

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.

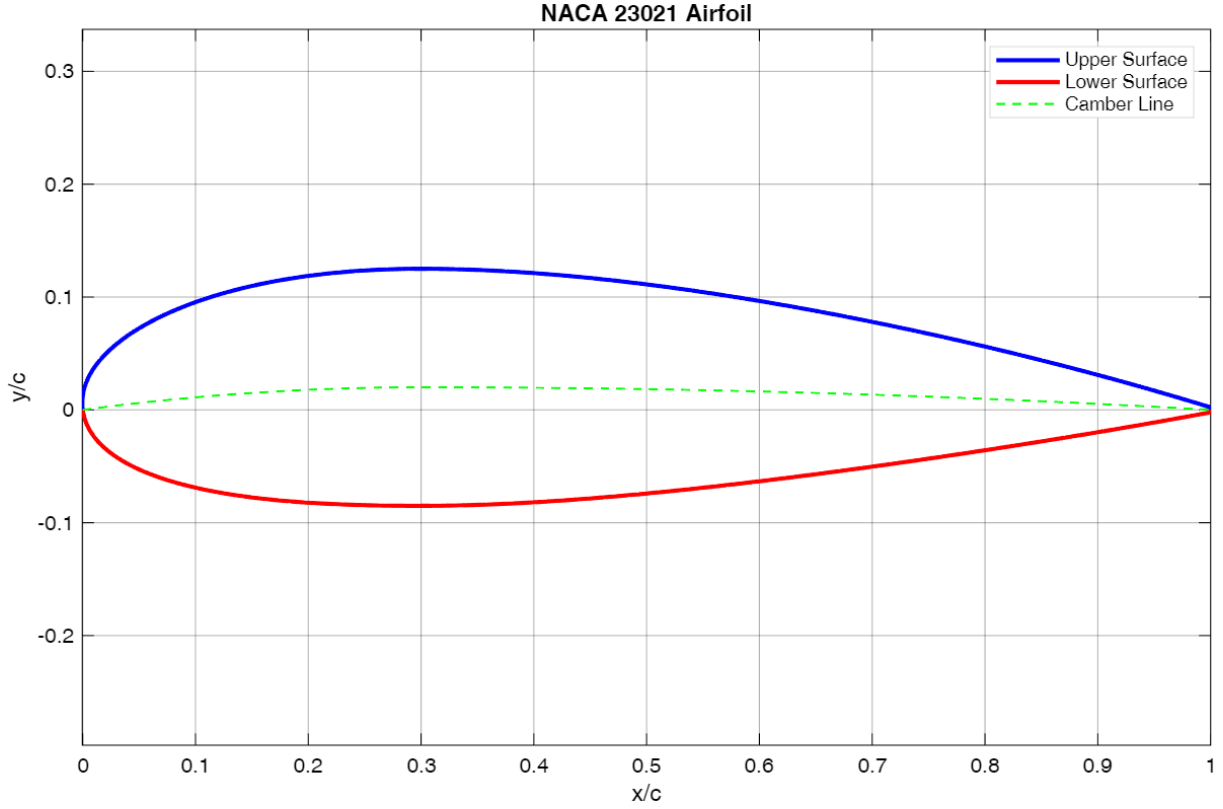


Figure 1: Enter Caption

Figure 2: NACA 23021 Airfoil Geometry

2 Governing Equations

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

$$\Psi_{xx} + \Psi_{yy} = 0 \quad (1)$$

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

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

Where:

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

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:

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

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.

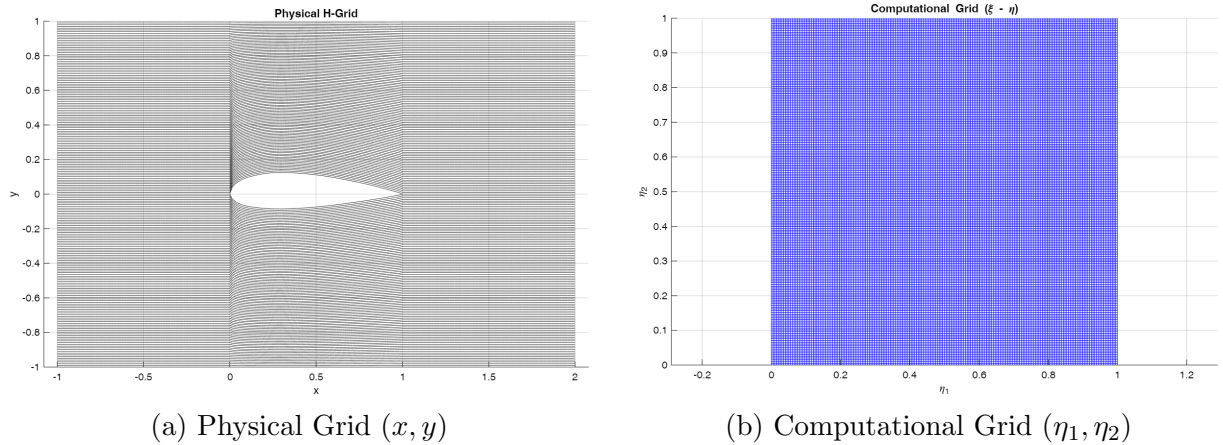


Figure 3: Grid Generation using H-Grid Topology

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}).

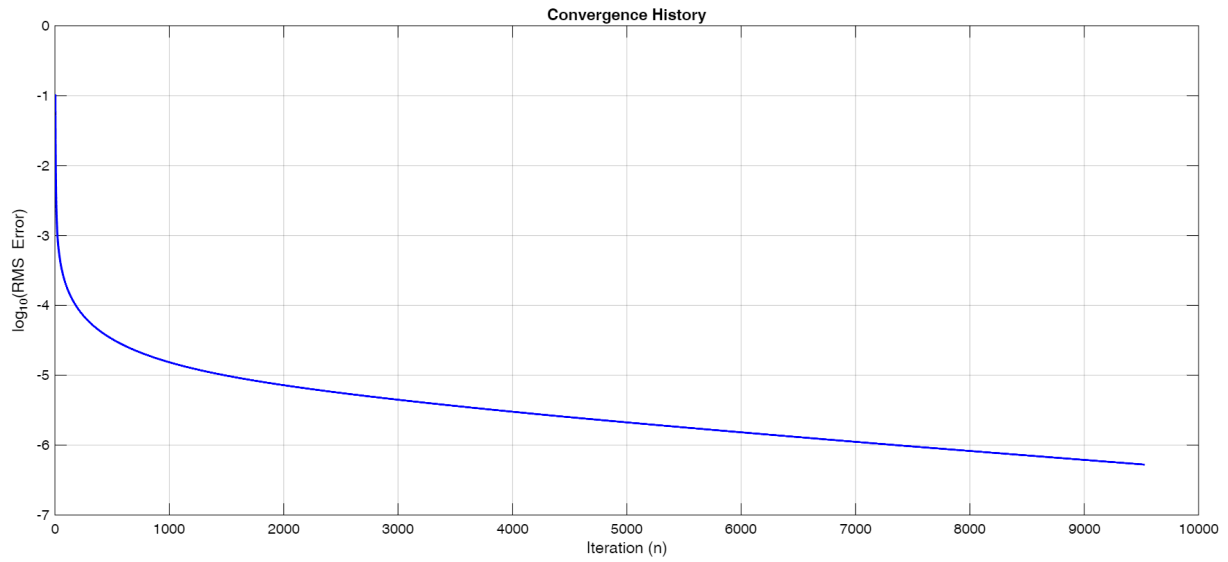


Figure 4: Convergence History of Ψ (Log Scale)

