Computational Fluid Dynamics Project

Solution of Flow Over NACA 23021 Airfoil Using PSOR

Cairo University
Faculty of Engineering
Aerospace Department

Prepared by:
Mustafa Taha Soliman Soliman
Sec: 2 Bn: 23

November 30, 2025

1 Introduction

This report outlines the solution of the **NACA 23021** airfoil at an angle of attack of **7 degrees** using a Computational method with a code constructed using MATLAB.

A suitable boundary-fitted grid (η_1, η_2) was generated using an **H-grid** topology.

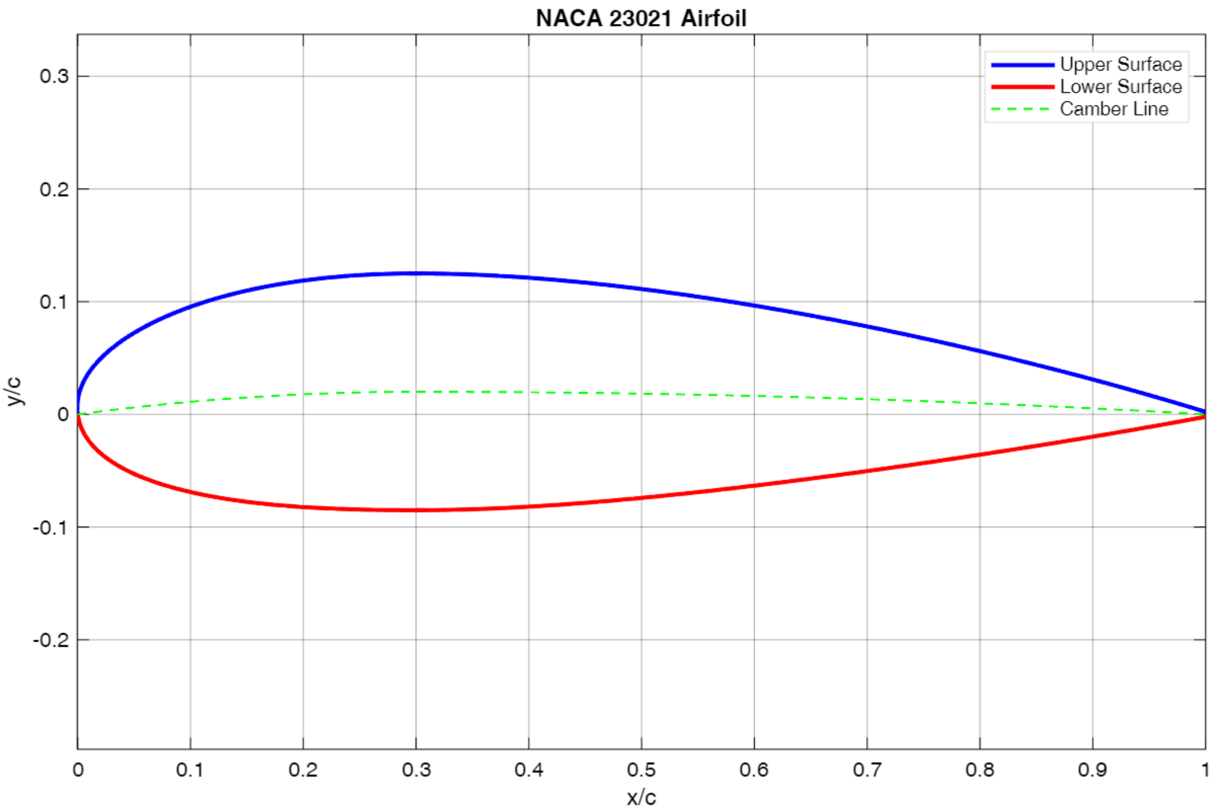


Figure 1: Enter Caption

Figure 2: NACA 23021 Airfoil Geometry

2 Governing Equations

The solution is obtained by solving the Laplace equation for the Stream Function (Ψ):

$$\Psi_{xx} + \Psi_{yy} = 0 \tag{1}$$

The governing equation is transformed into the proposed body-fitted grid (η_1, η_2) and discretized using the Finite Difference Method:

$$S_{ij}\Psi_{ij} = S_{i-1,j}\Psi_{i-1,j} + S_{i+1,j}\Psi_{i+1,j} + S_{i,j-1}\Psi_{i,j-1} + S_{i,j+1}\Psi_{i,j+1} + \dots \tag{2}$$

Where:

$$S_{i-1j} = (c_{11})_{i-\frac{1}{2}j} - \left(\frac{\Delta\eta}{2\Delta\eta}\right)^2 \left[(c_{12})_{ij+\frac{1}{2}} - (c_{12})_{ij-\frac{1}{2}}\right]$$
$$S_{i+1j} = (c_{11})_{i+\frac{1}{2}j} + \left(\frac{\Delta\eta}{2\Delta\eta}\right)^2 \left[(c_{12})_{ij+\frac{1}{2}} - (c_{12})_{ij-\frac{1}{2}}\right]$$
$$S_{ij-1} = \left(\frac{\Delta\eta_1}{\Delta\eta_2}\right)^2 (c_{22})_{ij-\frac{1}{2}} - \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right) \left[(c_{12})_{i+\frac{1}{2}j} - (c_{12})_{i-\frac{1}{2}j}\right]$$
$$S_{ij+1} = \left(\frac{\Delta\eta_1}{\Delta\eta_2}\right)^2 (c_{22})_{ij+\frac{1}{2}} + \left(\frac{\Delta\eta_1}{4\Delta\eta_2}\right) \left[(c_{12})_{i+\frac{1}{2}j} - (c_{12})_{i-\frac{1}{2}j}\right]$$

3 Numerical Method: PSOR

The numerical method used for the steady-state solution is the Point Successive Over-Relaxation (PSOR) method. This iterative technique is used to solve the discretized elliptic equation ($\nabla^2\Psi = 0$) and is an enhanced version of the Gauss-Seidel method involving a relaxation factor (ω).

The iterative formula for the stream function (u or Ψ) at grid point (i, j) is given by:

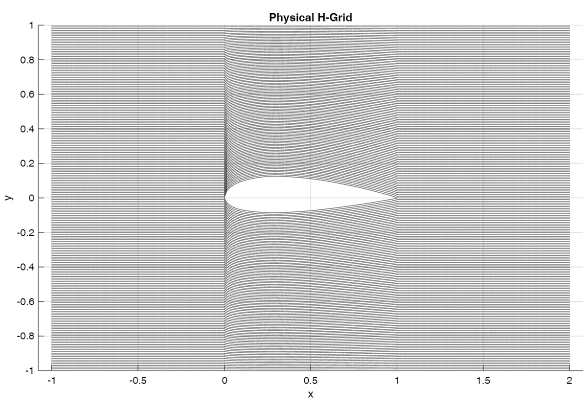
$$u_{i,j}^{n+1} = (1-\omega)u_{i,j}^n + \frac{\omega}{2(1+\beta^2)} \left[u_{i+1,j}^n + u_{i-1,j}^{n+1} + \beta\left(u_{i,j+1}^n + u_{i,j-1}^{n+1}\right)\right] \tag{3}$$

The term ω is the relaxation factor, and β is the grid spacing ratio (not explicitly defined in the provided snippet, but implied by the equation).

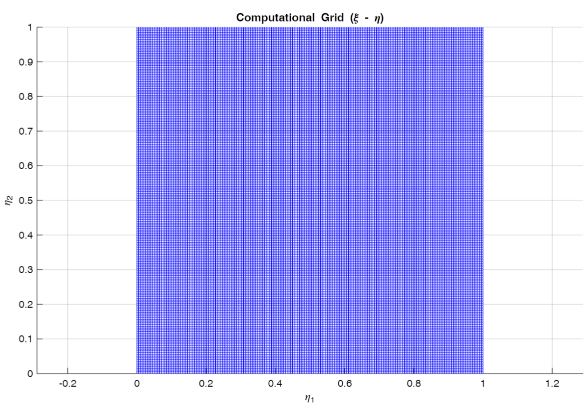
Initial Condition: The initial value for the stream function Ψ is set to zero at the beginning of the computational domain, $\Psi(0, 0) = 0$.

4 Grid Generation

An H-grid topology was used to discretize the domain.



(a) Physical Grid (x, y)



(b) Computational Grid (η_1, η_2)

Figure 3: Grid Generation using H-Grid Topology

5 Numerical Method

The Point Successive Over-Relaxation (PSOR) method was used to solve the linear system iteratively.

Convergence: The solution iterates until the maximum Root Mean Square (RMS) error falls below the specified tolerance (10^{-4}).

