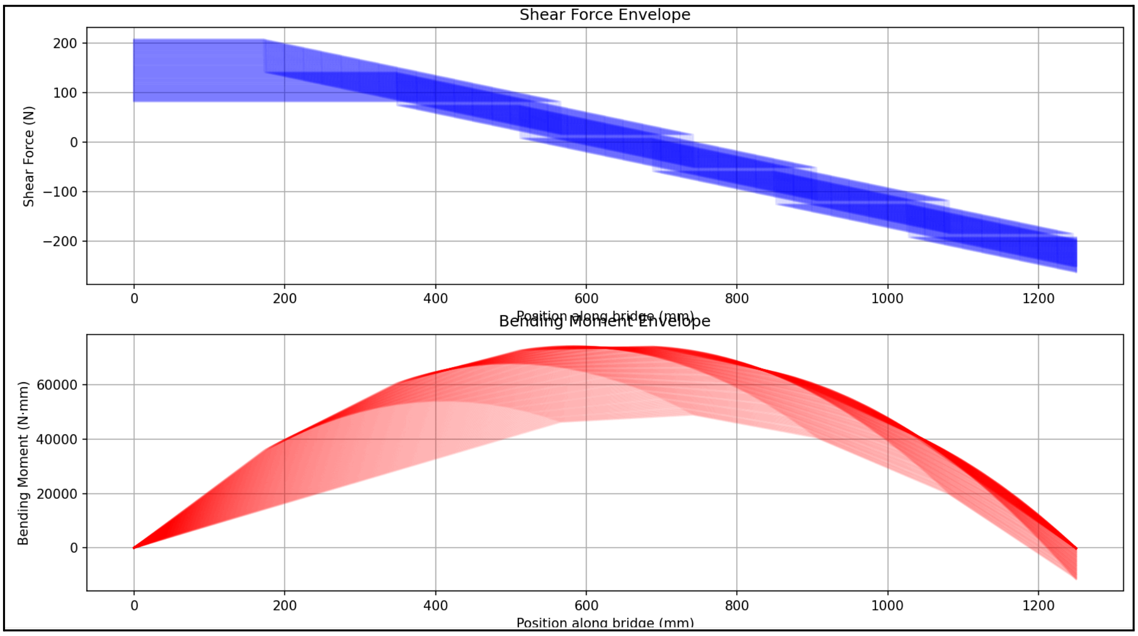
Stresses:
Flexural Stress (Tension): 6.46 MPa
Flexural Stress (Compression): 1.98 MPa
Maximum Shear Stress: 1.22 MPa
Maximum Glue Shear Stress: 0.18 MPa

