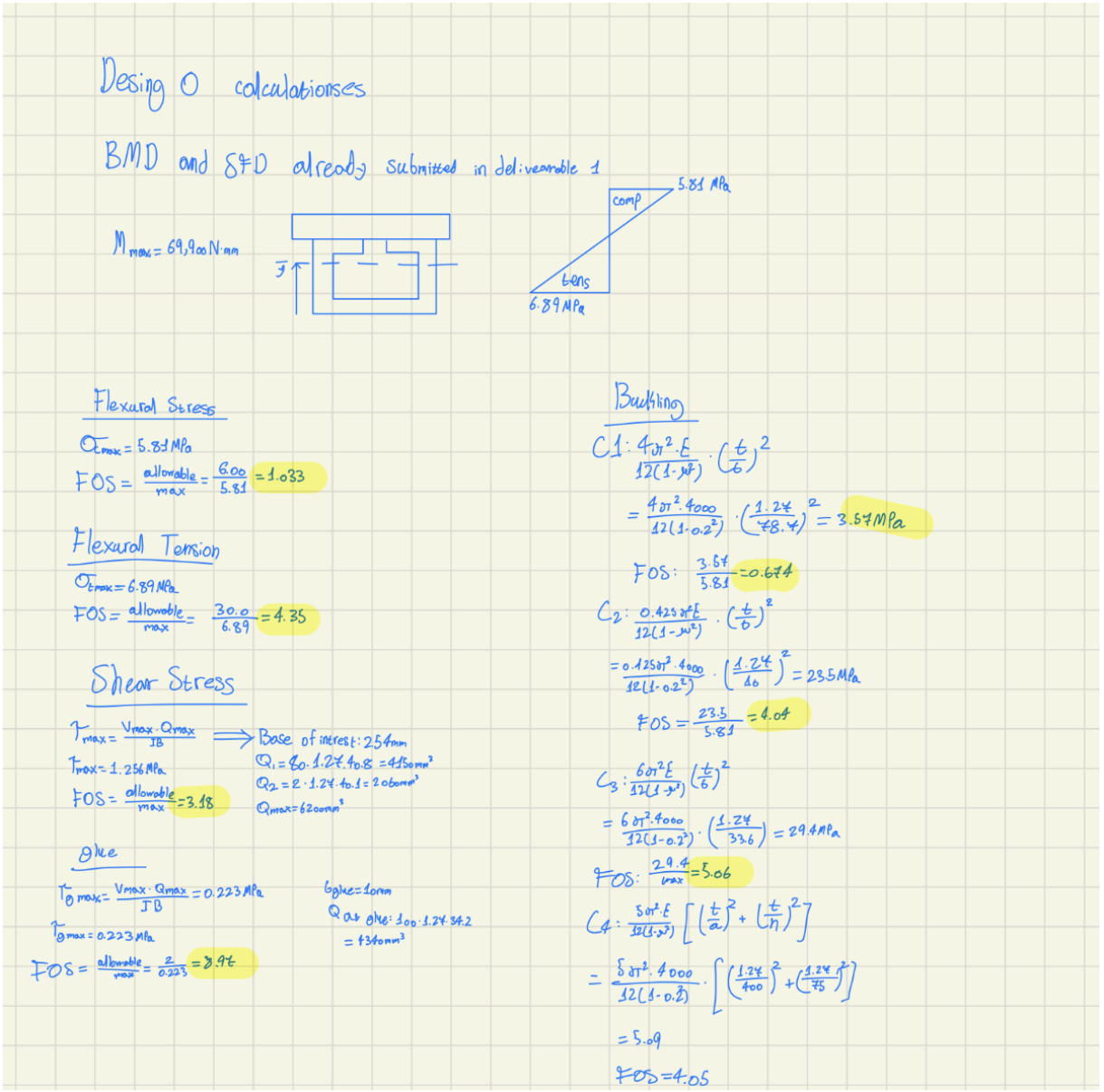
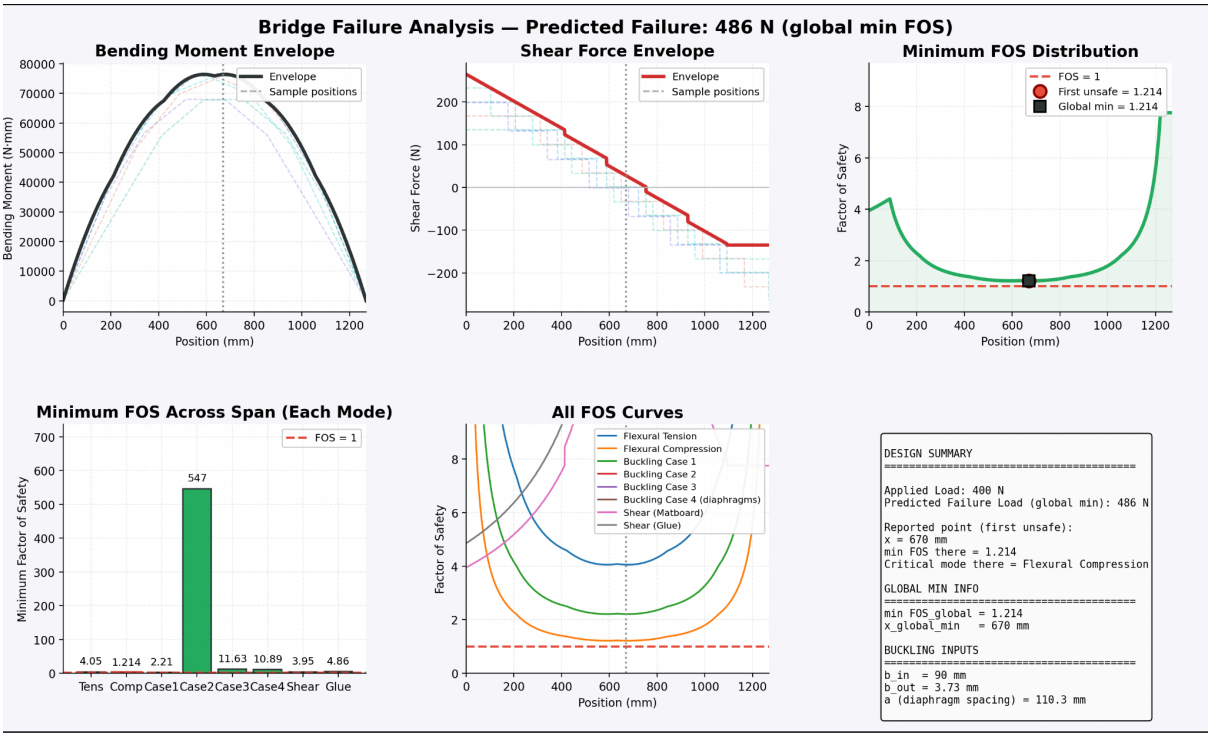
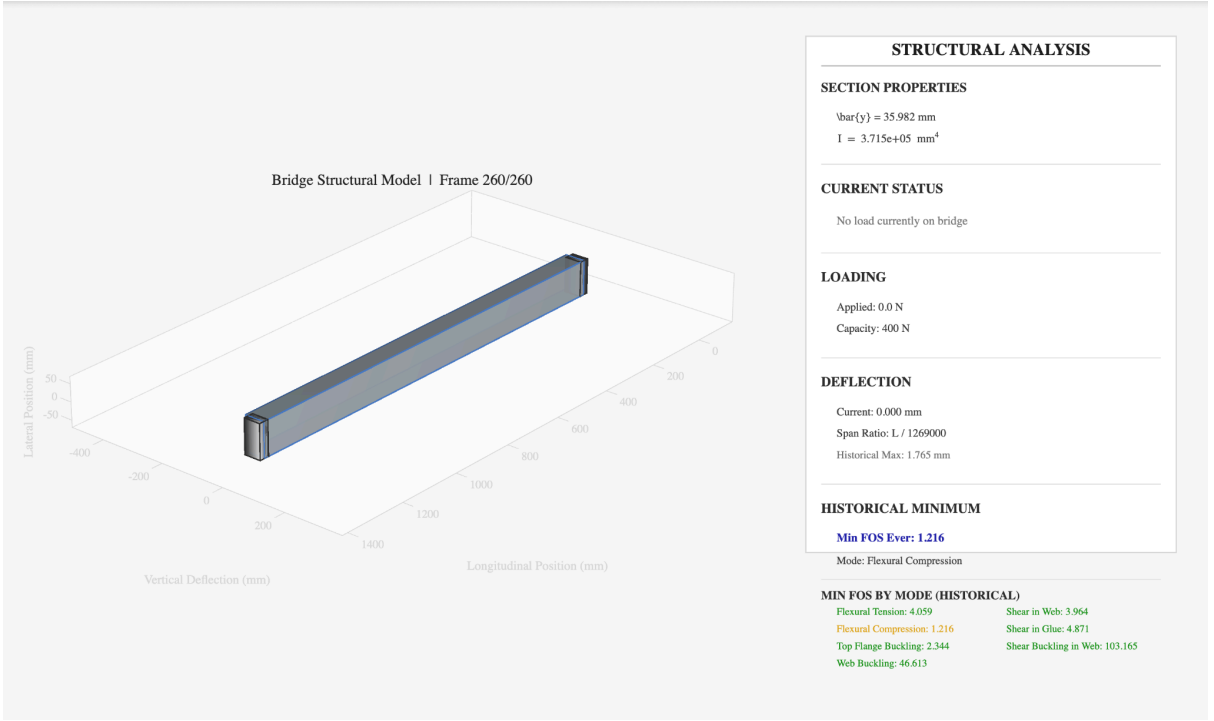


- **Go to next page**
- **Mindful of rounding errors (matlab and python has a lot of those, we tried including a slide rule precision function but it didn't do much)**

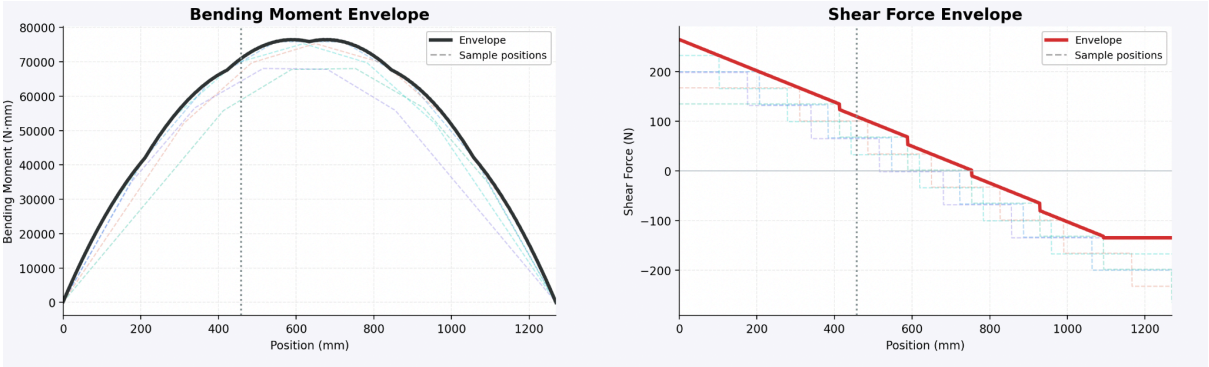
Outputs for codes:

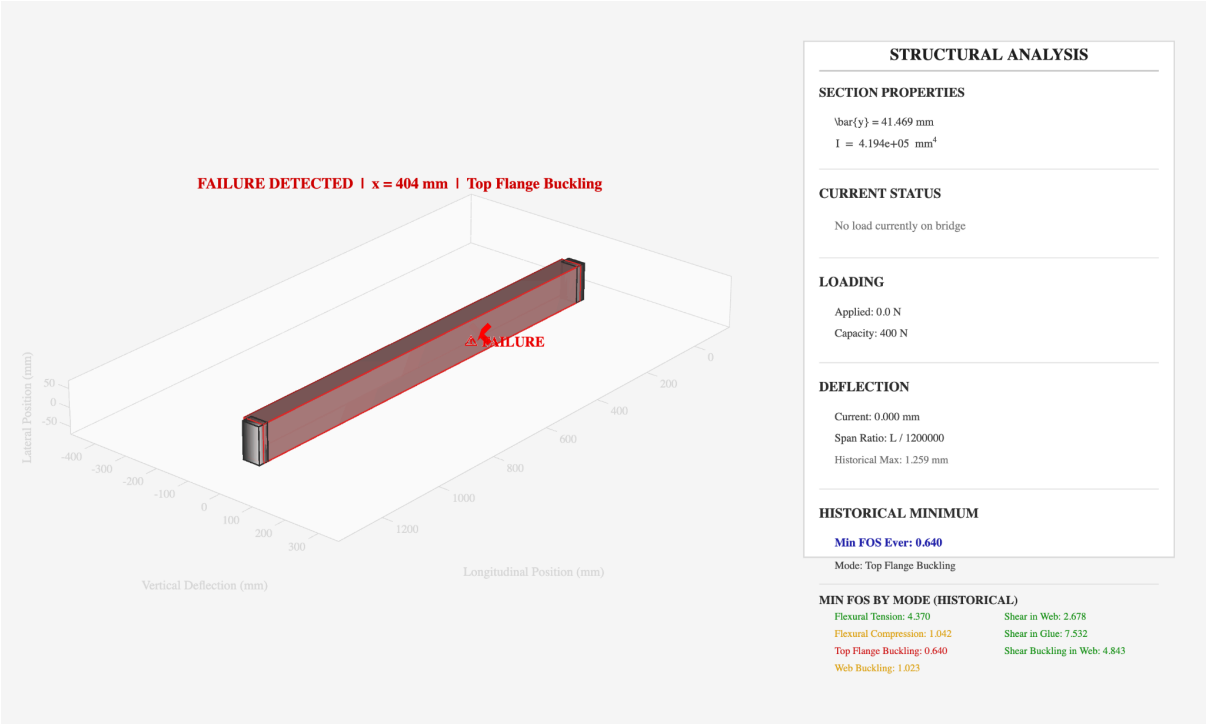
- **Analysis for our final iteration**





Design 0 Output:





Go to the next page for matlab code (most of the analysis + visualisation), you should run this, it's cool!

Go to page 19 for matlab design 0 cross sectionfile
Go to page 22 for matlab our cross section file (run this as well)
Go to page 26 for python main file
Go to page 37 for python design 0 file
Go to page 40 for python our cross section file

```
% loadcase1.m
% includes anaylsis similar to the one in python (but not as deep)
% includes a 3D visual of the bridge as it deflects using MAT2
% some boundaries were set to avoid drifting
% note: my way of doing the train load probably wasn't the smartest but it
% works, much better methods can be employed
% this might be quite heavy to run, if you're experiencing issues go to the
% variable nsteps and change it to 180/120 or whatever you need
clear; clc; close all;
% defining section, a new file was made for each different cross-section
% to make it easy to switch from one cross section to another
SEC_FUNC = @section;
sec = SEC_FUNC();
L       = sec.L;
E       = sec.E;
I       = sec.I;
ybar    = sec.ybar;
y_top   = sec.y_top;
y_bot   = sec.y_bot;
Q_web   = sec.Q_web;
Q_glue  = sec.Q_glue;
t_web   = sec.t_web;
t_glue  = sec.t_glue;
caps    = sec.cap;
fprintf('\n========== BRIDGE SIMULATION STARTING ==========\n');
fprintf('Span: %.0f mm | E: %.0f MPa | I: %.1f mm^4\n', L, E, I);
fprintf('Total train load: 400 N (6 axles × %.1f N)\n\n', 400/6);
n  = 600;
x  = linspace(0, L, n+1);
%% Train Load for Loadcase1
total_load = 400;
num_axles  = 6;
P_axle     = total_load / num_axles;
A = sec.spacing_A; % these are the two different spacings between the
wheels
B = sec.spacing_B;
offsets = [0, A, A+B, A+B+A, A+B+A+B, A+B+A+B+A];
train_length = offsets(end);
%% setting up visulization of the beam
W_bridge   = 100;
H_bridge   = 60;
zmin = -W_bridge/2;
zmax =  W_bridge/2;
y_top_face0 = 0;
% Animation setup
nSteps   = 260;
front_start = -train_length;
front_end   = L + 60;
front_pos = linspace(front_start, front_end, nSteps);
hFig = figure('Name','Bridge Structural Analysis','NumberTitle','off');
clf(hFig);
```

```
set(hFig, 'Color', [0.97 0.97 0.97], 'Units','normalized', 'Position',
[0.03 0.08 0.94 0.84]);
ax = subplot('Position', [0.05 0.08 0.58 0.88]);
hold(ax,'on'); grid(ax,'on'); box(ax,'on');
view(ax, 130, 25);
camproj(ax,'perspective');
xlabel(ax, 'Longitudinal Position (mm)', 'FontSize', 12, 'FontWeight',
'normal', 'FontName', 'Segoe UI');
ylabel(ax, 'Vertical Deflection (mm)', 'FontSize', 12, 'FontWeight',
'normal', 'FontName', 'Segoe UI');
zlabel(ax, 'Lateral Position (mm)', 'FontSize', 12, 'FontWeight', 'normal',