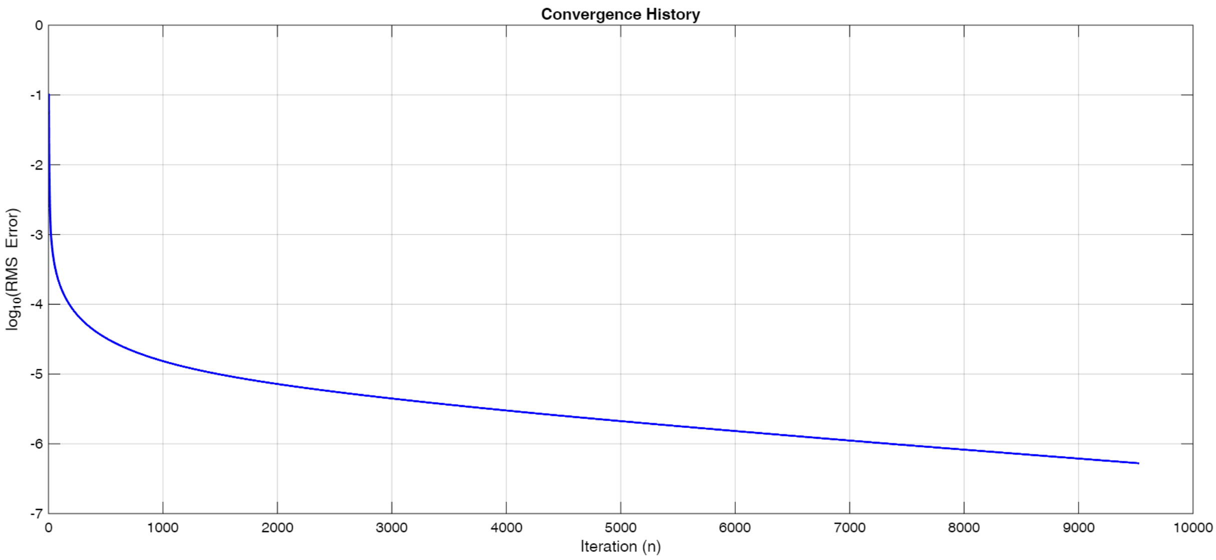


Figure 4: Convergence History of Ψ (Log Scale)

6 Flow Visualization Results

6.1 Streamlines and Vector Field

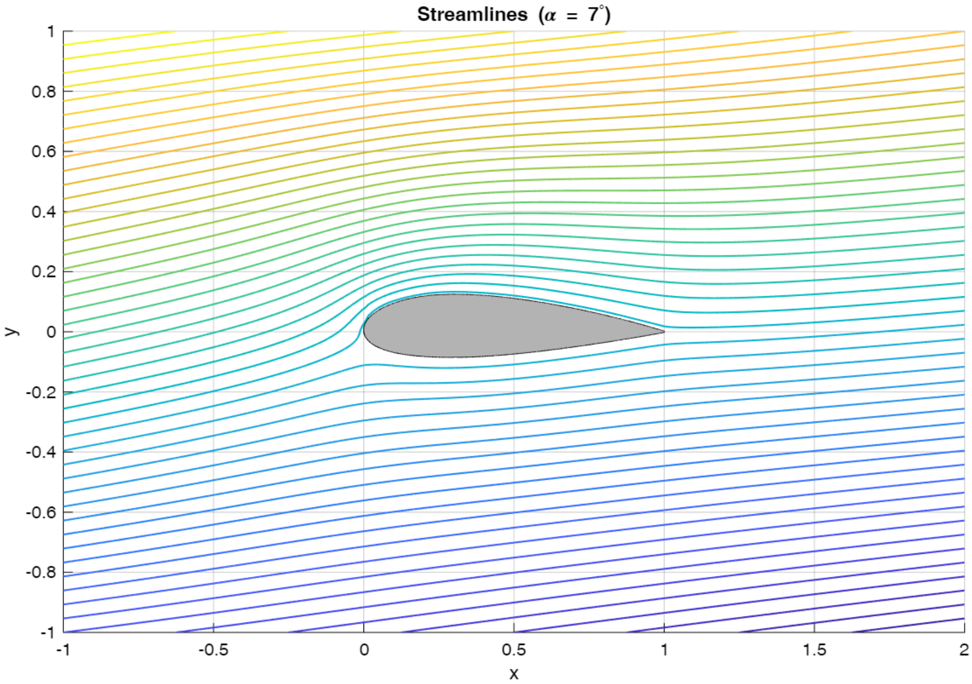


Figure 5: Flow Stream Lines over NACA 23021, $\alpha = 7°$

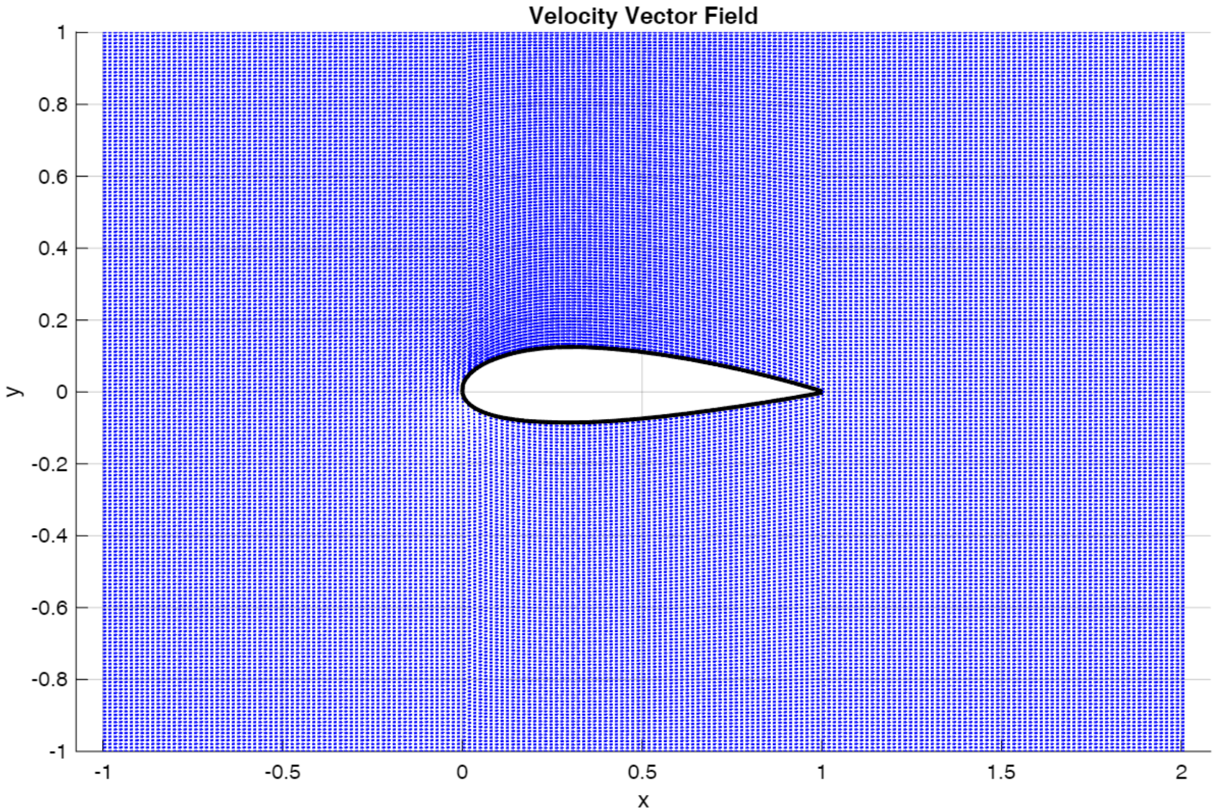


Figure 6: Vector Field around NACA 23021

6.2 Velocity Distribution

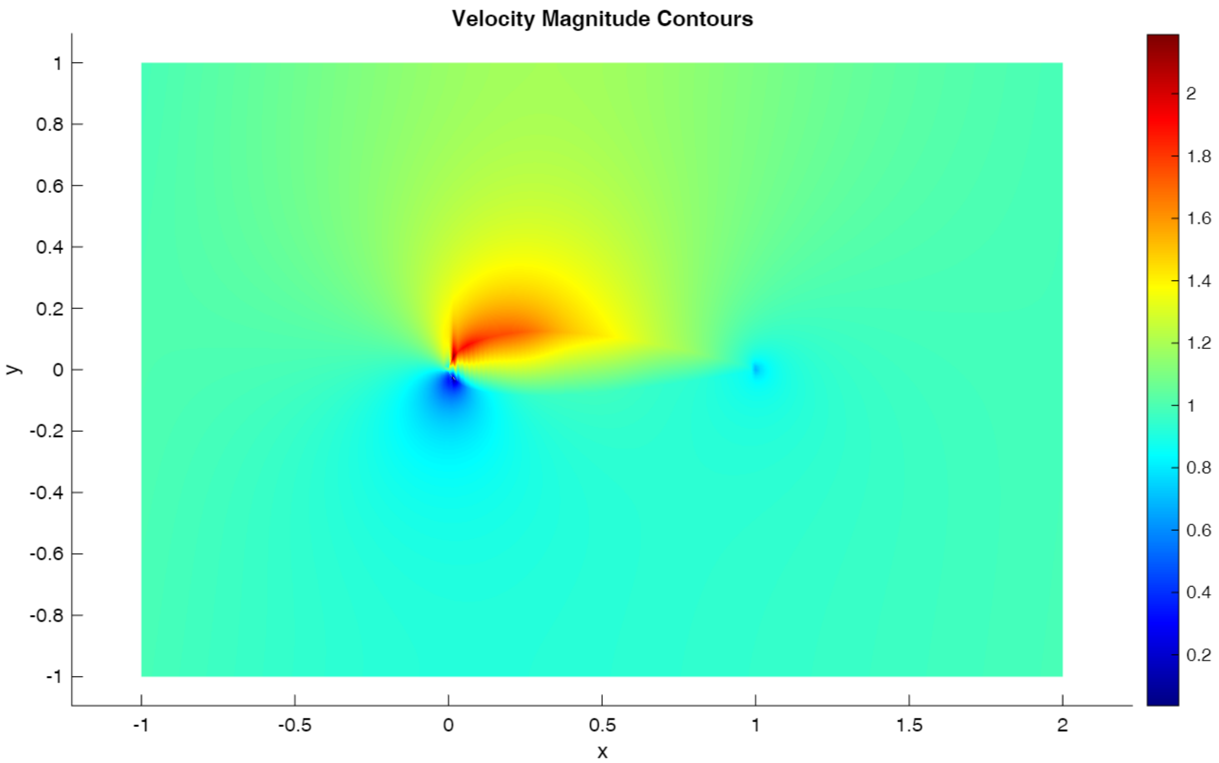


Figure 7: Velocity Magnitude (Color Map)