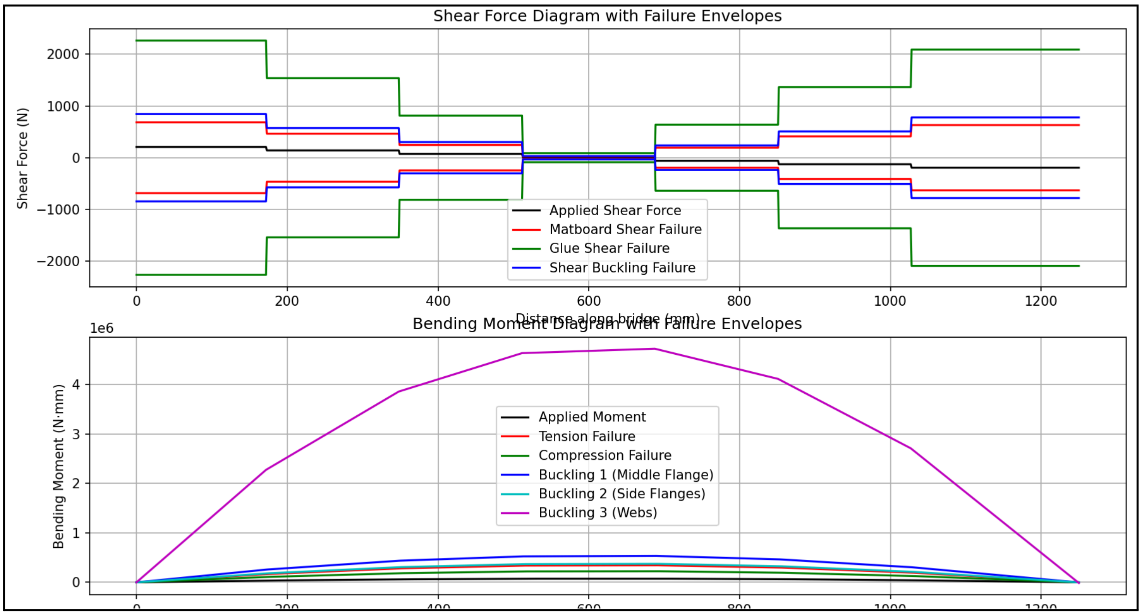
Predicted failure load: 1212.2 N

Factors of Safety:
FOS against tension: 4.643 (GOOD)
FOS against compression: 3.030 (GOOD) - *The lowest FOS is 3.0*
FOS against shear: 3.284 (GOOD)
FOS against glue failure: 10.871 (GOOD)
FOS against buckling in middle flange: 7.206 (GOOD)
FOS against buckling in the side flanges: 5.057 (GOOD)
FOS against buckling in webs: 63.640 (GOOD)
FOS against shear buckling: 4.357 (GOOD)

Moving Load Analysis Results:
Maximum Shear Force (Moving): 263.04 N (leftmost wheel at 222.0 mm)
Maximum Bending Moment (Moving): 74370.33 N·mm (leftmost wheel at 76.0 mm)


Shear Force Diagram
Shear Force
Shear Force (N)
Position along bridge (mm)
Bending Moment Diagram
Bending Moment
Bending Moment (N·mm)
Position along bridge (mm)


Shear Force Envelope
Shear Force (N)
Position along bridge (mm)
Bending Moment Envelope
Bending Moment (N·mm)
Position along bridge (mm)



Code Calculations of Load Case 2, Final Design

Pass 1: [67.5, 67.5, 67.5, 67.5, 91, 91] (load positions)

Buckling Capacities:
Case 1 - Web Buckling: 14.27 MPa
Case 2 - Side Flange Buckling: 10.01 MPa
Case 3 - Middle Flange Buckling: 126.00 MPa
Shear Buckling Capacity: 5.31 MPa

Central Locomotive Location:
Maximum Shear Force: 229.74 N at x = 1028.1 mm
Maximum Bending Moment: 83191.62 N·mm at x = 687.5 mm

Stresses:
Flexural Stress (Tension): 7.24 MPa
Flexural Stress (Compression): 2.22 MPa
Maximum Shear Stress: 1.35 MPa
Maximum Glue Shear Stress: 0.20 MPa

Predicted failure load: 1222.5 N

Factors of Safety:
FOS against tension: 4.144 (GOOD)
FOS against compression: 2.705 (GOOD) - *The lowest FOS is 2.7*
FOS against shear: 2.973 (GOOD)
FOS against glue failure: 9.843 (GOOD)
FOS against buckling in middle flange: 6.432 (GOOD)
FOS against buckling in side flanges: 4.513 (GOOD)
FOS against buckling in webs: 56.800 (GOOD)
FOS against shear buckling: 3.945 (GOOD)

Moving Load Analysis Results:
Maximum Shear Force (Moving): 310.02 N
Maximum Bending Moment (Moving): 83191.62 N·mm

Pass 2: [77.625, 77.625, 81, 81, 100.1, 100.1] (load positions)

Buckling Capacities:
Case 1 - Web Buckling: 14.27 MPa
Case 2 - Side Flange Buckling: 10.01 MPa
Case 3 - Middle Flange Buckling: 126.00 MPa
Shear Buckling Capacity: 5.31 MPa

Central Locomotive Location:
Maximum Shear Force: 260.60 N at x = 1028.1 mm
Maximum Bending Moment: 95937.80 N·mm at x = 687.5 mm

Stresses:
Flexural Stress (Tension): 8.35 MPa
Flexural Stress (Compression): 2.56 MPa
Maximum Shear Stress: 1.53 MPa
Maximum Glue Shear Stress: 0.23 MPa

Predicted failure load: 1213.6 N

Factors of Safety:
FOS against tension: 3.593 (GOOD)
FOS against compression: 2.345 (GOOD) *- The lowest FOS is 2.3*
FOS against shear: 2.621 (GOOD)
FOS against glue failure: 8.677 (GOOD)
FOS against buckling in middle flange: 5.577 (GOOD)
FOS against buckling in side flanges: 3.914 (GOOD)
FOS against buckling in webs: 49.254 (GOOD)
FOS against shear buckling: 3.478 (GOOD)

