CIV102 Matboard Bridge Design Report

Candy Ma, Bobby Xiao, Emiliano Rioja

TA: Junyan Xiao

Dated: November 23, 2025

Table of Contents

-Introduction

-1 Methodology (p2)

-1.1 Design procedure (p2)

-1.2 Constraints and Requirements (p2)

-1.3 Usage of Digital Software (p3)

-2 Overall Design Decisions (p3)

-3 Iterative Design Process (p4)

-3.1 Design 0 (p4)

-3.2 Design 1 (p5)

-3.3 Design 2 (p5)

-3.4 Design 3 (p5)

-3.5 Design 4 (p6)

-3.6 Design 5 (p6)

-3.7 Design 6 (p8)

-4 Conclusion (p9)

Appendix

- Code for SFE (Shear Force Envelope) and BME (Bending Moment Envelope)
- Code for all FOS output

Glossary

FOS – Factor of Safety

Introduction

The purpose of this project is to optimize the performance of a beam bridge under the load of a moving train, analysis and iterations of the design were conducted and documented in this report. An optimal beam bridge design (1250*122.5*100) under the constraints and requirements of CIV102 was found at the end of this report. Bridge designing is a practical method to help students learn and apply knowledge. Team 306 sufficiently applied their knowledge gained in course CIV102 and devoted themselves to building the bridge. This report will cover the methodology, major design decisions, and the iterations of the bridge.

Pic 1 Two group members holding the bridge as if they were working out.

1. Methodology

1.1 Design Procedure

A specific design process was followed to ensure the efficiency and structure of teamwork.

A. Preparation

Shear force and bending moment are analyzed at each 1mm position along the bridge. The locations of maximum shear force and bending moments on the bridge are then found using Python and verified through hand calculation. With these data, the *FOS* for design 0 under load case 1 are then computed through Python code [2] and verified through hand calculation. This phase marks

B. Iterative Design

Iterative design process is applied to improve design 0. The minimum *FOS* is computed through Python code [2], which determines the failure mode of the bridge. Applying CIV102 knowledge, we determined the optimal design within material and construction constraints.

C. Construction

Matboard cutting plan is sketched on iPad and then finalized using SolidWorks. Supplies such as gloves, cutting mat, and *3M AURA™ Respirator 9205+* face masks were used to ensure safety during construction process. Minor changes to the bridge structure were made during construction process. The details will be discussed in 4.8.

1.2 Constraints and Requirements

The contact cement used is *LePage Heavy Duty Contact Cement* [1] while the matboard itself provided with the following dimensions (the measurements are translated to mm with respect to slide rule precision):

32” = 813 mm
40” = 1016 mm
0.05” = 1.270 mm

yielding the surface area of:

826008 mm^2

Furthermore, the final bridge would also have to fulfill certain boundary conditions including:

Total length: 1250 – 1270 mm
Maximum height: 200 mm
Height of cross section: multiple of 20 mm
Flat portion: 60 mm
Minimum deck width: 100 mm

The maximum height and flat-deck conditions are imposed at the bridge supports located at both ends. After satisfying these boundary conditions, the completed bridge is tested using a three-segment train with an initial total load of 400 N. The load is then increased incrementally after each test, as shown in Figures 1 and 2.

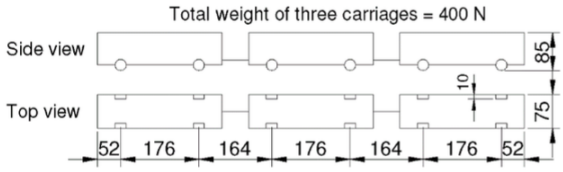

Figure 1. Train dimensions

Figure 2. Loading schematic of train movement

1.3 Usage of Digital Software

In the design, most calculations were computed with Python codes [2].

```
#calculate reaction forces based on the positions of the load: by equilibrium of the fo
def reaction_calc(center_position):  4 usages
    #sum of forces that's on the bridge is 0
    forcr_list2 = [] #a two dimension list, second dimension: force,loc
    loc_middle_car_force = [center_position - 88, center_position + 88]  # [0] is on the left
    loc_last_car_force = [center_position - 428, center_position - 252]
    loc_first_car_force = [center_position + 252, center_position + 428]
    list_load = [loc_first_car_force, loc_middle_car_force, loc_last_car_force]
    # force_dic = {"first_car": [91, loc_first_car_force], "middle_car": [67.5, loc_middle_car_force],
    #              "last_car": [67.5, loc_last_car_force]}  # forces is the sigle point force at one w
    force_dic = {"first_car": [67.5, loc_first_car_force], "middle_car": [67.5, loc_middle_car_force],
                 "last_car": [91, loc_last_car_force]}  # forces is the sigle point force at one wheel
    # force_dic = {"first_car": [66.6, loc_first_car_force], "middle_car": [66.6, loc_middle_car_force
    #              "last_car": [66.6, loc_last_car_force]}
    #find the forces on the bridge
    for i in range(0,len(list_load)):
        for j in range(0,len(list_load[i])):
            if list_load[i][j] >= 0 and list_load[i][j] <= length:
                forcr_list2.append([list(force_dic.values())[i][0],list(force_dic.values())[i][1][j]])
```

Figure 1. A screenshot of the reaction force calculation function. The marked areas compute the different load cases, which allow the code to output results based on various load cases easily.

