Sensitivity Analysis to find Airfoil of Desired C_L

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import matplotlib.cm as cm

#
==========================================================================
==
# FIXED PARAMETERS
#
==========================================================================
==
U_inf = 80.0       # Free-stream velocity [m/s]
R = 1.5            # Circle radius [m]
b = 0.8            # Joukowski parameter [m]
rho = 1.225        # Air density [kg/m³]
x0 = 0.0           # Circle center x (symmetric airfoil)
y0 = 0.0           # Circle center y (symmetric airfoil)

#
==========================================================================
==
# ANALYTICAL FUNCTIONS WITH ANGLE OF ATTACK
#
==========================================================================
==
def generate_symmetric_airfoil(R, b, n_points=500):
    """Generate symmetric Joukowski airfoil"""
    theta = np.linspace(0, 2*np.pi, n_points)
    z_circle = R * np.exp(1j * theta)
    z_airfoil = z_circle + b**2 / z_circle
    return np.real(z_airfoil), np.imag(z_airfoil)

def velocity_with_aoa(theta, R, b, U_inf, alpha_rad, Gamma):
    """Calculate velocity on airfoil surface with angle of attack"""
    z = R * np.exp(1j * theta)
    z0 = x0 + 1j*y0

    # Complex velocity in z-plane with angle of attack
    W_z = U_inf * np.exp(-1j*alpha_rad) - \
          U_inf * np.exp(1j*alpha_rad) * R**2/(z - z0)**2 + \
          (1j*Gamma)/(2*np.pi*(z - z0))
```

```
    # Derivative of Joukowski transformation
    dz_prime_dz = 1 - b**2 / z**2

    # Velocity in airfoil plane
    W_z_prime = W_z / dz_prime_dz

    return W_z_prime

def pressure_coefficient_with_aoa(theta, R, b, U_inf, alpha_rad, Gamma):
    """Calculate pressure coefficient with angle of attack"""
    W = velocity_with_aoa(theta, R, b, U_inf, alpha_rad, Gamma)
    V = np.abs(W)
    return 1 - (V/U_inf)**2

def calculate_circulation_for_aoa(alpha_rad, R, U_inf, b):
    """Calculate required circulation for Kutta condition at given α"""
    # For symmetric airfoil, circulation to satisfy Kutta condition
    # Γ = 4πRU∞ sin(α + β) where β depends on airfoil shape
    # Simplified: Γ = 4πRU∞ sin(α) for symmetric airfoil at small α
    return 4 * np.pi * R * U_inf * np.sin(alpha_rad)

def calculate_lift_coefficient(alpha_rad, R, b, U_inf):
    """Calculate lift coefficient using thin airfoil theory
approximation"""
    # For symmetric airfoil: C_L = 2πα
    # More accurate: C_L = 2π sin(α) for larger angles
    c = 2 * (R + b**2/R)  # Chord length

    # Thin airfoil theory: C_L = 2πα (radians)
    C_L_thin = 2 * np.pi * alpha_rad

    # With Kutta-Joukowski: C_L = 2Γ/(U∞c)
    Gamma = calculate_circulation_for_aoa(alpha_rad, R, U_inf, b)
    C_L_KJ = 2 * Gamma / (U_inf * c)

    return C_L_thin, C_L_KJ, Gamma, c

def calculate_stall_onset(alpha_deg):
    """Estimate stall onset based on angle of attack"""
    # Simple stall model
    if abs(alpha_deg) < 10:
        return "Attached flow"
    elif 10 <= abs(alpha_deg) < 15:
        return "Trailing edge separation"
    elif 15 <= abs(alpha_deg) < 20:
```

```
        return "Partial leading edge separation"
    else:
        return "Fully separated flow (stalled)"

#
==========================================================================
==
# MAIN ANALYSIS - ANGLE OF ATTACK EFFECTS
#
==========================================================================
==
# Define range of angle of attack values
alpha_deg_values = np.linspace(-15, 25, 17)  # -15° to 25° in 2.5°
increments
alpha_rad_values = np.deg2rad(alpha_deg_values)