'FontName', 'Segoe UI');
set(ax, 'FontName', 'Segoe UI', 'FontSize', 10.5, 'Color', [0.99 0.99
0.99]);
set(ax, 'GridAlpha', 0.12, 'GridLineStyle', '-', 'LineWidth', 0.6);
% Side panel for anayalsis
ax_text = subplot('Position', [0.66 0.08 0.32 0.88]);
axis(ax_text,'off');
failed      = false;
fail_x      = NaN;
fail_mode   = '';
FOS_at_fail = NaN;
P_on_fail   = NaN;
eps_small   = 1e-10;
min_FOS_history = inf;
critical_mode_history = '';
max_deflection_history = 0;
pause_done = false;   % so we only pause once at first failure
modes = { ...
    'Flexural Tension', ...
    'Flexural Compression', ...
    'Top Flange Buckling', ...
    'Web Buckling', ...
    'Shear in Web', ...
    'Shear in Glue', ...
    'Shear Buckling in Web'};
nModes = numel(modes);
min_FOS_mode_history = inf(nModes,1);
% panel layout constants (kept tight so everything fits)
panel_left   = 0.06;
panel_right  = 0.94;
panel_top    = 0.955;
panel_bot    = 0.045;
for k = 1:nSteps
    front_x = front_pos(k);
    axle_x  = front_x + offsets;
    on_span = (axle_x >= 0) & (axle_x <= L);
    load_x  = axle_x(on_span);
    load_P  = P_axle * ones(1, numel(load_x));
    P_on    = sum(load_P);
    if isempty(load_x)
        V  = zeros(size(x));
        M  = zeros(size(x));
```

```
        RA = 0; RB = 0;
    else
        [V, M, RA, RB] = compute_SFD_BMD_local(x, L, load_x, load_P);
    end
    % defaults so code never breaks when no load
    FOS_global    = inf;
    critical_mode = 'None';
    fail_location = NaN;
    % ================================================================
    % FOS CALCULATION (RUNS EVEN AFTER FAILURE)
    % ================================================================
    if P_on > 0
        stress_top = -M .* y_top ./ I;
        stress_bot =  M .* y_bot ./ I;
        stress_bot_tens = max(stress_bot, 0);   % tension only at bottom
        stress_top_comp = max(-stress_top, 0);  % compression only at top
        tau_web  = abs(V) .* Q_web  ./ (I * t_web);
        tau_glue = abs(V) .* Q_glue ./ (I * t_glue);
        FOS_tension     = caps.tension          ./ (stress_bot_tens +
eps_small);
        FOS_compression = caps.compression      ./ (stress_top_comp +
eps_small);
        FOS_flange_buck = caps.flange_buckling ./ (stress_top_comp +
eps_small);
        FOS_web_buck    = caps.web_buckling     ./ (stress_top_comp +
eps_small);
        FOS_shear       = caps.shear_board      ./ (tau_web  + eps_small);
        FOS_glue        = caps.shear_glue       ./ (tau_glue + eps_small);
        FOS_shear_buck  = caps.shear_buckling  ./ (tau_web  + eps_small);
        FOS_frame_mins = [ ...
            min(FOS_tension); ...
            min(FOS_compression); ...
            min(FOS_flange_buck); ...
            min(FOS_web_buck); ...
            min(FOS_shear); ...
            min(FOS_glue); ...
            min(FOS_shear_buck)];
        % keep updating historical mins no matter what
        min_FOS_mode_history = min(min_FOS_mode_history, FOS_frame_mins);
        FOS_stack = cat(1, FOS_tension, FOS_compression, FOS_flange_buck,
...