6 Flow Visualization Results

6.1 Streamlines and Vector Field

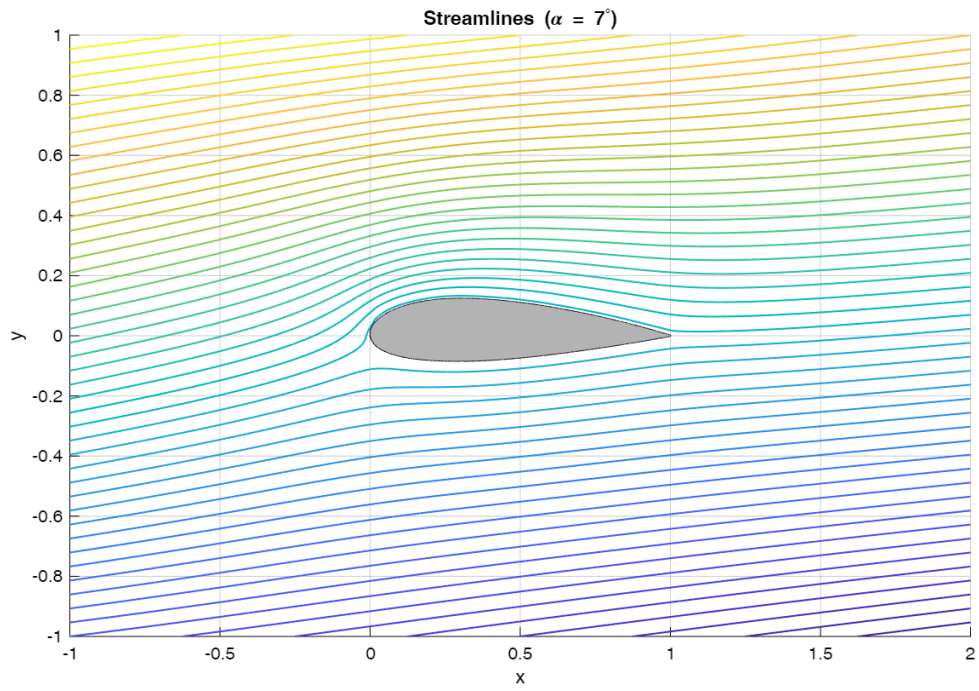


Figure 5: Flow Stream Lines over NACA 23021, $\alpha = 7^\circ$

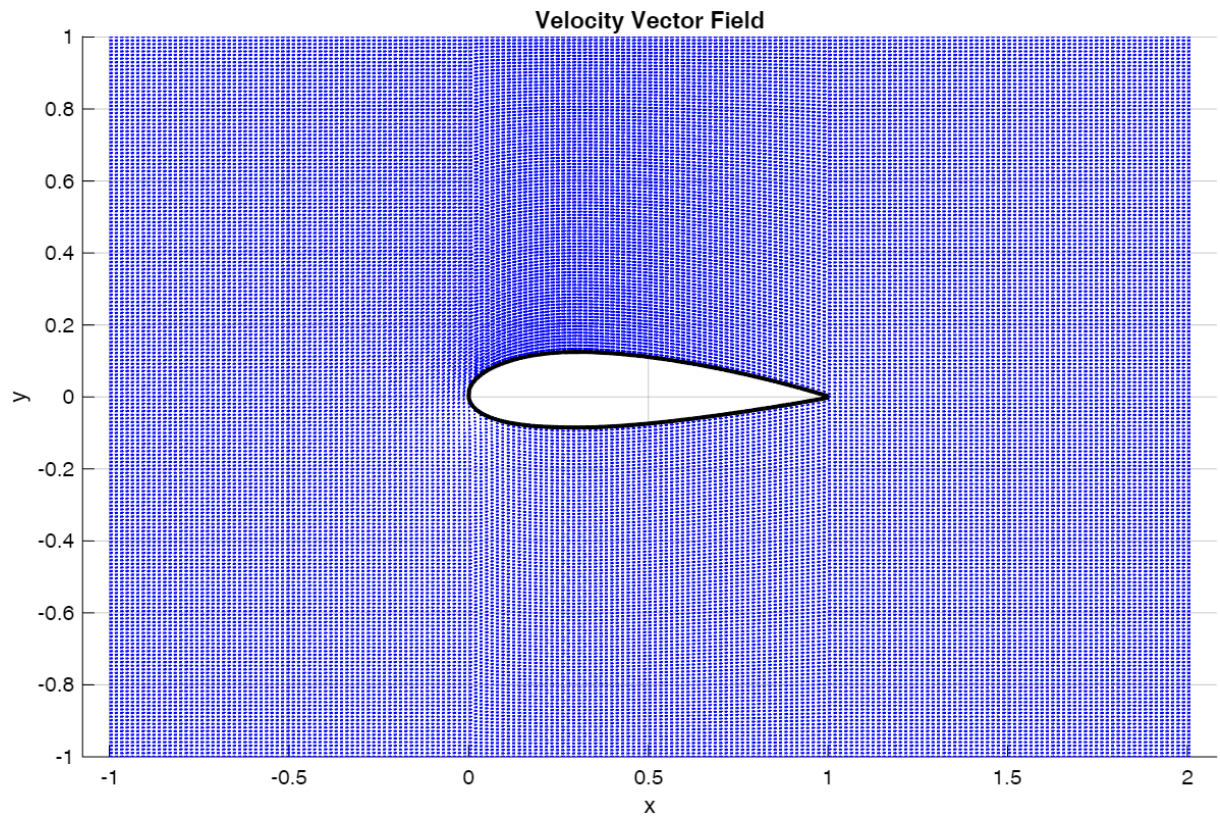


Figure 6: Vector Field around NACA 23021

6.2 Velocity Distribution

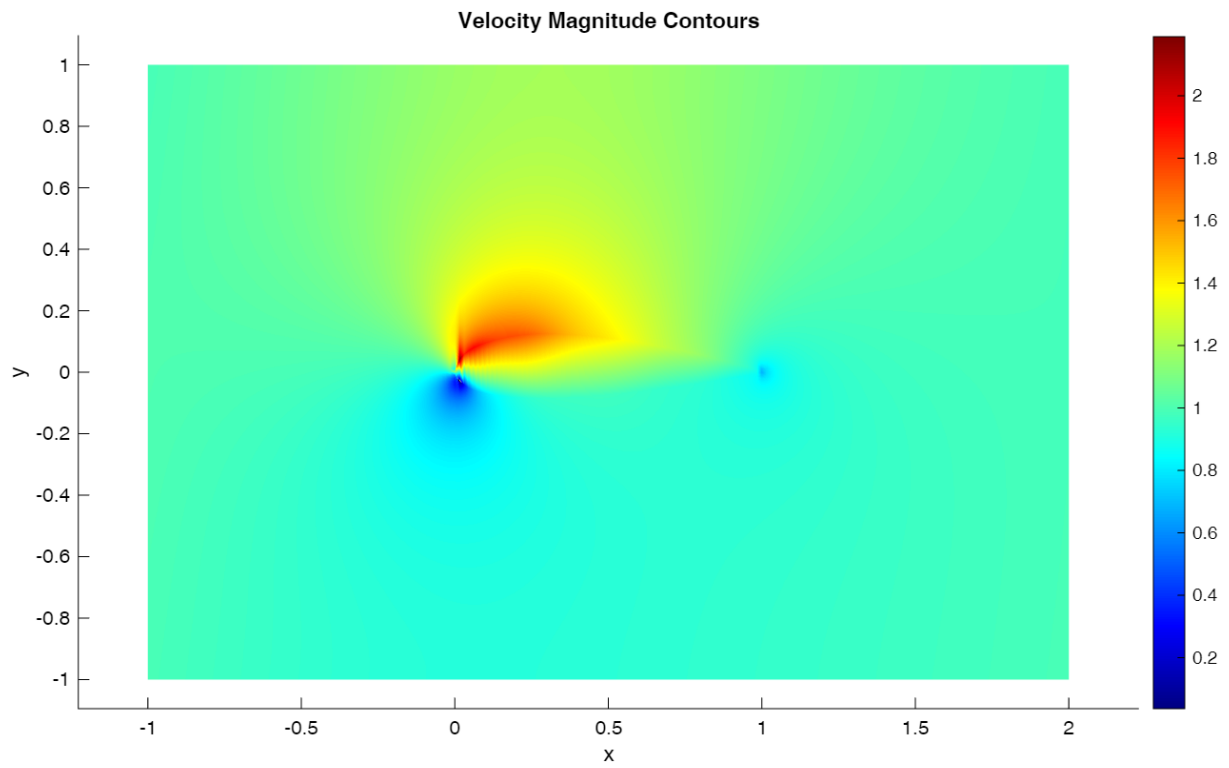


Figure 7: Velocity Magnitude (Color Map)

