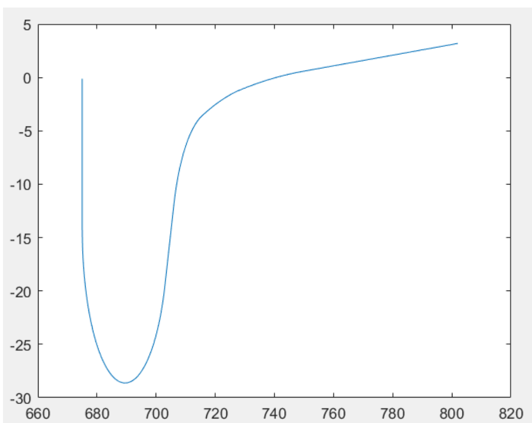
Assignment 1 : Kinematics of Railway Wheelset

Khushi Agrawal (210514), Shashank Singh Tomar (210965)

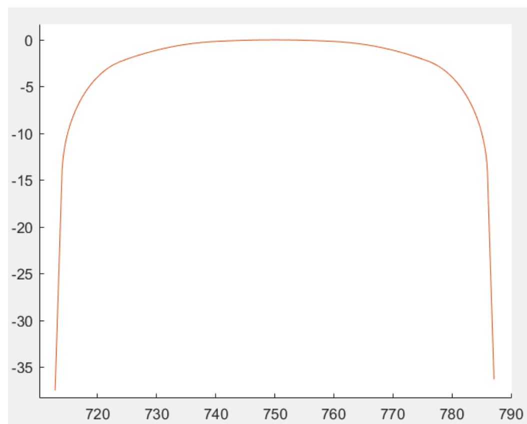
We will broadly describe the steps of our solution here, The code implementation is provided at the end.

1. Plotting the rail wheel profile

Using the engineering drawings provided we coded equations for line, circle in MATLAB to represent the profiles exactly. The initial orientation of our profiles was:



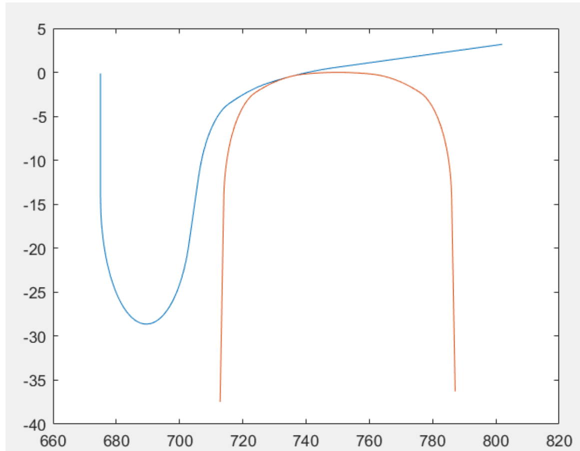
Wheel Profile



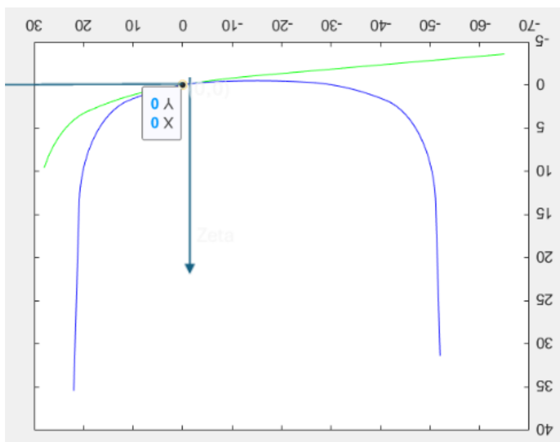
Rail Profile

2. Finding the contact point

We found the contact point by aligning the x,y and x*,y* axes (aligning center points). Then we found the y coordinate for rail, wheel with equal slope for rail, wheel profile by using Newton Raphson. The corresponding z distance between the two profiles was then shifted.