                           FOS_web_buck, FOS_shear, FOS_glue,
FOS_shear_buck);
        [FOS_min, mode_idx] = min(FOS_stack, [], 1);
        [FOS_global, iFail] = min(FOS_min);
        fail_location = x(iFail);
        critical_mode = modes{mode_idx(iFail)};
        if FOS_global < min_FOS_history
            min_FOS_history = FOS_global;
            critical_mode_history = critical_mode;
        end
        % only *set* failure once, but keep sim running after
        if ~failed && FOS_global <= 1.0
```

```
            failed      = true;
            fail_x      = x(iFail);
            fail_mode   = critical_mode;
            FOS_at_fail = FOS_global;
            P_on_fail   = P_on;
            fprintf('\n!!! BRIDGE FAILURE DETECTED !!!\n');
            fprintf('Frame: %d/%d\n', k, nSteps);
            fprintf('Location: x = %.1f mm\n', fail_x);
            fprintf('Failure mode: %s\n', fail_mode);
            fprintf('Load on span: %.1f N (%.1f%% of total)\n', ...
                    P_on_fail, 100*P_on_fail/total_load);
            fprintf('FOS at failure: %.4f\n\n', FOS_at_fail);
        end
    end
    % Deflection calculation (MAT2)
    w = zeros(size(x));
    if P_on > 0 && max(abs(M)) > 1e-10
        kappa = M ./ (E * I);
        p_fit = polyfit(x, kappa, 1);
        kappa = kappa - polyval(p_fit, x);
        theta = cumtrapz(x, kappa);
        theta = theta - (theta(end) / L) * x;
        w = cumtrapz(x, theta);
        w = w - w(1) - ((w(end) - w(1)) / L) * x;
        w(1) = 0;
        w(end) = 0;
        if max(abs(w)) > L/50
            w = w * (L/50) / max(abs(w));
        end
    end
    max_deflection = max(abs(w));
    if max_deflection > max_deflection_history
        max_deflection_history = max_deflection;
    end
    if max_deflection > 1e-6
        if max_deflection > L/100
            scale_factor = 0.15 * L / max_deflection;
        else
            scale_factor = 0.25 * L / max_deflection;
        end
    else
        scale_factor = 1;
    end
    w_smooth = w * scale_factor;
    % 3D Animation
    cla(ax);
    hold(ax,'on'); grid(ax,'on');
    sup_h = 50;
    draw_support_block(ax, 0, sup_h, W_bridge);
    draw_support_block(ax, L, sup_h, W_bridge);
    if ~failed
        draw_box_beam_smooth(ax, x, w_smooth, y_top_face0, H_bridge, zmin,
zmax, ...
```

```
                            [0.90 0.95 0.99], [0.28 0.48 0.78]);
    else
        draw_box_beam_smooth(ax, x, w_smooth, y_top_face0, H_bridge, zmin,
zmax, ...
                            [1.0 0.72 0.72], [0.87 0.17 0.17]);
        if fail_x >= min(x) && fail_x <= max(x)
            y_fail = interp1(x, w_smooth, fail_x, 'linear', 0) +
y_top_face0;
            plot3(ax, fail_x, y_fail, 0, 'rx', 'MarkerSize', 24,
'LineWidth', 5);
            text(fail_x, y_fail + 75, 0, '⚠ FAILURE', ...
                'FontSize', 13, 'FontWeight', 'bold', 'Color', [0.92 0 0],
...
                'HorizontalAlignment', 'center', 'Parent', ax, 'FontName',
'Segoe UI');
        end
    end
    if ~isempty(load_x)
        for j = 1:numel(load_x)
            px = load_x(j);
            if px >= min(x) && px <= max(x)
                y_top_here = interp1(x, w_smooth, px, 'linear', 0) +
y_top_face0;
            else
                y_top_here = y_top_face0;
            end
            arrow_len = 60;
            quiver3(ax, px, y_top_here + H_bridge + 15, 0, 0, -arrow_len, 0,
...
                    'Color', [0.94 0.14 0.14], 'LineWidth', 3.4,
'MaxHeadSize', 1.6);
        end
    end
    xlim(ax, [-150, L+150]);
    ylim(ax, [-0.4*L, 0.3*L]);
    zlim(ax, [-W_bridge*0.7, W_bridge*0.7]);
    view(ax, 130, 25);
    axis(ax,'vis3d');
    daspect(ax,[1 1 1]);
    camlight(ax,'headlight');
    lighting(ax,'gouraud');
    if ~failed
        title(ax, sprintf('Bridge Structural Model  |  Frame %d/%d', k,
nSteps), ...
              'FontWeight', 'normal', 'FontSize', 14, 'FontName', 'Segoe
UI', 'Color', [0.15 0.15 0.15]);
    else
        title(ax, sprintf('FAILURE DETECTED  |  x = %.0f mm  |  %s', fail_x,
fail_mode), ...
              'FontWeight', 'bold', 'Color', [0.82 0 0], 'FontSize', 14,
'FontName', 'Segoe UI');
    end
    % updating analysis as position changes
```

```
    cla(ax_text);
    axis(ax_text,'off');
    rectangle('Position', [0.02 0.02 0.96 0.96], 'FaceColor', 'white', ...
              'EdgeColor', [0.85 0.85 0.85], 'LineWidth', 1.2, 'Parent',
ax_text);
    text(0.50, panel_top, 'STRUCTURAL ANALYSIS', ...
         'FontSize', 15, 'FontWeight', 'bold', 'Parent', ax_text,
'FontName', 'Segoe UI', ...
         'HorizontalAlignment', 'center', 'Color', [0.12 0.12 0.12]);
    line([panel_left panel_right], [panel_top-0.03 panel_top-0.03], ...
         'Color', [0.75 0.75 0.75], 'LineWidth', 1.2, 'Parent', ax_text);
    ypos = panel_top - 0.07;
    % ================================================================
    % ADDED: SECTION PROPERTIES BLOCK (constant)
    % ================================================================
    text(panel_left, ypos, 'SECTION PROPERTIES', ...
         'FontSize', 12.0, 'FontWeight', 'bold', ...
         'Parent', ax_text, 'FontName', 'Segoe UI', ...
         'Color', [0.18 0.18 0.18]);
    ypos = ypos - 0.055;
    text(panel_left+0.04, ypos, sprintf('\\bar{y} = %.3f mm', ybar), ...
         'FontSize', 10.5, 'Parent', ax_text, 'FontName', 'Segoe UI', ...
         'Color', [0.25 0.25 0.25]);
    ypos = ypos - 0.038;
    text(panel_left+0.04, ypos, sprintf('I = %.3e mm^4', I), ...
         'FontSize', 10.5, 'Parent', ax_text, 'FontName', 'Segoe UI', ...
         'Color', [0.25 0.25 0.25]);
    ypos = ypos - 0.050;
    line([panel_left panel_right], [ypos ypos], 'Color', [0.88 0.88 0.88],
...
         'LineWidth', 0.8, 'Parent', ax_text);
    ypos = ypos - 0.045;
    % ================================================================
    text(panel_left, ypos, 'CURRENT STATUS', 'FontSize', 12.5, 'FontWeight',
'bold', ...
         'Parent', ax_text, 'FontName', 'Segoe UI', 'Color', [0.18 0.18
0.18]);
    ypos = ypos - 0.06;
    if P_on > 0
        if FOS_global >= 1.5
            fos_color = [0.10 0.58 0.10];
            status_bg = [0.90 0.97 0.90];
        elseif FOS_global >= 1.0
            fos_color = [0.88 0.65 0.10];
            status_bg = [1.0 0.97 0.88];
        else
            fos_color = [0.82 0.08 0.08];
            status_bg = [1.0 0.92 0.92];
        end
        rectangle('Position', [panel_left ypos-0.012 0.86 0.06],
'FaceColor', status_bg, ...
                  'EdgeColor', 'none', 'Parent', ax_text);
```

```
        text(panel_left+0.04, ypos+0.018, sprintf('Min FOS: %.3f',
FOS_global), ...
             'FontSize', 14, 'FontWeight', 'bold', 'Parent', ax_text, ...
             'FontName', 'Segoe UI', 'Color', fos_color);
        ypos = ypos - 0.055;
        text(panel_left+0.04, ypos, sprintf('Critical Mode: %s',
critical_mode), ...
             'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
        ypos = ypos - 0.042;
        text(panel_left+0.04, ypos, sprintf('Location: x = %.1f mm',
fail_location), ...
             'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
    else
        text(panel_left+0.04, ypos, 'No load currently on bridge', ...
             'FontSize', 10.8, 'Color', [0.48 0.48 0.48], 'Parent', ax_text,
'FontName', 'Segoe UI');
    end
    ypos = ypos - 0.06;
    line([panel_left panel_right], [ypos ypos], 'Color', [0.88 0.88 0.88],
...
         'LineWidth', 0.8, 'Parent', ax_text);
    ypos = ypos - 0.045;
    text(panel_left, ypos, 'LOADING', 'FontSize', 12.5, 'FontWeight',
'bold', ...
         'Parent', ax_text, 'FontName', 'Segoe UI', 'Color', [0.18 0.18
0.18]);
    ypos = ypos - 0.055;
    text(panel_left+0.04, ypos, sprintf('Applied: %.1f N', P_on), ...
         'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
    ypos = ypos - 0.04;
    text(panel_left+0.04, ypos, sprintf('Capacity: %.0f N', total_load), ...
         'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
    ypos = ypos - 0.055;
    line([panel_left panel_right], [ypos ypos], 'Color', [0.88 0.88 0.88],
...
         'LineWidth', 0.8, 'Parent', ax_text);
    ypos = ypos - 0.045;
    text(panel_left, ypos, 'DEFLECTION', 'FontSize', 12.5, 'FontWeight',
'bold', ...
         'Parent', ax_text, 'FontName', 'Segoe UI', 'Color', [0.18 0.18
0.18]);
    ypos = ypos - 0.055;
    text(panel_left+0.04, ypos, sprintf('Current: %.3f mm', max_deflection),
...
         'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
    ypos = ypos - 0.04;
    text(panel_left+0.04, ypos, sprintf('Span Ratio: L / %.0f',
L/max(max_deflection, 0.001)), ...
```

```
        'FontSize', 10.2, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
   ypos = ypos - 0.04;
   text(panel_left+0.04, ypos, sprintf('Historical Max: %.3f mm',
max_deflection_history), ...
        'FontSize', 9.8, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.45 0.45 0.45]);
   ypos = ypos - 0.055;
   line([panel_left panel_right], [ypos ypos], 'Color', [0.88 0.88 0.88],
...
        'LineWidth', 0.8, 'Parent', ax_text);
   ypos = ypos - 0.045;
   text(panel_left, ypos, 'HISTORICAL MINIMUM', 'FontSize', 12.5,
'FontWeight', 'bold', ...
        'Parent', ax_text, 'FontName', 'Segoe UI', 'Color', [0.18 0.18
0.18]);
   ypos = ypos - 0.055;
   if min_FOS_history < inf
       text(panel_left+0.04, ypos, sprintf('Min FOS Ever: %.3f',
min_FOS_history), ...