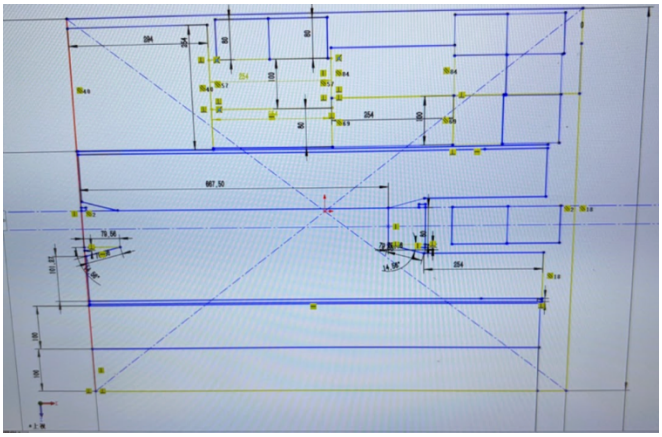

Figure 2. A screenshot of the matboard cutting plan in SolidWorks.

2 Overall Design Decisions

Objective	Decision	Justification
Cross-section shape	π beam	Because it's easy for calculations.
The moment and shear vary along the bridge; lack of materials.	Change cross sections along the bridge	For the middle of the bridge, there's no need for that much resistance for shear force, but the moment is high; for the two ends of the bridge, the need to resist the shear force is high, but the moment is low. Therefore, by varying the cross sections along the

		bridge, more materials can be saved.
Difficulty glueing pieces together.	Increase the glue tag to 7 mm.	FOS of 5mm glue tab is 12.8. FOS of 7 mm glue tab is: 17.93
We want the bridge to be as stable as possible with the limited quality of construction.	Add a bottom beam to secure the diaphragms and the webs.	A bottom beam can help glue the diaphragms together in a better position.

3 Iterative Design Process

Based on given initial Design 0, our team applied iterative design process to calculate the eight factors of safety for each iteration and calculate its failure load. Through analysis, we tackled the minimum factor of safety in each iteration by applying CIV102 knowledge to improve failure load.

3.1 Design 0

Design 0 was an initial design provided in the project handout in which it consisted of dimensions:

Top flange width: 100 mm
Cross-section width: 80 mm
Base of top flange to base: 75 mm
Flange thickness: 1.270 mm

And a schematic was provided, as shown in Figure 3.

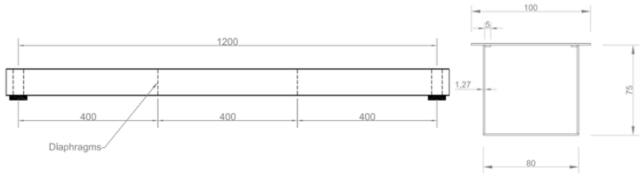

Figure 3. Schematic of Design 0

The *FOS* of Design 0 is calculated using Python [3],

Flexural Tension: 3.88
Flexural Compression: 0.923
Centroidal Shear Stress: 2.26
Glue Shear Stress: 6.33
Plate Buckling Case 1: 0.531
Plate Buckling Case 2: 3.61
Plate Buckling Case 3: 4.525
Shear Buckling: 2.87

where the minimum *FOS* is 0.531, which determines the failure mode of case 1 plate buckling, yielding,

Failure Load: 240 N

To tackle the failure mode of case 1 plate buckling, we analyzed the equation,

$$\sigma = \frac{4\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2$$

where E is the Young's modulus of 4000 MPa, μ is the Poisson's ratio of 0.2, t is the thickness of the flange, and b is the length of the flange between the webs.

Analyzing the above equation, σ is directly proportional to t. Increasing the thickness of the flange would increase the σ, which would increase the minimum *FOS* and the subsequent failure load.

3.2 Design 1

Changes applied to Design 0:

Thicken the top flange to 2.54 mm

Flexural Tension: 4.20
Flexural Compression: 1.486
Centroidal Shear Stress: 2.28
Glue Shear Stress: 5.25
Plate Buckling Case 1: 3.42
Plate Buckling Case 2: 23.3
Plate Buckling Case 3: 12.67
Shear Buckling: 2.90

where the minimum *FOS* is 1.486, which determines the failure mode of flexural compression, yielding,

Failure Load: 672 N

Equation of flexural compression,

$$\sigma = \frac{My}{I}$$

where M is the maximum moment, y is the distance from $\bar{y}$ to the top flange, and I is the moment of inertia.

Since M cannot be changed from altering cross sections, we can only modify y and I. Analyzing the above equation, σ and y are directly proportional while σ and I are inversely proportional. Therefore, increasing I would decrease σ, which would increase the *FOS* for flexural compression. Considering the calculation of I using the Parallel Axis Theorem, increasing web height would increase I.

3.3 Design 2

Changes applied to Design 1:

Increase web height to 80 mm

Flexural Tension: 4.57
Flexural Compression: 1.597
Centroidal Shear Stress: 3.54
Glue Shear Stress: 8.21
Plate Buckling Case 1: 3.68
Plate Buckling Case 2: 25.0
Plate Buckling Case 3: 11.68
Shear Buckling: 3.97

where the minimum *FOS* is 1.597, which determines the failure mode of flexural compression, yielding,

Failure Load: 722 N

The changes were made to increase the second moment of area, which as stated before is the only way to change flexural compression, by increasing I, compression decrease proportionally increasing the factor of safety.

3.4 Design 3

Changes applied to Design 2:

Increase web height to 160 mm; Decrease diaphragm spacing to 200 mm (5 diaphragms)

Flexural Tension: 11.75
Flexural Compression: 3.56
Centroidal Shear Stress: 4.60
Glue Shear Stress: 12.25
Plate Buckling Case 1: 8.21
Plate Buckling Case 2: 55.8
Plate Buckling Case 3: 5.11

Shear Buckling: 2.04

where the minimum *FOS* is 2.04, which determines the failure mode of shear buckling, yielding,

Failure Load: 920 N

Equation of shear buckling:

$$\sigma = \frac{5\pi^2 E}{12(1-\mu^2)}\left(\left(\frac{t}{h}\right)^2 + \left(\frac{t}{a}\right)^2\right)$$

These changes were made to improve the FOS of flexural stress, by increasing the second moment of Area *I,* but the failure was now because of shear buckling, because the height increased.