Then we calculated the axis according to the requirement of our equations:



3. Solving the equations

$$\xi_w = f(\eta_w), \quad \xi_r = g(\eta_r) \qquad (1)$$

$$u_y - y_0 - r_0\phi - \eta_{wr} + \eta_{rr} = 0 \qquad (2)$$

$$u_z + l\phi + \zeta_{wr} - \zeta_{rr} = 0 \qquad (3)$$

$$\phi - \delta_{wr} + \delta_{rr} = 0 \qquad (4)$$

$$u_y - y_0 - r_0\phi + \eta_{wl} - \eta_{rl} = 0 \qquad (5)$$

$$u_z - l\phi + \zeta_{wl} - \zeta_{rl} = 0 \qquad (6)$$

$$\phi + \delta_{wl} - \delta_{rl} = 0 \qquad (7)$$

$$\tan\delta_{wr} = d\zeta_{wr} / d\eta_{wr} \qquad (8)$$

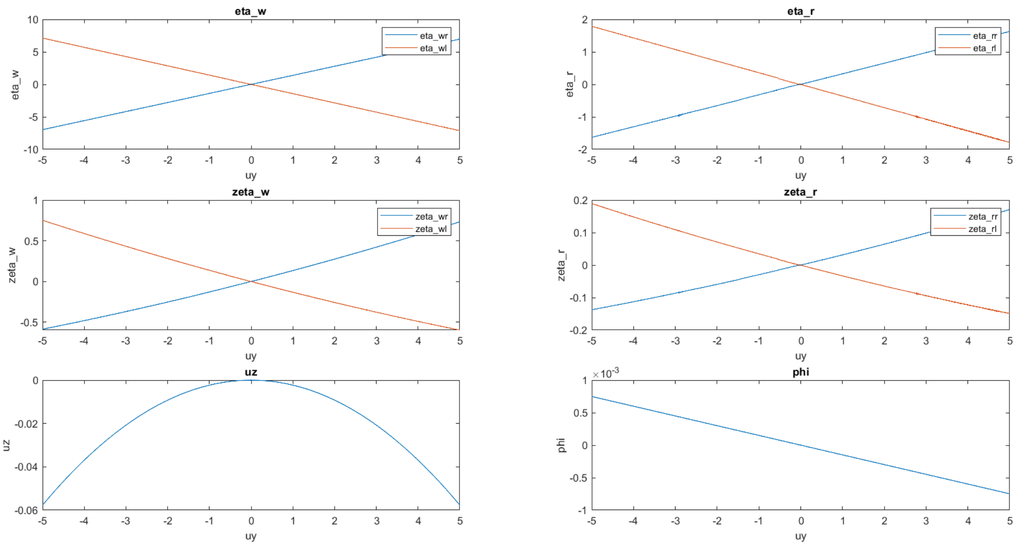
$$\tan\delta_{wl} = d\zeta_{wl} / d\eta_{wl} \qquad (9)$$

$$\tan\delta_{rr} = d\zeta_{rr} / d\eta_{rr} \qquad (10)$$

$$\tan\delta_{rl} = d\zeta_{rl} / d\eta_{rl} \qquad (11)$$

We solved these equations for η_r, η_w, ζ_r, ζ_w, ϕ, u_z (both left and right side) using the fsolve() routine in MATLAB by varying u_y from -5 to 5 with a step size of 0.00001.