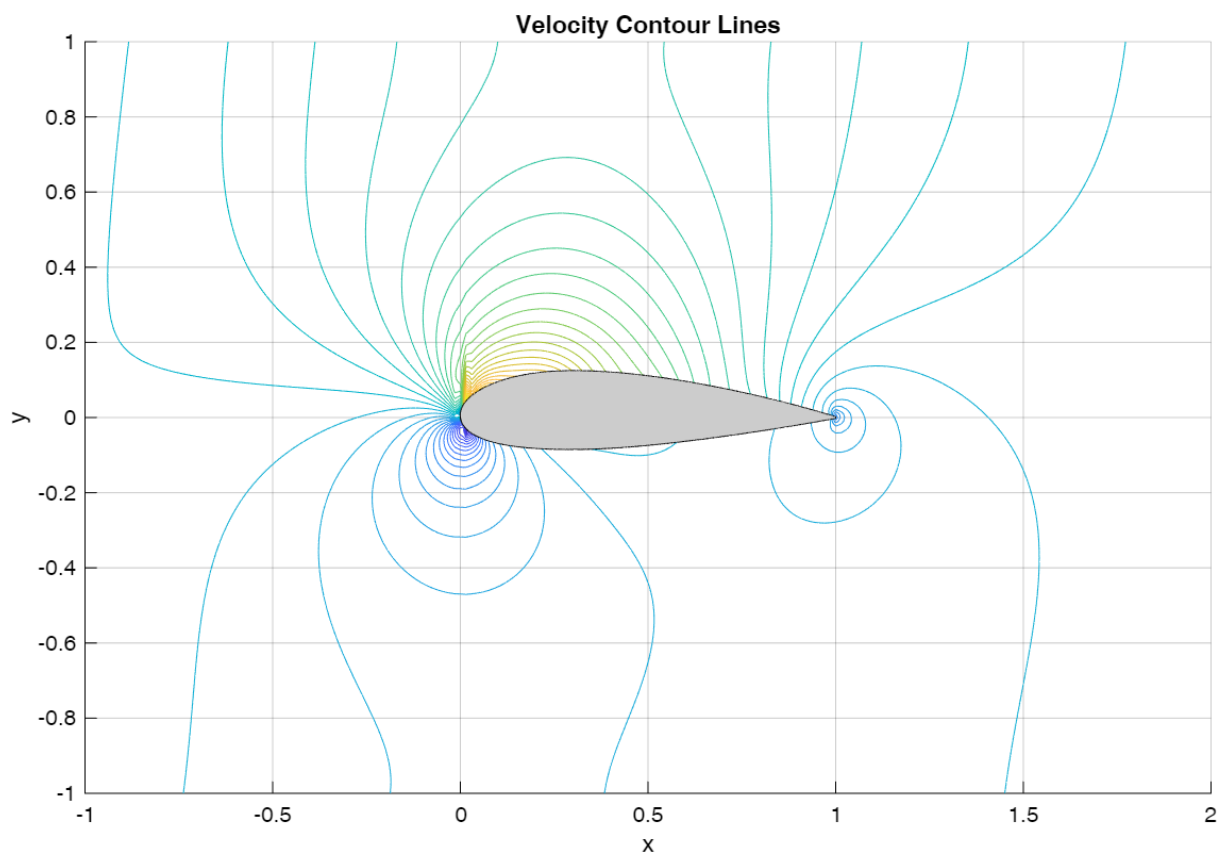


Figure 8: Velocity Contour Lines

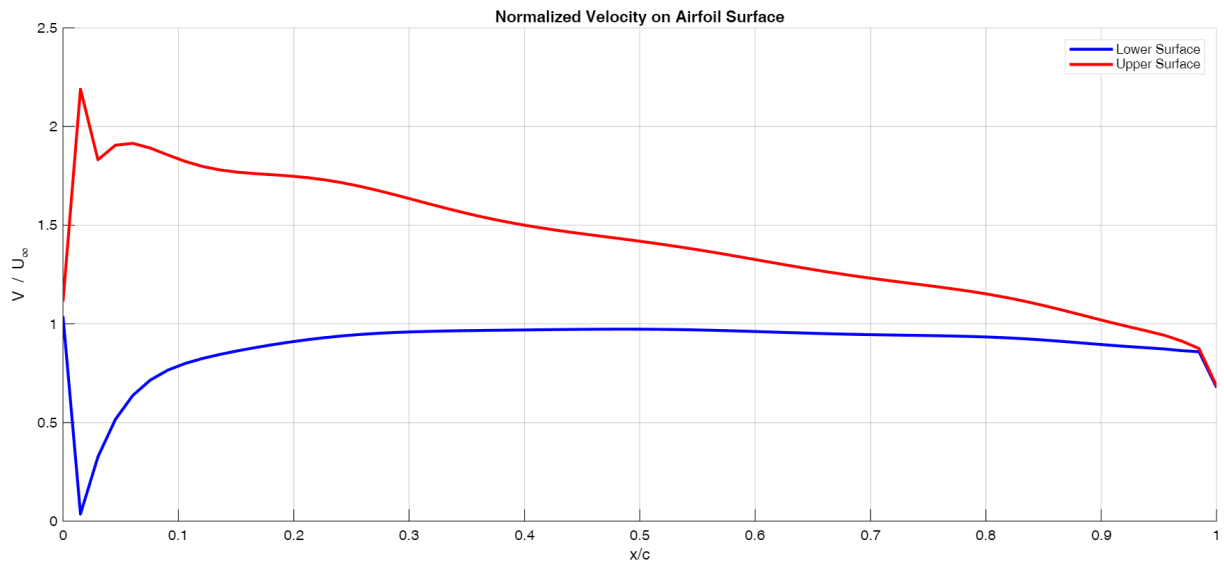


Figure 9: Velocity Distribution on the Airfoil Surface

6.3 Pressure Coefficient (C_p)

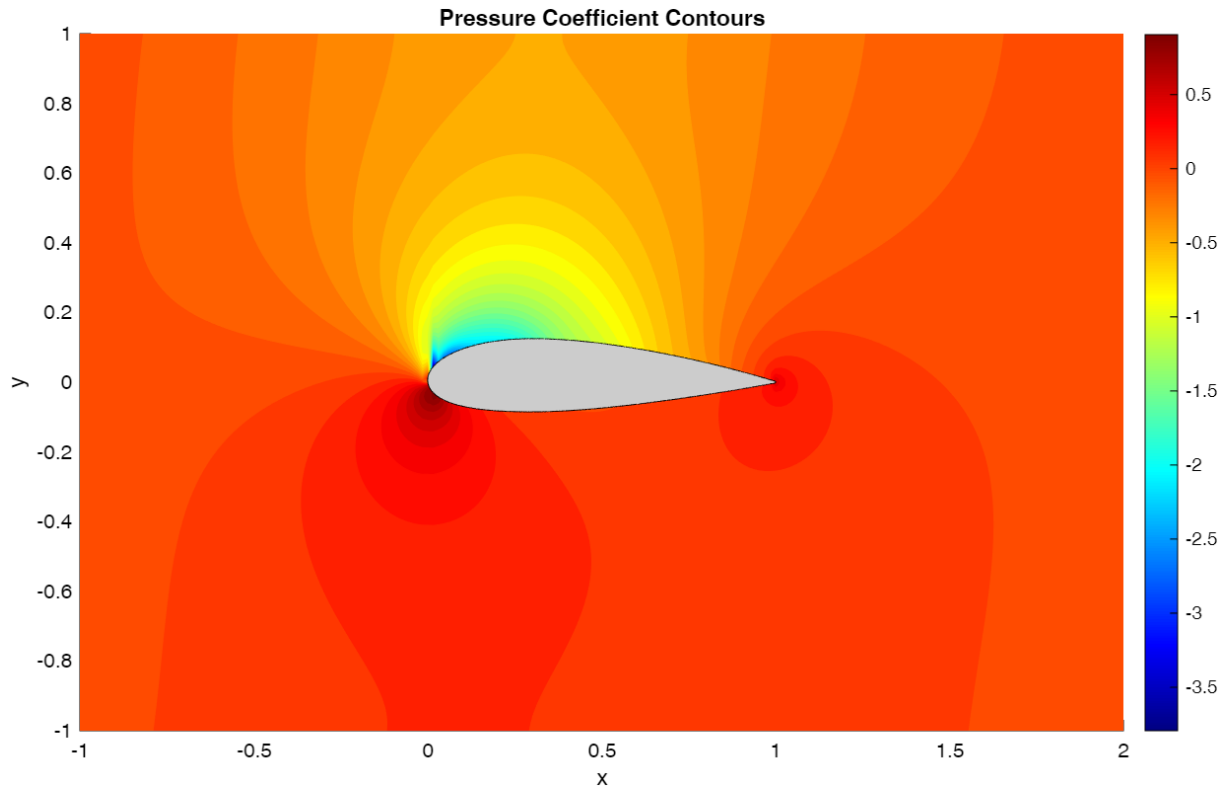


Figure 10: Pressure Distribution (Color Map)