3.5 Design 4

Changes applied to Design 3:

Decrease diaphragm spacing to 120 mm (9 diaphragms); Increase glue tab width to 7 mm

Flexural Tension: 11.78
Flexural Compression: 3.61
Centroidal Shear Stress: 6.72
Glue Shear Stress: 25.4
Plate Buckling Case 1: 8.31
Plate Buckling Case 2: 56.5
Plate Buckling Case 3: 5.25
Shear Buckling: 5.04

where the minimum *FOS* is 3.61, which determines the failure mode of flexural compression, yielding,

Failure Load: 1631 N

We calculated surface area of the material required for this design, yielding 838910 mm^2, which exceeds the material constraint of 826008 mm^2. Therefore, even if this design was a good design, we could not build it because of material constraint.

3.6 Design 5

To minimize the material usage while maximize the failure load so that we could have enough material to build the bridge, we were thinking of decreasing all the web height, however, keep the middle one higher and the two side one lower. Another layer of top beam could be added to compensate the loss in second moment of area.
Changes applied to Design 4:

Separate the bridge into 3 sections with 2 cross sections with variable depth (section 1 and 2 with 100 mm height and section 3 with 120 mm height); Change the diaphragm spacing between the sections; Add a third layer of top flange in section 3 (middle section)

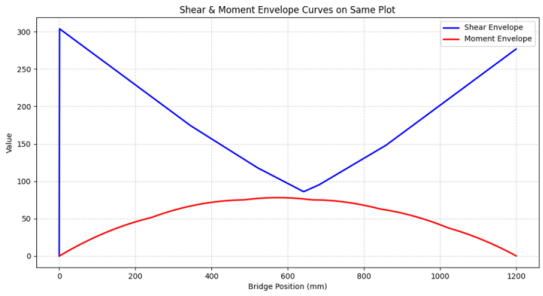

Figure 5. The graph of bending moment and shear force along the bridge.

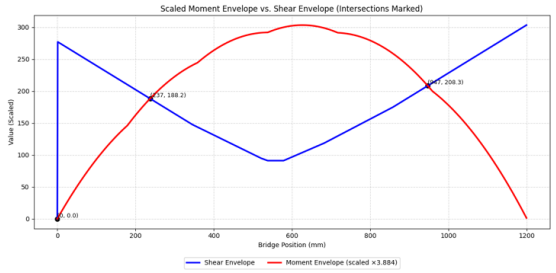

Figure 6. The graph of bending moment and shear force along the bridge. The moment is scaled up to the maximum shear. The intersections represent division of three sections of the bridge, where x_1 = 237 mm and x_2 = 947 mm.

Figure 6 shows that the bridge's end regions (x = 0–x_1 and x_2–1200 mm) carry high shear forces and low bending moments, while the central region between x_1 and x_2 exhibits the opposite pattern. Accordingly, we designate the central span between x_1 and x_2 as Section 3, and the regions from x = 0–x_1 = 237 mm and x_2–1200 mm as Sections 1 and 2, respectively.

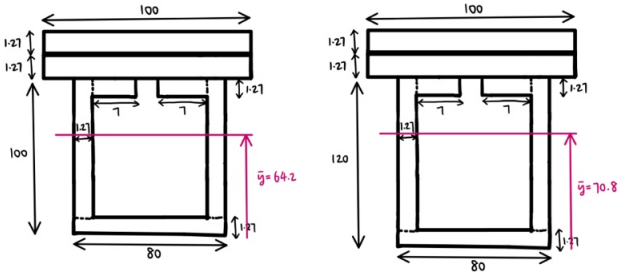

Figure 7. Two different cross sections with different web height: the left side is the cross section for section 1 and 2; the right side is the cross section for section 3.

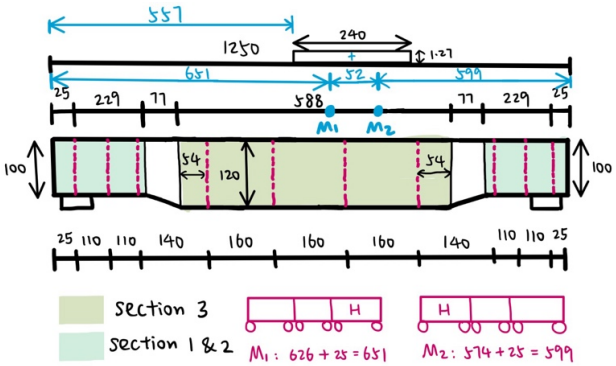

Figure 8. 2D drawing of the diaphragm spacing along the bridge (side view). M_1 represents the location of the maximum moment when the heavier car is at the front when travelling forward; M_2 represents the location of the maximum moment when the heavier train is at the end when travelling forward.

The difference between and M_2 is 52 mm. The length of the third top layer is 240 mm, which is much larger than 52 mm, to ensure that it covers all maximum moment regions. The addition of the third top layer increases the $\bar{y}$ length from 75.6 mm to 83.2 mm, which subsequently increases the compressive stress from 2.37 MPa to 1.773 MPa, and the factor of safety of compressive stress increases from 2.53 to 3.38.

We decided to make the diaphragm spacing in section 1 and 2 smaller to increase *FOS* of shear buckling of the web because the shear force is the largest on the ends of the bridge.