4. Finally we plotted our results in MATLAB



Codes

1. main.m

This code uses all the other subroutines to carry out the procedure described above.

```
% Initial configuration of rail and wheel dimensions and shifts
L_rail = 1500/2; % Half-length of the rail (initial value adjusted by dividing
by 2)
L_wheel = 1350/2; % Half-length of the wheel (initial value adjusted by
dividing by 2)
y_shift_r = -172; % Vertical shift for the rail
y_shift_w = -420; % Vertical shift for the wheel

% Contact configuration setup
x_guess = 721.5; % Initial guess for the x-coordinate of the contact point
x_contact = newton(@contact_pt, x_guess, L_wheel, y_shift_w, L_rail,
y_shift_r); % Finding x-contact point using Newton's method
y_contact_wheel = rightwheel(x_contact, L_wheel, y_shift_w); % y-coordinate of
wheel contact
y_contact_rail = rightrail(x_contact, L_rail, y_shift_r); % y-coordinate of
rail contact
% Shifting axis to align with the z-coordinate of the contact point
y_shift_r_contact = -y_contact_rail + y_shift_r; % Adjusted rail vertical shift
y_shift_w_contact = -y_contact_wheel + y_shift_w; % Adjusted wheel vertical
shift

% Motion configuration - shifting the coordinate system for analysis
```

```matlab
% Defining the eta-zeta axis for both rail (stationary) and wheel (moving)
L_rail_eta_zeta = L_rail - x_contact; % Length of the rail in the new
coordinate system
L_wheel_eta_zeta = L_wheel - x_contact; % Length of the wheel in the new
coordinate system
% Now both profiles are considered to be at 0,0 in their respective coordinates
% Plot to check contact configuration after axis shift
figure
plot_profile(L_rail_eta_zeta, L_wheel_eta_zeta, y_shift_r_contact,
y_shift_w_contact);
hold off

% Initializing data storage and guesses for iterative solution process
data = [];
guess1 = [0,0,0,0,0,0,0,0,0,0]; % Initial guess for the solver
guess2 = [0,0,0,0,0,0,0,0,0,0]; % Second guess for the solver
% Loop to solve the system for positive displacement (uy from 0 to 5)
for uy = 0:0.00001:5
   % Solving the system for current uy value using iterative guesses
   [solutions, fval] = solve_system(L_wheel_eta_zeta, y_shift_w_contact,
L_rail_eta_zeta, y_shift_r_contact, uy, guess2 + (guess2 - guess1)/2);
   guess1 = guess2; % Update previous guess
   guess2 = solutions; % Update current guess
   data = [data ; uy, solutions]; % Store the results
   disp(uy); % Display current uy value
end

% Data storage and guesses for solving negative displacement
data_neg = [];
guess1 = [0,0,0,0,0,0,0,0,0,0]; % Reset initial guess for the solver
guess2 = [0,0,0,0,0,0,0,0,0,0]; % Reset second guess for the solver
% Loop to solve the system for negative displacement (uy from 0 to -5)
for uy = 0:-0.00001:-5
   % Solving the system for current uy value using iterative guesses
   [solutions, fval] = solve_system(L_wheel_eta_zeta, y_shift_w_contact,
L_rail_eta_zeta, y_shift_r_contact, uy, guess2 + (guess2 - guess1)/2);
   guess1 = guess2; % Update previous guess
   guess2 = solutions; % Update current guess
   data_neg = [uy, solutions; data_neg]; % Store the results in reverse order
   disp(uy); % Display current uy value
end

% Combine positive and negative data sets
data = [data_neg; data];
% Extracting x-axis data (uy values)
x = data(:, 1);

% Extracting individual columns for plotting
eta_wr = data(:, 2);
```

```
eta_rr = data(:, 3);
eta_wl = data(:, 4);
eta_rl = data(:, 5);
zeta_wr = data(:, 6);
zeta_rr = data(:, 7);
zeta_wl = data(:, 8);
zeta_rl = data(:, 9);
uz = data(:, 10);
phi = data(:, 11);
% Plot 1: eta_w (wheel eta values)
figure;
subplot(3, 2, 1);
plot(x, eta_wr, x, eta_wl);
title('eta\_w');
xlabel('uy');
ylabel('eta\_w');
legend('eta\_wr','eta\_wl')
% Plot 2: eta_r (rail eta values)
subplot(3, 2, 2);
plot(x, eta_rr, x, eta_rl);
title('eta\_r');
xlabel('uy');
ylabel('eta\_r');
legend('eta\_rr','eta\_rl')
% Plot 3: zeta_w (wheel zeta values)
subplot(3, 2, 3);
plot(x, zeta_wr, x, zeta_wl);
title('zeta\_w');
xlabel('uy');
ylabel('zeta\_w');
legend('zeta\_wr','zeta\_wl')
% Plot 4: zeta_r (rail zeta values)
subplot(3, 2, 4);
plot(x, zeta_rr, x, zeta_rl);
title('zeta\_r');
xlabel('uy');
ylabel('zeta\_r');
legend('zeta\_rr','zeta\_rl')
% Plot 5: uz (displacement uz values)
subplot(3, 2, 5);
plot(x, uz);
title('uz');
xlabel('uy');
ylabel('uz');
% Plot 6: phi (rotation phi values)
subplot(3, 2, 6);
plot(x, phi);
title('phi');
xlabel('uy');
```

```
ylabel('phi');
```