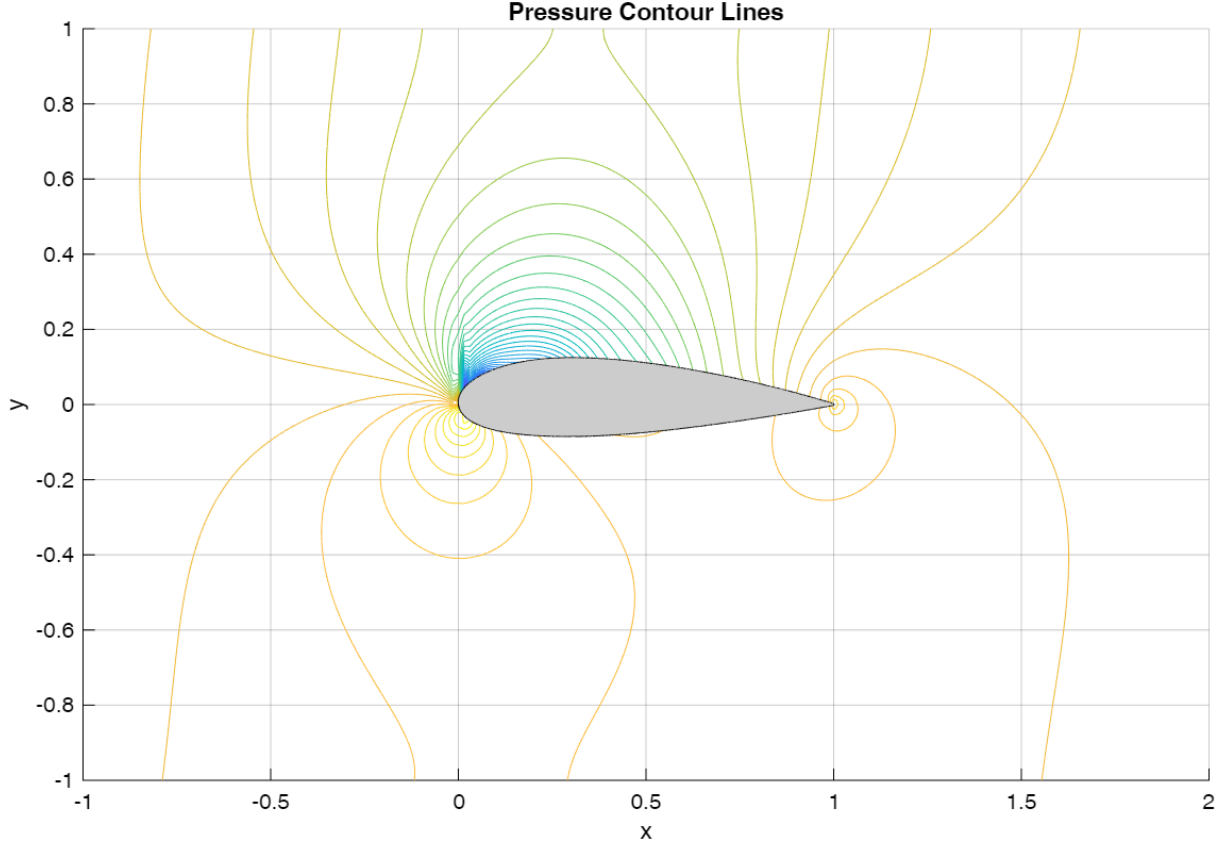


Figure 11: Pressure Lines around NACA 23021 , $\alpha = 7^\circ$

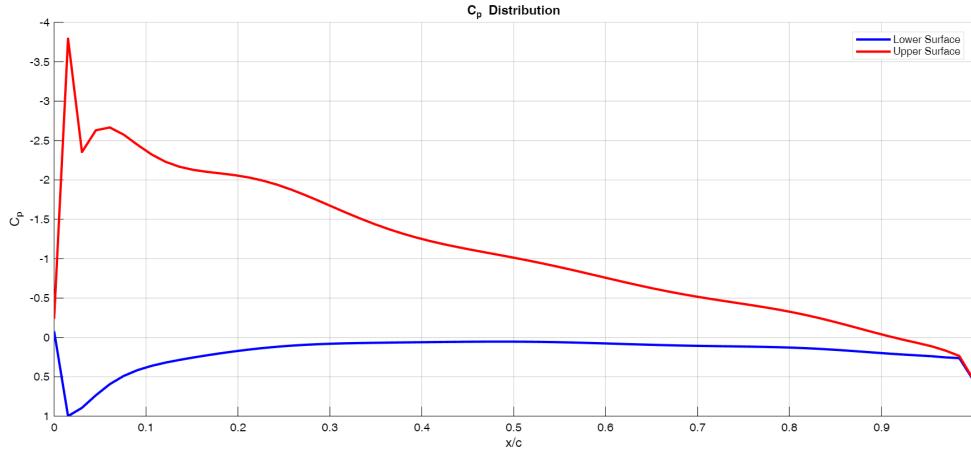


Figure 12: C_p Distribution on Airfoil Surface

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].

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

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

$$C_l = C_n \cos(\alpha) - C_a \sin(\alpha) \quad (5)$$

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]:

$$C_a \sin(\alpha) \ll C_n \cos(\alpha) \quad (6)$$

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

$$C_l \approx C_n \cos(\alpha) \quad (7)$$

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

$$C_d = C_n \sin(\alpha) \quad (8)$$

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]:

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

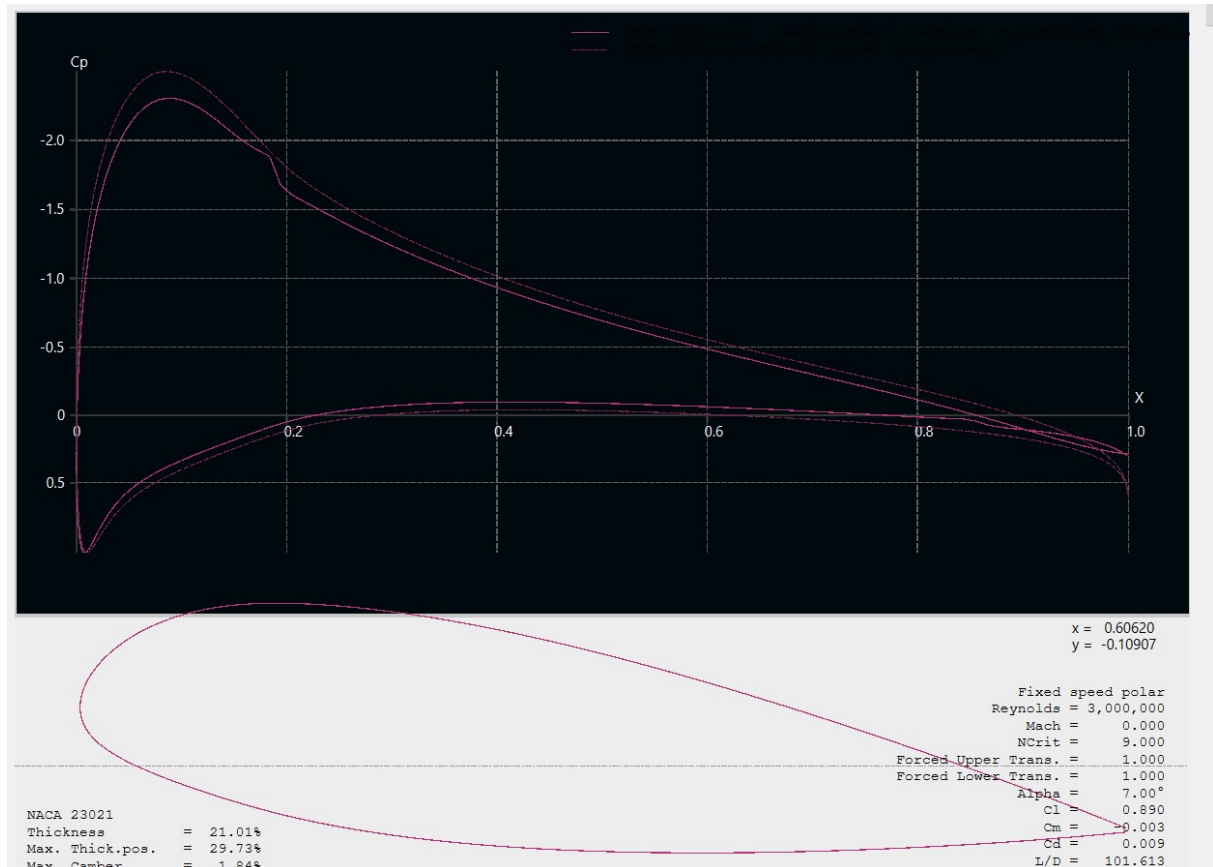


Figure 13: XFLR5 Validation: C_p Distribution on Airfoil Surface

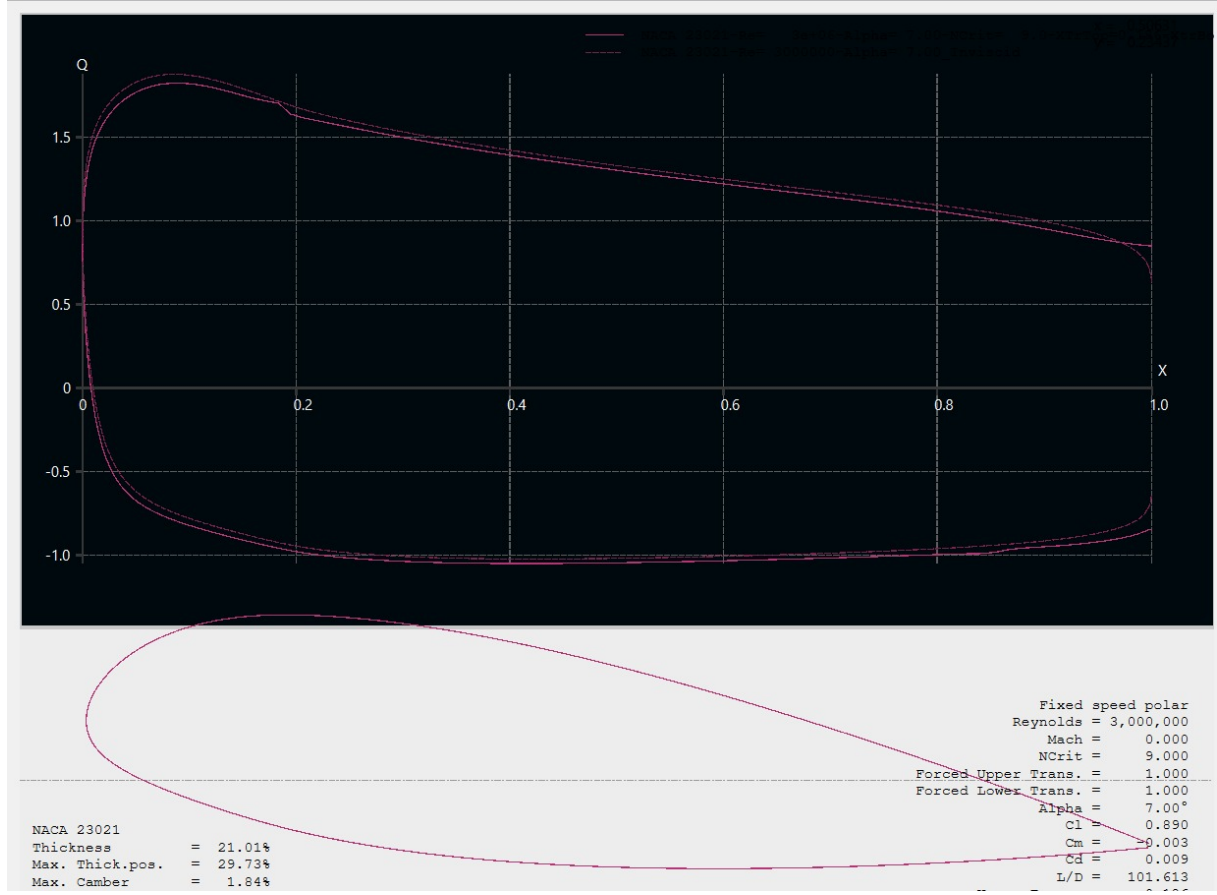


Figure 14: XFLR5 Validation: Velocity Ratio (Q/U_∞) Distribution on Airfoil Surface