            'FontSize', 10.8, 'FontWeight', 'bold', 'Parent', ax_text, ...
            'FontName', 'Segoe UI', 'Color', [0.15 0.15 0.68]);
       ypos = ypos - 0.042;
       text(panel_left+0.04, ypos, sprintf('Mode: %s',
critical_mode_history), ...
            'FontSize', 9.8, 'Parent', ax_text, 'FontName', 'Segoe UI',
'Color', [0.25 0.25 0.25]);
   else
       text(panel_left+0.04, ypos, 'Awaiting data...', ...
            'FontSize', 10.2, 'Color', [0.50 0.50 0.50], 'Parent', ax_text,
'FontName', 'Segoe UI');
   end
   % ======================
   % MIN FOS BY MODE section  (2 columns)
   % ======================
   ypos = ypos - 0.035;
   line([panel_left panel_right], [ypos ypos], 'Color', [0.88 0.88 0.88],
...
        'LineWidth', 0.8, 'Parent', ax_text);
   ypos = ypos - 0.030;
   text(panel_left, ypos, 'MIN FOS BY MODE (HISTORICAL)', ...
        'FontSize', 11.0, 'FontWeight', 'bold', ...
        'Parent', ax_text, 'FontName', 'Segoe UI', ...
        'Color', [0.18 0.18 0.18]);
   ypos = ypos - 0.030;
   % two-column layout
   nLeft  = ceil(nModes/2);      % 4
   nRight = nModes - nLeft;      % 3
   x_left  = panel_left + 0.04;
   x_right = panel_left + 0.48; % shift right column over
   line_step = 0.032;            % vertical spacing per row
   for ii = 1:nModes
       val = min_FOS_mode_history(ii);
```

```
        if isfinite(val)
            txt = sprintf('%s: %.3f', modes{ii}, val);
        else
            txt = sprintf('%s: --', modes{ii});
        end
        if val >= 1.5
            c = [0.10 0.58 0.10];
        elseif val >= 1.0
            c = [0.88 0.65 0.10];
        else
            c = [0.82 0.08 0.08];
        end
        if ii <= nLeft
            y_row = ypos - (ii-1)*line_step;
            x_col = x_left;
        else
            y_row = ypos - (ii-nLeft-1)*line_step;
            x_col = x_right;
        end
        text(x_col, y_row, txt, ...
            'FontSize', 9.2, 'Parent', ax_text, ...
            'FontName', 'Segoe UI', 'Color', c);
    end
    drawnow;
    if mod(k, 30) == 0
        if P_on > 0
            fprintf('Frame %d/%d | FOS: %.3f | Load: %.0f N | Deflection:
%.2f mm\n', ...
                    k, nSteps, FOS_global, P_on, max_deflection);
        else
            fprintf('Frame %d/%d | No load\n', k, nSteps);
        end
    end
    % pause once on failure, then keep going
    if failed && ~pause_done
        pause(2);
        pause_done = true;
    end
end
if ~failed
    fprintf('\n========== SIMULATION COMPLETE ==========\n');
    fprintf('Bridge survived full train passage!\n');
    fprintf('Minimum FOS achieved: %.3f\n', min_FOS_history);
    fprintf('Critical mode: %s\n', critical_mode_history);
    fprintf('Maximum deflection: %.3f mm (L/%.0f)\n\n', ...
            max_deflection_history, L/max(max_deflection_history, 0.001));
    fprintf('--- MIN FOS BY MODE (HISTORICAL) ---\n');
    for ii = 1:nModes
        if isfinite(min_FOS_mode_history(ii))
            fprintf('%s: %.3f\n', modes{ii}, min_FOS_mode_history(ii));
        else
            fprintf('%s: --\n', modes{ii});
        end
```

```
    end
    fprintf('\n');
else
    fprintf('\n========== FINAL FAILURE SUMMARY ==========\n');
    fprintf('Failure location: x = %.1f mm\n', fail_x);
    fprintf('Failure mode: %s\n', fail_mode);
    fprintf('Load at failure: %.1f N (%.1f%% of 400 N)\n', ...
            P_on_fail, 100*P_on_fail/400);
    fprintf('FOS at failure: %.4f\n', FOS_at_fail);
    fprintf('Maximum deflection reached: %.3f mm (L/%.0f)\n\n', ...
            max_deflection_history, L/max(max_deflection_history, 0.001));
    fprintf('--- MIN FOS BY MODE (HISTORICAL, INCLUDING POST-FAILURE)
---\n');
    for ii = 1:nModes
        if isfinite(min_FOS_mode_history(ii))
            fprintf('%s: %.3f\n', modes{ii}, min_FOS_mode_history(ii));
        else
            fprintf('%s: --\n', modes{ii});
        end
    end
    fprintf('\n');
end
%% Local funcs
function [V, M, RA, RB] = compute_SFD_BMD_local(x, L, load_x, load_P)
    n = numel(x);
    V = zeros(1,n);
    M = zeros(1,n);
    if isempty(load_x)
        RA = 0; RB = 0;
        return;
    end
    load_x_snap = interp1(x, x, load_x, 'nearest', 'extrap');
    load_x_snap = max(0, min(L, load_x_snap));
    Ptot    = sum(load_P);
    momentA = sum(load_P .* load_x_snap);
    RB = momentA / L;
    RA = Ptot - RB;
    V(:) = RA;
    for i = 1:numel(load_x_snap)
        V(x >= load_x_snap(i)) = V(x >= load_x_snap(i)) - load_P(i);
    end
    M = cumtrapz(x, V);
    M = M - (M(end) - M(1)) * (x / L) - M(1);
    M(1)   = 0;
    M(end) = 0;
end
function draw_box_beam_smooth(ax, xline, wline, y_top0, H, zmin, zmax,
faceColor, edgeColor)
    yTop = wline + y_top0;
    yBot = yTop - H;
    n    = numel(xline);
    [X_top, Z_top] = meshgrid(xline, [zmin, zmax]);
    Y_top = repmat(yTop, 2, 1);
```

```
    surf(ax, X_top, Y_top, Z_top, ...
        'FaceColor', faceColor, 'EdgeColor', 'none', 'FaceAlpha', 0.94);
    [X_bot, Z_bot] = meshgrid(xline, [zmin, zmax]);
    Y_bot = repmat(yBot, 2, 1);
    surf(ax, X_bot, Y_bot, Z_bot, ...
        'FaceColor', faceColor*0.89, 'EdgeColor', 'none', 'FaceAlpha',
0.94);
    X_side = repmat(xline, 2, 1);
    Z_left = zmin * ones(2, n);
    Y_left = [yBot; yTop];
    surf(ax, X_side, Y_left, Z_left, ...
        'FaceColor', faceColor*0.82, 'EdgeColor', 'none', 'FaceAlpha',
0.91);
    Z_right = zmax * ones(2, n);
    Y_right = [yBot; yTop];
    surf(ax, X_side, Y_right, Z_right, ...
        'FaceColor', faceColor*0.82, 'EdgeColor', 'none', 'FaceAlpha',
0.91);
    xL = xline(1);
    xR = xline(end);
    vertsL = [ ...
        xL, yBot(1), zmin;
        xL, yTop(1), zmin;
        xL, yTop(1), zmax;
        xL, yBot(1), zmax ];
    patch(ax, 'Vertices', vertsL, 'Faces', [1 2 3 4], ...
        'FaceColor', faceColor*0.77, 'EdgeColor', edgeColor, ...
        'LineWidth', 1.1, 'FaceAlpha', 0.94);
    vertsR = [ ...
        xR, yBot(end), zmin;
        xR, yTop(end), zmin;
        xR, yTop(end), zmax;
        xR, yBot(end), zmax ];
    patch(ax, 'Vertices', vertsR, 'Faces', [1 2 3 4], ...
        'FaceColor', faceColor*0.77, 'EdgeColor', edgeColor, ...
        'LineWidth', 1.1, 'FaceAlpha', 0.94);
    plot3(ax, xline, yTop, zmin*ones(1,n), 'Color', edgeColor, 'LineWidth',
1.4);
    plot3(ax, xline, yTop, zmax*ones(1,n), 'Color', edgeColor, 'LineWidth',
1.4);
    plot3(ax, xline, yBot, zmin*ones(1,n), 'Color', edgeColor*0.72,
'LineWidth', 1.0);
    plot3(ax, xline, yBot, zmax*ones(1,n), 'Color', edgeColor*0.72,
'LineWidth', 1.0);
    plot3(ax, [xL xL], [yBot(1) yTop(1)], [zmin zmin], 'Color', edgeColor,
'LineWidth', 1.1);
    plot3(ax, [xL xL], [yBot(1) yTop(1)], [zmax zmax], 'Color', edgeColor,
'LineWidth', 1.1);
    plot3(ax, [xR xR], [yBot(end) yTop(end)], [zmin zmin], 'Color',
edgeColor, 'LineWidth', 1.1);
    plot3(ax, [xR xR], [yBot(end) yTop(end)], [zmax zmax], 'Color',
edgeColor, 'LineWidth', 1.1);
end
```

```
function draw_support_block(ax, x0, height, width)
   zmin = -width/2;
   zmax =  width/2;
   verts = [x0-15, 0,       zmin;
            x0+15, 0,       zmin;
            x0+15, -height, zmin;
            x0-15, -height, zmin;
            x0-15, 0,       zmax;
            x0+15, 0,       zmax;
            x0+15, -height, zmax;
            x0-15, -height, zmax];
   faces = [1 2 3 4;
            5 6 7 8;
            1 2 6 5;
            2 3 7 6;
            3 4 8 7;
            4 1 5 8];
   patch(ax, 'Vertices', verts, 'Faces', faces, ...
         'FaceColor', [0.38 0.38 0.38], 'EdgeColor', [0.18 0.18 0.18], ...
         'LineWidth', 1.3, 'FaceAlpha', 0.90);
end
```

```
function sec = section()
% this is the cross-section for design 0
% dfines span, material, geometry, capacities, and section properties.