Moving Load Analysis Results:
Maximum Shear Force (Moving): 352.50 N
Maximum Bending Moment (Moving): 95937.80 N·mm

Pass 3: [89.26875, 89.26875, 97.2, 97.2, 110.11, 110.11] (load positions)

Buckling Capacities:
Case 1 - Web Buckling: 14.27 MPa
Case 2 - Side Flange Buckling: 10.01 MPa
Case 3 - Middle Flange Buckling: 126.00 MPa
Shear Buckling Capacity: 5.31 MPa

Central Locomotive Location:
Maximum Shear Force: 297.10 N at x = 0.0 mm
Maximum Bending Moment: 110804.55 N·mm at x = 687.5 mm

Stresses:
Flexural Stress (Tension): 9.64 MPa
Flexural Stress (Compression): 2.95 MPa
Maximum Shear Stress: 1.74 MPa
Maximum Glue Shear Stress: 0.26 MPa

Predicted failure load: 1204.5 N

Factors of Safety:
FOS against tension: 3.111 (GOOD)
FOS against compression: 2.031 (GOOD) *- The lowest FOS is 2.0*
FOS against shear: 2.299 (GOOD)
FOS against glue failure: 7.611 (GOOD)
FOS against buckling in middle flange: 4.829 (GOOD)
FOS against buckling in side flanges: 3.389 (GOOD)
FOS against buckling in webs: 42.645 (GOOD)
FOS against shear buckling: 3.050 (GOOD)

Moving Load Analysis Results:
Maximum Shear Force (Moving): 401.40 N
Maximum Bending Moment (Moving): 110804.55 N·mm

Full script/program with comments, user-defined functions, and required packages.