# Generate base airfoil (same geometry for all α)
x_airfoil_base, y_airfoil_base = generate_symmetric_airfoil(R, b)
chord = np.max(x_airfoil_base) - np.min(x_airfoil_base)

# Calculate parameters for each angle of attack
results = []
for alpha_deg, alpha_rad in zip(alpha_deg_values, alpha_rad_values):
    # Calculate lift coefficients
    C_L_thin, C_L_KJ, Gamma, c = calculate_lift_coefficient(alpha_rad, R,
b, U_inf)

    # Calculate pressure distribution at key points
    theta_points = np.array([0, np.pi/2, np.pi, 3*np.pi/2])  # TE, top,
LE, bottom
    Cp_points = []
    V_points = []

    for theta in theta_points:
        W = velocity_with_aoa(theta, R, b, U_inf, alpha_rad, Gamma)
        V = np.abs(W)
        Cp = 1 - (V/U_inf)**2
        Cp_points.append(Cp)
        V_points.append(V)

    # Calculate full pressure distribution for plotting
    theta_full = np.linspace(0, 2*np.pi, 200)
    Cp_full = np.array([pressure_coefficient_with_aoa(t, R, b, U_inf,
alpha_rad, Gamma)
                        for t in theta_full])
```

```
    # Determine flow state
    flow_state = calculate_stall_onset(alpha_deg)

    results.append({
        'alpha_deg': alpha_deg,
        'alpha_rad': alpha_rad,
        'C_L_thin': C_L_thin,
        'C_L_KJ': C_L_KJ,
        'Gamma': Gamma,
        'Cp_TE': Cp_points[0],      # Trailing edge
        'Cp_top': Cp_points[1],     # Top (max thickness)
        'Cp_LE': Cp_points[2],      # Leading edge
        'Cp_bottom': Cp_points[3],  # Bottom (max thickness)
        'V_TE': V_points[0],
        'V_top': V_points[1],
        'V_LE': V_points[2],
        'V_bottom': V_points[3],
        'Cp_full': Cp_full,
        'theta_full': theta_full,
        'flow_state': flow_state,
        'chord': c
    })

#
==========================================================================
==
# VISUALIZATION: COMPREHENSIVE ANGLE OF ATTACK ANALYSIS
#
==========================================================================
==
fig = plt.figure(figsize=(18, 12))
fig.suptitle('Impact of Angle of Attack on Joukowski Airfoil
Aerodynamics\n' +
             f'R={R}m, b={b}m, U∞={U_inf}m/s, Chord={chord:.2f}m',
             fontsize=16, fontweight='bold', y=0.98)

gs = GridSpec(3, 3, height_ratios=[1, 1, 1], hspace=0.3, wspace=0.3)

# Color map for different angles of attack
colors = cm.RdYlBu(np.linspace(0, 1, len(alpha_deg_values)))

# Plot 1: Lift Curve (C_L vs α)
ax1 = plt.subplot(gs[0, 0])
alpha_list = [res['alpha_deg'] for res in results]
```

```
C_L_KJ_list = [res['C_L_KJ'] for res in results]
C_L_thin_list = [res['C_L_thin'] for res in results]

# Plot experimental/theoretical curves
ax1.plot(alpha_list, C_L_KJ_list, 'b-o', linewidth=2.5, markersize=6,
         label='Kutta-Joukowski Theory', markerfacecolor='white')

ax1.plot(alpha_list, C_L_thin_list, 'r--', linewidth=2, alpha=0.7,
         label='Thin Airfoil Theory (C_L = 2πα)')

# Mark stall region (approximate)
stall_start = 12  # Approximate stall angle
stall_region = [a for a in alpha_list if a >= stall_start]
if stall_region:
    stall_min = min(stall_region)
    stall_max = max(stall_region)
    ax1.axvspan(stall_min, stall_max, alpha=0.2, color='red', label='Stall
Region')

ax1.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax1.set_ylabel('Lift Coefficient, C_L', fontsize=11)
ax1.set_title('Lift Curve: C_L vs Angle of Attack', fontsize=12)
ax1.grid(True, alpha=0.2)
ax1.legend(loc='upper left', fontsize=9)
ax1.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
ax1.axvline(x=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)