   %% span and material
   sec.L  = 1200;     % span [mm]
```

```
sec.E  = 4000;      % MPa (N/mm^2), example matboard E
sec.mu = 0.2;       % Poisson's ratio
%% Cross-section geometry
geom.top_flange_width      = 100;   % mm
geom.top_flange_thickness  = 1.27;  % mm
geom.bottom_flange_width   = 80;    % mm
geom.bottom_flange_thickness = 1.27;
geom.web_height            = 75;    % mm (overall web depth)
geom.web_thickness         = 1.27;  % mm
geom.side_flange_width     = 5;     % mm (glue tab width)
% train spacing
sec.spacing_A = 176;   % mm
sec.spacing_B = 164;   % mm
% section properties
sp = compute_section_properties_like_python(geom);
sec.ybar   = sp.ybar;
sec.y_top  = sp.y_top;       % ybar -> top surface
sec.y_bot  = sp.y_bot;       % ybar -> bottom surface
sec.I      = sp.I;
sec.Q_web  = sp.Q_centroid;
sec.Q_glue = sp.Q_glue;
sec.t_web  = sp.t_shear;
sec.t_glue = sp.t_glue;
%% matieral properites (given)
tension_cap        = 30;    % flexural tension capacity
compression_cap    = 6;     % flexural compression capacity
shear_cap_matboard = 4;     % shear in matboard
shear_cap_glue     = 2;     % shear in glue
% local flange buckling
E  = sec.E;
mu = sec.mu;
overhang = (geom.top_flange_width - geom.bottom_flange_width) / 2;
k1 = 0.425;
sigma_overhang = k1 * pi^2 * E / (12*(1 - mu^2)) * ...
                 (geom.top_flange_thickness / overhang)^2;
between = geom.bottom_flange_width - 2 * geom.web_thickness;
k1b = 4.0;
sigma_between = k1b * pi^2 * E / (12*(1 - mu^2)) * ...
                (geom.top_flange_thickness / between)^2;
sigma_flange_crit = min(sigma_overhang, sigma_between);
% web buckling
k2 = 6.0;
sigma_web_crit = k2 * pi^2 * E / (12*(1 - mu^2)) * ...
                 (geom.web_thickness / geom.web_height)^2;
% shear buckling in web (with diaphragm spacing)
n_diaphragms = 10;
spacing_d    = sec.L / (n_diaphragms + 1);
k_s = 5.0;
tau_buckling = k_s * pi^2 * E / (12*(1 - mu^2)) * ...
    ((geom.web_thickness / geom.web_height)^2 + ...
     (geom.web_thickness / spacing_d)^2);
% pack capacities into struct
cap.tension         = tension_cap;
```

```
    cap.compression     = compression_cap;
    cap.flange_buckling = sigma_flange_crit;
    cap.web_buckling    = sigma_web_crit;
    cap.shear_board     = shear_cap_matboard;
    cap.shear_glue      = shear_cap_glue;
    cap.shear_buckling  = tau_buckling;
    sec.cap = cap;
end
% some helper functions
function sp = compute_section_properties_like_python(geom)
    bf_top = geom.top_flange_width;
    tf_top = geom.top_flange_thickness;
    bf_bot = geom.bottom_flange_width;
    tf_bot = geom.bottom_flange_thickness;
    tw     = geom.web_thickness;
    h_web  = geom.web_height;
    side_w = geom.side_flange_width;
    % bottom flange
    A_b = bf_bot * tf_bot;
    y_b = tf_bot / 2;
    % clear web height (between flanges)
    clear_web_h = h_web - tw;
    % two webs
    A_w = 2 * clear_web_h * tw;
    y_w = tf_bot + clear_web_h / 2;
    % two side tabs at top
    A_tabs = 2 * side_w * tw;
    y_tabs = h_web + tf_top / 2;
    % top flange
    A_t = bf_top * tf_top;
    y_t = h_web + tf_top / 2;
    % centroid
    A_total = A_b + A_w + A_tabs + A_t;
    ybar    = (A_b*y_b + A_w*y_w + A_tabs*y_tabs + A_t*y_t) / A_total;
    % second moment of area about NA
    I_b = (bf_bot * tf_bot^3)/12 + A_b * (y_b - ybar)^2;
    I_w = 2 * ((tw * clear_web_h^3)/12 + (clear_web_h * tw) * (y_w -
ybar)^2);
    I_tabs = 2 * ((side_w * tw^3)/12 + (side_w * tw) * (y_tabs - ybar)^2);
    I_t = (bf_top * tf_top^3)/12 + A_t * (y_t - ybar)^2;
    I_total = I_b + I_w + I_tabs + I_t;
    % distances to extreme fibres
    y_top_dist = (h_web + tf_top) - ybar;
    y_bot_dist = ybar;
    % Q for web shear (first moment of area above neutral axis)
    % This includes: top flange + side tabs + portion of web above NA
    Q_top_flange = A_t * (y_t - ybar);
    Q_tabs = A_tabs * (y_tabs - ybar);

    % Calculate web contribution above NA
    web_top = tf_bot + clear_web_h;
    if ybar < web_top
        % NA is within or below the web
```

```
        web_above_h = web_top - ybar;
        if web_above_h > clear_web_h
            web_above_h = clear_web_h;
        end
        % First moment: Area * distance from NA to centroid of that area
        web_cg_above = ybar + web_above_h / 2;
        Q_web_part = 2 * tw * web_above_h * (web_cg_above - ybar);
    else
        % NA is above web (unlikely but handle it)
        Q_web_part = 0;
    end

    Q_centroid = Q_top_flange + Q_tabs + Q_web_part;
    % Q for glue line (first moment of top flange about NA)
    % The glue connects top flange to webs
    Q_glue = Q_top_flange;
    % shear thicknesses
    t_shear = 2 * tw;       % two webs carry shear
    t_glue  = 2 * side_w;  % two glue lines (both sides)
    sp = struct( ...
        'ybar',       ybar, ...
        'y_top',      y_top_dist, ...
        'y_bot',      y_bot_dist, ...
        'I',          I_total, ...
        'Q_centroid',Q_centroid, ...
        'Q_glue',     Q_glue, ...
        't_shear',    t_shear, ...
        't_glue',     t_glue);
end
```

Design 0 Cross section file

```
function sec = section()
% this is the cross-section for design 0
% dfines span, material, geometry, capacities, and section properties.
   %% span and material
   sec.L  = 1200;      % span [mm]
   sec.E  = 4000;      % MPa (N/mm^2), example matboard E
   sec.mu = 0.2;       % Poisson's ratio
   %% Cross-section geometry
   geom.top_flange_width      = 100;    % mm
   geom.top_flange_thickness  = 1.27;   % mm
   geom.bottom_flange_width   = 80;     % mm
   geom.bottom_flange_thickness = 1.27;
   geom.web_height            = 75;     % mm (overall web depth)
   geom.web_thickness         = 1.27;   % mm
   geom.side_flange_width     = 5;      % mm (glue tab width)
   % train spacing
   sec.spacing_A = 176;    % mm
   sec.spacing_B = 164;    % mm
   % section properties
   sp = compute_section_properties_like_python(geom);
   sec.ybar   = sp.ybar;
   sec.y_top  = sp.y_top;       % ybar -> top surface
   sec.y_bot  = sp.y_bot;       % ybar -> bottom surface
   sec.I      = sp.I;
   sec.Q_web  = sp.Q_centroid;
   sec.Q_glue = sp.Q_glue;
   sec.t_web  = sp.t_shear;
   sec.t_glue = sp.t_glue;
   %% matieral properites (given)
   tension_cap        = 30;    % flexural tension capacity
   compression_cap    = 6;     % flexural compression capacity
   shear_cap_matboard = 4;     % shear in matboard
   shear_cap_glue     = 2;     % shear in glue
   % local flange buckling
   E  = sec.E;
   mu = sec.mu;
   overhang = (geom.top_flange_width - geom.bottom_flange_width) / 2;
   k1 = 0.425;
   sigma_overhang = k1 * pi^2 * E / (12*(1 - mu^2)) * ...
                    (geom.top_flange_thickness / overhang)^2;
   between = geom.bottom_flange_width - 2 * geom.web_thickness;
   k1b = 4.0;
   sigma_between = k1b * pi^2 * E / (12*(1 - mu^2)) * ...
                   (geom.top_flange_thickness / between)^2;
   sigma_flange_crit = min(sigma_overhang, sigma_between);
   % web buckling
   k2 = 6.0;
   sigma_web_crit = k2 * pi^2 * E / (12*(1 - mu^2)) * ...
                    (geom.web_thickness / geom.web_height)^2;
```

```
    % shear buckling in web (with diaphragm spacing)
    n_diaphragms = 10;
    spacing_d    = sec.L / (n_diaphragms + 1);
    k_s = 5.0;
    tau_buckling = k_s * pi^2 * E / (12*(1 - mu^2)) * ...
        ((geom.web_thickness / geom.web_height)^2 + ...
         (geom.web_thickness / spacing_d)^2);
    % pack capacities into struct
    cap.tension         = tension_cap;
    cap.compression     = compression_cap;
    cap.flange_buckling = sigma_flange_crit;
    cap.web_buckling    = sigma_web_crit;
    cap.shear_board     = shear_cap_matboard;
    cap.shear_glue      = shear_cap_glue;
    cap.shear_buckling  = tau_buckling;
    sec.cap = cap;
end
% some helper functions
function sp = compute_section_properties_like_python(geom)
    bf_top = geom.top_flange_width;
    tf_top = geom.top_flange_thickness;
    bf_bot = geom.bottom_flange_width;
    tf_bot = geom.bottom_flange_thickness;
    tw     = geom.web_thickness;
    h_web  = geom.web_height;
    side_w = geom.side_flange_width;
    % bottom flange
    A_b = bf_bot * tf_bot;
    y_b = tf_bot / 2;
    % clear web height (between flanges)
    clear_web_h = h_web - tw;
    % two webs
    A_w = 2 * clear_web_h * tw;
    y_w = tf_bot + clear_web_h / 2;
    % two side tabs at top
    A_tabs = 2 * side_w * tw;
    y_tabs = h_web + tf_top / 2;
    % top flange
    A_t = bf_top * tf_top;
    y_t = h_web + tf_top / 2;
    % centroid
    A_total = A_b + A_w + A_tabs + A_t;
    ybar    = (A_b*y_b + A_w*y_w + A_tabs*y_tabs + A_t*y_t) / A_total;
    % second moment of area about NA
    I_b = (bf_bot * tf_bot^3)/12 + A_b * (y_b - ybar)^2;
    I_w = 2 * ((tw * clear_web_h^3)/12 + (clear_web_h * tw) * (y_w -
ybar)^2);
    I_tabs = 2 * ((side_w * tw^3)/12 + (side_w * tw) * (y_tabs - ybar)^2);
    I_t = (bf_top * tf_top^3)/12 + A_t * (y_t - ybar)^2;
    I_total = I_b + I_w + I_tabs + I_t;
    % distances to extreme fibres
    y_top_dist = (h_web + tf_top) - ybar;
    y_bot_dist = ybar;
```

```
    % Q for web shear (first moment of area above neutral axis)
    % This includes: top flange + side tabs + portion of web above NA
    Q_top_flange = A_t * (y_t - ybar);
    Q_tabs = A_tabs * (y_tabs - ybar);

    % Calculate web contribution above NA
    web_top = tf_bot + clear_web_h;
    if ybar < web_top
        % NA is within or below the web
        web_above_h = web_top - ybar;
        if web_above_h > clear_web_h
            web_above_h = clear_web_h;
        end
        % First moment: Area * distance from NA to centroid of that area
        web_cg_above = ybar + web_above_h / 2;
        Q_web_part = 2 * tw * web_above_h * (web_cg_above - ybar);
    else
        % NA is above web (unlikely but handle it)
        Q_web_part = 0;
    end

    Q_centroid = Q_top_flange + Q_tabs + Q_web_part;
    % Q for glue line (first moment of top flange about NA)
    % The glue connects top flange to webs
    Q_glue = Q_top_flange;
    % shear thicknesses
    t_shear = 2 * tw;       % two webs carry shear
    t_glue  = 2 * side_w;  % two glue lines (both sides)
    sp = struct( ...