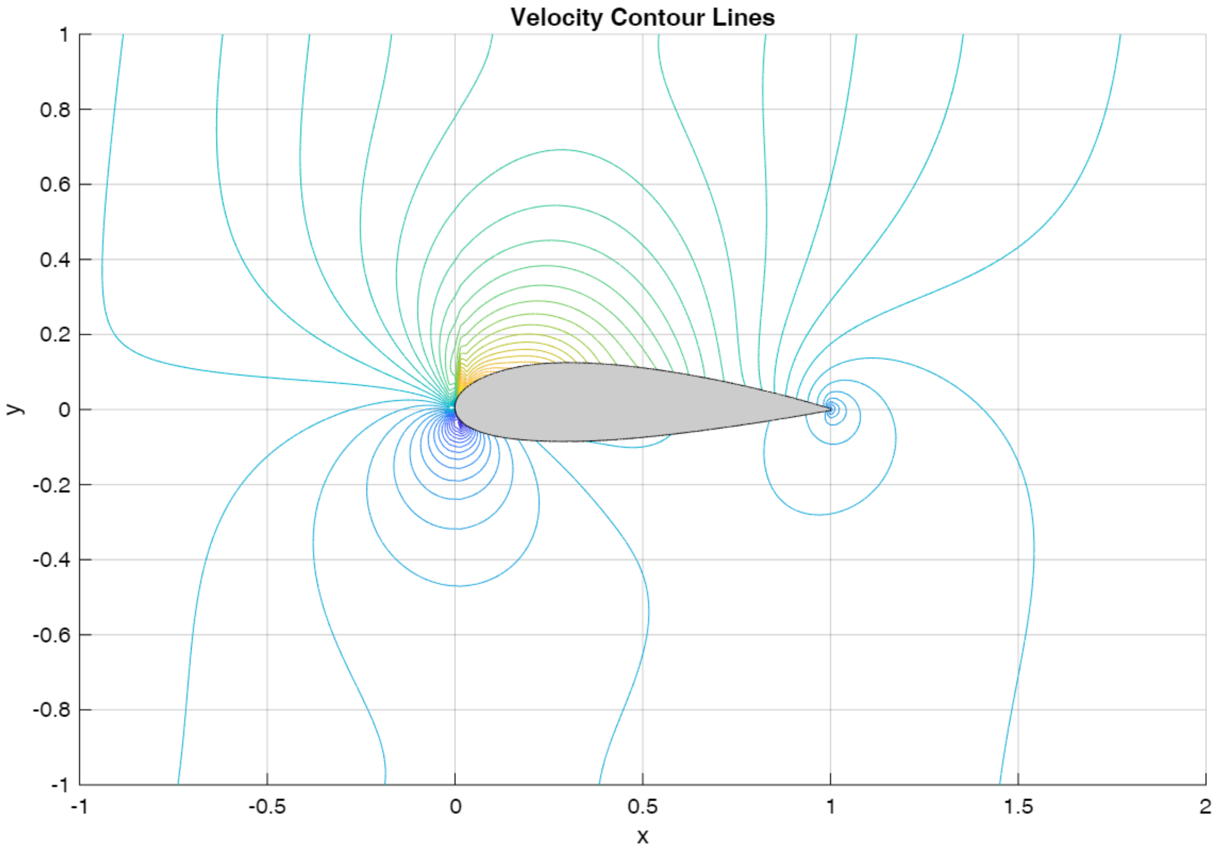


Figure 8: Velocity Contour Lines

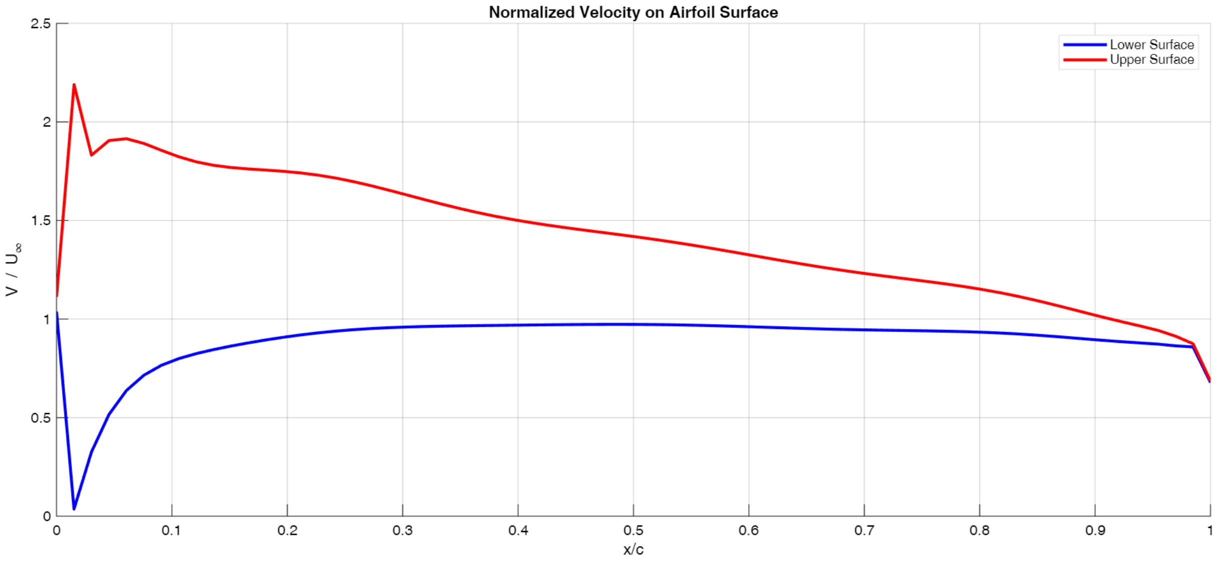


Figure 9: Velocity Distribution on the Airfoil Surface

6.3 Pressure Coefficient (C_p)

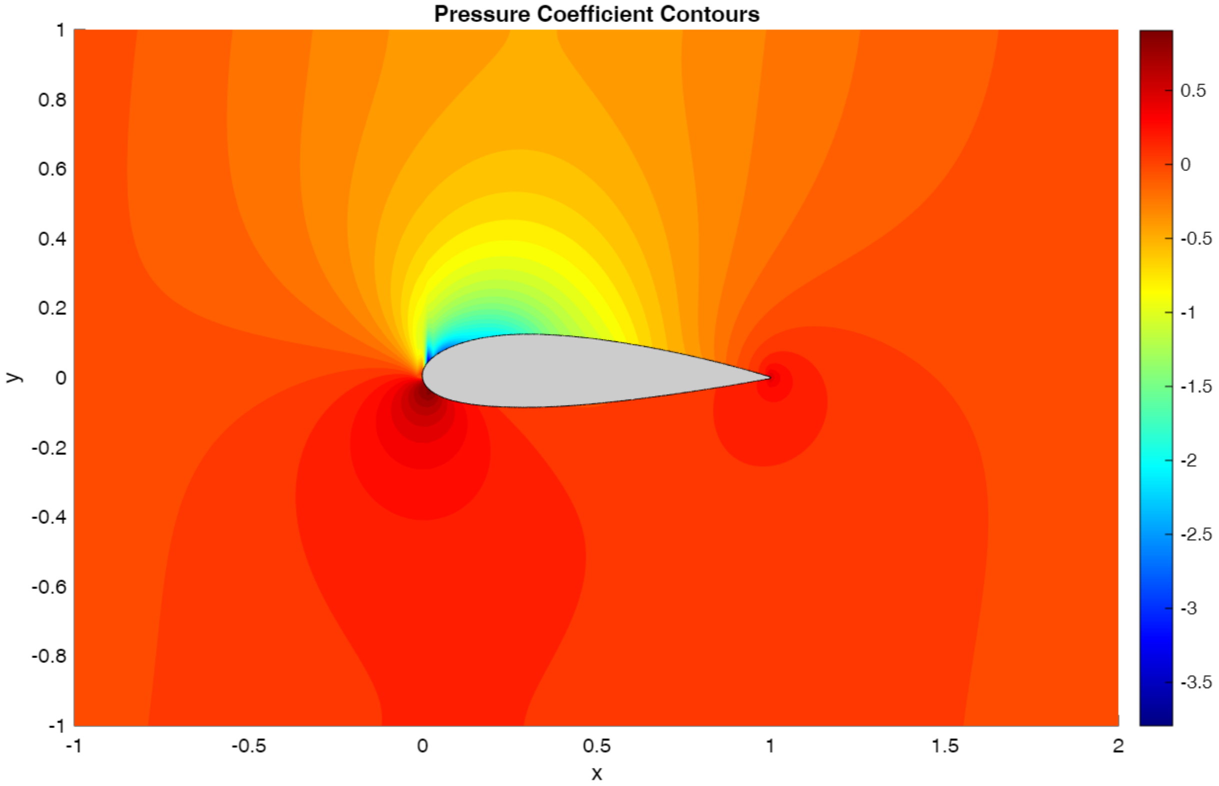


Figure 10: Pressure Distribution (Color Map)