[Python Ver. 3 using matplotlib to graph SFD, BMD, & Failure Capacities]

```
#required packages
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import List, Tuple

@dataclass
# This class defines the geometric properties of a bridge, such as dimensions and #flange
thicknesses.
class BridgeGeometry:
    """Class to store bridge geometric properties"""
    length: float  # mm
    web_height: float  # mm (previously height, now only web height)
    width: float  # mm
    top_flange_thickness: float  # mm
    bottom_flange_thickness: float  # mm
    web_thickness: float  # mm
    bottom_flange_width: float  # mm
```

```
    @property
    def total_height(self) -> float:
        """Calculate total height as web_height + top_flange_thickness +
bottom_flange_thickness"""
        total = self.web_height + self.top_flange_thickness + self.bottom_flange_thickness +
1.27

        return self.web_height + self.top_flange_thickness + self.bottom_flange_thickness +
1.27

@dataclass
# This class represents load cases, including the total weight and positions/loads of #wheels.
class LoadCase:
    """Class to store load case information"""
    total_weight: float  # N
    wheel_positions: np.ndarray  # mm
    wheel_loads: np.ndarray  # N

# This class provides methods for analyzing bridge performance under different load
#conditions.
class BridgeAnalysis:
    def __init__(self, geometry: BridgeGeometry):
        self.geometry = geometry
        self.E = 4000  # MPa
        self.mu = 0.2
        self.sigma_ult_tension = 30  # MPa
        self.sigma_ult_compression = 6  # MPa
        self.tau_max = 4  # MPa
        self.discretization = 1250  # number of points along bridge, separated by mm
        self.x = np.linspace(0, geometry.length, self.discretization + 1)

# Calculates cross-sectional properties including the centroid, moment of inertia, and
#shear-related properties.
    def calculate_section_properties(self) -> Tuple[float, float, float,float]:

        #Calculate y_bar, I and Q exactly as shown in the hand calculations, using only
       # geometric relationships rather than hardcoded values.
      """
        Returns:
           Tuple[float, float, float]: (y_bar, I, Q_cent)
              - y_bar: Distance from bottom to neutral axis
              - I: Second moment of area
              - Q_cent: First moment of area at centroid for shear calculations
              - Q_glue :
        """
        # Define geometry values from class properties
        h = self.geometry.total_height  # Total height
        w = self.geometry.width   # Total width
        t_top = self.geometry.top_flange_thickness
        t_bottom = self.geometry.bottom_flange_thickness
        t_web = self.geometry.web_thickness
        bottom_w = self.geometry.bottom_flange_width  # Bottom flange width

        # Calculate derived dimensions
        web_spacing = bottom_w - t_web  # Distance between web centers
```

```
overhang = (w - bottom_w) / 2  # Overhang length each side
h_web = h - (t_top + t_bottom)  # Height of web (clear height between flanges)
top_small_width = overhang + t_web / 2  # Width of small top piece including #half web

# Areas calculation
A1 = t_top * w  # Top flange
A2 = t_top * top_small_width  # Small top piece (includes half web thickness)
A3 = h_web * t_web  # Web
A4 = t_bottom * bottom_w  # Bottom flange

A5 = 1.27 * 80

# Y coordinates calculation (from bottom)
y_B1 = h - t_top / 2  # Top flange centroid
y_B2 = h - t_top - t_top / 2  # Small top piece centroid
y_B3 = t_bottom + h_web / 2  # Web centroid
y_B4 = t_bottom / 2  # Bottom flange centroid
y_B5 = h - 1.27/2 #ADDED top flange


# Calculate y_bar
numerator = (A1 * y_B1 + 2 * A2 * y_B2 + 2 * A3 * y_B3 + A4 * y_B4 + A5 * y_B5)
denominator = (A1 + 2 * A2 + 2 * A3 + A4 + A5)
y_bar = numerator / denominator

# Calculate I using parallel axis theorem
I = (
   # Top flange
   A1 * (y_B1 - y_bar)**2 + w * t_top**3 / 12 +
   # Two small top pieces
   2 * (A2 * (y_B2 - y_bar)**2 + top_small_width * t_top**3 / 12) +
   # Two webs
   2 * (A3 * (y_B3 - y_bar)**2 + t_web * h_web**3 / 12) +
   # Bottom flange
   A4 * (y_B4 - y_bar)**2 + bottom_w * t_bottom**3 / 12 +
   # added piece
   A5 + (y_B5 - y_bar)**2 + bottom_w * 1.27**3 / 12
)

# Calculate Q at centroid for shear stress
Q_cent = (
   # Top flange contribution
   A1 * (h - t_top / 2 - y_bar) +
   # Top small pieces contribution (both sides)
   2 * A2 * (h - t_top - t_top / 2 - y_bar) +
   # Web contribution above centroid
   2 * (t_web * (h - y_bar - t_top) *
       (h - t_top - (h - y_bar - t_top)/2 - y_bar)) +
   # added piece
   A5 * ( h - t_top - 1.27/2 - y_bar)