        'ybar',       ybar, ...
        'y_top',      y_top_dist, ...
        'y_bot',      y_bot_dist, ...
        'I',          I_total, ...
        'Q_centroid',Q_centroid, ...
        'Q_glue',     Q_glue, ...
        't_shear',    t_shear, ...
        't_glue',     t_glue);
end
```

Our Final Iteration Cross Section

```
function sec = section2()
% our final cross section
%  L = 1269 mm, H_total = 60 mm
% our cross section used webs total (each 1.27 mm thick)
% bottom flange is single layer
% Top flange is double layered
   %% span + material
   sec.L  = 1269;      % mm
   sec.E  = 4000;      % MPa
   sec.mu = 0.2;
   %% train spacings
   sec.spacing_A = 176;   % mm
   sec.spacing_B = 164;   % mm
   %% geometry
   g.t        = 1.27;       % base matboard thickness [mm]
   g.b_top    = 100.0;      % top flange width [mm]
   g.b_bot    = 92.54;      % bottom flange width [mm]
   g.tab_w    = 5.0;        % glue tab width each side [mm]
   g.H_total  = 60.0;       % total height [mm]
   g.n_webs   = 4;          % number of webs (each thickness t)
   % flange thicknesses
   g.t_bot = g.t;           % bottom flange = 1 layer
   g.t_top = 2*g.t;         % top flange = 2 layers stacked
   % clear web height between flanges
   g.h_web = g.H_total - g.t_bot - g.t_top;  % mm (clear web height)
   % diaphragm spacing a for Case 4
   g.a_diaphragm = 40.0;    % mm
   % paired webs act as one plate (no spacing)
   g.web_pair_thickness = 2*g.t;  % 2.54 mm
   %% Section properties (MATCHING PYTHON)
   sp = compute_section_properties_python_match(g);
   sec.ybar   = sp.ybar;
   sec.y_top  = sp.y_top;
   sec.y_bot  = sp.y_bot;
   sec.I      = sp.I;
   sec.Q_web  = sp.Q_web;
   sec.Q_glue = sp.Q_glue;
   % Effective shear widths b in tau = VQ/(I b)
   sec.t_web  = sp.b_web_eff;   % total web shear width
   sec.t_glue = sp.b_glue_eff;  % total glue width
   %% Material capacities (MPa)
   tension_cap        = 30;
   compression_cap    = 6;
   shear_cap_matboard = 4;
   shear_cap_glue     = 2;
   E  = sec.E;
   mu = sec.mu;
   % buckling parameters
```

```
    t_flange = g.t_top;                 % thickness of top flange plate
(total)
    b_web    = g.web_pair_thickness;    % paired web plate thickness
    h_web    = g.h_web;                 % clear web height
    ytop     = sec.y_top;               % distance NA -> top fibre
    % buckling cases
    % case 1
    b_in = g.b_bot - 2*b_web;
    k1 = 4.0;
    sigma_case1 = k1*pi^2*E/(12*(1-mu^2)) * (t_flange/b_in)^2;
    % case 2
    b_out = (g.b_top - g.b_bot)/2;
    k2 = 0.4254;
    sigma_case2 = k2*pi^2*E/(12*(1-mu^2)) * (t_flange/b_out)^2;
    sigma_flange_crit = min(sigma_case1, sigma_case2);
    % case 3
    k3 = 6.0;
    sigma_web_crit = k3*pi^2*E/(12*(1-mu^2)) * (b_web/ytop)^2;
    % case 4
    a  = g.a_diaphragm;
    k4 = 5.0;
    tau_buckling = k4*pi^2*E/(12*(1-mu^2)) * ( (b_web/h_web)^2 + (b_web/a)^2
);
    %% material properites + caps
    cap.tension          = tension_cap;
    cap.compression      = compression_cap;
    cap.flange_buckling = sigma_flange_crit;  % governing case 1/2
    cap.web_buckling     = sigma_web_crit;    % case 3
    cap.shear_board      = shear_cap_matboard;
    cap.shear_glue       = shear_cap_glue;
    cap.shear_buckling   = tau_buckling;      % case 4
    sec.cap = cap;
end
% Section properties
function sp = compute_section_properties_python_match(g)
    t      = g.t;
    bt     = g.b_top;
    tt     = g.t_top;      % top flange total thickness (2t)
    bb     = g.b_bot;
    tb     = g.t_bot;      % bottom flange thickness (t)
    hw     = g.h_web;      % clear web height
    tab_w  = g.tab_w;
    n_webs = g.n_webs;
    % Y-LEVELS (matching Python)
    y_bot_top  = tb;                    % top of bottom flange
    y_web_top  = tb + hw;               % top of webs = underside of top
flange
    y_top_top  = y_web_top + tt;        % very top surface (should = 60)
    % Centroids of each component
    y_bot_c    = tb / 2.0;
    y_webs_c   = tb + hw / 2.0;

    % Tabs live INSIDE at top, under the double top flange
```

```
    y_tabs_c   = y_web_top - t / 2.0;

    y_top_c    = y_web_top + tt / 2.0;
    % areas (tabs ARE structural)
    A_bot  = bb * tb;
    A_webs = n_webs * t * hw;
    A_tabs = 2 * (tab_w * t);
    A_top  = bt * tt;
    A_total = A_bot + A_webs + A_tabs + A_top;
    % ybar
    ybar = (A_bot*y_bot_c + A_webs*y_webs_c + A_tabs*y_tabs_c +
A_top*y_top_c) / A_total;
    % intertia
    I_bot = (bb*tb^3)/12 + A_bot*(y_bot_c - ybar)^2;

    I_web_each = (t*hw^3)/12 + (t*hw)*(y_webs_c - ybar)^2;
    I_webs = n_webs * I_web_each;

    I_tab_each = (tab_w*t^3)/12 + (tab_w*t)*(y_tabs_c - ybar)^2;
    I_tabs = 2 * I_tab_each;

    I_top = (bt*tt^3)/12 + A_top*(y_top_c - ybar)^2;
    I_total = I_bot + I_webs + I_tabs + I_top;
    % shear free areas (Q)
    y_top_dist = y_top_top - ybar;
    y_bot_dist = ybar;
    % Q at centroid
    Q_web = 0;

    Q_web = Q_web + A_top*(y_top_c - ybar);

    if ybar < y_web_top
        Q_web = Q_web + A_tabs*(y_tabs_c - ybar);
    end

    if ybar < y_web_top
        h_above = y_web_top - ybar;
        A_webs_above = n_webs * t * h_above;
        y_webs_above_c = ybar + h_above/2;
        Q_web = Q_web + A_webs_above*(y_webs_above_c - ybar);
    end
    % Q for glue
    Q_glue = A_top*(y_top_c - ybar);
    b_web_eff  = n_webs * t;
    b_glue_eff = 2 * tab_w;
    sp.ybar      = ybar;
    sp.y_top     = y_top_dist;
    sp.y_bot     = y_bot_dist;
    sp.I         = I_total;
    sp.Q_web     = Q_web;
    sp.Q_glue    = Q_glue;
    sp.b_web_eff = b_web_eff;
    sp.b_glue_eff= b_glue_eff;
```

```
end
```

Python Main file

```
"""
In-depth anaylsis of the bridge, this was actually very fun to make
matlab follows the same logic but its cooler and im better at python so this
anaylsis was much easier
"""
import numpy as np
import matplotlib.pyplot as plt
from math import pi

# import the cross-section calculations
import section as secmod
from section import compute_section_properties


def sig_str(x):  # sliderule precision :), some rounding errors occuered though
    if x == 0:
        return "0"
    x_abs = abs(x)
    sci = f"{x_abs:.5e}"
    first_sig_digit = sci[0]
    return f"{x:.4g}" if first_sig_digit == "1" else f"{x:.3g}"


#general bridge inputs
L = 1269

spacing_A = 176
spacing_B = 164

TOTAL_LOAD = 400
num_loads = 6
per_load = TOTAL_LOAD / num_loads
load_values = [per_load] * num_loads

num_points = 1201
x = np.linspace(0, L, num_points)

slide_step = 1.0

# load positions along the bridge
offsets = np.array([
    0,
    spacing_A,
    spacing_A + spacing_B,
```

```
    spacing_A + spacing_B + spacing_A,
    spacing_A + spacing_B + spacing_A + spacing_B,
    spacing_A + spacing_B + spacing_A + spacing_B + spacing_A
])

group_length = offsets[-1]
start_positions = np.arange(0, L - group_length + 1e-9, slide_step)


# SFD and BMD calculations
def compute_sfd(load_positions, load_vals):
    total_load = sum(load_vals)
    moment_A = sum(p * m for p, m in zip(load_positions, load_vals))

    RB = moment_A / L
    RA = total_load - RB

    V = np.ones_like(x) * RA
    for p, m in zip(load_positions, load_vals):
        V[x >= p] -= m
    return V


def compute_bmd(V):
    dx = x[1] - x[0]
    M = np.cumsum(V) * dx
    M[-1] = 0
    return M


#envleopes
env_M_pos = np.full_like(x, -np.inf)
env_M_neg = np.full_like(x, np.inf)
env_V_pos = np.full_like(x, -np.inf)
env_V_neg = np.full_like(x, np.inf)

sample_idx = [
    0,
    len(start_positions) // 4,
    len(start_positions) // 2,
    3 * len(start_positions) // 4,
    len(start_positions) - 1
]

sample_M_list = []
```

```
sample_V_list = []

for i, p0 in enumerate(start_positions):
    loads_here = p0 + offsets
    V = compute_sfd(loads_here, load_values)
    M = compute_bmd(V)

    env_M_pos = np.maximum(env_M_pos, M)
    env_M_neg = np.minimum(env_M_neg, M)
    env_V_pos = np.maximum(env_V_pos, V)
    env_V_neg = np.minimum(env_V_neg, V)

    if i in sample_idx:
        sample_M_list.append(M.copy())
        sample_V_list.append(V.copy())

print("Envelope computation complete.")