2. rightwheel.m

This code stores the wheel profile

```
function [ywr] = rightwheel(xwr, L, y_shift)

% Function to calculate the y-coordinate of the wheel profile at a given
x-coordinate.
% xwr: x-coordinate of the wheel profile
% L: horizontalshift applied to align the rail profile
% y_shift: vertical shift applied to align the wheel profile

xa = L; ya = y_shift - 14; % Point A coordinates
xb = L + 27.96; yb = y_shift - 19.39; % Point B coordinates
xc = L + 30.98; yc = y_shift - 11.84; % Point C coordinates
xd = L + 40.48; yd = y_shift - 3.48; % Point D coordinates
xe = L + 52.17; ye = y_shift - 1.19; % Point E coordinates
xf = L + 74.52; yf = y_shift + 0.69; % Point F coordinates
xg = L + 127; yg = y_shift + 3.31; % Point G coordinates

% Determine which section of the wheel profile xwr falls into
if xwr < xb && xwr >= xa
   % Section between points A and B, circular arc
   [c1, c2, c3, c4] = findCircleCenters(xa, ya, xb, yb, 14.5); % Find circle
centers for the arc
   ywr = c2 - sqrt((14.5^2) - (xwr - c1)^2); % Calculate y-coordinate using
circle equation
elseif xwr < xc && xwr >= xb
   % Section between points B and C, straight line
   ywr = yb + (yc - yb) * (xwr - xb) / (xc - xb); % Linear interpolation
between points B and C
elseif xwr < xd && xwr >= xc
   % Section between points C and D, circular arc
   [c1, c2, c3, c4] = findCircleCenters(xc, yc, xd, yd, 14); % Find circle
centers for the arc
   ywr = c4 + sqrt((14^2) - (xwr - c3)^2); % Calculate y-coordinate using
circle equation
elseif xwr < xe && xwr >= xd
   % Section between points D and E, circular arc
   [c1, c2, c3, c4] = findCircleCenters(xd, yd, xe, ye, 100); % Find circle
centers for the arc
   ywr = c4 + sqrt((100^2) - (xwr - c3)^2); % Calculate y-coordinate using
circle equation
elseif xwr < xf && xwr >= xe
   % Section between points E and F, circular arc
   [c1, c2, c3, c4] = findCircleCenters(xe, ye, xf, yf, 330); % Find circle
centers for the arc
```

```
    ywr = c4 + sqrt((330^2) - (xwr - c3)^2); % Calculate y-coordinate using
circle equation
elseif xwr <= xg && xwr >= xf
    % Section between points F and G, straight line
    ywr = yf + (yg - yf) * (xwr - xf) / (xg - xf); % Linear interpolation
between points F and G
else
    % If xwr is outside the defined profile, assume derailment
    ywr = 0;
    disp('wheel derailed'); % Display message indicating derailment
end
end
```

3. rightrail.m

Similar to rightwheel.m

```
function yrr = rightrail(xrr, L, y_shift)

% Function to calculate the y-coordinate of the right rail profile at a given
x-coordinate.
% yrr: y-coordinate of the rail profile
% xrr: x-coordinate of the rail profile
% L: horizontal shift applied to align the rail profile
% y_shift: vertical shift applied to align the rail profile

xa = L - 37.18; ya = 172 - 37.48 + y_shift; % Point A coordinates
xb = L - 36.02; yb = 172 - 14.30 + y_shift; % Point B coordinates
xc = L - 26.05; yc = 172 - 2.3 + y_shift; % Point C coordinates
xd = L - 10.25; yd = 172 - 0.18 + y_shift; % Point D coordinates
xg = L + 37.18; yg = 172 - 37.48 + y_shift; % Point G coordinates
xi = L + 36.02; yi = 172 - 14.30 + y_shift; % Point I coordinates
xf = L + 26.05; yf = 172 - 2.3 + y_shift; % Point F coordinates
xe = L + 10.25; ye = 172 - 0.18 + y_shift; % Point E coordinates

% Determine which section of the rail profile xrr falls into
if xrr <= xb && xrr >= xa
    % Section between points A and B, straight line
    yrr = ya + (yb - ya) * (xrr - xa) / (xb - xa); % Linear interpolation
between points A and B
elseif xrr <= xc && xrr > xb
    % Section between points B and C, circular arc
    [c1, c2, c3, c4] = findCircleCenters(xb, yb, xc, yc, 13); % Find circle
centers for the arc
    yrr = c4 + sqrt((13^2) - (xrr - c3)^2); % Calculate y-coordinate using
circle equation
elseif xrr <= xd && xrr > xc
    % Section between points C and D, circular arc
```

```
    [c5, c6, c7, c8] = findCircleCenters(xd, yd, xc, yc, 80); % Find circle
centers for the arc
    yrr = c6 + sqrt((80^2) - (xrr - c5)^2); % Calculate y-coordinate using
circle equation
elseif xrr <= xe && xrr > xd
    % Section between points D and E, circular arc
    [c5, c6, c7, c8] = findCircleCenters(xd, yd, xe, ye, 300); % Find circle
centers for the arc
    yrr = c8 + sqrt((300^2) - (xrr - c7)^2); % Calculate y-coordinate using
circle equation
elseif xrr <= xg && xrr >= xi
    % Section between points G and I, straight line
    yrr = yg + (yg - yi) * (xrr - xg) / (xg - xi); % Linear interpolation
between points G and I
elseif xrr <= xi && xrr > xf
    % Section between points I and F, circular arc
    [c1, c2, c3, c4] = findCircleCenters(xf, yf, xi, yi, 13); % Find circle
centers for the arc
    yrr = c4 + sqrt((13^2) - (xrr - c3)^2); % Calculate y-coordinate using
circle equation
elseif xrr <= xf && xrr > xe
    % Section between points F and E, circular arc
    [c5, c6, c7, c8] = findCircleCenters(xf, yf, xe, ye, 80); % Find circle
centers for the arc
    yrr = c6 + sqrt((80^2) - (xrr - c5)^2); % Calculate y-coordinate using
circle equation
else
    % If xrr is outside the defined profile, assume derailment
    yrr = 0;
    disp('rail derailed'); % Display message indicating derailment
end
end
```