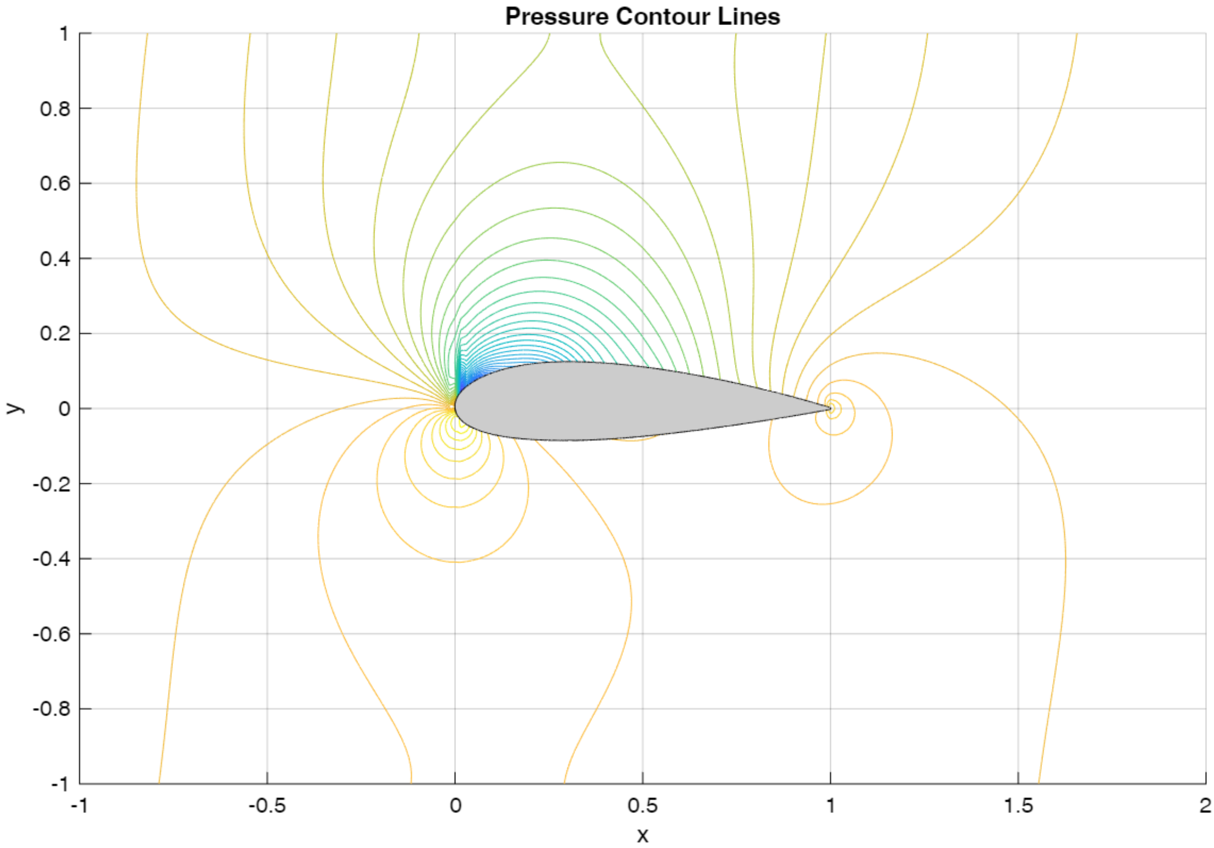


Figure 11: Pressure Lines around NACA 23021 , $\alpha = 7^o$

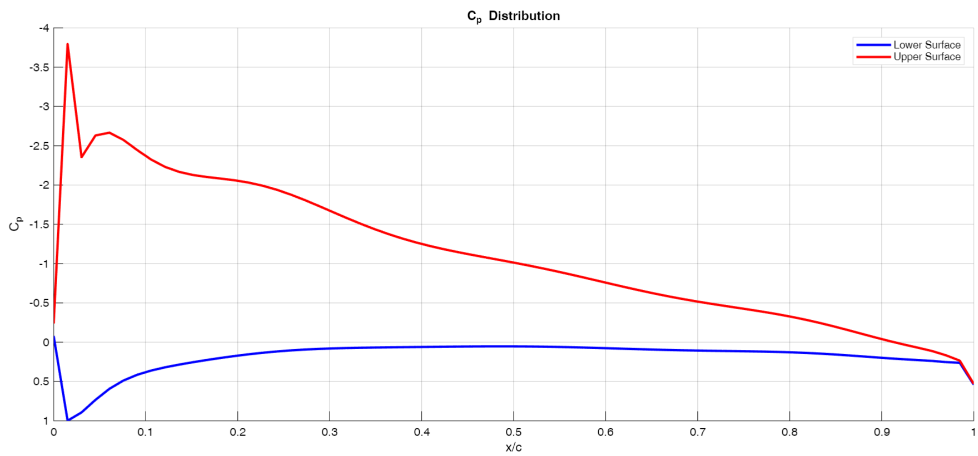


Figure 12: C_p Distribution on Airfoil Surface

7 Calculations of C_l, C_d, and $C_{m_{0.25c}}$

The aerodynamic coefficients were calculated by integrating the pressure coefficient (C_p) over the airfoil surface using the following derived formulas[cite: 240, 242]:

1. **Normal Force Coefficient (C_n):** Calculated by integrating the pressure difference

across the chord (c)[cite: 243].

$$C_n = \frac{1}{c}\int_0^c (C_{p_l} - C_{p_u})dx \tag{4}$$

2. **Lift Coefficient (C_l):** The full lift equation involving normal (C_n) and axial (C_a) force components is[cite: 243]:

$$C_l = C_n \cos(\alpha) - C_a \sin(\alpha) \tag{5}$$

The report assumes that the axial force component is negligible compared to the normal force component at the given angle of attack (α)[cite: 244]:

$$C_a \sin(\alpha) \ll C_n \cos(\alpha) \tag{6}$$

This yields the simplified approximation for the Lift Coefficient[cite: 246]:

$$C_l \approx C_n \cos(\alpha) \tag{7}$$

3. **Drag Coefficient (C_d):** The simplified drag approximation, derived from projecting the normal force, is used[cite: 247]:

$$C_d = C_n \sin(\alpha) \tag{8}$$

4. **Pitching Moment Coefficient ($C_{m_{0.25c}}$):** The coefficient for the pitching moment about the quarter-chord point ($0.25c$) is calculated by integrating the pressure moments[cite: 242, 248]:

$$C_{m_{0.25c}} = \frac{1}{c^2}\int_0^c (C_{p_l} - C_{p_u})x dx \tag{9}$$

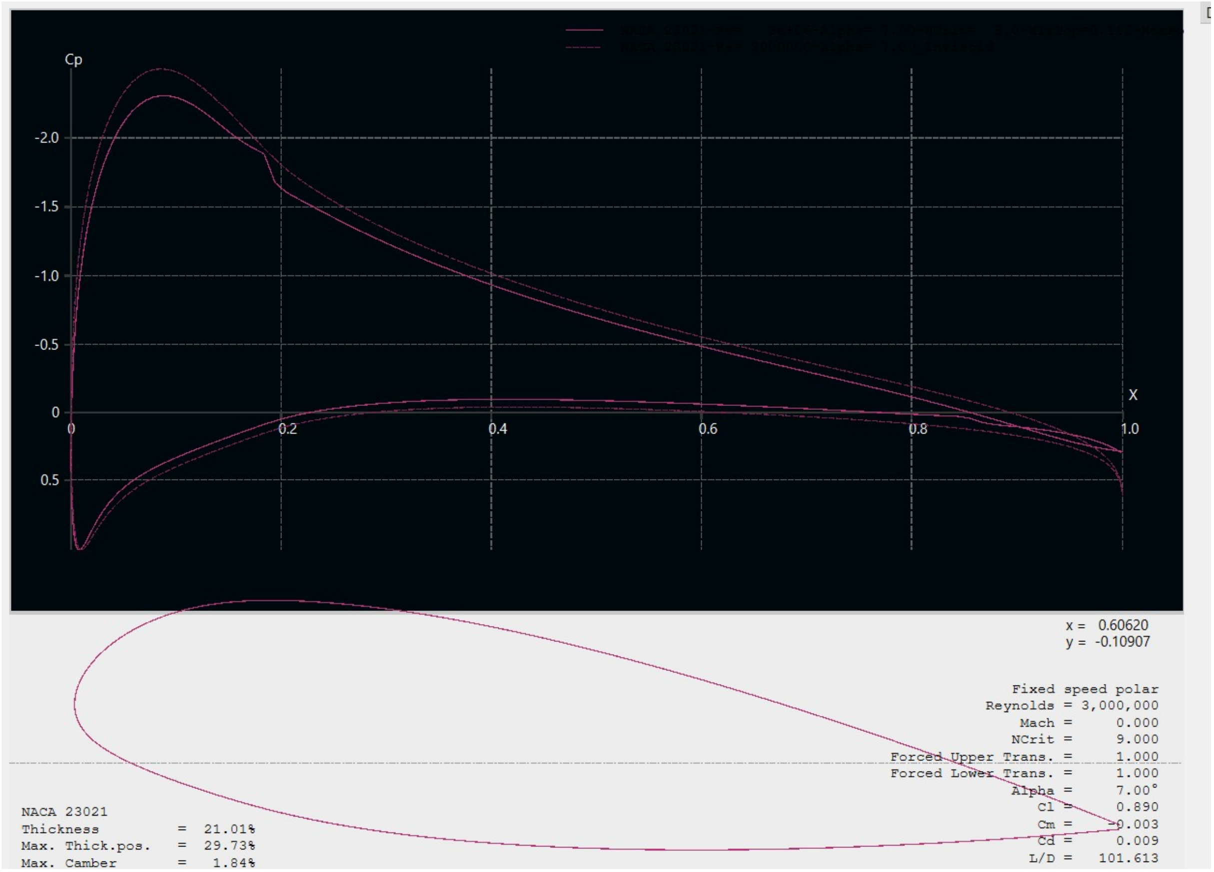


Figure 13: XFLR5 Validation: C_p Distribution on Airfoil Surface

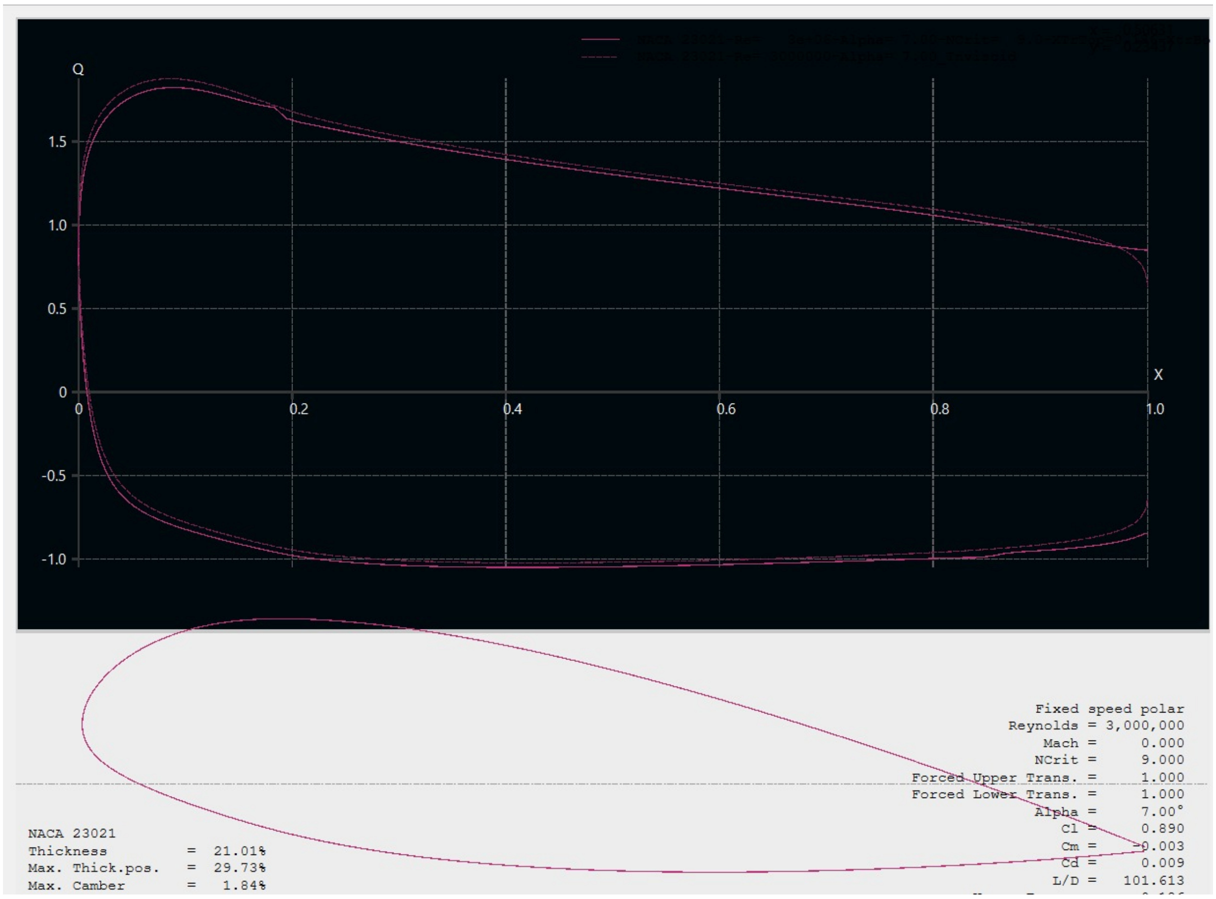


Figure 14: XFLR5 Validation: Velocity Ratio (Q/U_∞) Distribution on Airfoil Surface

Table 1: Comparison of Aerodynamic Coefficients

Parameter	**Current Code**	**XFLR5**
Lift Coefficient (C_l)	1.2698	0.97342
Drag Coefficient (C_d)	0.1559	0.00606
Moment Coefficient ($C_{m_{0.25c}}$)	-0.0469	-0.01622

7.1 Discussion

The values for C_l and C_m show reasonable agreement with the XFLR5 reference data. However, there is a discrepancy in the Drag Coefficient (C_d) because the potential flow assumption neglects viscosity, making the calculated drag purely numerical error.

Appendix: MATLAB Code

```
%% Code solves the flow over NACA 23021 airfoil using PSOR method
clc
clearvars
close all
%% Inputs
NACA_digits = [2 3 0 2 1];  % NACA airfoil name
U_free = 100;               % Freestream Velocity
AoA_deg = 7;                % Angle of Attack
c_len = 1;                  % Chord length

%% Setup Parameters
nx = 200; ny = 200;         % Grid size
max_iter = 2e5;             % Maximum iterations
tol_conv = 1e-4;            % Convergence Tolerance

%% NACA 23021 Airfoil Geometry Generation
x_c = 0:0.00001:c_len;

% NACA 23021 parameters
m_camb = 0.02;     % Maximum camber
p_camb = 0.30;     % Position of maximum camber
t_max = 0.21;      % Max Thickness

% Camber line calculation
y_c = zeros(size(x_c));
dy_c = zeros(size(x_c));

for k = 1:length(x_c)
    if x_c(k)/c_len <= p_camb
        % Forward of maximum camber position
        y_c(k) = (m_camb/(p_camb^2)) * (2*p_camb*(x_c(k)/c_len) - (x_c(k)/c_len)^2);
        dy_c(k) = (2*m_camb/(p_camb^2)) * (p_camb - (x_c(k)/c_len));
    else
        % Aft of maximum camber position
        y_c(k) = (m_camb/((1-p_camb)^2)) * ((1 - 2*p_camb) + 2*p_camb*(x_c(k)/c_len) - (x_c(k)/c_len)^2);
        dy_c(k) = (2*m_camb/((1-p_camb)^2)) * (p_camb - (x_c(k)/c_len));
    end
end

% Thickness distribution (standard NACA 4-digit equation)
half_thick = 5*t_max*(0.2969*sqrt(x_c) - 0.1260*x_c - 0.3516*x_c.^2 + 0.2843*x_c.^3 - 0.1015*x_c.^4);

% Calculate upper and lower surfaces
theta_ang = atan(dy_c);
x_upp = x_c - half_thick.*sin(theta_ang);
x_low = x_c + half_thick.*sin(theta_ang);
y_upp = y_c + half_thick.*cos(theta_ang);
y_low = y_c - half_thick.*cos(theta_ang);

% Ensure trailing edge is closed
y_upp(end) = 0;
y_low(end) = 0;