)

Q_glue = ( A1* (h - t_top / 2 - y_bar)) + A5 * ( h - t_top - 1.27/2 - y_bar)
return y_bar, I, Q_cent, Q_glue
```

```
# Computes reaction forces at bridge supports using equilibrium equations.
    def calculate_reactions(self, load_case: LoadCase) -> Tuple[float, float]:
        """
        Calculate reaction forces at supports
        Uses moment equilibrium about left pin (x=0) to find right reaction,
        then vertical force equilibrium to find left reaction.

        Args:
            load_case: LoadCase object containing wheel positions and their individual loads

        Returns:
            Tuple[float, float]: (R_left, R_right) forces in Newtons
        """
        # Extract individual wheel loads and positions
        wheel_positions = load_case.wheel_positions  # Array of x positions
        wheel_loads = load_case.wheel_loads  # Array of corresponding loads

        # Calculate right reaction using moment about left pin (x=0)
        # Σ M_left = 0
        # R_right * 1200 = Σ(P_i * x_i) for all wheels i
        moment_sum = 0
        for pos, load in zip(wheel_positions, wheel_loads):
            moment_sum += load * pos

        R_right = moment_sum / 1200  # Right reaction force

        # Calculate left reaction using vertical equilibrium
        # Σ F_y = 0
        # R_left + R_right - Σ(P_i) = 0
        total_load = sum(wheel_loads)
        R_left = total_load - R_right

        return R_left, R_right

# Determines maximum shear force and bending moment on the bridge and their #locations.
    def find_max_loads(self, train_positions: np.ndarray, wheel_loads: np.ndarray) ->
Tuple[float, float, float, float]:
        """
        Calculate maximum shear force and bending moment and their locations
        """
        L = self.geometry.length
        R_right = np.sum(wheel_loads * train_positions) / L
        R_left = np.sum(wheel_loads) - R_right

        x = np.linspace(0, L, 1201)  # 1mm spacing
        shear = np.zeros_like(x)
        moment = np.zeros_like(x)

        for i, xi in enumerate(x):
            if xi >= 0:
                shear[i] += R_left

            for pos, load in zip(train_positions, wheel_loads):
                if xi >= pos:
                    shear[i] -= load
```

```
            if i > 0:
                dx = x[1] - x[0]
                moment[i] = moment[i-1] + shear[i] * dx

        max_shear = np.max(np.abs(shear))
        max_moment = np.max(moment)
        x_max_shear = x[np.argmax(np.abs(shear))]
        x_max_moment = x[np.argmax(moment)]

        return max_shear, max_moment, x_max_shear, x_max_moment

# Computes stresses (tensile, compressive, shear, and glue) under the maximum #loading
conditions.
    def calculate_stresses(self, V_max: float, M_max: float) -> Tuple[float, float, float, float]:
        """
        Calculate all stresses using maximum shear and maximum moment

        Returns:
            Tuple[float, float, float, float]: (sigma_tension, sigma_compression, tau_max,
taug_max)
        """
        y_bar, I, Q_cent,Q_glue = self.calculate_section_properties()

        # Normal stresses using maximum moment
        sigma_tension = M_max * y_bar / I
        sigma_compression = M_max * (self.geometry.total_height - y_bar) / I

        # Shear stress using maximum shear force and VQ/Ib formula
        tau_max = V_max * Q_cent / (I * (2 * self.geometry.web_thickness))
        taug_max = V_max * Q_glue / (I * (2 * (self.geometry.web_thickness + 5)))


        return sigma_tension, sigma_compression, tau_max, taug_max

# Calculates buckling capacities based on geometry and material properties.
    def calculate_buckling_capacities(self) -> Tuple[float, float, float, float, float, float, float,
float]:
        """
        Calculate buckling capacities using cross-section geometry
        Returns: (S_tens, S_comp, T_max, T_gmax, S_buck1, S_buck2, S_buck3, T_buck)
        """
        # Material properties (from class initialization)
        E = self.E  # MPa
        mu = self.mu

        # Material strengths (using class properties)
        S_tens = self.sigma_ult_tension
        S_comp = self.sigma_ult_compression
        T_max = self.tau_max
        T_gmax = 2  # MPa

        # Get cross-section properties
        t_top = self.geometry.top_flange_thickness
        t_bottom = self.geometry.bottom_flange_thickness
```

```
        t_web = self.geometry.web_thickness
        h = self.geometry.total_height
        w = self.geometry.width
        bottom_width = self.geometry.bottom_flange_width  # mm

        # Common term in buckling equations
        common_term = (np.pi**2 * E) / (12 * (1 - mu**2))