4. findCircleCenters.m

Helper function for finding profiles

```
function [center1X, center1Y, center2X, center2Y] = findCircleCenters(x1, y1,
x2, y2, r)
    % This function finds the two possible centers of a circle given two points
on the circle and the radius.
    % Calculate the distance between the two points
    d = sqrt((x2 - x1)^2 + (y2 - y1)^2);
    % Check if the given radius is valid for the given distance
    if r < d / 2
        error('The radius is too small to form a circle with the given
points.');
    end
```

```
    % Calculate the midpoint of the line segment
    xm = (x1 + x2) / 2;
    ym = (y1 + y2) / 2;
    % Calculate the distance from the midpoint to the circle center
    h_distance = sqrt(r^2 - (d / 2)^2);
    % Find the direction vector perpendicular to the line segment
    dx = x2 - x1;
    dy = y2 - y1;
    % Perpendicular vector components
    perp_dx = -dy;
    perp_dy = dx;
    % Normalize the perpendicular vector
    norm = sqrt(perp_dx^2 + perp_dy^2);
    perp_dx = perp_dx / norm;
    perp_dy = perp_dy / norm;
    % Possible centers of the circle
    center1X = xm + h_distance * perp_dx;
    center1Y = ym + h_distance * perp_dy;
    center2X = xm - h_distance * perp_dx;
    center2Y = ym - h_distance * perp_dy;
end
```

5. diffr.m

Helper function for finding slope at a point in the profile

```
function dy_dx = diffr(f, x_val, L, y)
    % f is a function handle for the function to differentiate
    % x_val is the point at which to evaluate the derivative
    % Define a small perturbation
    h = 1e-6;   % Perturbation size

    % Compute the numerical derivative using finite difference
    dy_dx = (f(x_val + h, L, y) - f(x_val - h, L , y)) / (2 * h);
end
```

6. contact_pt.m

Function that defines the necessary condition for contact used in newton.m

```
function z = contact_pt(x_c, L1, y1, L2, y2)
z = diffr(@rightwheel,x_c, L1, y1)-diffr(@rightrail,x_c, L2, y2);
z= norm(z);
end
```

7. newton.m

Standard Newton Raphson procedure

```
function q = newton(fun, q0, L1, y1, L2, y2)

% Function to perform Newton-Raphson iteration for finding the root of a
nonlinear function

f0 = feval(fun, q0, L1, y1, L2, y2);
count = 0; % Initialize iteration counter
h = 1e-6; % Step size for numerical differentiation
n = length(q0); % Number of unknowns
e = eye(n) * h; % Small perturbation matrix for finite difference approximation
D = e; % Initial Jacobian matrix

% Iterate until the function value is sufficiently small or max iterations are
reached
while ((norm(f0) > 1e-6) * (count < 6000))
   % Compute the Jacobian matrix using finite differences
   for k = 1:n
       % Approximate the derivative of the function with respect to each
variable
       D(:, k) = (feval(fun, q0 + e(:, k), L1, y1, L2, y2) - f0) / h;
   end

   % Update the guess using Newton-Raphson step
   q0 = q0 - D \ f0;

   % Evaluate the function at the new guess
   f0 = feval(fun, q0, L1, y1, L2, y2);

   % Increment the iteration counter
   count = count + 1;
end
% Check if the iteration limit was reached without convergence
if count == 6000
   q = q0 / 0; % Return NaN to indicate failure to converge
else
   q = q0; % Return the converged solution
end
```