Section 1 and 2:

Flexural Tension: 8.94
Flexural Compression: 3.03
Centroidal Shear Stress: 3.00
Glue Shear Stress: 10.23
Plate Buckling Case 1: 6.98

Plate Buckling Case 2: 47.4
Plate Buckling Case 3: 13.28
Shear Buckling: 2.88

where the minimum *FOS* is 2.88, which determines the failure mode of shear buckling, yielding,

Failure Load: 1302 N

Section 3 (2 top flange layers):

Flexural Tension: 7.87
Flexural Compression: 2.57
Centroidal Shear Stress: 5.15
Glue Shear Stress: 18.19
Plate Buckling Case 1: 5.91
Plate Buckling Case 2: 40.2
Plate Buckling Case 3: 7.32
Shear Buckling: 3.86

where the minimum *FOS* is 2.57, which determines the failure mode of shear buckling, yielding,

Failure Load: 1161 N

Section 3 (3 top flange layers):

Flexural Tension: 8.26
Flexural Compression: 3.42
Centroidal Shear Stress: 5.15
Glue Shear Stress: 16.47
Plate Buckling Case 1: 17.70
Plate Buckling Case 2: 120.4
Plate Buckling Case 3: 14.13
Shear Buckling: 3.86

where the minimum *FOS* is 3.42, which determines the failure mode of shear buckling, yielding,

Failure Load: 1544 N

The minimum failure load is: 1161N

3.7 Design 6

Changes applied to Design 5:

Section 1 and 2 will fail because of shear buckling, one way to increase the FOS of this is to add more diaphragms. Section 3 will fail because of the Flexural compression. Increase the second moment of area is a way to increase the FOS of this. Therefore, the following changes were made.

Add an I beam in the middle of section 3 and a half diaphragm on the splice of section 1 and the slanted section. This decision was made on site when constructing the bridge, because we found that we had some material left.

1. I beam (120mm*80mm*1.27mm)

The addition of this I beam was indented to reinforce the middle part of the bridge, where approximately the maximum moment is. We assume that it will increase the second moment of area as well as decrease the shear stress buckling.

Plate Buckling Case 3: Down
Glue Shear Stress: Up
Flexural Compression: Up

2. Half diaphragm

We assume this half diaphragm will decrease the interval of diaphragm at the section 1, thus increasing the FOS of shear buckling. Another reason is that we thought that the strength between section one and three would decrease significantly because two parts were completely separated.

Therefore, a diaphragm can decrease the chance of shear buckling a lot.

4 Conclusion

Throughout the project the team had to apply the knowledge gained from the course into a real-life design, which meant not only apply the known equations, whereas understand the background of each one and their relation to optimize the iterations and decisions needed to produce the best bridge possible. Understanding how to approach the variables and uncertainties within the project is a key takeaway from how engineering works, having not only a material constraint, but also a time one as well, the team had to plan and prioritize the approaches taken to deliver the best possible bridge, something that team 306 managed to approach in the best way possible, knowing how to work with a team, and tackling the problems from different angles and views, until all the decisions were made, every iteration polished, and the dock firmly glued to finalize the creation of the bridge: “The pineapple juice 1”.