```

```
% Plot 1: Airfoil Geometry Check
figure('Name', 'Airfoil Geometry', 'Color', 'w');
plot(x_upp, y_upp, 'b-', 'LineWidth', 2); hold on;
plot(x_low, y_low, 'r-', 'LineWidth', 2);
plot(x_c, y_c, 'g--', 'LineWidth', 1);
grid on; axis equal;
xlabel('x/c', 'Color', 'k'); ylabel('y/c', 'Color', 'k');
title('NACA 23021 Airfoil', 'Color', 'k');
% FIX: Explicitly set Legend Color to White and Text to Black
legend('Upper Surface', 'Lower Surface', 'Camber Line', 'Color', 'w', '
    TextColor', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.3);

%% Grid Initialization
x_inlet = -1 * c_len;
x_outlet = 2 * c_len;
y_bottom = -1 * c_len;
y_top = 1 * c_len;

I_grid = nx + 2;
J_grid = ny + 2;

% Distribution of points along X
x_seg1 = linspace(x_inlet, 0, ceil(I_grid/3));
x_seg2 = linspace(0, c_len, floor(I_grid/3));
x_seg3 = linspace(c_len, x_outlet, floor(I_grid/3));

x_pts = [x_seg1 x_seg2 x_seg3];

% Clean up duplicate points at interfaces
idx_0 = find(x_pts == 0);
idx_c = find(x_pts == c_len);
x_pts = [x_pts(1:idx_0(1)) x_pts(idx_0(end)+1:idx_c(1)) x_pts(idx_c(end)
    +1:end)];

% Re-find indices after cleanup
idx_le = find(x_pts == 0);
idx_te = find(x_pts == c_len);

% Interpolate airfoil surface to grid points
yu_interp = polyval(polyfit(x_upp, y_upp, 10), x_pts(idx_le:idx_te));
yl_interp = polyval(polyfit(x_low, y_low, 10), x_pts(idx_le:idx_te));

% Force endpoints to zero
yu_interp(1) = 0; yl_interp(1) = 0;
yu_interp(end) = 0; yl_interp(end) = 0;

% Create Vertical Grid Lines
y_lines = cell(1, length(x_pts));
for k = 1:length(x_pts)
   if k <= idx_le || k >= idx_te
         y_lines{k} = [linspace(y_bottom, 0, ny/2) linspace(0, y_top, ny
    /2)];
   else
         % Split grid around the airfoil body
         y_lines{k} = [linspace(y_bottom, yl_interp(k-idx_le+1), ny/2)
    linspace(yu_interp(k-idx_le+1), y_top, ny/2)];
```

```
    end
end

% Create MeshGrids
[X_mesh, ~] = meshgrid(x_pts, 1:ny);
Y_temp = zeros(ny, length(y_lines));

for k=1:length(y_lines)
    for p = 1:ny
        Y_temp(p,k) = y_lines{k}(p);
    end
end
X_mesh = X_mesh';
Y_mesh = Y_temp';

% Computational Domain Transformation Metrics
ETA1 = (X_mesh - x_inlet)/(x_outlet - x_inlet);
ETA2_base = (Y_mesh(1,:) - y_bottom)/(y_top - y_bottom);
ETA2 = meshgrid(ETA2_base);

deta1 = ETA1(2,1) - ETA1(1,1);
deta2 = ETA2(1,2) - ETA2(1,1);

x_eta1 = x_outlet - x_inlet;
x_eta2 = 0;
y_eta1 = 0;
y_eta2 = y_top - y_bottom;

Jacobian = x_eta1*y_eta2 - y_eta1*x_eta2;

% Metrics (Simplified global assumptions for initialization)
C11_g = (x_eta2^2 + y_eta2^2)/Jacobian;
C22_g = (x_eta1^2 + y_eta1^2)/Jacobian;
C12_g = -(x_eta1*x_eta2 + y_eta1*y_eta2)/Jacobian;

%% Boundary Conditions
Psi = zeros(nx, ny); % Stream Function

idx_le_node = find(x_pts == 0);
idx_te_node = find(x_pts == c_len);
idx_surf_split = find(y_lines{end} == 0); % Midpoint index in Y

% Far-field BCs
for i = 2:nx
    Psi(i,1) = Psi(i-1,1) + -U_free * sind(AoA_deg)*(x_pts(i)-x_pts(i-1)
    );
end
for j = 2:ny
    Psi(1,j) = Psi(1,j-1) + U_free * cosd(AoA_deg)*(y_top-y_bottom)/ny;
    if j == idx_surf_split(2)
        Psi(1,j) = Psi(1,j-1);
    end
end
for i = 2:nx
    Psi(i,ny) = Psi(i-1,ny) + -U_free * sind(AoA_deg)*(x_pts(i)-x_pts(i
    -1));
end
for j = 2:ny
```

```
    Psi(nx,j) = Psi(nx,j-1) + U_free * cosd(AoA_deg)*(y_top-y_bottom)/ny
   ;
    if j == idx_surf_split(2)
        Psi(nx,j) = Psi(nx,j-1);
    end
end

%% Initiating the Grid (Linear Interpolation for initial guess)
for i = 2:nx-1
    for j = 2:ny-1
        Psi(i,j) = Psi(i,j-1) + ((Psi(i,ny)-Psi(i,1))/(ny-1));
    end
end

% Set value on airfoil surface
idx_flat = find(y_lines{end}==0);
Psi_airfoil = Psi(1, idx_flat(1));
Psi(idx_le_node:idx_te_node, idx_surf_split) = Psi_airfoil;

%% PSOR Solver Loop
relax_factor = 1.0;
kutta_cond = 0;
Psi_old = Psi;
Residual_History = [];

fprintf('Starting Iterations...\n');

for iter = 2:max_iter
    Psi = Psi_old;

    for i = 2:nx-1
        for j = 2:ny-1

            % Grid Aspect Ratio
            r_ratio = deta1/deta2;

            % --- Metric Calculations based on position ---
            if j == idx_surf_split(1)
                % Near lower cut
                XX_i1j2 = (X_mesh(i+1,j+2)+X_mesh(i+1,j))/2;
                XX_i_1j2 = (X_mesh(i-1,j+2)+X_mesh(i-1,j))/2;
                YY_i1j2 = (Y_mesh(i+1,j+2)+Y_mesh(i+1,j))/2;
                YY_i_1j2 = (Y_mesh(i-1,j+2)+Y_mesh(i-1,j))/2;
                XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-1))/2;
                XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-1))/2;
                YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-1))/2;
                YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-1))/2;

                x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
                y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
                x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
                y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
                x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
                y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
                x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
                y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;

                XX_i2j1 = (X_mesh(i+1,j+2)+X_mesh(i,j+2))/2;
```

```
                XX_i2j_1 = (X_mesh(i+1,j-1)+X_mesh(i,j-1))/2;
                YY_i2j1 = (Y_mesh(i+1,j+2)+Y_mesh(i,j+2))/2;
                YY_i2j_1 = (Y_mesh(i+1,j-1)+Y_mesh(i,j-1))/2;
                XX_i_2j1 = (X_mesh(i,j+2)+X_mesh(i-1,j+2))/2;
                XX_i_2j_1 = (X_mesh(i,j-1)+X_mesh(i-1,j-1))/2;
                YY_i_2j1 = (Y_mesh(i,j+2)+Y_mesh(i-1,j+2))/2;
                YY_i_2j_1 = (Y_mesh(i,j-1)+Y_mesh(i-1,j-1))/2;

                x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
                y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
                x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
                y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
                x_eta2_jp = (X_mesh(i,j+2)-X_mesh(i,j))/deta2;
                y_eta2_jp = (Y_mesh(i,j+2)-Y_mesh(i,j))/deta2;
                x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-1))/deta2;
                y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-1))/deta2;

            elseif j == idx_surf_split(2)
                % Near upper cut
                XX_i1j2 = (X_mesh(i+1,j+1)+X_mesh(i+1,j))/2;
                XX_i_1j2 = (X_mesh(i-1,j+1)+X_mesh(i-1,j))/2;
                YY_i1j2 = (Y_mesh(i+1,j+1)+Y_mesh(i+1,j))/2;
                YY_i_1j2 = (Y_mesh(i-1,j+1)+Y_mesh(i-1,j))/2;
                XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-2))/2;
                XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-2))/2;
                YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-2))/2;
                YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-2))/2;

                x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
                y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
                x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
                y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
                x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
                y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
                x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
                y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;

                XX_i2j1 = (X_mesh(i+1,j+1)+X_mesh(i,j+1))/2;
                XX_i2j_1 = (X_mesh(i+1,j-2)+X_mesh(i,j-2))/2;
                YY_i2j1 = (Y_mesh(i+1,j+1)+Y_mesh(i,j+1))/2;
                YY_i2j_1 = (Y_mesh(i+1,j-2)+Y_mesh(i,j-2))/2;
                XX_i_2j1 = (X_mesh(i,j+1)+X_mesh(i-1,j+1))/2;
                XX_i_2j_1 = (X_mesh(i,j-2)+X_mesh(i-1,j-2))/2;
                YY_i_2j1 = (Y_mesh(i,j+1)+Y_mesh(i-1,j+1))/2;
                YY_i_2j_1 = (Y_mesh(i,j-2)+Y_mesh(i-1,j-2))/2;

                x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
                y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
                x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
                y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
                x_eta2_jp = (X_mesh(i,j+1)-X_mesh(i,j))/deta2;
                y_eta2_jp = (Y_mesh(i,j+1)-Y_mesh(i,j))/deta2;
                x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-2))/deta2;
                y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-2))/deta2;

            else
                % Standard internal nodes
                XX_i1j2 = (X_mesh(i+1,j+1)+X_mesh(i+1,j))/2;
```

```
            XX_i_1j2 = (X_mesh(i-1,j+1)+X_mesh(i-1,j))/2;
            YY_i1j2 = (Y_mesh(i+1,j+1)+Y_mesh(i+1,j))/2;
            YY_i_1j2 = (Y_mesh(i-1,j+1)+Y_mesh(i-1,j))/2;
            XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-1))/2;
            XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-1))/2;
            YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-1))/2;
            YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-1))/2;

            x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
            y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
            x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
            y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
            x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
            y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
            x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
            y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;

            XX_i2j1 = (X_mesh(i+1,j+1)+X_mesh(i,j+1))/2;
            XX_i2j_1 = (X_mesh(i+1,j-1)+X_mesh(i,j-1))/2;
            YY_i2j1 = (Y_mesh(i+1,j+1)+Y_mesh(i,j+1))/2;
