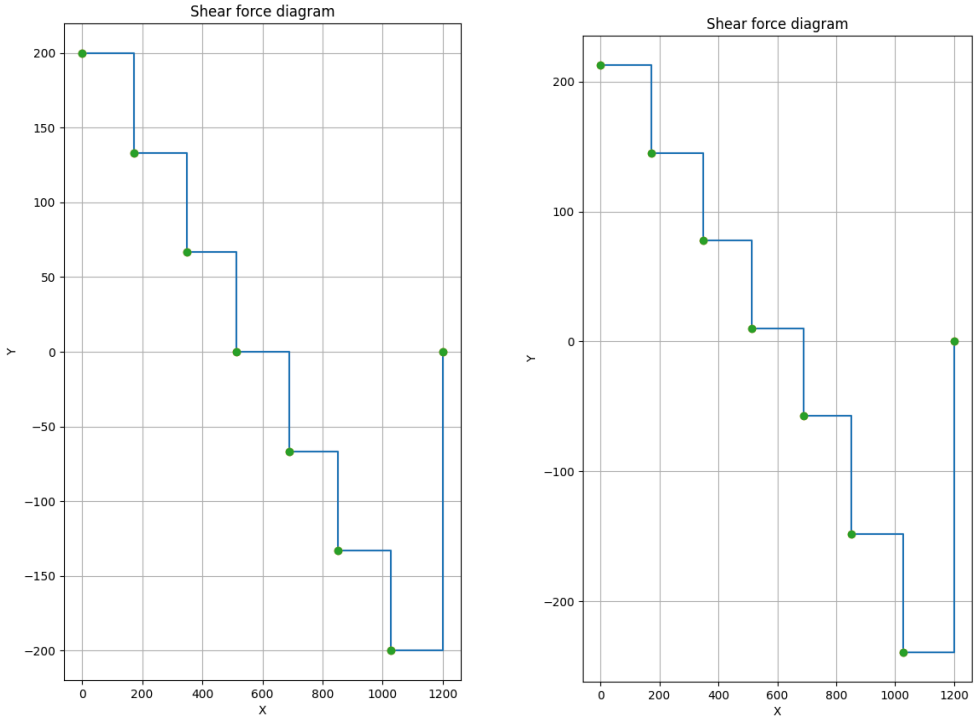
First Portion (Hand Calculations)

- [ ] FOS against failure and failure mode for Design 0 under a moving 400 N train loaded according to Load Case 1 (1.6.1)
- [ ] Any other hand calculations to determine the FOS against failure and failure mode of your final design under a train loaded according to the base case of Load Case 2 (1.6.2)

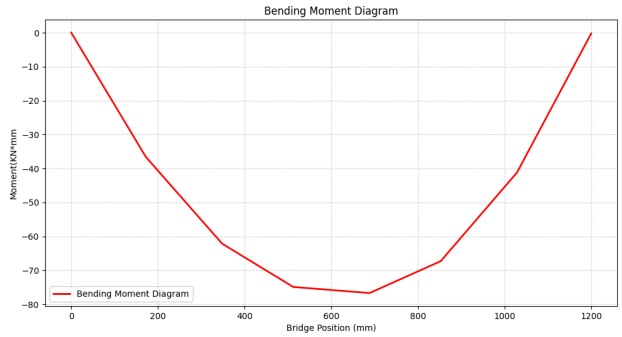
Second Portion (Programming)

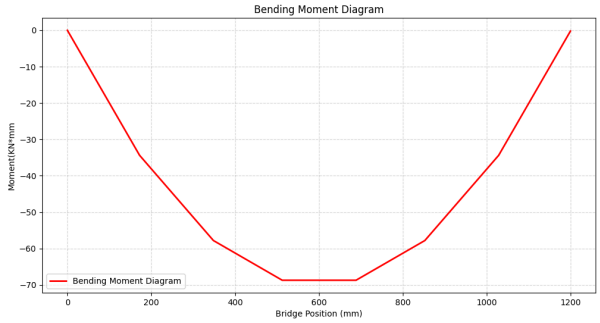
- [ ] Code output for all intermediate calculations (centroidal axis location, moment of area, etc.) performed by the hand calculations of Design 0 under a moving 400 N train loaded according to Load Case 1
- [ ] SFD of Design 0 for Load Case 1 and Load Case 2 (Bobby)