8. plot_profile.m

Used for plotting the two profiles

```
function data=plot_profile(L1,L2,y1,y2)

% Rail profile plot
data = [];
for i = (L1 - 37.18):0.1: (L1 + 37.18)
```

```
    y = rightrail(i, L1, y1);
    data = [data ; i, y];
end
plot(data(:,1),data(:,2),'-')
hold on

% Wheel profile plot
data = [];
for i = L2 : 0.1 : L2 + 127
    y = rightwheel(i, L2, y2);
    data = [data ; i, y];
end
plot(data(:,1),data(:,2),'-')
end
```

9. solve_system.m

Used in main.m for solving the 10 non linear equations using fsolve()

```
function [solution, fval] = solve_system(L_wheel_eta_zeta, y_shift_w_contact,
L_rail_eta_zeta, y_shift_r_contact, uy, guess)

% Function to solve the system of equations related to wheel and rail
interaction
% Inputs:
%   L_wheel_eta_zeta: Horizontal shift applied to the wheel profile
%   y_shift_w_contact: Vertical shift applied to the wheel profile
%   L_rail_eta_zeta: Horizontal shift applied to the rail profile
%   y_shift_r_contact: Vertical shift applied to the rail profile
%   uy: Lateral displacement input
%   guess: Initial guess vector for the solver
% Outputs:
%   solution: Vector of solved variables
%   fval: Function values at the solution

    y0 = 0; % Initial y-coordinate reference
    r0 = 450; % Radius of curvature for contact calculation
    l = 1350/2; % Half the wheelbase length

    % Define the system of equations as nested function
    function F = equations(x)

        % Define the unknowns
        eta_wr = x(1); % Eta coordinate of the right wheel
        eta_rr = x(2); % Eta coordinate of the right rail
        eta_wl = x(3); % Eta coordinate of the left wheel
        eta_rl = x(4); % Eta coordinate of the left rail
        zeta_wr = x(5); % Zeta coordinate of the right wheel
```

```
        zeta_rr = x(6); % Zeta coordinate of the right rail
        zeta_wl = x(7); % Zeta coordinate of the left wheel
        zeta_rl = x(8); % Zeta coordinate of the left rail
        uz = x(9); % Vertical displacement
        phi = x(10); % Rotation angle

        % Define the equations based on the problem statement
        eq1 = uy - y0 - r0 * phi - eta_wr + eta_rr; % Lateral displacement
balance on the right side
        eq2 = uz + l * phi + zeta_wr - zeta_rr; % Vertical displacement balance
on the right side
        eq3 = phi - atan(diffr(@rightwheel, eta_wr, L_wheel_eta_zeta,
y_shift_w_contact)) + atan(diffr(@rightrail, eta_rr, L_rail_eta_zeta,
y_shift_r_contact)); % Rotation angle balance on the right side
        eq4 = uy - y0 - r0 * phi + eta_wl - eta_rl; % Lateral displacement
balance on the left side
        eq5 = uz - l * phi + zeta_wl - zeta_rl; % Vertical displacement balance
on the left side
        eq6 = phi + atan(diffr(@rightwheel, -eta_wl, L_wheel_eta_zeta,
y_shift_w_contact)) - atan(diffr(@rightrail, -eta_rl, L_rail_eta_zeta,
y_shift_r_contact)); % Rotation angle balance on the left side

        eq7 = zeta_wr - (-1) * rightwheel(-eta_wr, L_wheel_eta_zeta,
y_shift_w_contact); % Right wheel profile match
        eq8 = zeta_rr - (-1) * rightrail(-eta_rr, L_rail_eta_zeta,
y_shift_r_contact); % Right rail profile match
        eq9 = zeta_wl - (-1) * rightwheel(-eta_wl, L_wheel_eta_zeta,
y_shift_w_contact); % Left wheel profile match
        eq10 = zeta_rl - (-1) * rightrail(-eta_rl, L_rail_eta_zeta,
y_shift_r_contact); % Left rail profile match
        % Return the system of equations
        F = [eq1; eq2; eq3; eq4; eq5; eq6; eq7; eq8; eq9; eq10];
    end

    x0 = guess; % Starting guesses can be any reasonable value
    [solution, fval] = fsolve(@equations, x0); % Use fsolve to find the solution

end
```