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

```
1 %% Code solves the flow over NACA 23021 airfoil using PSOR method
2 clc
3 clearvars
4 close all
5 %% Inputs
6 NACA_digits = [2 3 0 2 1]; % NACA airfoil name
7 U_free = 100; % Freestream Velocity
8 AoA_deg = 7; % Angle of Attack
9 c_len = 1; % Chord length
10
11 %% Setup Parameters
12 nx = 200; ny = 200; % Grid size
13 max_iter = 2e5; % Maximum iterations
14 tol_conv = 1e-4; % Convergence Tolerance
15
16 %% NACA 23021 Airfoil Geometry Generation
17 x_c = 0:0.00001:c_len;
18
19 % NACA 23021 parameters
20 m_camb = 0.02; % Maximum camber
21 p_camb = 0.30; % Position of maximum camber
22 t_max = 0.21; % Max Thickness
23
24 % Camber line calculation
25 y_c = zeros(size(x_c));
26 dy_c = zeros(size(x_c));
27
28 for k = 1:length(x_c)
29     if x_c(k)/c_len <= p_camb
30         % Forward of maximum camber position
31         y_c(k) = (m_camb/(p_camb^2)) * (2*p_camb*(x_c(k)/c_len) - (x_c(k)/c_len)^2);
32         dy_c(k) = (2*m_camb/(p_camb^2)) * (p_camb - (x_c(k)/c_len));
33     else
34         % Aft of maximum camber position
35         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);
36         dy_c(k) = (2*m_camb/((1-p_camb)^2)) * (p_camb - (x_c(k)/c_len));
37     end
38 end
39
40 % Thickness distribution (standard NACA 4-digit equation)
41 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);
42
43 % Calculate upper and lower surfaces
44 theta_ang = atan(dy_c);
45 x_upp = x_c - half_thick.*sin(theta_ang);
46 x_low = x_c + half_thick.*sin(theta_ang);
47 y_upp = y_c + half_thick.*cos(theta_ang);
48 y_low = y_c - half_thick.*cos(theta_ang);
49
50 % Ensure trailing edge is closed
51 y_upp(end) = 0;
52 y_low(end) = 0;
53
```

```

54 % Plot 1: Airfoil Geometry Check
55 figure('Name', 'Airfoil Geometry', 'Color', 'w');
56 plot(x_upp, y_upp, 'b-', 'LineWidth', 2); hold on;
57 plot(x_low, y_low, 'r-', 'LineWidth', 2);
58 plot(x_c, y_c, 'g--', 'LineWidth', 1);
59 grid on; axis equal;
60 xlabel('x/c', 'Color', 'k'); ylabel('y/c', 'Color', 'k');
61 title('NACA 23021 Airfoil', 'Color', 'k');
62 % FIX: Explicitly set Legend Color to White and Text to Black
63 legend('Upper Surface', 'Lower Surface', 'Camber Line', 'Color', 'w', '
    TextColor', 'k');
64 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.3);
65
66 %% Grid Initialization
67 x_inlet = -1 * c_len;
68 x_outlet = 2 * c_len;
69 y_bottom = -1 * c_len;
70 y_top = 1 * c_len;
71
72 I_grid = nx + 2;
73 J_grid = ny + 2;
74
75 % Distribution of points along X
76 x_seg1 = linspace(x_inlet, 0, ceil(I_grid/3));
77 x_seg2 = linspace(0, c_len, floor(I_grid/3));
78 x_seg3 = linspace(c_len, x_outlet, floor(I_grid/3));
79
80 x_pts = [x_seg1 x_seg2 x_seg3];
81
82 % Clean up duplicate points at interfaces
83 idx_0 = find(x_pts == 0);
84 idx_c = find(x_pts == c_len);
85 x_pts = [x_pts(1:idx_0(1)) x_pts(idx_0(end)+1:idx_c(1)) x_pts(idx_c(end)
    +1:end)];
86
87 % Re-find indices after cleanup
88 idx_le = find(x_pts == 0);
89 idx_te = find(x_pts == c_len);
90
91 % Interpolate airfoil surface to grid points
92 yu_interp = polyval(polyfit(x_upp, y_upp, 10), x_pts(idx_le:idx_te));
93 yl_interp = polyval(polyfit(x_low, y_low, 10), x_pts(idx_le:idx_te));
94
95 % Force endpoints to zero
96 yu_interp(1) = 0; yl_interp(1) = 0;
97 yu_interp(end) = 0; yl_interp(end) = 0;
98
99 % Create Vertical Grid Lines
100 y_lines = cell(1, length(x_pts));
101 for k = 1:length(x_pts)
102     if k <= idx_le || k >= idx_te
103         y_lines{k} = [linspace(y_bottom, 0, ny/2) linspace(0, y_top, ny
            /2)];
104     else
105         % Split grid around the airfoil body
106         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)];

```



```

107     end
108 end
109
110 % Create MeshGrids
111 [X_mesh, ~] = meshgrid(x_pts, 1:ny);
112 Y_temp = zeros(ny, length(y_lines));
113
114 for k=1:length(y_lines)
115     for p = 1:ny
116         Y_temp(p,k) = y_lines{k}(p);
117     end
118 end
119 X_mesh = X_mesh';
120 Y_mesh = Y_temp';
121
122 % Computational Domain Transformation Metrics
123 ETA1 = (X_mesh - x_inlet)/(x_outlet - x_inlet);
124 ETA2_base = (Y_mesh(1,:) - y_bottom)/(y_top - y_bottom);
125 ETA2 = meshgrid(ETA2_base);
126
127 deta1 = ETA1(2,1) - ETA1(1,1);
128 deta2 = ETA2(1,2) - ETA2(1,1);
129
130 x_eta1 = x_outlet - x_inlet;
131 x_eta2 = 0;
132 y_eta1 = 0;
133 y_eta2 = y_top - y_bottom;
134
135 Jacobian = x_eta1*y_eta2 - y_eta1*x_eta2;
136
137 % Metrics (Simplified global assumptions for initialization)
138 C11_g = (x_eta2^2 + y_eta2^2)/Jacobian;
139 C22_g = (x_eta1^2 + y_eta1^2)/Jacobian;
140 C12_g = -(x_eta1*x_eta2 + y_eta1*y_eta2)/Jacobian;
141
142 %% Boundary Conditions
143 Psi = zeros(nx, ny); % Stream Function
144
145 idx_le_node = find(x_pts == 0);
146 idx_te_node = find(x_pts == c_len);
147 idx_surf_split = find(y_lines{end} == 0); % Midpoint index in Y
148
149 % Far-field BCs
150 for i = 2:nx
151     Psi(i,1) = Psi(i-1,1) + -U_free * sind(AoA_deg)*(x_pts(i)-x_pts(i-1));
152 end
153 for j = 2:ny
154     Psi(1,j) = Psi(1,j-1) + U_free * cosd(AoA_deg)*(y_top-y_bottom)/ny;
155     if j == idx_surf_split(2)
156         Psi(1,j) = Psi(1,j-1);
157     end
158 end
159 for i = 2:nx
160     Psi(i,ny) = Psi(i-1,ny) + -U_free * sind(AoA_deg)*(x_pts(i)-x_pts(i-1));
161 end
162 for j = 2:ny

```

```

163     Psi(nx,j) = Psi(nx,j-1) + U_free * cosd(AoA_deg)*(y_top-y_bottom)/ny
164     ;
165     if j == idx_surf_split(2)
166         Psi(nx,j) = Psi(nx,j-1);
167     end
168 end
169 %% Initiating the Grid (Linear Interpolation for initial guess)
170 for i = 2:nx-1
171     for j = 2:ny-1
172         Psi(i,j) = Psi(i,j-1) + ((Psi(i,ny)-Psi(i,1))/(ny-1));
173     end
174 end
175
176 % Set value on airfoil surface
177 idx_flat = find(y_lines{end}==0);
178 Psi_airfoil = Psi(1, idx_flat(1));
179 Psi(idx_le_node:idx_te_node, idx_surf_split) = Psi_airfoil;
180
181 %% PSOR Solver Loop
182 relax_factor = 1.0;
183 kutta_cond = 0;
184 Psi_old = Psi;
185 Residual_History = [];
186
187 fprintf('Starting Iterations...\n');
188
189 for iter = 2:max_iter
190     Psi = Psi_old;
191
192     for i = 2:nx-1
193         for j = 2:ny-1
194
195             % Grid Aspect Ratio
196             r_ratio = deta1/deta2;
197
198             % --- Metric Calculations based on position ---
199             if j == idx_surf_split(1)
200                 % Near lower cut
201                 XX_i1j2 = (X_mesh(i+1,j+2)+X_mesh(i+1,j))/2;
202                 XX_i_1j2 = (X_mesh(i-1,j+2)+X_mesh(i-1,j))/2;
203                 YY_i1j2 = (Y_mesh(i+1,j+2)+Y_mesh(i+1,j))/2;
204                 YY_i_1j2 = (Y_mesh(i-1,j+2)+Y_mesh(i-1,j))/2;
205                 XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-1))/2;
206                 XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-1))/2;
207                 YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-1))/2;
208                 YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-1))/2;
209
210                 x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
211                 y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
212                 x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
213                 y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
214                 x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
215                 y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
216                 x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
217                 y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;
218
219                 XX_i2j1 = (X_mesh(i+1,j+2)+X_mesh(i,j+2))/2;