Appendix

1. Code for SFE and BME

```
import matplotlib.pyplot as plt
import numpy as np
#Deifine support location relative to the left
length = 1200
support1 = (length-1200)/2
support2 = length-(length-1200)/2
#our case
print(support1,support2)
#calculate reaction forces based on the positions of the load: by
equilibrium of the forces and equilibrium of the moment

def reaction_calc(center_position):
    #sum of forces that's on the bridge is 0
    forcr_list2 = [] #a two dimension list, second dimension: force,loc
    loc_middle_car_force = [center_position - 88, center_position + 88]
# [0] is on the left
    loc_last_car_force = [center_position - 428, center_position - 252]
    loc_first_car_force = [center_position + 252, center_position +
428]
    list_load = [loc_first_car_force, loc_middle_car_force,
loc_last_car_force]
    # force_dic = {"first_car": [91, loc_first_car_force],
"middle_car": [67.5, loc_middle_car_force],
    #              "last_car": [67.5, loc_last_car_force]}  # forces is
the sigle point force at one wheel.
    # force_dic = {"first_car": [67.5, loc_first_car_force],
"middle_car": [67.5, loc_middle_car_force],
    #              "last_car": [91, loc_last_car_force]}  # forces is
the sigle point force at one wheel.
    force_dic = {"first_car": [66.6, loc_first_car_force],
"middle_car": [66.6, loc_middle_car_force],
                 "last_car": [66.6, loc_last_car_force]}
    #find the forces on the bridge
    for i in range(0,len(list_load)):
        for j in range(0,len(list_load[i])):
            if list_load[i][j] >= 0 and list_load[i][j] <= length:

forcr_list2.append([list(force_dic.values())[i][0],list(force_dic.value
s())[i][1][j]])

    #Calculate the moment at support 1
    #Use the moment to get the yB second
    yB = 0
    for m in range(0,len(forcr_list2)):
        yB += forcr_list2[m][0]*(forcr_list2[m][1]-support1)
    yB = yB/(support2-support1)
    # print("YB:",yB)
    #Find ya by looping all the forces and
    yTotal = 0
    for k in forcr_list2:
        yTotal += k[0]
    # print(yTotal)
```

```
    yA = yTotal-yB
    # print("YA:",yA)
    forcr_list2 = [[-x, y] for x, y in forcr_list2]
    # print(forcr_list2)
    forcr_list2.append([yA,support1])
    forcr_list2.append([yB, support2])
    re_list = sorted(forcr_list2,key=lambda l:l[1])
    # print(re_list)
    return re_list
#input is the position of the center of the train
#output will be the intensity of reaction forces, and the location of
loads and forces
def graph_shear_force(shearforce):

    plt.figure(figsize=(6, 10))
    x = [p[1] for p in shearforce]
    y = [p[0] for p in shearforce]
    plt.step(x,y, where="post")
    plt.plot(x, y, 'o')
    plt.plot(x, y, 'o')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title("Shear force diagram")
    plt.grid(True)
    # plt.gca().invert_yaxis()  # optional: if you want y increasing
downward (like image coords)
    plt.show()
def data_process(forces):
    #forces is a set of force in a sequencial order from left to right.
if linear then: [force,"linear",[start,end]], if not: [force,location]
    # Calculate SFD Hard
    # calculate BMD
    # the graph of maximum moment on BMD
    # print(forces)
    shearforce = [] #two dimension list, second element is the loc,
first is the shear
    shear = 0
    for i in forces:
        # print(i)
        shear += i[0]
        shearforce.append([shear,i[1]])
    # print(shearforce)
    # graph_shear_force(shearforce)
    # graph_BMD(shearforce)
    # maximum bending moment
    Mmax = [-1000,0] # a two dimension list that the first element is
the moment the second is the position
    Moment = 0
    for i in range(1,len(shearforce)):
        distance = shearforce[i][1] - shearforce[i-1][1]
        Moment += distance* shearforce[i-1][0]/1000
        # print(Moment,distance)
        if Moment >= Mmax[0]:
            Mmax = [Moment, shearforce[i][1]]
    # print(Mmax)
    #shear max
    shearmax = [-10000000,0]
```

```
        for i in shearforce:
            if abs(i[0]) > shearmax[0]:
                shearmax = [abs(i[0]), i[1]]

        return [Mmax,shearmax]

    #input: the position of forces, amount of force
    #output:
    #looping through every position of the train, graph each maximum
    moments according to the positon of the train. find the maximum moment.
    def graph_max_moment(moment,loc_train_center):
        import matplotlib.pyplot as plt
        # print(moment[1])
        # Extract x and y data
        y = [p[0] for p in moment]  # moment values
        x = loc_train_center  # locations

        # Find maximum (x, y)
        max_y = max(y)
        i = y.index(max_y)
        max_x = x[i]
        print("max moment on bridge:",moment[i][1])
        # Print max coordinate
        print(f"Maximum moment occurs at x = {max_x}, moment = {max_y}")

        #Plot
        plt.figure(figsize=(8, 6))
        plt.plot(x, y, marker='.', linewidth=0.01)

        # Mark the max point
        plt.scatter(max_x, max_y, color='red')
        plt.text(max_x, max_y, f"({max_x:.2f}, {max_y:.2f})", color='red',
                 verticalalignment='bottom')

        plt.xlabel("Train Center Location (mm)")
        plt.ylabel("Moment (kN·mm)")
        plt.title("Moment Diagram vs. Train Center Location")
        plt.grid(True)
        plt.show()

    def graph_BMD(shearforce):
        Moment = 0
        Moment_list = [[0, shearforce[0][1]]]
        list_m = []
        list_x = []
        for i in range(1, len(shearforce)):
            distance = shearforce[i][1] - shearforce[i - 1][1]
            Moment += distance * shearforce[i - 1][0] / 1000