BMD_envelope = env_M_pos
SFD_envelope = env_V_pos


#import gemoetry
if hasattr(secmod, "get_geometry"):
    geometry = secmod.get_geometry()
elif hasattr(secmod, "geometry"):
    geometry = secmod.geometry
else:
    raise ValueError("section.py must define geometry or get_geometry().")



ybar, ytop, ybot, I, Q_centroid, Q_glue, t_shear, t_glue =
compute_section_properties(geometry)



def compute_ybar_I_details(geom):
    bf_top = geom["top_flange_width"]
    tf_top = geom["top_flange_thickness"]

    bf_bot = geom["bottom_flange_width"]
    tf_bot = geom["bottom_flange_thickness"]

    tw = geom["web_thickness"]
```

```
    h_web = geom["web_height"]

    side_w = geom.get("side_flange_width", 0.0)
    n_webs = geom.get("n_webs", 2)

    # clear web height between flanges
    clear_web_h = h_web - tw

    # Areas and centroids from bottom
    A_b = bf_bot * tf_bot
    y_b = tf_bot / 2.0

    A_w = n_webs * clear_web_h * tw
    y_w = tf_bot + clear_web_h / 2.0

    A_tabs = 2.0 * side_w * tw
    y_tabs = h_web + tf_top / 2.0

    A_t = bf_top * tf_top
    y_t = h_web + tf_top / 2.0

    A_total = A_b + A_w + A_tabs + A_t
    ybar_calc = (A_b*y_b + A_w*y_w + A_tabs*y_tabs + A_t*y_t) / A_total

    # Second moment about NA (horizontal)
    I_b = (bf_bot * tf_bot**3)/12.0 + A_b * (y_b - ybar_calc)**2

    I_w_single = (tw * clear_web_h**3)/12.0 + (clear_web_h * tw) * (y_w -
ybar_calc)**2
    I_w = n_webs * I_w_single

    I_tabs = 2.0 * ((side_w * tw**3)/12.0 + (side_w * tw) * (y_tabs - ybar_calc)**2)

    I_t = (bf_top * tf_top**3)/12.0 + A_t * (y_t - ybar_calc)**2

    I_total_calc = I_b + I_w + I_tabs + I_t

    return ybar_calc, I_total_calc, {
        "A_b": A_b, "A_w": A_w, "A_tabs": A_tabs, "A_t": A_t,
        "y_b": y_b, "y_w": y_w, "y_tabs": y_tabs, "y_t": y_t,
        "clear_web_h": clear_web_h, "n_webs": n_webs
    }


ybar_calc, I_calc, details = compute_ybar_I_details(geometry)
```

```
print("\n" + "="*60)
print("CROSS-SECTION PROPERTIES")
print("="*60)
print(f"ȳ (ybar from section): {sig_str(ybar)} mm")
print(f"ȳ (ybar calc check):   {sig_str(ybar_calc)} mm")
print(f"y_top: {sig_str(ytop)} mm")
print(f"y_bot: {sig_str(ybot)} mm")
print(f"I (from section): {sig_str(I)} mm⁴ = {sig_str(I/1e6)} × 10⁶ mm⁴")
print(f"I (calc check):   {sig_str(I_calc)} mm⁴ = {sig_str(I_calc/1e6)} × 10⁶ mm⁴")
print(f"Q_centroid: {sig_str(Q_centroid)} mm³")
print(f"Q_glue: {sig_str(Q_glue)} mm³")
print(f"t_shear: {sig_str(t_shear)} mm")
print(f"t_glue: {sig_str(t_glue)} mm")
print("="*60 + "\n")



def compute_fos_all():
    E = 4000
    mu = 0.2

    tension_cap = 30
    compression_cap = 6
    shear_cap_matboard = 4
    shear_cap_glue = 2

    eps = 1e-10

    stress_top = -BMD_envelope * ytop / I   # compression at top for +M
    stress_bot =  BMD_envelope * ybot / I   # tension at bottom for +M

    tau_cent = SFD_envelope * Q_centroid / (I * t_shear)
    tau_gl   = SFD_envelope * Q_glue     / (I * t_glue)

    stress_bot_tens = np.maximum(stress_bot, 0.0)
    FOS_tension     = tension_cap / (stress_bot_tens + eps)
    FOS_compression = compression_cap / (np.abs(stress_top) + eps)

    t_flange = 2 * geometry["top_flange_thickness"]
    b_web    = geometry.get("web_pair_thickness", geometry["web_thickness"])
    h_web    = geometry["web_height"]

    if "b_in" in geometry:
```

```
        b_in = geometry["b_in"]
    else:
        b_in = geometry["bottom_flange_width"] - 2.0 * b_web

    sigma_case1 = (
        4.0 * pi**2 * E / (12.0 * (1 - mu**2))
        * (t_flange / b_in)**2
    )
    FOS_buck_case1 = sigma_case1 / (np.abs(stress_top) + eps)

    if "b_out" in geometry:
        b_out = geometry["b_out"]
    else:
        b_out = (geometry["top_flange_width"] - geometry["bottom_flange_width"]) /
2.0

    sigma_case2 = (
        0.425 * 4.0 * pi**2 * E / (12.0 * (1 - mu**2))
        * (t_flange / b_out)**2
    )
    FOS_buck_case2 = sigma_case2 / (np.abs(stress_top) + eps)

    sigma_case3 = (
        6.0 * pi**2 * E / (12.0 * (1 - mu**2))
        * (b_web / ytop)**2
    )
    FOS_buck_case3 = sigma_case3 / (np.abs(stress_top) + eps)

    n_diaphragms = 10.5
    a = L / (n_diaphragms + 1.0)
    tau_case4 = (
        5.0 * pi**2 * E / (12.0 * (1 - mu**2))
        * ((b_web / h_web)**2 + (b_web / a)**2)
    )
    FOS_buck_case4 = tau_case4 / (np.abs(tau_cent) + eps)

    FOS_shear_mat  = shear_cap_matboard / (np.abs(tau_cent) + eps)
    FOS_shear_glue = shear_cap_glue     / (np.abs(tau_gl) + eps)

    FOS_min = np.minimum.reduce([
        FOS_tension,
        FOS_compression,
        FOS_buck_case1,
        FOS_buck_case2,
        FOS_buck_case3,
```

```
        FOS_buck_case4,
        FOS_shear_mat,
        FOS_shear_glue
    ])

    return {
        "tension": FOS_tension,
        "compression": FOS_compression,
        "buck_case1": FOS_buck_case1,
        "buck_case2": FOS_buck_case2,
        "buck_case3": FOS_buck_case3,
        "buck_case4": FOS_buck_case4,
        "shear_matboard": FOS_shear_mat,
        "shear_glue": FOS_shear_glue,
        "min": FOS_min,
        "sigma_case1": sigma_case1,
        "sigma_case2": sigma_case2,
        "sigma_case3": sigma_case3,
        "tau_case4": tau_case4,
        "a": a,
        "b_in": b_in,
        "b_out": b_out
    }


FOS = compute_fos_all()

min_FOS_global = np.min(FOS["min"])
idx_min_global = np.argmin(FOS["min"])
x_min_global = x[idx_min_global]
P_failure = TOTAL_LOAD * min_FOS_global

below1 = np.where(FOS["min"] < 1.0)[0]
if len(below1) > 0:
    fail_idx = below1[0]
    fail_x = x[fail_idx]
    failed_under_current_load = True
else:
    fail_idx = idx_min_global
    fail_x = x_min_global
    failed_under_current_load = False

min_FOS_at_reported = FOS["min"][fail_idx]

modes = [
```

```
    "Flexural Tension",
    "Flexural Compression",
    "Buckling Case 1 (Top flange inboard)",
    "Buckling Case 2 (Top flange outstand)",
    "Buckling Case 3 (Web compression)",
    "Buckling Case 4 (Shear w/ diaphragms)",
    "Shear (Matboard)",
    "Shear (Glue)"
]

vals_at_reported = [
    FOS["tension"][fail_idx],
    FOS["compression"][fail_idx],
    FOS["buck_case1"][fail_idx],
    FOS["buck_case2"][fail_idx],
    FOS["buck_case3"][fail_idx],
    FOS["buck_case4"][fail_idx],
    FOS["shear_matboard"][fail_idx],
    FOS["shear_glue"][fail_idx]
]

mode_reported = modes[np.argmin(vals_at_reported)]


print("\n" + "="*60)
print("ANALYSIS RESULTS")
print("="*60)
print(f"Applied load: {sig_str(TOTAL_LOAD)} N")
print(f"Predicted failure load: {sig_str(P_failure)} N")
print(f"Reported location: {sig_str(fail_x)} mm | min FOS there =
{sig_str(min_FOS_at_reported)}")
print(f"Critical mode there: {mode_reported}")
print("="*60 + "\n")


# pLOTTING
plt.style.use("default")

fig = plt.figure(figsize=(21, 10))
fig.patch.set_facecolor("#f7f9fc")

gs = fig.add_gridspec(
    2, 2,
    hspace=0.35, wspace=0.25,
    left=0.05, right=0.97,
```

```
    top=0.92, bottom=0.08,
    height_ratios=[1.0, 0.8]
)

colors = ["#6C5CE7", "#00CEC9", "#0984E3", "#E17055", "#00B894"]
env_color_m = "#2D3436"
env_color_v = "#D63031"

from matplotlib.lines import Line2D

def style_ax(ax):
    ax.grid(True, alpha=0.20, linestyle="--", linewidth=0.7)
    ax.set_axisbelow(True)
    ax.tick_params(labelsize=10)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)


# Plot 1: BMD envelope
ax1 = fig.add_subplot(gs[0, 0])
for i, M in enumerate(sample_M_list):
    ax1.plot(x, M, "--", linewidth=1.0, alpha=0.30, color=colors[i % len(colors)])
env_line, = ax1.plot(x, BMD_envelope, linewidth=3.0, color=env_color_m,
label="Envelope")
ax1.set_title("Bending Moment Envelope", fontsize=14, fontweight="bold")
ax1.set_xlabel("Position (mm)")
ax1.set_ylabel("Bending Moment (N·mm)")
ax1.axvline(fail_x, linestyle=":", linewidth=1.6, color="#7f8c8d")
sample_line = Line2D([0], [0], linestyle="--", color="gray", alpha=0.6,
label="Sample positions")