```

```

220     XX_i2j_1 = (X_mesh(i+1,j-1)+X_mesh(i,j-1))/2;
221     YY_i2j1 = (Y_mesh(i+1,j+2)+Y_mesh(i,j+2))/2;
222     YY_i2j_1 = (Y_mesh(i+1,j-1)+Y_mesh(i,j-1))/2;
223     XX_i_2j1 = (X_mesh(i,j+2)+X_mesh(i-1,j+2))/2;
224     XX_i_2j_1 = (X_mesh(i,j-1)+X_mesh(i-1,j-1))/2;
225     YY_i_2j1 = (Y_mesh(i,j+2)+Y_mesh(i-1,j+2))/2;
226     YY_i_2j_1 = (Y_mesh(i,j-1)+Y_mesh(i-1,j-1))/2;
227
228     x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
229     y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
230     x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
231     y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
232     x_eta2_jp = (X_mesh(i,j+2)-X_mesh(i,j))/deta2;
233     y_eta2_jp = (Y_mesh(i,j+2)-Y_mesh(i,j))/deta2;
234     x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-1))/deta2;
235     y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-1))/deta2;
236
237     elseif j == idx_surf_split(2)
238         % Near upper cut
239         XX_i1j2 = (X_mesh(i+1,j+1)+X_mesh(i+1,j))/2;
240         XX_i_1j2 = (X_mesh(i-1,j+1)+X_mesh(i-1,j))/2;
241         YY_i1j2 = (Y_mesh(i+1,j+1)+Y_mesh(i+1,j))/2;
242         YY_i_1j2 = (Y_mesh(i-1,j+1)+Y_mesh(i-1,j))/2;
243         XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-2))/2;
244         XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-2))/2;
245         YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-2))/2;
246         YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-2))/2;
247
248         x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
249         y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
250         x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
251         y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
252         x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
253         y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
254         x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
255         y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;
256
257         XX_i2j1 = (X_mesh(i+1,j+1)+X_mesh(i,j+1))/2;
258         XX_i2j_1 = (X_mesh(i+1,j-2)+X_mesh(i,j-2))/2;
259         YY_i2j1 = (Y_mesh(i+1,j+1)+Y_mesh(i,j+1))/2;
260         YY_i2j_1 = (Y_mesh(i+1,j-2)+Y_mesh(i,j-2))/2;
261         XX_i_2j1 = (X_mesh(i,j+1)+X_mesh(i-1,j+1))/2;
262         XX_i_2j_1 = (X_mesh(i,j-2)+X_mesh(i-1,j-2))/2;
263         YY_i_2j1 = (Y_mesh(i,j+1)+Y_mesh(i-1,j+1))/2;
264         YY_i_2j_1 = (Y_mesh(i,j-2)+Y_mesh(i-1,j-2))/2;
265
266         x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
267         y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
268         x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
269         y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
270         x_eta2_jp = (X_mesh(i,j+1)-X_mesh(i,j))/deta2;
271         y_eta2_jp = (Y_mesh(i,j+1)-Y_mesh(i,j))/deta2;
272         x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-2))/deta2;
273         y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-2))/deta2;
274
275     else
276         % Standard internal nodes
277         XX_i1j2 = (X_mesh(i+1,j+1)+X_mesh(i+1,j))/2;

```

```

278     XX_i_1j2 = (X_mesh(i-1,j+1)+X_mesh(i-1,j))/2;
279     YY_i1j2 = (Y_mesh(i+1,j+1)+Y_mesh(i+1,j))/2;
280     YY_i_1j2 = (Y_mesh(i-1,j+1)+Y_mesh(i-1,j))/2;
281     XX_i1j_2 = (X_mesh(i+1,j)+X_mesh(i+1,j-1))/2;
282     XX_i_1j_2 = (X_mesh(i-1,j)+X_mesh(i-1,j-1))/2;
283     YY_i1j_2 = (Y_mesh(i+1,j)+Y_mesh(i+1,j-1))/2;
284     YY_i_1j_2 = (Y_mesh(i-1,j)+Y_mesh(i-1,j-1))/2;
285
286     x_eta1_ip = (X_mesh(i+1,j)-X_mesh(i,j))/deta1;
287     y_eta1_ip = (Y_mesh(i+1,j)-Y_mesh(i,j))/deta1;
288     x_eta1_im = (X_mesh(i,j)-X_mesh(i-1,j))/deta1;
289     y_eta1_im = (Y_mesh(i,j)-Y_mesh(i-1,j))/deta1;
290     x_eta1_jp = (XX_i1j2-XX_i_1j2)/2/deta1;
291     y_eta1_jp = (YY_i1j2-YY_i_1j2)/2/deta1;
292     x_eta1_jm = (XX_i1j_2-XX_i_1j_2)/2/deta1;
293     y_eta1_jm = (YY_i1j_2-YY_i_1j_2)/2/deta1;
294
295     XX_i2j1 = (X_mesh(i+1,j+1)+X_mesh(i,j+1))/2;
296     XX_i2j_1 = (X_mesh(i+1,j-1)+X_mesh(i,j-1))/2;
297     YY_i2j1 = (Y_mesh(i+1,j+1)+Y_mesh(i,j+1))/2;
298     YY_i2j_1 = (Y_mesh(i+1,j-1)+Y_mesh(i,j-1))/2;
299     XX_i_2j1 = (X_mesh(i,j+1)+X_mesh(i-1,j+1))/2;
300     XX_i_2j_1 = (X_mesh(i,j-1)+X_mesh(i-1,j-1))/2;
301     YY_i_2j1 = (Y_mesh(i,j+1)+Y_mesh(i-1,j+1))/2;
302     YY_i_2j_1 = (Y_mesh(i,j-1)+Y_mesh(i-1,j-1))/2;
303
304     x_eta2_ip = (XX_i2j1-XX_i2j_1)/2/deta2;
305     y_eta2_ip = (YY_i2j1-YY_i2j_1)/2/deta2;
306     x_eta2_im = (XX_i_2j1-XX_i_2j_1)/2/deta2;
307     y_eta2_im = (YY_i_2j1-YY_i_2j_1)/2/deta2;
308     x_eta2_jp = (X_mesh(i,j+1)-X_mesh(i,j))/deta2;
309     y_eta2_jp = (Y_mesh(i,j+1)-Y_mesh(i,j))/deta2;
310     x_eta2_jm = (X_mesh(i,j)-X_mesh(i,j-1))/deta2;
311     y_eta2_jm = (Y_mesh(i,j)-Y_mesh(i,j-1))/deta2;
312     end
313
314     % Metric calculation logic
315     J_ip = x_eta1_ip*y_eta2_ip - y_eta1_ip*x_eta2_ip;
316     C11_ip = (x_eta2_ip^2 + y_eta2_ip^2)/J_ip;
317     C22_ip = (x_eta1_ip^2 + y_eta1_ip^2)/J_ip;
318     C12_ip = -(x_eta1_ip*x_eta2_ip + y_eta1_ip*y_eta2_ip)/J_ip;
319
320     J_im = x_eta1_im*y_eta2_im - y_eta1_im*x_eta2_im;
321     C11_im = (x_eta2_im^2 + y_eta2_im^2)/J_im;
322     C22_im = (x_eta1_im^2 + y_eta1_im^2)/J_im;
323     C12_im = -(x_eta1_im*x_eta2_im + y_eta1_im*y_eta2_im)/J_im;
324
325     J_jp = x_eta1_jp*y_eta2_jp - y_eta1_jp*x_eta2_jp;
326     C11_jp = (x_eta2_jp^2 + y_eta2_jp^2)/J_jp;
327     C22_jp = (x_eta1_jp^2 + y_eta1_jp^2)/J_jp;
328     C12_jp = -(x_eta1_jp*x_eta2_jp + y_eta1_jp*y_eta2_jp)/J_jp;
329
330     J_jm = x_eta1_jm*y_eta2_jm - y_eta1_jm*x_eta2_jm;
331     C11_jm = (x_eta2_jm^2 + y_eta2_jm^2)/J_jm;
332     C22_jm = (x_eta1_jm^2 + y_eta1_jm^2)/J_jm;
333     C12_jm = -(x_eta1_jm*x_eta2_jm + y_eta1_jm*y_eta2_jm)/J_jm;
334
335     % Coefficients

```

```

336     Sij = C11_ip + C11_im + r_ratio^2 * (C22_jp + C22_jm);
337     Sim = C11_im - r_ratio^2 * (C12_jp - C12_jm) / 4;
338     Sip = C11_ip + r_ratio^2 * (C12_jp - C12_jm) / 4;
339     Sjm = r_ratio^2 * C22_jm - r_ratio * (C12_ip - C12_im) / 4;
340     Sjp = r_ratio^2 * C22_jp + r_ratio * (C12_ip - C12_im) / 4;
341     Smm = r_ratio * (C12_im + C12_jm) / 4;
342     Smp = -r_ratio * (C12_im + C12_jp) / 4;
343     Spm = -r_ratio * (C12_ip + C12_jm) / 4;
344     Spp = r_ratio * (C12_ip + C12_jp) / 4;
345
346     % Update Stream Function
347     if i >= idx_le_node && i <= idx_te_node && (j ==
idx_surf_split(1) || j == idx_surf_split(2))
348         Psi(i,j) = Psi_airfoil;
349     elseif j == idx_surf_split(1) && (i<idx_le_node || i>
idx_te_node)
350         % Wake/Inlet Line Lower Cut
351         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;
352         Psi(i,j) = Psi(i,j) + relax_factor*(E_delta-Psi(i,j));
353     elseif j == idx_surf_split(2) && (i<idx_le_node || i>
idx_te_node)
354         % Wake/Inlet Line Upper Cut (match lower)
355         Psi(i,j) = Psi(i,j-1);
356     else
357         % General Domain
358         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;
359         Psi(i,j) = Psi(i,j) + relax_factor*(E_delta-Psi(i,j));
360     end
361 end
362 end
363
364 % Kutta Condition Check
365 if abs(Psi(idx_te_node+1, idx_surf_split(1)) - Psi_airfoil) <=
tol_conv
366     kutta_cond = 1;
367 else
368     Psi_airfoil = Psi(idx_te_node+1, idx_surf_split(1));
369     kutta_cond = 0;
370 end
371
372 % Convergence Check
373 Residual = sqrt((Psi - Psi_old).^2 ./ nx ./ ny);
374 max_res = max(max(Residual));
375 Residual_History(iter-1) = max_res;
376
377 if max_res <= tol_conv && kutta_cond == 1
378     fprintf('Converged at iteration: %d\n', iter);
379     fprintf('Max Error: %e\n', max_res);
380     break
381 elseif iter == max_iter
382     fprintf('Max iterations reached without full convergence.\n');
383 end
384
385 Psi_old = Psi;