        # Case 1: Middle section of top flange (k=4)
        b1 = bottom_width - t_web  # Approximate value for the bottom flange width #minus the
web thickness

        # Case 2: Edge sections of top flange (k=0.425)
        overhang = (w - bottom_width) / 2  # Length of each side overhang of the flange
        b2 = overhang + t_web / 2

        # Case 3: Web sections (k=6)
        y_bar = self.calculate_section_properties()[0]  # Using existing method
        middle_section_height = h - t_top  # Height of the middle section
        b3 = middle_section_height - y_bar  # Height above the neutral axis for web

        # Case 4: Shear buckling
        h_clear = h - (t_top + t_bottom)  # Clear height between flanges
        a = 225  # Diaphragm spacing, maximum spacing

        # Calculate buckling stresses
        S_buck1 = 4 * common_term * (t_top / b1) ** 2
        S_buck2 = 0.425 * common_term * (t_top / b2) ** 2
        S_buck3 = 6 * common_term * (t_web / b3) ** 2
        T_buck = 5 * common_term * ((t_web / a) ** 2 + (t_web / h_clear) ** 2)

        return S_tens, S_comp, T_max, T_gmax, S_buck1, S_buck2, S_buck3, T_buck

# Computes factors of safety (FOS) against different failure modes.
    def calculate_all_fos(self, V_max: float, M_max: float) -> Tuple[float, float, float, float, float,
float, float, float]:
        """Calculate all factors of safety"""
        sigma_tension, sigma_compression, tau_max, taug_max =
self.calculate_stresses(V_max, M_max)
        S_tens, S_comp, T_max, T_gmax, S_buck1, S_buck2, S_buck3, T_buck =
self.calculate_buckling_capacities()

        FOS_tens = S_tens / sigma_tension
        FOS_comp = S_comp / sigma_compression
        FOS_shear = T_max/ tau_max
        FOS_glue = T_gmax / taug_max
        FOS_buck1 = S_buck1 / sigma_compression
        FOS_buck2 = S_buck2 / sigma_compression
        FOS_buck3 = S_buck3 / sigma_compression
        FOS_buckV = T_buck/ tau_max

        return FOS_tens, FOS_comp, FOS_shear, FOS_glue, FOS_buck1, FOS_buck2,
FOS_buck3, FOS_buckV

# Predicts the failure load of the bridge by analyzing the minimum factor of safety.
```

```
    def calculate_failure_load(self, load_case: LoadCase) -> float:
        """ Calculate failure load based on minimum FOS"""
        max_shear, max_moment, _, _ = self.find_max_loads(
            load_case.wheel_positions,
            load_case.wheel_loads
        )
        FOS_values = self.calculate_all_fos(max_shear, max_moment)
        min_FOS = min(FOS_values)
        P_fail = load_case.total_weight * min_FOS

        return P_fail

# Performs a comprehensive analysis of the bridge, including stresses, FOS, and #failure
predictions.
    def analyze_bridge(self, load_case: LoadCase):
        # Print buckling capacities
        S_tens, S_comp, T_max, T_gmax, S_buck1, S_buck2, S_buck3, T_buck =
self.calculate_buckling_capacities()
        print(f"\nBuckling Capacities:")
        print(f"Case 1 - Web Buckling: {S_buck1:.2f} MPa")
        print(f"Case 2 - Side Flange Buckling: {S_buck2:.2f} MPa")
        print(f"Case 3 - Middle Flange Buckling: {S_buck3:.2f} MPa")
        print(f"Shear Buckling Capacity: {T_buck:.2f} MPa")

        max_shear, max_moment, x_shear, x_moment = self.find_max_loads(
            load_case.wheel_positions,
            load_case.wheel_loads
        )

        # Calculate stresses
        sigma_tension, sigma_compression, tau_max, taug_max =
self.calculate_stresses(max_shear, max_moment)

        # Calculate factors of safety
        FOS_values = self.calculate_all_fos(max_shear, max_moment)
        P_fail = self.calculate_failure_load(load_case)

        failure_modes = [
            "tension",
            "compression",
            "shear",
            "glue failure",
            "buckling in middle flange",
            "buckling in side flanges",
            "buckling in webs",
            "shear buckling"
        ]

        print(f"\nCentral Locomotive Location:")
        print(f"Maximum Shear Force: {max_shear:.2f} N at x = {x_shear:.1f} mm")
        print(f"Maximum Bending Moment: {max_moment:.2f} N·mm at x = {x_moment:.1f}
mm")

        print(f"\nStresses:")
        print(f"Flexural Stress (Tension): {sigma_tension:.2f} MPa")
```

```
        print(f"Flexural Stress (Compression): {sigma_compression:.2f} MPa")
        print(f"Maximum Shear Stress: {tau_max:.2f} MPa")
        print(f"Maximum Glue Shear Stress: {taug_max:.2f} MPa")

        print(f"\nPredicted failure load: {P_fail:.1f} N")
        print("\nFactors of Safety:")
        for mode, fos in zip(failure_modes, FOS_values):