(left: SFD of case one when the train is at the center. Right: SFD of case two when the train is at the middle)

- [ ] BMD of Design 0 for Load Case 1 and for your final design under Load Case 2 (Bobby)





(Up: BMD of case two when the train is at the middle; Down: BMD of case one when the train is at the center)

- [ ] Code output of all FOS values along the bridge for Load Case 1 and for your final design under Load Case 2
- [ ] Include the entire script: comments, formatting, user-defined functions; indicate installed packages (ex. NumPy) (Bobby, Candy)

1.Code for all the graphing and calculation for reaction forces, moment, and shear:
Installed package: Matplotlib, Numpy
Code:

```
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.values(
))[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,reaction_
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])
```

2. Code for calculating the FOS (math was imported)

```
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
```

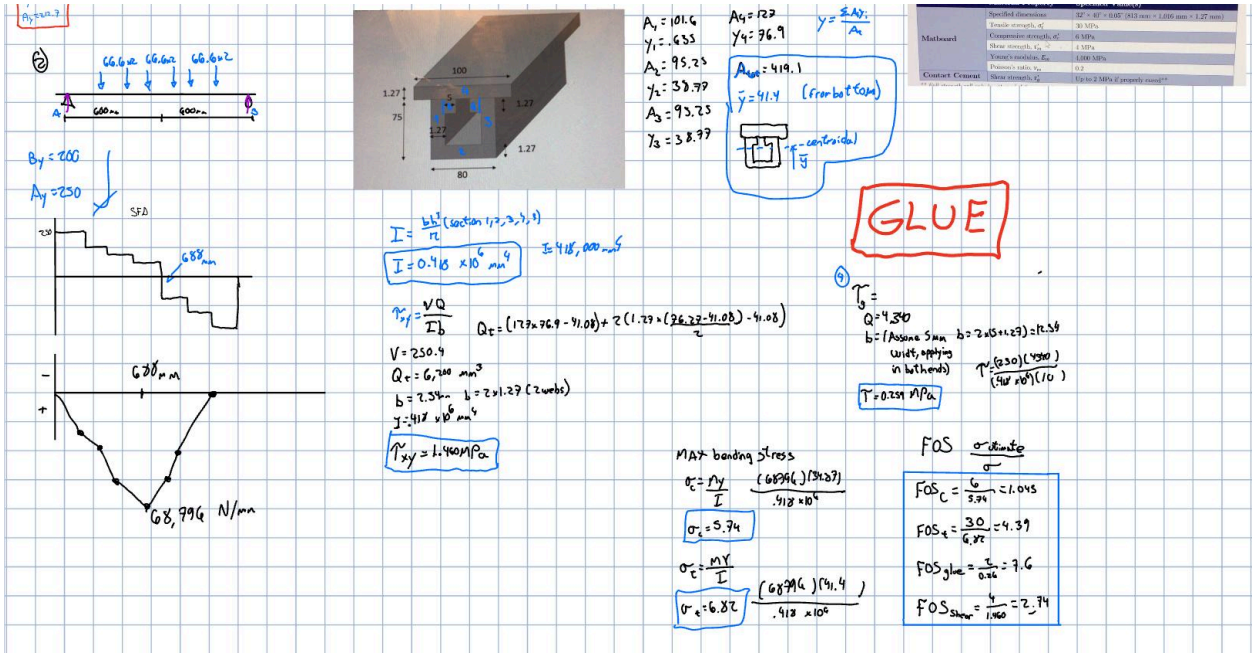
```
    y1 = h_bot + h1/2
    y2 = h2/2
    y3_4 = h2 + h3_4/2
    y5_6 = h_bot - h5_6 + h5_6/2
    # y bar
    y_bar = (A1 * y1 + A2 * y2 + A3_4 * y3_4 + A5_6 * y5_6)/At
    # Second Moment of Inertia
    I1 =  (b1 * (h1) ** 3)/12
    I2 = (b2 * (h2) ** 3)/12
    I3_4 = 2 * ((b3_4 * h3_4 ** 3)/12)
    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,
```

