source_code.py

```python
''' wheel spacing: 52 176 164 176 164 176 52
0 < x_train < 240 mm
force exerted by each wheel = 400/6 N
'''
import numpy as np
import matplotlib.pyplot as plt
import math


L = 1200


def create_x_bridge():
    ''' Return a list x_bridge of length L'''
    global L
    x_bridge = []

    for i in range(L + 2):
        x_bridge.append(i)


    return x_bridge



def loc_of_wheels(x_train):
    ''' Return the location of wheels given x_train'''
    w1 = 52 + x_train
    w2 = 52 + 176 + x_train
    w3 = 52 + 176 + 164 + x_train
    w4 = 52 + 176 + 164 + 176 + x_train
    w5 = 52 + 176 + 164 + 176 +164 + x_train
    w6 = 52 + 176 + 164 + 176 +164 + 176 + x_train


    return w1, w2, w3, w4, w5, w6

# print(loc_of_wheels(164))
# L = 1200
# w1, w2, w3, w4, w5, w6 = loc_of_wheels(164)
def reaction_forces(w1, w2, w3, w4, w5, w6):
    global L
    By = (400/ 6 * (w1 + w2 + w3 + w4 + w5 + w6)) / L
    Ay = 400 - By

```

```
    return Ay, By

# print(reaction_forces(w1, w2, w3, w4, w5, w6))



def create_V_M_lists(x_train):
    ''' Return the shear force and bending moment along the beam when
    the train is at x_train
    V in N
    M in Nmm
    '''

    global L
    w1, w2, w3, w4, w5, w6 = loc_of_wheels(x_train)
    Ay, By = reaction_forces(w1, w2, w3, w4, w5, w6)
    x_bridge = create_x_bridge()

    V = [0]

    for i in range(1, w1 + 1):
        V.append(Ay)

    for i in range(w1 + 1, w2 + 1):
        V.append(Ay - 400 / 6)

    for i in range(w2 + 1, w3 + 1):
        V.append(Ay - 400 / 6 * 2)

    for i in range(w3 + 1, w4 + 1):
        V.append(Ay - 400 / 6 * 3)

    for i in range(w4 + 1, w5 + 1):
        V.append(Ay - 400 / 6 * 4)

    for i in range(w5 + 1, w6 + 1):
        V.append(Ay - 400 / 6 * 5)

    for i in range(w6 + 1, L + 1):
        V.append(-By)

    V.append(0)

    M = [0]

```

```
    for i in range(1, w1 + 1):
        M.append(Ay * i)

    for i in range(w2 - w1):
        M.append(Ay * w1 + V[i + w1] * i)

    for i in range(w3 - w2):
        M.append(Ay * w1 + V[w2] * (w2 - w1) + V[i + w2] * i)

    for i in range(w4 - w3):
        M.append(Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[i + w3] * i)

    for i in range(w5 - w4):
        M.append(Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3) +\
        V[i + w4] * i)

    for i in range(w6 - w5):
        M.append(Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3) +\
        V[w5] * (w5 - w4) + V[i + w5] * i)

    for i in range(L - w6):
        M.append(Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3) +\
        V[w5] * (w5 - w4) + V[w6] * (w6 - w5) - By * i)

    M.append(0)


    return V, M







def graph_SFD_BMD(x_train):

    '''Take in the position of the train, x_train and graphs the SFD and BMD'''

    w1, w2, w3, w4, w5, w6 = loc_of_wheels(x_train)
    Ay, By = reaction_forces(w1, w2, w3, w4, w5, w6)
    x_bridge = create_x_bridge()
    x_axis = [0] * (L + 2)
```

```
    V, M = create_V_M_lists(x_train)

    plt.figure(f"SFD (x = {x_train})")
    plt.plot(x_bridge, x_axis)
    plt.title("Shear Force Diagram")
    plt.ylabel("Shear Force (N)")
    plt.xlabel("Distance Along Bridge (mm)")

    # prints the shear force at each location
    print("x = 0, V = 0")
    print(f"x = {w1}, V = {Ay}")