            Moment_list.append([Moment, shearforce[i][1]])
        # determine the left side moment, right side moment,  expression
        print(Moment_list)
        for k in range(0,length,1):
            lf_m = 0
            lf_m_loc = 0
            rf_m = 0
            rf_m_loc = 0
            for i in range(1, len(Moment_list)):
```

```
            if Moment_list[i - 1][1] <= k and Moment_list[i - 1][1] >=
lf_m_loc:

                lf_m = Moment_list[i - 1][0]
                lf_m_loc = Moment_list[i - 1][1]
                rf_m = Moment_list[i][0]
                rf_m_loc = Moment_list[i][1]

        if rf_m_loc - lf_m_loc != 0:
            point_m = (rf_m - lf_m) / (rf_m_loc - lf_m_loc) * (k -
lf_m_loc) + lf_m
        else:
            point_m = lf_m
        # print(point_m)
        list_m.append(-point_m)
        list_x.append(k)
    # Plot
    plt.figure(figsize=(12, 6))
    plt.plot(list_x, list_m, label="Bending Moment Diagram",
color="red", linewidth=2)
    plt.xlabel("Bridge Position (mm)")
    plt.ylabel("Moment(KN*mm")
    plt.title("Bending Moment Diagram")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.legend()
    plt.show()

def plot_shear_moment_same(list_point_maxnmin_shear,
list_point_maxnmin_moment,
                           abs_shear=True):
    # Extract envelope values
    x_vals = [p[0][0] for p in list_point_maxnmin_shear]  # bridge
positions

    shear_vals = [max(abs(p[0][1]), abs(p[1][1])) if abs_shear else
max(p[0][1], p[1][1])
                  for p in list_point_maxnmin_shear]

    moment_vals = [max(p[0][1], p[1][1]) for p in
list_point_maxnmin_moment]

    # Plot
    plt.figure(figsize=(12, 6))
    plt.plot(x_vals, shear_vals, label="Shear Envelope", color="blue",
linewidth=2)
    plt.plot(x_vals, moment_vals, label="Moment Envelope", color="red",
linewidth=2)
    plt.xlabel("Bridge Position (mm)")
    plt.ylabel("Value")
    plt.title("Shear & Moment Envelope Curves on Same Plot")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.legend()
    plt.show()

#find the envelop shear force
def graph_max_shear(shearforces,loc_train_center):
    #shearforces: two dimension vector, first element shear, second is
```

```
    loc
        import matplotlib.pyplot as plt
        # Extract x and y data
        y = [p[0] for p in shearforces]  # shear values
        x = loc_train_center  # locations

        # Find maximum (x, y)
        max_y = max(y)
        i = y.index(max_y)
        max_x = x[i]
        print("max on bridge loc:",shearforces[i][1])
        # Print max coordinate
        print(f"Maximum shear occurs at x = {max_x}, shear = {max_y}")
        # Plot
        plt.figure(figsize=(8, 6))
        plt.plot(x, y, marker='o', linewidth=1)
        # Mark the max point
        plt.scatter(max_x, max_y, color='red')
        plt.text(max_x, max_y, f"({max_x:.2f}, {max_y:.2f})", color='red',
                 verticalalignment='bottom')
        plt.xlabel("Train Center Location (mm)")
        plt.ylabel("Shear Force (N)")
        plt.title("Shear Force Envelope Diagram")
        plt.grid(True)
        plt.show()

    def calc_shear_along_bridge(position,forces):
        # get the shear of a specific *position* accoding to the forces
    list
        #output the shear force
        shearforce = []  # two dimension list, second element is the loc,
    first is the shear
        shear = 0
        for i in forces:
            # print(i)
            shear += i[0]
            shearforce.append([shear, i[1]])
        # print(shearforce)
        point_shear = 0
        least_left = -100000
        for i in shearforce:
            if i[1] < position and i[1]>least_left:
                least_left = i[1]
                point_shear = i[0]
                if position == 200:
                    print(point_shear)
        #Shear = The closest left shear
        return point_shear

    def cal_moment_along_bridge(position,forces):
        shearforce = []  # two dimension list, second element is the loc,
    first is the shear
        shear = 0
        for i in forces:
            # print(i)
            shear += i[0]
            shearforce.append([shear, i[1]])
```

```
    Moment = 0
    Moment_list = [[0,0]]
    for i in range(1, len(shearforce)):
        distance = shearforce[i][1] - shearforce[i - 1][1]
        Moment += distance * shearforce[i - 1][0] / 1000
        Moment_list.append([Moment,shearforce[i][1]])
    #determine the left side moment, right side moment,  expression
    lf_m = 0
    lf_m_loc = 0
    rf_m = 0
    rf_m_loc = 0

    for i in range(1,len(Moment_list)):
        if Moment_list[i-1][1] <= position and Moment_list[i-1][1] >=
lf_m_loc:
            lf_m = Moment_list[i-1][0]
            lf_m_loc = Moment_list[i-1][1]
            rf_m = Moment_list[i][0]
            rf_m_loc = Moment_list[i][1]

    if rf_m_loc-lf_m_loc != 0:
        point_m = (rf_m-lf_m)/(rf_m_loc-lf_m_loc)*(position -
lf_m_loc)+lf_m
    else:
        point_m = lf_m
    # print(point_m)
    return point_m
        # print(Moment,distance)
def scatter_color_by_y(data, name):
    xs = []
    ys = []

    for (x_pair, y_pair) in data:
        a, b = x_pair
        c, d = y_pair
        xs.extend([a, c])
        ys.extend([b, d])

    xs = np.array(xs)
    ys = np.array(ys)

    # Max & Min values
    y_max = np.max(ys)
    y_min = np.min(ys)
    # Index lists (may contain multiple points)
    max_idx = np.where(ys == y_max)[0]
    min_idx = np.where(ys == y_min)[0]
    plt.figure(figsize=(10, 6))
    # MAIN SCATTER — only scatter contributing to colorbar
    sc = plt.scatter(xs, ys, c=ys, cmap='coolwarm', s=25, alpha=0.85)
    plt.colorbar(sc, label=name)