ax1.legend(handles=[env_line, sample_line], fontsize=9, framealpha=0.95, loc="upper
right")
style_ax(ax1)
ax1.set_xlim(0, L)


# Plot 2: SFD envelope
ax2 = fig.add_subplot(gs[0, 1])
for i, V in enumerate(sample_V_list):
    ax2.plot(x, V, "--", linewidth=1.0, alpha=0.30, color=colors[i % len(colors)])
env_v_line, = ax2.plot(x, SFD_envelope, linewidth=3.0, color=env_color_v,
label="Envelope")
ax2.axhline(0, linestyle="-", linewidth=0.8, color="#B2BEC3")
ax2.axvline(fail_x, linestyle=":", linewidth=1.6, color="#7f8c8d")
ax2.set_title("Shear Force Envelope", fontsize=14, fontweight="bold")
```

```
ax2.set_xlabel("Position (mm)")
ax2.set_ylabel("Shear Force (N)")
ax2.set_xlim(0, L)
sample_v_line = Line2D([0], [0], linestyle="--", color="gray", alpha=0.6,
label="Sample positions")
ax2.legend(handles=[env_v_line, sample_v_line], fontsize=9, framealpha=0.95,
loc="upper right")
style_ax(ax2)


# Plot 3: Wide analysis summary (spans entire bottom row)
ax3 = fig.add_subplot(gs[1, :])
ax3.axis("off")

summary_text = f"""
DESIGN SUMMARY
{'='*60}

Applied Load: {sig_str(TOTAL_LOAD)} N
Predicted Failure Load (global min FOS): {sig_str(P_failure)} N

Reported point (first unsafe):
x = {sig_str(fail_x)} mm
min FOS there = {sig_str(min_FOS_at_reported)}
Critical mode there = {mode_reported}

SECTION CALCS (ybar + I)
{'='*60}
Areas:
A_bottom = {sig_str(details["A_b"])} mm^2
A_webs   = {sig_str(details["A_w"])} mm^2  (n_webs = {details["n_webs"]})
A_tabs   = {sig_str(details["A_tabs"])} mm^2
A_top    = {sig_str(details["A_t"])} mm^2

Centroid from bottom:
ȳ_calc = (ΣA_i y_i)/(ΣA_i) = {sig_str(ybar_calc)} mm
ȳ_used (section file)      = {sig_str(ybar)} mm

Second moment about NA:
I_calc = Σ(I_i + A_i d_i^2) = {sig_str(I_calc)} mm^4
I_used (section file)       = {sig_str(I)} mm^4

BUCKLING INPUTS
{'='*60}
b_in  = {sig_str(FOS["b_in"])} mm
```

```
b_out = {sig_str(FOS["b_out"])} mm
a (diaphragm spacing) = {sig_str(FOS["a"])} mm
"""

ax3.text(
    0.02, 0.98,
    summary_text,
    fontsize=9.6,
    va="top",
    family="monospace",
    bbox=dict(boxstyle="round", facecolor="white", alpha=0.97)
)

plt.suptitle(
    f"Bridge Failure Analysis — Predicted Failure: {sig_str(P_failure)} N",
    fontsize=16,
    fontweight="bold"
)

plt.tight_layout(rect=[0.03, 0.06, 0.97, 0.94])
plt.show()
```

Python Design 0 File

```python
# section.py
# cross section geometry of design 0

import numpy as np

geometry = {
    "top_flange_width": 100.0,
    "top_flange_thickness": 1.27,

    "bottom_flange_width": 80.0,
    "bottom_flange_thickness": 1.27,

    "web_thickness": 1.27,
    "web_height": 75.0 - 2*1.27,  # clear web height = 72.46 mm

    "side_flange_width": 5.0,
    "tab_thickness": 1.27,

    # Required for buckling calculations:
    "b_in": 80.0 - 2*1.27,  # 77.46 mm - clear width between webs (Case 1)
    "b_out": (100.0 - 80.0) / 2.0  # 10.0 mm - overhang (Case 2)
}

def get_geometry():
    return geometry


def compute_section_properties(geom):

    bf_top = geom["top_flange_width"]
    tf_top = geom["top_flange_thickness"]

    bf_bot = geom["bottom_flange_width"]
    tf_bot = geom["bottom_flange_thickness"]

    t_web  = geom["web_thickness"]
    h_web  = geom["web_height"]

    tab_w  = geom["side_flange_width"]
    tab_t  = geom.get("tab_thickness", t_web)

    # Y-levels
    y_bot_top  = tf_bot                        # top of bottom flange
```

```
y_web_top  = tf_bot + h_web              # top of side webs
y_tab_top  = y_web_top + tab_t           # top of tabs
y_top_top  = y_tab_top + tf_top          # very top surface (76.27 mm)

# Areas + centroids
# Bottom flange
A_bot = bf_bot * tf_bot
y_bot_c = tf_bot / 2.0

# Side webs
A_webs = 2 * (t_web * h_web)
y_webs_c = y_bot_top + h_web / 2.0

# Glue tabs (horizontal elements)
A_tabs = 2 * (tab_w * tab_t)
y_tabs_c = y_web_top + tab_t / 2.0

# Top flange
A_top = bf_top * tf_top
y_top_c = y_tab_top + tf_top / 2.0

# Neutral axis
A_total = A_bot + A_webs + A_tabs + A_top

ybar = (
    A_bot  * y_bot_c
    + A_webs * y_webs_c
    + A_tabs * y_tabs_c
    + A_top  * y_top_c
) / A_total

# Moment of inertia
I_bot = (bf_bot * tf_bot**3) / 12.0 + A_bot * (y_bot_c - ybar)**2

I_web_each = (t_web * h_web**3) / 12.0 + (t_web * h_web) * (y_webs_c - ybar)**2
I_webs = 2 * I_web_each

I_tab_each = (tab_w * tab_t**3) / 12.0 + (tab_w * tab_t) * (y_tabs_c - ybar)**2
I_tabs = 2 * I_tab_each

I_top = (bf_top * tf_top**3) / 12.0 + A_top * (y_top_c - ybar)**2

I_total = I_bot + I_webs + I_tabs + I_top

# Extreme fiber distances
```

```
    y_top_dist = y_top_top - ybar
    y_bot_dist = ybar

    # Q at centroid (for shear through webs)
    Q_centroid = 0.0

    # Top flange contribution
    if ybar < y_top_top:
        Q_centroid += A_top * (y_top_c - ybar)

    # Tabs contribution
    if ybar < y_tab_top:
        Q_centroid += A_tabs * (y_tabs_c - ybar)

    # Webs portion above NA
    if ybar < y_web_top:
        h_above = y_web_top - ybar
        A_web_above = 2 * (t_web * h_above)
        y_web_above_c = ybar + h_above / 2.0
        Q_centroid += A_web_above * (y_web_above_c - ybar)

    # Q for glue line (top flange only)
    A_glue = bf_top * tf_top
    d_glue = y_top_c - ybar
    Q_glue = A_glue * d_glue

    # Shear widths
    t_shear = 2 * t_web      # two side webs carry shear
    t_glue  = 2 * tab_w      # two glue tabs

    return ybar, y_top_dist, y_bot_dist, I_total, Q_centroid, Q_glue, t_shear,
t_glue
```

Final iteration python Code

```
import numpy as np

geometry = {
    "top_flange_width": 100.0,
    "top_flange_thickness": 1.27,     # single ply
    "bottom_flange_width": 92.54,
    "bottom_flange_thickness": 1.27,
    "web_thickness": 1.27,
    "web_height": 56.19,              # clear height
    "tab_width": 5.0,
    "n_webs": 4
}

def get_geometry():
    return geometry


def compute_section_properties(geom):

    # pull geometry
    bf_top = geom["top_flange_width"]
    tf_top = geom["top_flange_thickness"]
    bf_bot = geom["bottom_flange_width"]
    tf_bot = geom["bottom_flange_thickness"]
    web_t  = geom["web_thickness"]
    web_h  = geom["web_height"]
    tab_w  = geom["tab_width"]
    n_webs = geom["n_webs"]

    # top flange is double ply
    tf_top_total = 2 * tf_top

    y_bot_top  = tf_bot                          # top of bottom flange
    y_web_top  = tf_bot + web_h                  # top of webs = underside of top
flange
    y_top_top  = y_web_top + tf_top_total        # very top surface (should equal 60)


    y_bot_c  = tf_bot / 2.0
    y_webs_c = tf_bot + web_h / 2.0


    y_tabs_c = y_web_top - web_t / 2.0
```

```
y_top_c = y_web_top + tf_top_total / 2.0

A_bot  = bf_bot * tf_bot
A_webs = n_webs * web_t * web_h
A_tabs = 2 * (tab_w * web_t)
A_top  = bf_top * tf_top_total

A_total = A_bot + A_webs + A_tabs + A_top

ybar = (
    A_bot  * y_bot_c +
    A_webs * y_webs_c +
    A_tabs * y_tabs_c +
    A_top  * y_top_c
) / A_total

I_bot = (bf_bot * tf_bot**3)/12 + A_bot*(y_bot_c - ybar)**2

I_web_each = (web_t * web_h**3)/12 + (web_t*web_h)*(y_webs_c - ybar)**2
I_webs = n_webs * I_web_each

I_tab_each = (tab_w * web_t**3)/12 + (tab_w*web_t)*(y_tabs_c - ybar)**2
I_tabs = 2 * I_tab_each

I_top = (bf_top * tf_top_total**3)/12 + A_top*(y_top_c - ybar)**2

I_total = I_bot + I_webs + I_tabs + I_top


y_top_dist = y_top_top - ybar
y_bot_dist = ybar


Q_centroid = 0.0

# top flange
Q_centroid += A_top * (y_top_c - ybar)

# tabs
if ybar < y_web_top:
    Q_centroid += A_tabs * (y_tabs_c - ybar)

# webs above NA
if ybar < y_web_top:
```

```
        h_above = y_web_top - ybar
        A_webs_above = n_webs * web_t * h_above
        y_webs_above_c = ybar + h_above/2
        Q_centroid += A_webs_above * (y_webs_above_c - ybar)

    # glue-line Q: only top flange above glue line
    Q_glue = A_top * (y_top_c - ybar)

    # shear thicknesses
    t_shear = n_webs * web_t
    t_glue  = 2 * tab_w

    return ybar, y_top_dist, y_bot_dist, I_total, Q_centroid, Q_glue, t_shear,
t_glue
```