    print(f"x = {w2}, V = {Ay - 400 / 6}")
    print(f"x = {w2}, V = {Ay - 400 / 6 * 2}")
    print(f"x = {w3}, V = {Ay - 400 / 6 * 3}")
    print(f"x = {w4}, V = {Ay - 400 / 6 * 4}")
    print(f"x = {w5}, V = {Ay - 400 / 6 * 5}")
    print(f"x = {w6}, V = {-By}")
    print(f"x = {L}, V = 0")

    plt.plot(x_bridge, V)
    plt.show()

    plt.figure(f"BMD (x = {x_train})")
    plt.title("Bending Moment Diagram")
    plt.xlabel("Distance Along Bridge (mm)")
    plt.ylabel("Bending Moment (Nmm)")
    plt.plot(x_bridge, x_axis)

    # graphs the bending moment at each location
    print("x = 0, M = 0")
    print(f"x = {w1}, M = {Ay * w1}")
    print(f"x = {w2}, M = {Ay * w1 + V[w2] * (w2 - w1)}")
    print(f"x = {w3}, M = {Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2)}")
    print(f"x = {w4}, M = {Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3)}")
    print(f"x = {w5}, M = {Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3) + V[w5] * (w5 - w4)}")
    print(f"x = {w6}, M = {Ay * w1 + V[w2] * (w2 - w1) + V[w3] * (w3 - w2) + V[w4] * (w4 - w3) + V[w5] * (w5 - w4) + V[w6] * (w6 - w5)}")
    print(f"x = {L}, M = 0")

    plt.plot(x_bridge, M)
    plt.show()

```

```python

def create_V_M_envelopes():
    ''' Iterate through lists V, M and find the maximum V and M at
    every location on the bridge

    Return 2 dictionaries:
    all_V_at_every_loc
    and
    all_V_at_every_loc

    which store the positions on the bridge as keys and the list of
    possible V's and M's as values

    '''
    global L

    all_V_at_every_loc = dict()
    all_M_at_every_loc = dict()
    # all_V_at_every_loc[x_bridge] = list of all V's in all 240 loading positions
    for x_bridge in range(L + 2):
        all_V_at_every_loc[x_bridge] = []
        all_M_at_every_loc[x_bridge] = []

    max_V = 0
    max_M = 0
    max_V_bridge = 0
    max_M_bridge = 0
    max_V_train = 0
    max_M_train = 0

    for x_train in range(241):
        V, M = create_V_M_lists(x_train)

        for x_bridge in range(L + 2):
            # list of V and M values at location x_bridge
            # iterate through x_bridge and find the force at x_train, x_bridge
            all_V_at_every_loc[x_bridge].append(abs(V[x_bridge]))
            all_M_at_every_loc[x_bridge].append(abs(M[x_bridge]))

            if V[x_bridge] > max_V:
                max_V = V[x_bridge]
```

```
                max_V_bridge = x_bridge
                max_V_train = x_train
            if M[x_bridge] > max_M:
                max_M = M[x_bridge]
                max_M_bridge = x_bridge
                max_M_train = x_train



    # put the max V and M values at every location to a new list
    # list index = position on bridge

    V_envelope = []
    M_envelope = []
    for x_bridge in range(L + 2):
        V_envelope.append(max(all_V_at_every_loc[x_bridge]))
        M_envelope.append(max(all_M_at_every_loc[x_bridge]))


    return V_envelope, M_envelope







def graph_V_M_envelopes():
    '''Graphs the shear force and bending moment envelopes'''

    global L

    x_bridge = create_x_bridge()
    V_envelope, M_envelope = create_V_M_envelopes()


    plt.figure("Shear Force Envelope")
    plt.title("Shear Force Envelope")
    plt.ylabel("Maximum Shear Force During Loading (N)")