            status = "CRITICAL" if fos < 1.1 else "OK" if fos < 2.0 else "GOOD"
            print(f"FOS against {mode}: {fos:.3f} ({status})")

# Generates shear force and bending moment diagrams for the given load case.
    def plot_results(self, load_case: LoadCase):
        """Plot SFD and BMD diagrams"""
        L = self.geometry.length
        x = np.linspace(0, L, 1201)

        R_right = np.sum(load_case.wheel_loads * load_case.wheel_positions) / L
        R_left = np.sum(load_case.wheel_loads) - R_right

        shear = np.zeros_like(x)
        moment = np.zeros_like(x)

        for i, xi in enumerate(x):
            if xi >= 0:
                shear[i] += R_left

            for pos, load in zip(load_case.wheel_positions, load_case.wheel_loads):
                if xi >= pos:
                    shear[i] -= load

            if i > 0:
                dx = x[1] - x[0]
                moment[i] = moment[i-1] + shear[i] * dx

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

        ax1.plot(x, shear, 'b-', label='Shear Force')
        ax1.grid(True)
        ax1.set_xlabel('Position along bridge (mm)')
        ax1.set_ylabel('Shear Force (N)')
        ax1.set_title('Shear Force Diagram')
        ax1.legend()

        ax2.plot(x, moment, 'r-', label='Bending Moment')
        ax2.grid(True)
        ax2.set_xlabel('Position along bridge (mm)')
        ax2.set_ylabel('Bending Moment (N·mm)')
        ax2.set_title('Bending Moment Diagram')
        ax2.legend()

        plt.tight_layout()
        plt.show()

# Simulates and plots results for a moving train across the bridge.
    def plot_moving_train_results(self, load_case: LoadCase):
```

```
"""Plot SFD and BMD diagrams for moving train across the bridge"""
L = self.geometry.length
x = np.linspace(0, L, 1201)
increment = 2  # mm increment for moving train

all_shear = []
all_moment = []

# Variables to track maximum values and their positions
shear_force_max_moving = 0
bending_moment_max_moving = 0
max_shear_train_position = 0
max_moment_train_position = 0

# Define start and end of train movement
start_position = 0
end_position = L - (load_case.wheel_positions[-1] - load_case.wheel_positions[0])

# Loop over each train position in 2 mm increments
for position in np.arange(start_position, end_position + increment, increment):
    new_positions = load_case.wheel_positions + position
    R_right = np.sum(load_case.wheel_loads * new_positions) / L
    R_left = np.sum(load_case.wheel_loads) - R_right

    shear = np.zeros_like(x)
    moment = np.zeros_like(x)

    for i, xi in enumerate(x):
        if xi >= 0:
            shear[i] += R_left

        for pos, load in zip(new_positions, load_case.wheel_loads):
            if xi >= pos:
                shear[i] -= load

        if i > 0:
            dx = x[1] - x[0]
            moment[i] = moment[i - 1] + shear[i] * dx

    # Update maximum values if current values are larger
    current_max_shear = np.max(np.abs(shear))
    current_max_moment = np.max(moment)

    if current_max_shear > shear_force_max_moving:
        shear_force_max_moving = current_max_shear
        max_shear_train_position = position

    if current_max_moment > bending_moment_max_moving:
        bending_moment_max_moving = current_max_moment
        max_moment_train_position = position

    all_shear.append(shear)
    all_moment.append(moment)

# Print maximum values and their positions
```

```
        print(f"\nMoving Load Analysis Results:")
        print(f"Maximum Shear Force (Moving): {shear_force_max_moving:.2f} N (leftmost
wheel at {max_shear_train_position:.1f} mm)")
        print(f"Maximum Bending Moment (Moving): {bending_moment_max_moving:.2f} N·mm
(leftmost wheel at {max_moment_train_position:.1f} mm)")

        # Plotting the SFD and BMD for the moving train
        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

        for shear in all_shear:
            ax1.plot(x, shear, 'b-', alpha=0.1)
        ax1.set_title("Shear Force Envelope")
        ax1.set_xlabel("Position along bridge (mm)")
        ax1.set_ylabel("Shear Force (N)")
        ax1.grid(True)

        for moment in all_moment:
            ax2.plot(x, moment, 'r-', alpha=0.1)
        ax2.set_title("Bending Moment Envelope")
        ax2.set_xlabel("Position along bridge (mm)")
        ax2.set_ylabel("Bending Moment (N·mm)")
        ax2.grid(True)

        plt.tight_layout()
        plt.show()


# Calculates shear and moment failure capacities at each point along the bridge.
    def calculate_failure_capacities(self, load_case: LoadCase) -> tuple:
        """Calculate failure capacities for shear and moment at each point along bridge"""