```

```

386 end
387
388 %% Velocity Calculation
389 vel_u = zeros(nx, ny);
390 vel_v = zeros(nx, ny);
391
392 for j = 1:ny
393     for i = 1:nx
394         % Calculate local derivatives based on boundary location
395         if i == 1
396             if j == idx_surf_split(1)
397                 x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))
398                 /2/deta1;
399                 y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))
400                 /2/deta1;
401                 x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
402                 y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;
403             elseif j == idx_surf_split(2)
404                 x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))
405                 /2/deta1;
406                 y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))
407                 /2/deta1;
408                 x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
409                 y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
410             elseif j == 1
411                 x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))
412                 /2/deta1;
413                 y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))
414                 /2/deta1;
415                 x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))
416                 /2/deta2;
417                 y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))
418                 /2/deta2;
419             elseif j == ny
420                 x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))
421                 /2/deta1;
422                 y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))
423                 /2/deta1;
424                 x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))
425                 /2/deta2;
426                 y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))
427                 /2/deta2;
428             else
429                 x_eta1 = (-3*X_mesh(i,j)+4*X_mesh(i+1,j)-X_mesh(i+2,j))
430                 /2/deta1;
431                 y_eta1 = (-3*Y_mesh(i,j)+4*Y_mesh(i+1,j)-Y_mesh(i+2,j))
432                 /2/deta1;
433                 x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
434                 y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
435             end
436         elseif i == nx
437             % (Similar logic blocks for right boundary omitted for
438             % brevity but logic retained from original structure)
439             if j == idx_surf_split(1)
440                 x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))
441                 /2/deta1;
442                 y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))
443                 /2/deta1;

```

```

427         x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
428         y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;
429     elseif j == idx_surf_split(2)
430         x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))
/2/deta1;
431         y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))
/2/deta1;
432         x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
433         y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
434     elseif j == 1
435         x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))
/2/deta1;
436         y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))
/2/deta1;
437         x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))
/2/deta2;
438         y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))
/2/deta2;
439     elseif j == ny
440         x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))
/2/deta1;
441         y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))
/2/deta1;
442         x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))
/2/deta2;
443         y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))
/2/deta2;
444     else
445         x_eta1 = (3*X_mesh(i,j)-4*X_mesh(i-1,j)+X_mesh(i-2,j))
/2/deta1;
446         y_eta1 = (3*Y_mesh(i,j)-4*Y_mesh(i-1,j)+Y_mesh(i-2,j))
/2/deta1;
447         x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
448         y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
449     end
450     elseif j == 1
451         x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
452         y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
453         x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))/2/
deta2;
454         y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))/2/
deta2;
455     elseif j == ny
456         x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
457         y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
458         x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))/2/
deta2;
459         y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))/2/
deta2;
460     else
461         % Internal or split lines
462         if j == idx_surf_split(1) && (i<idx_le_node || i>
idx_te_node)
463             x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
464             y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
465             x_eta2 = (X_mesh(i,j+2)-X_mesh(i,j-1))/2/deta2;
466             y_eta2 = (Y_mesh(i,j+2)-Y_mesh(i,j-1))/2/deta2;

```

```

467         elseif j == idx_surf_split(2) && (i<idx_le_node || i>
idx_te_node)
468             x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
469             y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
470             x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-2))/2/deta2;
471             y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-2))/2/deta2;
472         elseif j == idx_surf_split(1)
473             x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
474             y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
475             x_eta2 = (3*X_mesh(i,j)-4*X_mesh(i,j-1)+X_mesh(i,j-2))
/2/deta2;
476             y_eta2 = (3*Y_mesh(i,j)-4*Y_mesh(i,j-1)+Y_mesh(i,j-2))
/2/deta2;
477         elseif j == idx_surf_split(2)
478             x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
479             y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
480             x_eta2 = (-3*X_mesh(i,j)+4*X_mesh(i,j+1)-X_mesh(i,j+2))
/2/deta2;
481             y_eta2 = (-3*Y_mesh(i,j)+4*Y_mesh(i,j+1)-Y_mesh(i,j+2))
/2/deta2;
482         else
483             x_eta1 = (X_mesh(i+1,j)-X_mesh(i-1,j))/2/deta1;
484             y_eta1 = (Y_mesh(i+1,j)-Y_mesh(i-1,j))/2/deta1;
485             x_eta2 = (X_mesh(i,j+1)-X_mesh(i,j-1))/2/deta2;
486             y_eta2 = (Y_mesh(i,j+1)-Y_mesh(i,j-1))/2/deta2;
487         end
488     end
489
490     J_det = x_eta1*y_eta2 - y_eta1*x_eta2;
491     eta1_x_loc = y_eta2/J_det;
492     eta2_x_loc = -y_eta1/J_det;
493     eta1_y_loc = -x_eta2/J_det;
494     eta2_y_loc = x_eta1/J_det;
495
496     % Apply derivatives to Stream Function
497     if i > 1 && i ~= nx && j > 1 && j ~= ny && (i<idx_le_node || i>
idx_te_node)
498         if j == idx_surf_split(1)
499             dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
500             dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
501         elseif j == idx_surf_split(2)
502             dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
503             dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
504         else
505             dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
506             dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
507         end
508     elseif i == 1 && j > 1 && j ~= ny
509         % Inlet
510         if j == idx_surf_split(1)
511             dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/
deta1;
512             dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
513         elseif j == idx_surf_split(2)
514             dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/
deta1;
515             dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
516         else

```



```

517         dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/
deta1;
518         dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
519     end
520     elseif i > 1 && i ~= nx && j == 1
521         dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
522         dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
523     elseif i == nx && j > 1 && j ~= ny
524         if j == idx_surf_split(1)
525             dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/
deta1;
526             dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
527         elseif j == idx_surf_split(2)
528             dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/
deta1;
529             dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
530         else
531             dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/
deta1;
532             dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
533         end
534     elseif i > 1 && i ~= nx && j == ny
535         dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
536         dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
537     elseif i == 1 && j == 1
538         dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
539         dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
540     elseif i == 1 && j == ny
541         dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
542         dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
543     elseif i == nx && j == 1
544         dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
545         dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
546     elseif i == nx && j == ny
547         dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
548         dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
549     elseif j == idx_surf_split(2) && (i>idx_le_node && i<idx_te_node
) % Upper Surface
550         dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
551         dPsi_dEta2 = (-3*Psi(i,j)+4*Psi(i,j+1)-Psi(i,j+2))/2/deta2;
552     elseif j == idx_surf_split(1) && (i>idx_le_node && i<idx_te_node
) % Lower Surface
553         dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;
554         dPsi_dEta2 = (3*Psi(i,j)-4*Psi(i,j-1)+Psi(i,j-2))/2/deta2;
555     elseif j == idx_surf_split(2) && i == idx_le_node % L.E
556         dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
557         dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
558     elseif j == idx_surf_split(1) && i == idx_le_node % L.E
559         dPsi_dEta1 = (3*Psi(i,j)-4*Psi(i-1,j)+Psi(i-2,j))/2/deta1;
560         dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
561     elseif j == idx_surf_split(2) && i == idx_te_node % T.E
562         dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
563         dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-2))/2/deta2;
564     elseif j == idx_surf_split(1) && i == idx_te_node % T.E
565         dPsi_dEta1 = (-3*Psi(i,j)+4*Psi(i+1,j)-Psi(i+2,j))/2/deta1;
566         dPsi_dEta2 = (Psi(i,j+2)-Psi(i,j-1))/2/deta2;
567     else
568         dPsi_dEta1 = (Psi(i+1,j)-Psi(i-1,j))/2/deta1;