    plt.xlabel("Distance Along Bridge (mm)")
    x_axis = [0] * (L + 2)
    plt.plot(x_bridge, x_axis)
    plt.plot(x_bridge, V_envelope)
    plt.show()

    print(f"x = 200, V = {V_envelope[201]}")
    print(f"x = 300, V = {V_envelope[301]}")
```

```python
    print(f"x = 400, V = {V_envelope[401]}")
    print(f"x = 600, V = {V_envelope[601]}")
    print(f"x = 800, V = {V_envelope[801]}")
    print(f"x = 1000, V = {V_envelope[1001]}")

    max_V = V_envelope[0]
    max_V_loc = 0
    for loc in range(len(V_envelope)):
        if V_envelope[loc] > max_V:
            max_V = V_envelope[loc]
            max_V_loc = loc

    print(f"Max V = {max_V}, location = {max_V_loc}")



    plt.figure("Bending Moment Envelope")
    plt.title("Bending Moment Envelope")
    plt.ylabel("Maximum Bending Moment During Loading (N)")
    plt.xlabel("Distance Along Bridge (mm)")
    plt.plot(x_bridge, M_envelope)
    plt.show()

    print(f"x = 200, M = {M_envelope[201]}")
    print(f"x = 300, M = {M_envelope[301]}")
    print(f"x = 400, M = {M_envelope[401]}")
    print(f"x = 600, M = {M_envelope[601]}")
    print(f"x = 800, M = {M_envelope[801]}")
    print(f"x = 1000, M = {M_envelope[1001]}")

    max_M = M_envelope[0]
    max_M_loc = 0
    for loc in range(301):
        if M_envelope[loc] > max_M:
            max_M = M_envelope[loc]
            max_M_loc = loc

    print(f"Max M = {max_M} at 301, location = {max_M_loc}")




def centroidal(dim):
    '''Take in a nested list dim = [[w1, h1, yb1], [w2, h2, yb2], ..., [wn, hn, ybn]]
    Return the centroidal axis of the shape, yb, in mm
    Assume sections are rectangular
```

```

    '''
    # yb = ∑(wi * hi * ybi)/∑(wi * hi)
    denominator = 0
    numerator = 0
    for i in range(len(dim)):
        denominator += (dim[i][0] * dim[i][1] * dim[i][2])
        numerator += (dim[i][0] * dim[i][1])
    return denominator / numerator



def I_calculator(dim, yb):
    ''' Take in a nested list dim = [[w1, h1, yb1], [w2, h2, yb2], ..., [wn, hn, ybn]]
    and y-bar, yb
    Return the second moment of area, I, in mm^4
    Assume sections are rectangular

    '''
    # I = ∑(Ioi + d^2A)
    I = 0
    for i in range(len(dim)):
        I += ((dim[i][0] * dim[i][1] ** 3) / 12 + ((dim[i][2] - yb) ** 2) * dim[i][0] * dim[i][1])

    return I






# 8 FAILURE CHECKS

def tens_comp_demand(M, y_top, yb, I):
    # check compression on top of beam
    comp = M * (y_top - yb) / I
    tens = M * yb / I

    return comp, tens

def shear_demand(V, Q, I, b):
    # check shear
    return V* Q / I / b


def thin_plate_mid(b, t):
```

```
    # mid flange
    sigma_crit = 4*(math.pi**2)*(4000*10**6)/12/(1-0.2**2)*(t / b)**2

    return sigma_crit

def thin_plate_side(b, t):
    # side flange
    sigma_crit = 0.425 * (math.pi**2)*(4000*10**6)/12/(1-0.2**2)*(t / b)**2
    return sigma_crit

def thin_plate_web(b, t):
    # web members
    sigma_crit = 6 * (math.pi**2)*(4000*10**6)/12/(1-0.2**2)*(t / b)**2

    return sigma_crit




def shear_thin_plate(h, w, a):
    # shear thin plate buckling
    sigma_crit = 5 * (math.pi**2)*(4000*10**6)/12/(1-0.2**2)*((w / h)**2 + (w / a)**2)

    return sigma_crit



# DEMAND

def create_tens_envelope(y_tot, yb, I):
    '''Take in the total height of the cross-section, y_tot,