            YY_i2j_1 = (Y_mesh(i+1,j-1)+Y_mesh(i,j-1))/2;
            XX_i_2j1 = (X_mesh(i,j+1)+X_mesh(i-1,j+1))/2;
            XX_i_2j_1 = (X_mesh(i,j-1)+X_mesh(i-1,j-1))/2;
            YY_i_2j1 = (Y_mesh(i,j+1)+Y_mesh(i-1,j+1))/2;
            YY_i_2j_1 = (Y_mesh(i,j-1)+Y_mesh(i-1,j-1))/2;

            x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
            y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
            x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
            y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
            x_eta2_jp = (X_mesh(i,j+1)-X_mesh(i,j))/deta2;
            y_eta2_jp = (Y_mesh(i,j+1)-Y_mesh(i,j))/deta2;
            x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-1))/deta2;
            y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-1))/deta2;
        end

        % Metric calculation logic
        J_ip = x_eta1_ip*y_eta2_ip - y_eta1_ip*x_eta2_ip;
        C11_ip = (x_eta2_ip^2 + y_eta2_ip^2)/J_ip;
        C22_ip = (x_eta1_ip^2 + y_eta1_ip^2)/J_ip;
        C12_ip = -(x_eta1_ip*x_eta2_ip + y_eta1_ip*y_eta2_ip)/J_ip;

        J_im = x_eta1_im*y_eta2_im - y_eta1_im*x_eta2_im;
        C11_im = (x_eta2_im^2 + y_eta2_im^2)/J_im;
        C22_im = (x_eta1_im^2 + y_eta1_im^2)/J_im;
        C12_im = -(x_eta1_im*x_eta2_im + y_eta1_im*y_eta2_im)/J_im;

        J_jp = x_eta1_jp*y_eta2_jp - y_eta1_jp*x_eta2_jp;
        C11_jp = (x_eta2_jp^2 + y_eta2_jp^2)/J_jp;
        C22_jp = (x_eta1_jp^2 + y_eta1_jp^2)/J_jp;
        C12_jp = -(x_eta1_jp*x_eta2_jp + y_eta1_jp*y_eta2_jp)/J_jp;

        J_jm = x_eta1_jm*y_eta2_jm - y_eta1_jm*x_eta2_jm;
        C11_jm = (x_eta2_jm^2 + y_eta2_jm^2)/J_jm;
        C22_jm = (x_eta1_jm^2 + y_eta1_jm^2)/J_jm;
        C12_jm = -(x_eta1_jm*x_eta2_jm + y_eta1_jm*y_eta2_jm)/J_jm;

        % Coefficients
```

```
            Sij = C11_ip + C11_im + r_ratio^2 * (C22_jp + C22_jm);
            Sim = C11_im - r_ratio^2 * (C12_jp - C12_jm) / 4;
            Sip = C11_ip + r_ratio^2 * (C12_jp - C12_jm) / 4;
            Sjm = r_ratio^2 * C22_jm - r_ratio * (C12_ip - C12_im) / 4;
            Sjp = r_ratio^2 * C22_jp + r_ratio * (C12_ip - C12_im) / 4;
            Smm = r_ratio * (C12_im + C12_jm) / 4;
            Smp = -r_ratio * (C12_im + C12_jp) / 4;
            Spm = -r_ratio * (C12_ip + C12_jm) / 4;
            Spp = r_ratio * (C12_ip + C12_jp) / 4;

            % Update Stream Function
            if i >= idx_le_node && i <= idx_te_node && (j == idx_surf_split(1) || j == idx_surf_split(2))
                Psi(i,j) = Psi_airfoil;
            elseif j == idx_surf_split(1) && (i<idx_le_node || i> idx_te_node)
                % Wake/Inlet Line Lower Cut
                E_delta = (Sim*Psi(i-1,j)+Sip*Psi(i+1,j)+Sjm*Psi(i,j-1)+Sjp*Psi(i,j+2)+Smm*Psi(i-1,j-1)+Smp*Psi(i-1,j+2)+Spm*Psi(i+1,j-1)+Spp*Psi(i+1,j+2))/Sij;
                Psi(i,j) = Psi(i,j) + relax_factor*(E_delta-Psi(i,j));
            elseif j == idx_surf_split(2) && (i<idx_le_node || i> idx_te_node)
                % Wake/Inlet Line Upper Cut (match lower)
                Psi(i,j) = Psi(i,j-1);
            else
                % General Domain
                E_delta = (Sim*Psi(i-1,j)+Sip*Psi(i+1,j)+Sjm*Psi(i,j-1)+Sjp*Psi(i,j+1)+Smm*Psi(i-1,j-1)+Smp*Psi(i-1,j+1)+Spm*Psi(i+1,j-1)+Spp*Psi(i+1,j+1))/Sij;
                Psi(i,j) = Psi(i,j) + relax_factor*(E_delta-Psi(i,j));
            end
        end
    end

    % Kutta Condition Check
    if  abs(Psi(idx_te_node+1, idx_surf_split(1)) - Psi_airfoil) <= tol_conv
        kutta_cond = 1;
    else
        Psi_airfoil = Psi(idx_te_node+1, idx_surf_split(1));
        kutta_cond = 0;
    end

    % Convergence Check
    Residual = sqrt((Psi - Psi_old).^2 ./ nx ./ ny);
    max_res = max(max(Residual));
    Residual_History(iter-1) = max_res;

    if max_res <= tol_conv && kutta_cond == 1
        fprintf('Converged at iteration: %d\n', iter);
        fprintf('Max Error: %e\n', max_res);
        break
    elseif iter == max_iter
        fprintf('Max iterations reached without full convergence.\n');
    end

    Psi_old = Psi;
```

```matlab
end

%% Velocity Calculation
vel_u = zeros(nx, ny);
vel_v = zeros(nx, ny);

for j = 1:ny
    for i = 1:nx
        % Calculate local derivatives based on boundary location
        if i == 1
            if j == idx_surf_split(1)
                x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))/2/deta1;
                y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;
            elseif j == idx_surf_split(2)
                x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))/2/deta1;
                y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
            elseif j == 1
                x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))/2/deta1;
                y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))/2/deta1;
                x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))/2/deta2;
                y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))/2/deta2;
            elseif j == ny
                x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))/2/deta1;
                y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))/2/deta1;
                x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))/2/deta2;
                y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))/2/deta2;
            else
                x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))/2/deta1;
                y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
            end
        elseif i == nx
             % (Similar logic blocks for right boundary omitted for brevity but logic retained from original structure)
             if j == idx_surf_split(1)
                x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))/2/deta1;
                y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))/2/deta1;
```

```
                x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;
            elseif j == idx_surf_split(2)
                x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))/2/deta1;
                y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
            elseif j == 1
                x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))/2/deta1;
                y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))/2/deta1;
                x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))/2/deta2;
                y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))/2/deta2;
            elseif j == ny
                x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))/2/deta1;
                y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))/2/deta1;
                x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))/2/deta2;
                y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))/2/deta2;
            else
                x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))/2/deta1;
                y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
            end
        elseif j == 1
            x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
            y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
            x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))/2/deta2;
            y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))/2/deta2;
        elseif j == ny
            x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
            y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
            x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))/2/deta2;
            y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))/2/deta2;
        else
            % Internal or split lines
             if j == idx_surf_split(1) && (i<idx_le_node || i>idx_te_node)
                x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
                y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;
```

```
            elseif j == idx_surf_split(2) && (i<idx_le_node || i>idx_te_node)
                x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
                y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
            elseif j == idx_surf_split(1)
                x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
                y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
                x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))/2/deta2;
                y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))/2/deta2;
            elseif j == idx_surf_split(2)
                x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
                y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
                x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))/2/deta2;
                y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))/2/deta2;
            else
                x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
                y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
                x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
                y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
            end
        end

        J_det = x_eta1*y_eta2 - y_eta1*x_eta2;
        eta1_x_loc = y_eta2/J_det;
        eta2_x_loc = -y_eta1/J_det;
        eta1_y_loc = -x_eta2/J_det;
        eta2_y_loc = x_eta1/J_det;

        % Apply derivatives to Stream Function
        if i > 1 && i ~= nx && j > 1 && j ~= ny && (i<idx_le_node || i>idx_te_node)
             if j == idx_surf_split(1)
                dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
             elseif j == idx_surf_split(2)
                dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
             else
                dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
             end
        elseif i == 1 && j > 1 && j ~= ny