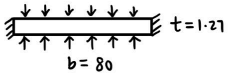
```
    "tau_buckV": tau_buckV,
    # Factor of Safety
    "fos_tens": fos_tens,
    "fos_comp": fos_comp,
    "fos_shear": fos_shear,
    "fos_glue": fos_glue,
    "fos_buck1": fos_buck1,
    "fos_buck2": fos_buck2,
    "fos_buck3": fos_buck3,
    "fos_buckV": fos_buckV,
    # Min FOS & Failure Load
    "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}")
```

Hand calculation of design 0 (load case 1)



Case 1 (buckling of the compressive flange between the webs)

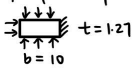
$\sigma_{crit} = \frac{4\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2$ $= 3.45$ MPa



$FOS = \frac{\sigma_{crit}}{\sigma_c} = \frac{3.45}{5.94} = 0.581$

case 2 (buckling of the tips of the compressive flange)

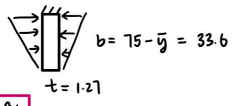
$\sigma_{crit} = \frac{0.425\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2$ $= 23.5$ MPa



$FOS = \frac{\sigma_{crit}}{\sigma_c} = \frac{23.5}{5.94} = 3.96$

case 3 (buckling of the webs due to the flexural stresse)

$\sigma_{crit} = \frac{6\pi^2 E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2$ $= 29.4$ MPa



$FOS = \frac{\sigma_{crit}}{\sigma_c} = \frac{29.4}{5.94} = 4.95$

case 4 (shear buckling of the web)

$\tau_{crit} = \frac{5\pi^2 E}{12(1-\mu^2)}\left(\left(\frac{t}{h}\right)^2 + \left(\frac{t}{a}\right)^2\right)$ $= 5.09$ MPa

h = 75
a = 400
t = 1.27

$FOS = \frac{\tau_{crit}}{\tau_{cent}} = \frac{5.09}{1.44} = 3.53$

FOStens = 4.45
FOScomp = 1.01
FOSflex_buck_1 = 0.581
FOSflex_buck_2 = 3.96
FOSflex_buck_3 = 4.95
FOSshear = 2.77
FOSglue = 7.40
FOSshear_buck = 3.53

Minimum FOS: FOSflex_buck_1

Failure Mode: Case 1 buckling of the compressive flange between the webs
DESIGN 0 Calculation: (LOAD CASE 1)
M: 69380.000
V: 256.740
y_bar: 41.431
Q_cent: 6193.283
Q_glue: 4343.896
I: 418352.209
Area: 428.574
s_comp: 5.778
s_tens: 6.871
tau_cent: 1.496
tau_glue: 0.267

y_top: 34.839
s_buck1: 3.455
s_buck2: 23.491
s_buck3: 29.430
tau_buckV: 5.086
fos_tens: 4.366
fos_comp: 1.038
fos_shear: 2.673
fos_glue: 7.502
fos_buck1: 0.598
fos_buck2: 4.066
fos_buck3: 5.094
fos_buckV: 3.399
min_FOS: 0.598
Failure_Load: 239.165

DESIGN 0 (LOAD CASE 2):
M: 78100.000
V: 304.100
y_bar: 41.431
Q_cent: 6193.283
Q_glue: 4343.896
I: 418352.209
Area: 428.574
s_comp: 6.504
s_tens: 7.735
tau_cent: 1.772
tau_glue: 0.316
y_top: 34.839
s_buck1: 3.455
s_buck2: 23.491
s_buck3: 29.430
tau_buckV: 5.086
fos_tens: 3.879
fos_comp: 0.923
fos_shear: 2.257
fos_glue: 6.334
fos_buck1: 0.531
fos_buck2: 3.612
fos_buck3: 4.525
fos_buckV: 2.870
min_FOS: 0.531
Failure_Load: 240.082