    and the height of the centroidal axis, yb

    Return a list that has the maximum tensile force at each
    bridge location

    '''
    y_tot *= 10 ** -3
    yb *= 10 ** -3
    I *= 10 **-12
    M_envelope = create_V_M_envelopes()[1]
    tens_demand = []
    for M in M_envelope:
```

```
        tens_demand.append(tens_comp_demand(M*10**-3, y_tot, yb, I)[1]\
        *10**-6)
    return tens_demand

def create_comp_envelope(y_tot, yb, I):
    '''Take in the total height of the cross-section,
    y_tot, the height of the centroidal axis, yb, and
    the second moment of area, I in the zone of interest

    Return a list that has the maximum compressive force
    at each bridge location

    '''
    y_tot *= 10 ** -3
    yb *= 10 ** -3
    I *= 10 **-12
    M_envelope = create_V_M_envelopes()[1]
    comp_demand = []
    for M in M_envelope:
        comp_demand.append(tens_comp_demand(M*10**-3, y_tot,\
        yb, I)[0]*10**-6)
    return comp_demand

def create_shear_envelope(Q, I, b):
    ''' Take in Q, I and b

    Return a list of the maximum shear force at each location
    of the bridge

    '''


    Q *= 10**-9
    I *= 10 ** -12
    b *= 10 ** -3
    x_bridge = create_x_bridge()
    V_envelope = create_V_M_envelopes()[0]
    shear_stress_demand = []
    for V in V_envelope:
        shear_stress_demand.append(shear_demand(V, Q, I, b)\
        *10**-6)
    return shear_stress_demand
```

```python

def graph_demand_capacity(title, xlabel, ylabel, demand, capacity, start, end):
    ''' Creates a capacity-demand graph'''

    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    x_bridge = create_x_bridge()[start: end +1]
    plt.plot(x_bridge, capacity[start: end + 1])
    plt.plot(x_bridge, demand[start: end + 1])
    plt.show()


# FINAL DESIGN


def x_sectional(zone):
    global yb, I
    yb = centroidal(zone)
  #   print(f"yb = {round(yb, 3)}")
    I = I_calculator(zone, yb)
    # print(f"I = {round(I, 3)}")
    return yb, I


# demand
def calculate_demand(y_top, yb, I, Q_glue, Q_cent, start, end):
    '''Calculates all the stresses demanded'''

    global comp_stress_envelope, tens_stress_envelope, glue_shear_stress_envelope,\
    matboard_shear_stress_envelope, max_comp_stress, max_tens_stress,\
    matboard_max_shear_stress, glue_max_shear_stress

    comp_stress_envelope = create_comp_envelope(y_top, yb, I)
    tens_stress_envelope = create_tens_envelope(y_top, yb, I)
    glue_shear_stress_envelope = create_shear_envelope(Q_glue, I, 12.54)
    matboard_shear_stress_envelope = create_shear_envelope(Q_cent, I, 2.54)

    max_comp_stress = max(comp_stress_envelope[start: end + 1])
    max_tens_stress = max(tens_stress_envelope[start: end +
```

```python
1])
    matboard_max_shear_stress = max(matboard_shear_stress_envelope[start: end + 1])
    glue_max_shear_stress = max(glue_shear_stress_envelope[start: end + 1])


# capacity
def calculate_capacity(b_mid_flange, t_mid_flange, b_side_flange, t_side_flange, b_web, t_web, h_diaphragm, a_diaphragm):
    '''Calculate all the capacities demanded'''

    global comp_capacity, tens_capacity, shear_glue_capacity, \
    shear_matboard_capacity, mid_flange_capacity, side_flange_capacity,\
    web_capacity, shear_thin_plate_capacity