             % Inlet
             if j == idx_surf_split(1)
                dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
             elseif j == idx_surf_split(2)
                dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
             else
```

```
                dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
            end
        elseif i > 1 && i ~= nx && j == 1
            dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
            dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
        elseif i == nx && j > 1 && j ~= ny
            if j == idx_surf_split(1)
                dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
            elseif j == idx_surf_split(2)
                dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
            else
                dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
            end
        elseif i > 1 && i ~= nx && j == ny
            dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
            dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
        elseif i == 1 && j == 1
            dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
            dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
        elseif i == 1 && j == ny
            dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
            dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
        elseif i == nx && j == 1
            dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
            dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
        elseif i == nx && j == ny
            dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
            dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
        elseif j == idx_surf_split(2) && (i>idx_le_node && i<idx_te_node) % Upper Surface
            dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
            dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
        elseif j == idx_surf_split(1) && (i>idx_le_node && i<idx_te_node) % Lower Surface
            dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
            dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
        elseif j == idx_surf_split(2) && i == idx_le_node % L.E
            dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
            dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
        elseif j == idx_surf_split(1) && i == idx_le_node % L.E
            dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
            dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
        elseif j == idx_surf_split(2) && i == idx_te_node % T.E
            dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
            dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
        elseif j == idx_surf_split(1) && i == idx_te_node % T.E
            dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
            dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
        else
            dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
```

```
                dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
            end

            vel_u(i,j) = dPsi_dEta1*eta1_y_loc + dPsi_dEta2*eta2_y_loc;
            vel_v(i,j) = -dPsi_dEta1*eta1_x_loc - dPsi_dEta2*eta2_x_loc;
        end
end

Vel_mag = sqrt(vel_u.^2 + vel_v.^2);
Vel_airfoil_l = Vel_mag(idx_le_node:idx_te_node, idx_surf_split(1));
Vel_airfoil_u = Vel_mag(idx_le_node:idx_te_node, idx_surf_split(2));

%% Pressure Coefficient
Cp_field = 1 - (Vel_mag ./ U_free).^2;
Cp_u = 1 - (Vel_airfoil_u ./ U_free).^2;
Cp_l = 1 - (Vel_airfoil_l ./ U_free).^2;

%% Aerodynamic Coefficients (cl, cd, cm)
coeff_n = 0; coeff_m = 0;
num_pts = length(Cp_l);
for k = 1:num_pts-1
    cn_seg = 0.5 * ((Cp_l(k)-Cp_u(k)) + (Cp_l(k+1)-Cp_u(k+1))) / num_pts;
    coeff_n = coeff_n + cn_seg;

    pos_frac = linspace(0, num_pts, num_pts+1);
    coeff_m = coeff_m - cn_seg * (pos_frac(k)/num_pts - 0.25);
end

Cl = coeff_n * cosd(AoA_deg);
Cd = coeff_n * sind(AoA_deg);
Cm = coeff_m;

fprintf('------------------------------------\n');
fprintf('Aerodynamic Coefficients:\n');
fprintf('Cl = %.4f\n', Cl);
fprintf('Cd = %.4f\n', Cd);
fprintf('Cm = %.4f\n', Cm);
fprintf('------------------------------------\n');

%% --- PLOTTING SECTION ---

% 1. Airfoil Shape
figure('Name', 'Airfoil Shape Check', 'Color', 'w');
set(gcf, 'Color', 'w'); % Background White
hold on; grid on; axis equal;
plot(0:0.0001:c_len, polyval(polyfit(x_upp,y_upp,10),0:0.0001:c_len), 'k-', 'LineWidth', 2);
plot(0:0.0001:c_len, polyval(polyfit(x_low,y_low,10),0:0.0001:c_len), 'k-', 'LineWidth', 2);
% Mean Camber Line
plot(0:0.0001:c_len, (polyval(polyfit(x_upp,y_upp,10),0:0.0001:c_len)+polyval(polyfit(x_low,y_low,10),0:0.0001:c_len))/2, 'r--', 'LineWidth', 1.5);
title(['NACA ' sprintf('%d', NACA_digits) ' Airfoil Shape']);
xlabel('x/c', 'Color', 'k'); ylabel('y/c', 'Color', 'k');
% FIX: Explicitly set Legend Color to White and Text to Black
legend('Upper','Lower','Mean Camber', 'Color', 'w', 'TextColor', 'k');
```

```
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'FontSize', 10, 'GridColor', 'k', 'GridAlpha', 0.2);

% 2. Physical Grid
figure('Name', 'Physical Grid', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
plot(X_mesh, Y_mesh, 'Color', [0.2 0.2 0.2], 'LineWidth', 0.5); % Dark Grey lines for grid
for m = 1:length(x_pts)
    if m > idx_le_node && m < idx_te_node
        idx_cut = find(Y_temp(:,m) == yl_interp(m-idx_le_node+1));
        plot(X_mesh(1:idx_cut,m), Y_mesh(1:idx_cut,m), 'Color', [0.2 0.2 0.2]);
        plot(X_mesh(idx_cut+1:end,m), Y_mesh(idx_cut+1:end,m), 'Color', [0.2 0.2 0.2]);
    else
        plot(X_mesh(:,m), Y_mesh(:,m), 'Color', [0.2 0.2 0.2]);
    end
end
title('Physical H-Grid', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 3. Computational Grid
figure('Name', 'Computational Grid', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
plot(ETA1, ETA2, 'b-', 'LineWidth', 0.5);
plot(ETA2, ETA1, 'b-', 'LineWidth', 0.5);
title('Computational Grid (\xi - \eta)', 'Color', 'k');
xlabel('\eta_1', 'Color', 'k'); ylabel('\eta_2', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 4. Convergence History
figure('Name', 'Convergence', 'Color', 'w');
set(gcf, 'Color', 'w');
plot(2:1:iter, log10(Residual_History), 'b-', 'LineWidth', 1.5);
grid on;
title('Convergence History', 'Color', 'k');
xlabel('Iteration (n)', 'Color', 'k'); ylabel('log_{10}(RMS Error)', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 5. Streamlines
figure('Name', 'Streamlines', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
contour(X_mesh, Y_mesh, Psi, 50, 'LineWidth', 1.0);
fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.7 0.7 0.7], 'EdgeColor', 'k'); % Grey Fill
title(['Streamlines (\alpha = ' num2str(AoA_deg) '^{\circ})'], 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
```

```
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 6. Vector Field
figure('Name', 'Vector Field', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
q = quiver(X_mesh, Y_mesh, vel_u, vel_v);
q.Color = 'b';
q.LineWidth = 1.0;
plot(x_upp, y_upp, 'k', 'LineWidth', 2);
plot(x_low, y_low, 'k', 'LineWidth', 2);
title('Velocity Vector Field', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 7. Surface Velocity Distribution
figure('Name', 'Surface Velocity', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on;
plot(x_seg2, Vel_airfoil_l/U_free, 'b-', 'LineWidth', 2);
plot(x_seg2, Vel_airfoil_u/U_free, 'r-', 'LineWidth', 2);
title('Normalized Velocity on Airfoil Surface', 'Color', 'k');
xlabel('x/c', 'Color', 'k'); ylabel('V / U_{\infty}', 'Color', 'k');
% FIX: Explicitly set Legend Color to White and Text to Black
legend('Lower Surface', 'Upper Surface', 'Color', 'w', 'TextColor', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 8. Velocity Contour Lines (Restored)
figure('Name', 'Velocity Contour Lines', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
contour(X_mesh, Y_mesh, Vel_mag ./ U_free, 50);
fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], 'EdgeColor', 'k');
title('Velocity Contour Lines', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 9. Velocity Magnitude Color Map
figure('Name', 'Velocity Magnitude Color', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; axis equal;
surf(X_mesh, Y_mesh, Vel_mag ./ U_free);
view(2); shading interp; colormap jet;
cb = colorbar; cb.Color = 'k';
fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], 'EdgeColor', 'k');
title('Velocity Magnitude Contours', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k');

% 10. Pressure Coefficient Distribution
figure('Name', 'Cp Distribution', 'Color', 'w');
```

```
set(gcf, 'Color', 'w');
hold on; grid on;
plot(x_seg2, Cp_l, 'b-', 'LineWidth', 2);
plot(x_seg2, Cp_u, 'r-', 'LineWidth', 2);
set(gca, 'YDir','reverse');
title('C_p Distribution', 'Color', 'k');
xlabel('x/c', 'Color', 'k'); ylabel('C_p', 'Color', 'k');
% FIX: Explicitly set Legend Color to White and Text to Black
legend('Lower Surface', 'Upper Surface', 'Color', 'w', 'TextColor', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 11. Pressure Contour Lines (Restored)
figure('Name', 'Pressure Contour Lines', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; grid on; axis equal;
contour(X_mesh, Y_mesh, Cp_field, 50);
fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], 'EdgeColor', 'k');
title('Pressure Contour Lines', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', 'GridAlpha', 0.2);

% 12. Pressure Color Map
figure('Name', 'Pressure Color Map', 'Color', 'w');
set(gcf, 'Color', 'w');
hold on; axis equal;
contourf(X_mesh, Y_mesh, Cp_field, 50, 'LineColor', 'none');
cb = colorbar; cb.Color = 'k';
colormap jet;
fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], 'EdgeColor', 'k');
title('Pressure Coefficient Contours', 'Color', 'k');
xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k');
```