    # MAXIMA (highlight + annotate coordinates)
    plt.scatter(xs[max_idx], ys[max_idx],
                facecolors='none', edgecolors='red',
                s=120, linewidths=2.5, label="Maxima")
    for i in max_idx:
        plt.annotate(f"({xs[i]:.2f}, {ys[i]:.2f})",
```

```
                    (xs[i], ys[i]),
                    textcoords="offset points", xytext=(6, 6),
                    fontsize=9, color='red')
    # MINIMA
    plt.scatter(xs[min_idx], ys[min_idx],
                facecolors='none', edgecolors='blue',
                s=120, linewidths=2.5, label="Minima")
    if name != "Moment":
        # only annotate minima if NOT "Moment"
        for i in min_idx:
            plt.annotate(f"({xs[i]:.2f}, {ys[i]:.2f})",
                         (xs[i], ys[i]),
                         textcoords="offset points", xytext=(6, 6),
                         fontsize=9, color='blue')
    plt.xlabel("Bridge Position (mm)")
    plt.ylabel(name)
    plt.title(f"{name} vs. Bridge Length")
    plt.grid(True, linestyle='--', alpha=0.4)
    plt.legend()

    plt.show()

def plot_scaled_moment_shear(list_point_maxnmin_shear,
list_point_maxnmin_moment):

    # X positions (bridge coordinates)
    x_vals = [p[0][0] for p in list_point_maxnmin_shear]

    # Extract shear envelope (absolute)
    shear_vals = [max(abs(p[0][1]), abs(p[1][1])) for p in
list_point_maxnmin_shear]

    # Extract moment envelope (positive max)
    moment_vals = [max(p[0][1], p[1][1]) for p in
list_point_maxnmin_moment]
    # --- Scaling ---
    max_shear = max(shear_vals)
    max_moment = max(moment_vals)
    scale = max_shear / max_moment if max_moment != 0 else 1
    moment_scaled = [m * scale for m in moment_vals]
    # FIND INTERSECTIONS: shear_vals = moment_scaled
    intersections = []
    for i in range(1, len(x_vals)):
        x0, x1 = x_vals[i-1], x_vals[i]
        y0_s, y1_s = shear_vals[i-1], shear_vals[i]
        y0_m, y1_m = moment_scaled[i-1], moment_scaled[i]
        # Check for sign crossing (intersection between segments)
        if (y0_s - y0_m) * (y1_s - y1_m) <= 0:
            # Solve linear interpolation for intersection point
            # shear(x) = y0_s + t*(y1_s - y0_s)
            # moment(x)= y0_m + t*(y1_m - y0_m)
            # Find t where they equal
            numerator = (y0_s - y0_m)
            denominator = (y0_s - y0_m) - (y1_s - y1_m)
            if denominator != 0:
                t = numerator / denominator
                x_int = x0 + t * (x1 - x0)
```

```
                y_int = y0_s + t * (y1_s - y0_s)
                intersections.append((x_int, y_int))
    # Print intersection points
    print("\n=== Intersection Points (Scaled Moment = Shear) ===")
    if len(intersections) == 0:
        print("No intersections found.")
    else:
        for (xi, yi) in intersections:
            print(f"x = {xi:.2f} mm,  value = {yi:.2f}")
    plt.figure(figsize=(12, 6))
    plt.plot(x_vals, shear_vals, linewidth=2.5, color="blue",
label="Shear Envelope")
    plt.plot(x_vals, moment_scaled, linewidth=2.5, color="red",
label=f"Moment Envelope (scaled ×{scale:.3f})")
    # Plot intersection points
    for (xi, yi) in intersections:
        plt.scatter(xi, yi, color="black", s=50)
        plt.text(xi, yi, f"({xi:.0f}, {yi:.1f})", fontsize=9,
color="black", ha="left", va="bottom")

    plt.xlabel("Bridge Position (mm)")
    plt.ylabel("Value (Scaled)")
    plt.title("Scaled Moment Envelope vs. Shear Envelope (Intersections
Marked)")
    plt.grid(True, linestyle="--", alpha=0.5)

    # Centered legend
    plt.legend(loc="upper center", bbox_to_anchor=(0.5, -0.12), ncol=2)

    plt.tight_layout()
    plt.show()

def return_maxnmin(list,index):
    #design for a second dimension list sorting
    a = [p[index] for p in list]
    maxa = max(a)
    mina = min(a)
    indexmax = a.index(maxa)
    indexmin = a.index(mina)
    return list[indexmax],list[indexmin]
#design aspect, inertia at different location.
list_1 = []
list_2 = []
list_loc = []
list_point_maxnmin_shear = [] #Three dimensionn list. Second dimension:
bridge loc: first: loc, second: max, third: min
list_point_maxnmin_moment = []
for j in range(0,int(support2)-int(support1),1):
    #build up the list
    list_point_maxnmin_shear.append([])
    list_point_maxnmin_moment.append([])

for i in range(-428,length+428,1):
    list_1.append(data_process(reaction_calc(i))[0])
    list_2.append(data_process(reaction_calc(i))[1])
    list_loc.append(i)
    for k in range(0,int(support2)-int(support1),1):
```

```
            #add the whole data to the list
            # print(k-1,len(list_point_maxnmin_shear))

    list_point_maxnmin_shear[k].append([k,calc_shear_along_bridge(k,reactio
    n_calc(i))])
            list_point_maxnmin_moment[k].append([k,
    cal_moment_along_bridge(k, reaction_calc(i))])

    for m in range(0,len(list_point_maxnmin_shear)):
        list_point_maxnmin_shear[m] =
    return_maxnmin(list_point_maxnmin_shear[m],1)
        list_point_maxnmin_moment[m] =
    return_maxnmin(list_point_maxnmin_moment[m], 1)
    # print(list_point_maxnmin_shear)
    scatter_color_by_y(list_point_maxnmin_shear,"Shear Force")
    scatter_color_by_y(list_point_maxnmin_moment,"Moment")
    # # print(len(list_1),len(list_2))
    graph_max_moment(list_1,list_loc)
    graph_max_shear(list_2,list_loc)
    plot_shear_moment_same(list_point_maxnmin_shear,
    list_point_maxnmin_moment)
    plot_scaled_moment_shear(list_point_maxnmin_shear,
    list_point_maxnmin_moment)

    # print(graph_shear_along_bridge(600,reaction_calc(800)))
    # print(data_process(reaction_calc(600))[0])
```