```

```

569         dPsi_dEta2 = (Psi(i,j+1)-Psi(i,j-1))/2/deta2;
570     end
571
572     vel_u(i,j) = dPsi_dEta1*eta1_y_loc + dPsi_dEta2*eta2_y_loc;
573     vel_v(i,j) = -dPsi_dEta1*eta1_x_loc - dPsi_dEta2*eta2_x_loc;
574 end
575 end
576
577 Vel_mag = sqrt(vel_u.^2 + vel_v.^2);
578 Vel_airfoil_l = Vel_mag(idx_le_node:idx_te_node, idx_surf_split(1));
579 Vel_airfoil_u = Vel_mag(idx_le_node:idx_te_node, idx_surf_split(2));
580
581 %% Pressure Coefficient
582 Cp_field = 1 - (Vel_mag ./ U_free).^2;
583 Cp_u = 1 - (Vel_airfoil_u ./ U_free).^2;
584 Cp_l = 1 - (Vel_airfoil_l ./ U_free).^2;
585
586 %% Aerodynamic Coefficients (cl, cd, cm)
587 coeff_n = 0; coeff_m = 0;
588 num_pts = length(Cp_l);
589 for k = 1:num_pts-1
590     cn_seg = 0.5 * ((Cp_l(k)-Cp_u(k)) + (Cp_l(k+1)-Cp_u(k+1))) / num_pts
591     ;
592     coeff_n = coeff_n + cn_seg;
593
594     pos_frac = linspace(0, num_pts, num_pts+1);
595     coeff_m = coeff_m - cn_seg * (pos_frac(k)/num_pts - 0.25);
596 end
597
598 Cl = coeff_n * cosd(AoA_deg);
599 Cd = coeff_n * sind(AoA_deg);
600 Cm = coeff_m;
601
602 fprintf('-----\n');
603 fprintf('Aerodynamic Coefficients:\n');
604 fprintf('Cl = %.4f\n', Cl);
605 fprintf('Cd = %.4f\n', Cd);
606 fprintf('Cm = %.4f\n', Cm);
607 fprintf('-----\n');
608
609 %% --- PLOTTING SECTION ---
610
611 % 1. Airfoil Shape
612 figure('Name', 'Airfoil Shape Check', 'Color', 'w');
613 set(gcf, 'Color', 'w'); % Background White
614 hold on; grid on; axis equal;
615 plot(0:0.0001:c_len, polyval(polyfit(x_upper,y_upper,10),0:0.0001:c_len), 'k', 'LineWidth', 2);
616 plot(0:0.0001:c_len, polyval(polyfit(x_lower,y_lower,10),0:0.0001:c_len), 'k', 'LineWidth', 2);
617 % Mean Camber Line
618 plot(0:0.0001:c_len, (polyval(polyfit(x_upper,y_upper,10),0:0.0001:c_len)+
619     polyval(polyfit(x_lower,y_lower,10),0:0.0001:c_len))/2, 'r--', 'LineWidth', 1.5);
620 title(['NACA ' sprintf('%d', NACA_digits) ' Airfoil Shape']);
621 xlabel('x/c', 'Color', 'k'); ylabel('y/c', 'Color', 'k');
622 % FIX: Explicitly set Legend Color to White and Text to Black
623 legend('Upper','Lower','Mean Camber', 'Color', 'w', 'TextColor', 'k');

```

```

622 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'FontSize', 10, '
    GridColor', 'k', 'GridAlpha', 0.2);
623
624 % 2. Physical Grid
625 figure('Name', 'Physical Grid', 'Color', 'w');
626 set(gcf, 'Color', 'w');
627 hold on; grid on; axis equal;
628 plot(X_mesh, Y_mesh, 'Color', [0.2 0.2 0.2], 'LineWidth', 0.5); % Dark
    Grey lines for grid
629 for m = 1:length(x_pts)
630     if m > idx_le_node && m < idx_te_node
631         idx_cut = find(Y_temp(:,m) == yl_interp(m-idx_le_node+1));
632         plot(X_mesh(1:idx_cut,m), Y_mesh(1:idx_cut,m), 'Color', [0.2 0.2
0.2]);
633         plot(X_mesh(idx_cut+1:end,m), Y_mesh(idx_cut+1:end,m), 'Color',
[0.2 0.2 0.2]);
634     else
635         plot(X_mesh(:,m), Y_mesh(:,m), 'Color', [0.2 0.2 0.2]);
636     end
637 end
638 title('Physical H-Grid', 'Color', 'k');
639 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
640 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
641
642 % 3. Computational Grid
643 figure('Name', 'Computational Grid', 'Color', 'w');
644 set(gcf, 'Color', 'w');
645 hold on; grid on; axis equal;
646 plot(ETA1, ETA2, 'b-', 'LineWidth', 0.5);
647 plot(ETA2, ETA1, 'b-', 'LineWidth', 0.5);
648 title('Computational Grid (\xi - \eta)', 'Color', 'k');
649 xlabel('\eta_1', 'Color', 'k'); ylabel('\eta_2', 'Color', 'k');
650 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
651
652 % 4. Convergence History
653 figure('Name', 'Convergence', 'Color', 'w');
654 set(gcf, 'Color', 'w');
655 plot(2:1:iter, log10(Residual_History), 'b-', 'LineWidth', 1.5);
656 grid on;
657 title('Convergence History', 'Color', 'k');
658 xlabel('Iteration (n)', 'Color', 'k'); ylabel('log_{10}(RMS Error)', '
    Color', 'k');
659 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
660
661 % 5. Streamlines
662 figure('Name', 'Streamlines', 'Color', 'w');
663 set(gcf, 'Color', 'w');
664 hold on; grid on; axis equal;
665 contour(X_mesh, Y_mesh, Psi, 50, 'LineWidth', 1.0);
666 fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.7 0.7 0.7], '
    EdgeColor', 'k'); % Grey Fill
667 title(['Streamlines (\alpha = ' num2str(AoA_deg) '~{\circ})'], 'Color',
    'k');
668 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');

```

```

669 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
670
671 % 6. Vector Field
672 figure('Name', 'Vector Field', 'Color', 'w');
673 set(gcf, 'Color', 'w');
674 hold on; grid on; axis equal;
675 q = quiver(X_mesh, Y_mesh, vel_u, vel_v);
676 q.Color = 'b';
677 q.LineWidth = 1.0;
678 plot(x_upp, y_upp, 'k', 'LineWidth', 2);
679 plot(x_low, y_low, 'k', 'LineWidth', 2);
680 title('Velocity Vector Field', 'Color', 'k');
681 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
682 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
683
684 % 7. Surface Velocity Distribution
685 figure('Name', 'Surface Velocity', 'Color', 'w');
686 set(gcf, 'Color', 'w');
687 hold on; grid on;
688 plot(x_seg2, Vel_airfoil_l/U_free, 'b-', 'LineWidth', 2);
689 plot(x_seg2, Vel_airfoil_u/U_free, 'r-', 'LineWidth', 2);
690 title('Normalized Velocity on Airfoil Surface', 'Color', 'k');
691 xlabel('x/c', 'Color', 'k'); ylabel('V / U_{\infty}', 'Color', 'k');
692 % FIX: Explicitly set Legend Color to White and Text to Black
693 legend('Lower Surface', 'Upper Surface', 'Color', 'w', 'TextColor', 'k')
    ;
694 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
695
696 % 8. Velocity Contour Lines (Restored)
697 figure('Name', 'Velocity Contour Lines', 'Color', 'w');
698 set(gcf, 'Color', 'w');
699 hold on; grid on; axis equal;
700 contour(X_mesh, Y_mesh, Vel_mag ./ U_free, 50);
701 fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], '
    EdgeColor', 'k');
702 title('Velocity Contour Lines', 'Color', 'k');
703 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
704 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
705
706 % 9. Velocity Magnitude Color Map
707 figure('Name', 'Velocity Magnitude Color', 'Color', 'w');
708 set(gcf, 'Color', 'w');
709 hold on; axis equal;
710 surf(X_mesh, Y_mesh, Vel_mag ./ U_free);
711 view(2); shading interp; colormap jet;
712 cb = colorbar; cb.Color = 'k';
713 fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], '
    EdgeColor', 'k');
714 title('Velocity Magnitude Contours', 'Color', 'k');
715 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
716 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k');
717
718 % 10. Pressure Coefficient Distribution
719 figure('Name', 'Cp Distribution', 'Color', 'w');

```

```

720 set(gcf, 'Color', 'w');
721 hold on; grid on;
722 plot(x_seg2, Cp_l, 'b-', 'LineWidth', 2);
723 plot(x_seg2, Cp_u, 'r-', 'LineWidth', 2);
724 set(gca, 'YDir', 'reverse');
725 title('C_p Distribution', 'Color', 'k');
726 xlabel('x/c', 'Color', 'k'); ylabel('C_p', 'Color', 'k');
727 % FIX: Explicitly set Legend Color to White and Text to Black
728 legend('Lower Surface', 'Upper Surface', 'Color', 'w', 'TextColor', 'k')
    ;
729 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
730
731 % 11. Pressure Contour Lines (Restored)
732 figure('Name', 'Pressure Contour Lines', 'Color', 'w');
733 set(gcf, 'Color', 'w');
734 hold on; grid on; axis equal;
735 contour(X_mesh, Y_mesh, Cp_field, 50);
736 fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], '
    EdgeColor', 'k');
737 title('Pressure Contour Lines', 'Color', 'k');
738 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
739 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k', 'GridColor', 'k', '
    GridAlpha', 0.2);
740
741 % 12. Pressure Color Map
742 figure('Name', 'Pressure Color Map', 'Color', 'w');
743 set(gcf, 'Color', 'w');
744 hold on; axis equal;
745 contourf(X_mesh, Y_mesh, Cp_field, 50, 'LineColor', 'none');
746 cb = colorbar; cb.Color = 'k';
747 colormap jet;
748 fill([x_upp fliplr(x_low)], [y_upp fliplr(y_low)], [0.8 0.8 0.8], '
    EdgeColor', 'k');
749 title('Pressure Coefficient Contours', 'Color', 'k');
750 xlabel('x', 'Color', 'k'); ylabel('y', 'Color', 'k');
751 set(gca, 'Color', 'w', 'XColor', 'k', 'YColor', 'k');

```