    comp_capacity = [6] * 1201

    tens_capacity = [30] * 1201

    shear_glue_capacity = [2] * 1201

    shear_matboard_capacity = [4] * 1201

    mid_flange_capacity = [thin_plate_mid(b_mid_flange, t_mid_flange)\
    *10**-6] * 1201

    side_flange_capacity = [thin_plate_side(b_side_flange, t_side_flange)\
    *10**-6] * 1201

    web_capacity = [thin_plate_web(b_web, t_web)*10**-6] * 1201

    shear_thin_plate_capacity = [shear_thin_plate(h_diaphragm, 1.27, \
    a_diaphragm)*10**-6] * 1201



def graph_everything(start, end, zone):
    ''' Graphs all demands and capacities'''

```

```
    graph_demand_capacity("Compressive Stress vs Capacity", "Distance Along Bridge (mm)",\
     "Compressive Stress and Capacity", comp_stress_envelope, comp_capacity, start, end)
    print("compressive stress capacity:", comp_capacity[0])
    print("maximum compressive stress:", max_comp_stress)
    print(f"minimum FOS against compression failure: {6 / max_comp_stress}")

    graph_demand_capacity("Tensile Stress vs Capacity", "Distance Along Bridge (mm)",\
    "Tensile Stress and Capacity", tens_stress_envelope, tens_capacity, start, end)
    print("\ntensile stress capacity:", tens_capacity[0])
    print("maximum tensile stress:", max_tens_stress)
    print(f"minimum FOS against tension failure: {30 / max_tens_stress}")

    graph_demand_capacity("Glue Shear Demand vs Capacity", "Distance Along Bridge (mm)",\
    "Shear Stress and Capacity", glue_shear_stress_envelope, shear_glue_capacity, start, end)
    print("\nglue shear stress capacity:", shear_glue_capacity[0])
    print("maximum shear stress at glue location:", glue_max_shear_stress)
    print(f"minimum FOS against glue shear failure: {2 / glue_max_shear_stress}")

    graph_demand_capacity("Matboard Shear Demand vs Capacity", "Distance Along Bridge (mm)",\
    "Shear Stress and Capacity", matboard_shear_stress_envelope, shear_matboard_capacity, start, end)
    print("\nmatboard shear stress capacity: 4")
    print("maximum shear stress in matboard:", matboard_max_shear_stress)
    print(f"minimum FOS against matboard shear failure: {4 / matboard_max_shear_stress}")

    graph_demand_capacity("Mid-Flange Buckling Stress vs Capacity", "Distance Along Bridge(mm)",\
    "Buckling Stress and Capacity", comp_stress_envelope, mid_flange_capacity, start, end)
    print("\nmid-flange buckling capacity:", mid_flange_capacity[0])
    print("maximum shear stress in the mid-flange:",
```

```
matboard_max_shear_stress)
    print(f"minimum FOS against mid flange buckling: {mid_flange_capacity[0] / max_comp_stress}")

    graph_demand_capacity("Side-Flange Buckling Stress vs Capacity", "Distance Along Bridge (mm)",\
    "Buckling Stress and Capacity", comp_stress_envelope, side_flange_capacity, start, end)
    print("\nside-flange buckling capacity:", side_flange_capacity[0])
    print("maximum shear stress in the mid-flange:", matboard_max_shear_stress)
    print(f"minimum FOS against side flange buckling: {side_flange_capacity[0] / max_comp_stress}")

    graph_demand_capacity("Web Buckling Stress vs Capacity", "Distance Along Bridge (mm)",\
    "Buckling Stress and Capacity", comp_stress_envelope, web_capacity, start, end)
    print("\nweb buckling capacity:", web_capacity[0])