        # Get maximum loads
        max_shear, max_moment, _, _ = self.find_max_loads(
            load_case.wheel_positions,
            load_case.wheel_loads
        )

        # Get all factors of safety
        FOS_tens, FOS_comp, FOS_shear, FOS_glue, FOS_buck1, FOS_buck2, FOS_buck3,
FOS_buckV = self.calculate_all_fos(max_shear, max_moment)

        # Calculate moment and shear distributions
        L = self.geometry.length
        x = np.linspace(0, L, 1201)  # 1mm spacing
        shear = np.zeros_like(x)
        moment = np.zeros_like(x)

        R_right = np.sum(load_case.wheel_loads * load_case.wheel_positions) / L
        R_left = np.sum(load_case.wheel_loads) - R_right

        for i, xi in enumerate(x):
            if xi >= 0:
                shear[i] += R_left

            for pos, load in zip(load_case.wheel_positions, load_case.wheel_loads):
                if xi >= pos:
```

```
            shear[i] -= load

        if i > 0:
            dx = x[1] - x[0]
            moment[i] = moment[i-1] + shear[i] * dx

    # Calculate failure capacities
    Mf_tens = FOS_tens * moment
    Mf_comp = FOS_comp * moment
    Vf_shear = FOS_shear * shear
    Vf_glue = FOS_glue * shear
    Mf_buck1 = FOS_buck1 * moment
    Mf_buck2 = FOS_buck2 * moment
    Mf_buck3 = FOS_buck3 * moment
    Vf_buckV = FOS_buckV * shear

    return x, shear, moment, Mf_tens, Mf_comp, Vf_shear, Vf_glue, Mf_buck1, Mf_buck2,
Mf_buck3, Vf_buckV

  #function to plot capacities
# Visualizes failure capacity envelopes compared to actual forces along the bridge.
  def plot_failure_capacities(self, load_case: LoadCase):
    """Plot failure capacity envelopes against actual forces"""
    x, shear, moment, Mf_tens, Mf_comp, Vf_shear, Vf_glue, Mf_buck1, Mf_buck2,
Mf_buck3, Vf_buckV = self.calculate_failure_capacities(load_case)

    # Create figure with 2 subplots
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    # Plot shear failure envelopes
    ax1.plot(x, shear, 'k-', label='Applied Shear Force')
    ax1.plot(x, Vf_shear, 'r-', label='Matboard Shear Failure')
    ax1.plot(x, -Vf_shear, 'r-')
    ax1.plot(x, Vf_glue, 'g-', label='Glue Shear Failure')
    ax1.plot(x, -Vf_glue, 'g-')
    ax1.plot(x, Vf_buckV, 'b-', label='Shear Buckling Failure')
    ax1.plot(x, -Vf_buckV, 'b-')
    ax1.grid(True)
    ax1.set_xlabel('Distance along bridge (mm)')
    ax1.set_ylabel('Shear Force (N)')
    ax1.set_title('Shear Force Diagram with Failure Envelopes')
    ax1.legend()

    # Plot moment failure envelopes
    ax2.plot(x, moment, 'k-', label='Applied Moment')
    ax2.plot(x, Mf_tens, 'r-', label='Tension Failure')
    ax2.plot(x, Mf_comp, 'g-', label='Compression Failure')
    ax2.plot(x, Mf_buck1, 'b-', label='Buckling 1 (Middle Flange)')
    ax2.plot(x, Mf_buck2, 'c-', label='Buckling 2 (Side Flanges)')
    ax2.plot(x, Mf_buck3, 'm-', label='Buckling 3 (Webs)')
    ax2.grid(True)
    ax2.set_xlabel('Distance along bridge (mm)')
    ax2.set_ylabel('Bending Moment (N·mm)')
    ax2.set_title('Bending Moment Diagram with Failure Envelopes')
    ax2.legend()
```

```
        plt.tight_layout()
        plt.show()

# Main function to define geometry, load cases, and run the analysis with visualization.
def main():
    # Create geometry matching calculations
    geometry = BridgeGeometry(
        length=1250,
        web_height=73.73 + 1.19,
        width=140,
        top_flange_thickness=2.54,
        bottom_flange_thickness=1.27,
        web_thickness=1.27,
        bottom_flange_width=80  # mm
    )

    # Create bridge analysis object
    bridge = BridgeAnalysis(geometry)

    # Define load case
    wheel_positions = np.array([172, 348, 512, 688, 852, 1028])  # mm from left support
#LOAD CASE 1
    #wheel_loads = np.ones(6) * (400 / 6)  # 66.7N per wheel
    #load_case = LoadCase(400, wheel_positions, wheel_loads)
#LOAD CASE 2: PASS 1
    #wheel_loads = np.array([67.5, 67.5, 67.5, 67.5, 91, 91])
    #load_case = LoadCase(452, wheel_positions, wheel_loads)
#LOAD CASE 2: PASS 2
    #wheel_loads = np.array([77.625, 77.625, 81, 81, 100.1, 100.1])
    #load_case = LoadCase(517.45, wheel_positions, wheel_loads)
#LOAD CASE 2: PASS 3
    wheel_loads = np.array([89.26875, 89.26875, 97.2, 97.2, 110.11, 110.11])
    load_case = LoadCase(593.1575, wheel_positions, wheel_loads)

    # Analyze bridge
    bridge.analyze_bridge(load_case)
    bridge.plot_results(load_case)
    bridge.plot_moving_train_results(load_case)

    #exceute failure capacities and diagrams and output
    # Plot failure capacity envelopes
    bridge.plot_failure_capacities(load_case)

if __name__ == "__main__":
    main()
```