ITERATION 1: (DOUBLE THICKNESS)
M: 78100.000
V: 304.100
y_bar: 49.540
Q_cent: 7927.874
Q_glue: 6789.380
I: 541652.987
Area: 555.574
s_comp: 4.037
s_tens: 7.143
tau_cent: 1.752
tau_glue: 0.381
y_top: 28.000
s_buck1: 13.818
s_buck2: 93.964
s_buck3: 51.163
tau_buckV: 5.086
fos_tens: 4.200
fos_comp: 1.486
fos_shear: 2.283
fos_glue: 5.247
fos_buck1: 3.423
fos_buck2: 23.274
fos_buck3: 12.673
fos_buckV: 2.902
min_FOS: 1.486
Failure_Load: 671.745

Iteration 2 (increase height to 80 mm)
M: 78100.000
V: 208.300
y_bar: 52.512
Q_cent: 8605.302
Q_glue: 7304.636
I: 624304.226
Area: 568.274
s_comp: 3.757
s_tens: 6.569
tau_cent: 1.130
tau_glue: 0.244
y_top: 30.028
s_buck1: 13.818
s_buck2: 93.964

s_buck3: 43.890
tau_buckV: 4.491
fos_tens: 4.567
fos_comp: 1.597
fos_shear: 3.539
fos_glue: 8.206
fos_buck1: 3.678
fos_buck2: 25.014
fos_buck3: 11.684
fos_buckV: 3.973
min_FOS: 1.597
Failure_Load: 721.943

ITERATION 3 (DOUBLE HEIGHT TO 160mm; decrease diaphragm spacing to 200 > 5 diaphragms):
M: 78100.000
V: 304.100
y_bar: 97.944
Q_cent: 21755.711
Q_glue: 16084.898
I: 2996000.654
Area: 771.474
s_comp: 1.684
s_tens: 2.553
tau_cent: 0.869
tau_glue: 0.163
y_top: 64.596
s_buck1: 13.818
s_buck2: 93.964
s_buck3: 8.612
tau_buckV: 1.770
fos_tens: 11.750
fos_comp: 3.563
fos_shear: 4.601
fos_glue: 12.250
fos_buck1: 8.206
fos_buck2: 55.802
fos_buck3: 5.114
fos_buckV: 2.036
min_FOS: 2.036
Failure_Load: 920.477

ITERATION 3 (decrease diaphragm spacing to 120 > 9 diaphragms):

M: 78100.000
V: 208.300
y_bar: 98.345
Q_cent: 21895.402
Q_glue: 15982.841
I: 3015040.697
Area: 776.554
s_comp: 1.663
s_tens: 2.547
tau_cent: 0.596
tau_glue: 0.079
y_top: 64.195
s_buck1: 13.818
s_buck2: 93.964
s_buck3: 8.724
tau_buckV: 2.999
fos_tens: 11.776
fos_comp: 3.608
fos_shear: 6.717
fos_glue: 25.358
fos_buck1: 8.310
fos_buck2: 56.508
fos_buck3: 5.247
fos_buckV: 5.035
min_FOS: 3.608
Failure_Load: 1630.923

Thought process of optimizing:
SECTION 1 & 2 (two ends of the bridge): (h_bot = 100)
M: 53700.000
V: 302.300
y_bar: 64.490
Q_cent: 11563.586
Q_glue: 9342.098
I: 1031494.046
Area: 624.154
s_comp: 1.981
s_tens: 3.357
tau_cent: 1.334
tau_glue: 0.196
y_top: 38.050
s_buck1: 13.818
s_buck2: 93.964

s_buck3: 26.301
tau_buckV: 3.843
fos_tens: 8.936
fos_comp: 3.029
fos_shear: 2.998
fos_glue: 10.227
fos_buck1: 6.976
fos_buck2: 47.435
fos_buck3: 13.277
fos_buckV: 2.880
min_FOS: 2.880
Failure_Load: 1301.976