    print("maximum shear stress in the mid-flange:", matboard_max_shear_stress)
    print(f"minimum FOS against web buckling: {web_capacity[0] / max_comp_stress}")

    graph_demand_capacity("Shear Buckling Demand vs Capacity", "Distance Along Bridge (mm)",\
    "Shear Buckling Stress and Capacity", matboard_shear_stress_envelope, shear_thin_plate_capacity, start, end)
    print("\nshear buckling capacity:", shear_thin_plate_capacity[0])
    print("maximum shear stress:", matboard_max_shear_stress)
    print(f"minimum FOS against shear buckling: {shear_thin_plate_capacity[0] / matboard_max_shear_stress}")

# design 0
design0 = [100, 1.27, 75.635], [2.54, 75-1.27, 38.135], [10, 1.27, 74.365],\
[80, 1.27, 0.635]

# design iterations
design1 = [[100, 2.54, 76.905], [2.54, 75, 37.5], [10, 1.27, 73.73], [71.46,\
1.27, 0.635]]

design2 = [[200, 2.54, 76.27], [2.54, 75, 37.5], [20, 1.27,
```

```
74.365], [2.54, 1.27, 38.135],\
[77.46, 1.27, 0.635]]

design3 = [[100, 2.54, 82.54], [2.54, 90, 40.635], [55-2*1.27, 1.27, 1.27/2+80],\
[1.27, 80-1.27, (80+1.27)/2], [55, 1.27, 0.635]]

# final design
A = [[100, 2.54, 82.54], [75, 1.27, 80.635], [20, 1.27, 79.365],\
[2.54, 80, 40], [72.46, 1.27, 0.635]]

B = [[100, 2.54, 82.54], [2*11.27, 2.54, 80], [2*1.27, 78.73, 39.365],\
[72.46, 1.27, 0.635]]


design = design0
zone = A

if design == design0:
    Q_glue = 4344.035
    Q_cent = 6193.079
    x_sectional(design)
    calculate_demand(76.27, yb, I, Q_glue, Q_cent, 0, 1200)
    calculate_capacity(78.73, 1.27, 10.635, 1.27, 75 - 1.27/2 - yb, 1.27, 73.73, 400 )
    graph_everything(0, 1200, design0)

elif design == design1:
    Q_glue = 8402.955
    Q_cent = 9154.180828
    x_sectional(design1)
    calculate_demand(75+1.27*3, yb, I, Q_glue, Q_cent, 400, 800)
    calculate_capacity(80-1.27, 3.81, 10.635, 3.81, 75+3*1.27 - yb, 1.27, 75-1.27, 400)
    graph_everything(0, 1200, design1)

elif design == design2:
    x_sectional(design2)
    Q_glue = 4614.672
    Q_cent = 14227.9
    x_sectional(design2)
    calculate_demand(75+1.27*2, yb, I, Q_glue, Q_cent, 0, 1200)
    calculate_capacity(80-1.27*2, 2.54, 11.27, 2.54, 76.27 -
```

```
yb, 2.54, 75-1.27, 400)
    graph_everything(0, 1200, design2)

elif design == design3:
    x_sectional(design3)
    Q_glue = 12561
    Q_cent = 9632.232254
    x_sectional(design3)
    calculate_demand(83.81, yb, I, Q_glue, Q_cent, 0, 1200)
    calculate_capacity(55-1.27, 3.81, 23.135, 2.54, 81.27 - yb, 1.27, 80-1.27/2, 400)
    graph_everything(0, 1200, design3)

elif design == final:
    if zone == A:
        Q_glue = 6883.111
        Q_cent = 9549.54895
        x_sectional(A)
        calculate_demand(83.81, yb, I, Q_glue, Q_cent, 400, 800)
        calculate_capacity(73.73, 3.81, 13.77, 2.54, 80.635 - yb,1.27, 78.73, 360)
        graph_everything(400, 800, zone)
    else:
        Q_glue = 7590.497
        Q_cent = 8959.0227
        x_sectional(B)
        calculate_demand(83.81, yb, I, Q_glue, Q_cent, 0, 400)
        calculate_capacity(73.73, 2.54, 13.77, 2.54, 80.635 - yb, 1.27, 78.73, 250)
        graph_everything(0, 400, zone)


# GLUE FAILURE AT SPLICING LOCATIONS
sigma = create_shear_envelope(8959.0227*10**-9, 652622.7973685571*10**-12, 2.54*10**-3)[127]
FOS = 2/sigma
print(FOS) # 1.7211822386750777
```