3. Code for FOS calculations

```
import math
# 3. Maximum Moment and Shear Force
M =
V =


# 4. Bridge Geometry (Design Parameters)
t =
b1 =
b2 =
b3_4 = t
b5_6 =
h1 =
h2 = t
h_bot =
h3_4 = h_bot - t
h5_6 = t
a =

# 5. Calculate Cross-sectional Geometric Properties (y bar, Q, I)
def cross_section():
   A1 = b1 * h1
   A2 = b2 * h2
   A3_4 = 2 * h3_4 * b3_4
   A5_6 = 2 * h5_6 * b5_6
   # total area
   At = A1 + A2 + A3_4 + A5_6
```

```
    y1 = h_bot + h1/2
    y2 = h2/2
    y3_4 = h2 + h3_4/2
    y5_6 = h_bot - h5_6 + h5_6/2
    # y bar
    y_bar = (A1 * y1 + A2 * y2 + A3_4 * y3_4 + A5_6 * y5_6)/At
    # Second Moment of Inertia
    I1 =  (b1 * (h1) ** 3)/12
    I2 = (b2 * (h2) ** 3)/12
    I3_4 = 2 * ((b3_4 * h3_4 ** 3)/12)
    I5_6 = 2 * (b5_6 * h5_6 ** 3/12)
    It1 = I1 + I2 + I3_4 + I5_6
    # Distance of Individual Distance to y bar
    d1 = abs(y_bar - y1)
    d2 = abs(y_bar - y2)
    d3_4 = abs(y_bar - y3_4)
    d5_6 = abs(y_bar - y5_6)
    # Aidi^2
    Ad1 = A1 * d1 ** 2
    Ad2 = A2 * d2 ** 2
    Ad3_4 = A3_4 * d3_4 ** 2
    Ad5_6 = A5_6 * d5_6 ** 2
    Adt = Ad1 + Ad2 + Ad3_4 + Ad5_6
    # Total Moment of Inertia
    It2 = It1 + Adt
    # Q at centroidal axes
    Q_cent = b2 * h2 * (y_bar - h2/2) + 2 * b3_4 * (y_bar - h2) * (y_bar - h2)/2
    # Q at glue location
    Q_glue = b1 * h1 * (h_bot + h1/2 - y_bar)
    return y_bar, Q_cent, Q_glue, It2, At

# 6. Calculate Applied Stresses
def applied_stress(y_bar, Q_cent, Q_glue, It2):
    y_top = h_bot + h1 - y_bar
    y_bottom = y_bar
    # Max Compressive Stress (Top)
    s_comp = (M * y_top) / It2
    # Max Tensile Stress (Bottom)
    s_tens = (M * y_bottom) / It2
    # Max Shear Stress (at centroid)
    b_cent = 2 * b3_4
    tau_cent = (V * Q_cent)/(It2 * b_cent)
    # Max Shear Stress (at glue location)
    b_glue = 2 * (b5_6)
    tau_glue = (V * Q_glue)/(It2 * b_glue)
    return s_comp, s_tens, tau_cent, tau_glue, y_top

# 7. Calculate material/thin plate buckling capacities
def plate_buckling(y_bar, Q, It2):
    Vm = 0.2
    E = 4000
    # Case 1 buckling (compressive flange between the webs)
    s_buck1 = (4 * (math.pi) ** 2 * E) * (h1 / b2) ** 2/ (12 * (1 - Vm ** 2))
    # Case 2 buckling (tips of the compressive flange)
    s_buck2 = (0.425 * (math.pi) ** 2 * E) * (h1 / ((b1 - b2)/2)) ** 2/ (12 * (1 - Vm
** 2))
    # Case 3 buckling (webs due to flexural stresses)
    s_buck3 = (6 * (math.pi) ** 2 * E) *  (b3_4 / (h_bot - y_bar)) ** 2/ (12 * (1 - Vm
** 2))
    # Case 4 buckling (shear buckling of the webs)
    tau_buckV = ((5 * (math.pi) ** 2 * E) * ((b3_4/h_bot) ** 2 + (t/a) ** 2))/ (12 * (1
- Vm ** 2))
```

```
    return s_buck1, s_buck2, s_buck3, tau_buckV

# 8. FOS
def factor_of_safety(s_comp, s_tens, tau_cent, tau_glue, s_buck1, s_buck2, s_buck3,
tau_buckV):
    fos_tens = 30/s_tens
    fos_comp = 6 / s_comp
    fos_shear = 4 / tau_cent
    fos_glue = 2 / tau_glue
    fos_buck1 = s_buck1 / s_comp
    fos_buck2 = s_buck2 / s_comp
    fos_buck3 = s_buck3 / s_comp
    fos_buckV = tau_buckV / tau_cent
    return fos_tens, fos_comp, fos_shear, fos_glue, fos_buck1, fos_buck2, fos_buck3,
fos_buckV

# 9. Min FOS and the failure load Pfail
def min_FOS(fos_tens, fos_comp, fos_shear, fos_glue, fos_buck1, fos_buck2, fos_buck3,
fos_buckV, s_buck1, y_top):
    min_FOS = min(fos_tens, fos_comp, fos_shear, fos_glue, fos_buck1, fos_buck2,
fos_buck3, fos_buckV)
    Pf = 452 * min_FOS
    return min_FOS, Pf

# --- Cross-section properties ---
y_bar, Q_cent, Q_glue, It2, At = cross_section()

# --- Applied stresses ---
s_comp, s_tens, tau_cent, tau_glue, y_top = applied_stress(y_bar, Q_cent, Q_glue, It2)

# --- Buckling ---
s_buck1, s_buck2, s_buck3, tau_buckV = plate_buckling(y_bar, Q_cent, It2)

# --- Factor of Safety ---
fos_tens, fos_comp, fos_shear, fos_glue, fos_buck1, fos_buck2, fos_buck3, fos_buckV =
\
    factor_of_safety(s_comp, s_tens, tau_cent, tau_glue, s_buck1, s_buck2, s_buck3,
tau_buckV)

# --- Min FOS & Failure Load ---
min_fos, Pf = min_FOS(fos_tens, fos_comp, fos_shear, fos_glue, fos_buck1, fos_buck2,
fos_buck3, fos_buckV, s_buck1, y_top)

# --- Create dictionary of all variables ---
all_values = {
    # Load & Geometry
    "M": M,
    "V": V,
    # Cross-section
    "y_bar": y_bar,
    "Q_cent": Q_cent,
    "Q_glue": Q_glue,
    "I": It2,
    "Area": At,
    # Applied Stresses
    "s_comp": s_comp,
    "s_tens": s_tens,
    "tau_cent": tau_cent,
    "tau_glue": tau_glue,
    "y_top": y_top,
```

```
    # Buckling
    "s_buck1": s_buck1,
    "s_buck2": s_buck2,
    "s_buck3": s_buck3,
    "tau_buckV": tau_buckV,
    # Factor of Safety
    "fos_tens": fos_tens,
    "fos_comp": fos_comp,
    "fos_shear": fos_shear,
    "fos_glue": fos_glue,
    "fos_buck1": fos_buck1,
    "fos_buck2": fos_buck2,
    "fos_buck3": fos_buck3,
    "fos_buckV": fos_buckV,
    # Min FOS & Failure Load
    "min_FOS": min_fos,
    "Failure_Load": Pf}

# --- Print all variables ---
print("=== All Calculated Values ===")
for name, value in all_values.items():
    print(f"{name}: {value:.3f}")
```