SECTION 3 (middle of the bridge): (h_bot = 120) two LAYERS for cross-section
M: 78100.000
V: 208.300
y_bar: 75.969
Q_cent: 14740.356
Q_glue: 11506.544
I: 1556670.695
Area: 674.954
s_comp: 2.337
s_tens: 3.811
tau_cent: 0.777
tau_glue: 0.110
y_top: 46.571
s_buck1: 13.818
s_buck2: 93.964
s_buck3: 17.106
tau_buckV: 2.999
fos_tens: 7.871
fos_comp: 2.568
fos_shear: 5.151
fos_glue: 18.185
fos_buck1: 5.914
fos_buck2: 40.215
fos_buck3: 7.321
fos_buckV: 3.862
min_FOS: 2.568
Failure_Load: 1160.691

Three layers:
M: 78100.000
V: 208.300

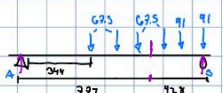
y_bar: 83.444
Q_cent: 16989.276
Q_glue: 14653.491
I: 1794880.965
Area: 801.954
s_comp: 1.756
s_tens: 3.631
tau_cent: 0.776
tau_glue: 0.121
y_top: 40.366
s_buck1: 31.091
s_buck2: 211.420
s_buck3: 24.817
tau_buckV: 2.999
fos_tens: 8.262
fos_comp: 3.416
fos_shear: 5.153
fos_glue: 16.465
fos_buck1: 17.702
fos_buck2: 120.370
fos_buck3: 14.130
fos_buckV: 3.863
min_FOS: 3.416
Failure_Load: 1544.055

HAND CALCULATIONS FOR FINAL DESIGN

Hand Calculations For Final Design

(Two Different Crossections throught the bridge)

Train @ 772 mm Center

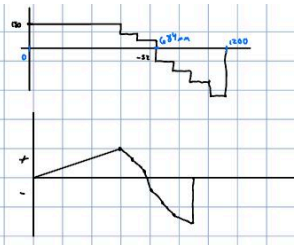


$\Sigma M = 0$ @A

$B_y = \frac{C(x_1)+D(x_2)+C(x_3)+E(x_4)+F(x_5)+E(x_6)+G(x_7)+H(x_8)}{1200} = 302.2$

$A_y + 149.8 = 130\ N$

$B_y = 302.2 = 302\ N$



For Sections 1 and 3 (height of the web: 100 mm)

Max Moment: 33700 Nmm

Max Shear: 302.9 N

$\bar{y} = 64.2$ mm

$I = 1025.264$ mm^4

Centroid

$\tau_{xy} = \frac{VQ}{Ib}$

$Q = 11488$

$b = (1.27 \times 2)$

$\tau_{xy} = 1.33$ MPa

Glue

$\tau_{glue} = \frac{VQ}{Ib}$

$Q = 9915$

$b = (2 \times 7)$

$\tau_{glue} = 0.287$

MAX bending stress

$\sigma_c = \frac{My}{I}$ $\quad \frac{(33700)(38.3)}{1025.264}$

$\sigma_c = 2$

$\sigma_t = \frac{MY}{I}$ $\quad \frac{(33700)(64.2)}{1025.264}$

$\sigma_t = 3.36$

FOS $\frac{\sigma_{ultimate} \text{ or } \tau_{ult}}{\sigma_{max}}$

$FOS_C = \frac{6}{2} = 3$

$FOS_t = \frac{30}{3.36} = 8.94$

$FOS_{glue} = \frac{\tau}{0.287} = 7.21$

$FOS_{Shear} = \frac{4}{1.33} = 3$

CASE 1 buckling of the compressive flange between the webs

$\frac{4\pi^2(4000)}{12(1-0.2^2)}\left(\frac{1.27\times2}{80}\right)^2 = FOS$ $\quad \frac{13.81}{2} = 6.92$

CASE 2 buckling of the tips of the compressive flange

$\frac{0.425\pi^2E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = FOS$ $\quad \frac{93.96}{2} = 46.9$ $\quad t = 1.27\times2$, $b = 10$

CASE 3 buckling of the webs due to flexural stresses

$\frac{6\pi^2E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = FOS$ $\quad \frac{23.8}{2} = 12.9$ $\quad b = 38.3$, $t = 1.27$

CASE 4 shear buckling of the web

$\frac{5\pi^2E}{12(1-\mu^2)}\left(\left(\frac{t}{h}\right)^2+\left(\frac{t}{a}\right)^2\right) = FOS$ $\quad \frac{3.84}{1.33} = 2.89$ $\quad t = 1.27$, $h = 100$, $a = 160$ (Max spacing between diaphragms)

For Section 3 (height of the web: 120mm)

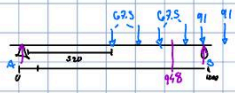
Max Moment: 78100 Nmm

Max Shear: 208.3 N

$\bar{y} = 75.64$ mm

$I = 1547030$ mm^4

Train @ 947 mm Center

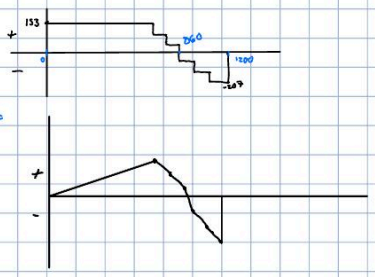


$\Sigma M = 0$ @A

$B_y = \frac{C(x_1)+D(x_2)+C(x_3)+E(x_4)+F(x_5)+E(x_6)+G(x_7)+H(x_8)}{1200} =$

$A_y = 133.2$

$B_y = 207.8$



Centroid

$\tau_{xy} = \frac{VQ}{Ib}$

$Q = 14640$

$b = (1.27\times2)$

$\tau_{xy} = 0.774$ MPa

Glue

$\tau_{glue} = \frac{VQ}{Ib}$

$Q = 8590$

$b = (2\times7)$

$\tau_{glue} = 0.185$ MPa

MAX bending stress

$\sigma_c = \frac{My}{I}$ $\quad \frac{(78100)(46.9)}{1547030}$

$\sigma_c = 2.37$

$\sigma_t = \frac{MY}{I}$ $\quad \frac{(78100)(75.6)}{1547030}$

$\sigma_t = 3.82$

FOS $\frac{\sigma_{ultimate} \text{ or } \tau_{ult}}{\sigma_{max}}$

$FOS_C = \frac{6}{2.37} = 2.53$

$FOS_t = \frac{30}{3.82} = 7.86$

$FOS_{glue} = \frac{\tau}{.185} = 17.93$

$FOS_{Shear} = \frac{4}{0.774} = 3.16$

CASE 1 buckling of the compressive flange between the webs

$\frac{4\pi^2(4000)}{12(1-0.2^2)}\left(\frac{1.27\times2}{80}\right)^2 = FOS$ $\quad \frac{13.8}{2.37} = 6.9$

CASE 2 buckling of the tips of the compressive flange

$\frac{0.425\pi^2E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = FOS$ $\quad \frac{93.96}{2.37} = 39.6$ $\quad t = 1.27\times2$, $b = 10$

CASE 3 buckling of the webs due to flexural stresses

$\frac{6\pi^2E}{12(1-\mu^2)}\left(\frac{t}{b}\right)^2 = FOS$ $\quad \frac{16.85}{2.37} = 7.11$ $\quad b = 44.36$, $t = 1.27$

CASE 4 shear buckling of the web

$\frac{5\pi^2E}{12(1-\mu^2)}\left(\left(\frac{t}{h}\right)^2+\left(\frac{t}{a}\right)^2\right) = FOS$ $\quad \frac{2.99}{0.774} = 3.87$ $\quad t = 1.27$, $h = 120$, $a = 160$ (Max spacing between diaphragms)