# Add lift curve slope
if len(alpha_list) > 1 and len(C_L_KJ_list) > 1:
    # Calculate approximate lift curve slope near α=0
    near_zero_idx = np.where(np.abs(alpha_list) <= 5)[0]
    if len(near_zero_idx) > 1:
        alpha_near_zero = np.array(alpha_list)[near_zero_idx]
        C_L_near_zero = np.array(C_L_KJ_list)[near_zero_idx]
        coeffs = np.polyfit(alpha_near_zero, C_L_near_zero, 1)
        dC_L_dalpha = coeffs[0]  # Per degree
        dC_L_dalpha_rad = dC_L_dalpha * (180/np.pi)  # Per radian

        ax1.text(0.05, 0.95, f'Lift curve slope:\n{dC_L_dalpha:.3f} per
degree\n' +
                 f'({dC_L_dalpha_rad:.2f} per radian)',
                 transform=ax1.transAxes, verticalalignment='top',
                 bbox=dict(boxstyle='round', facecolor='white',
alpha=0.8),
                 fontsize=9)
```

```
# Plot 2: Circulation vs Angle of Attack
ax2 = plt.subplot(gs[0, 1])
Gamma_list = [res['Gamma'] for res in results]

ax2.plot(alpha_list, Gamma_list, 'g-^', linewidth=2.5, markersize=6,
         label='Circulation, Γ', markerfacecolor='white')

# Theoretical curve: Γ = 4πRU∞ sin(α)
alpha_fine = np.linspace(min(alpha_deg_values), max(alpha_deg_values),
100)
Gamma_theory = 4 * np.pi * R * U_inf * np.sin(np.deg2rad(alpha_fine))
ax2.plot(alpha_fine, Gamma_theory, 'k--', linewidth=1.5, alpha=0.6,
         label='Theory: Γ = 4πRU∞sin(α)')

ax2.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax2.set_ylabel('Circulation, Γ [m²/s]', fontsize=11)
ax2.set_title('Circulation vs Angle of Attack', fontsize=12)
ax2.grid(True, alpha=0.2)
ax2.legend(loc='upper left', fontsize=9)
ax2.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
ax2.axvline(x=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)

# Plot 3: Pressure Coefficient at Key Points vs α
ax3 = plt.subplot(gs[0, 2])
Cp_LE_list = [res['Cp_LE'] for res in results]
Cp_top_list = [res['Cp_top'] for res in results]
Cp_bottom_list = [res['Cp_bottom'] for res in results]

ax3.plot(alpha_list, Cp_LE_list, 'r-o', linewidth=2, markersize=5,
         label='Leading Edge (Cp_LE)', markerfacecolor='white')
ax3.plot(alpha_list, Cp_top_list, 'b-s', linewidth=2, markersize=5,
         label='Top Surface (Cp_top)', markerfacecolor='white')
ax3.plot(alpha_list, Cp_bottom_list, 'g-^', linewidth=2, markersize=5,
         label='Bottom Surface (Cp_bottom)', markerfacecolor='white')

ax3.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax3.set_ylabel('Pressure Coefficient, C_p', fontsize=11)
ax3.set_title('Pressure Coefficient at Key Points vs α', fontsize=12)
ax3.grid(True, alpha=0.2)
ax3.legend(loc='best', fontsize=8)
ax3.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
ax3.axvline(x=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
ax3.invert_yaxis()  # Negative Cp (suction) upwards
```

```
# Plot 4: Velocity at Key Points vs α
ax4 = plt.subplot(gs[1, 0])
V_LE_list = [res['V_LE'] for res in results]
V_top_list = [res['V_top'] for res in results]
V_bottom_list = [res['V_bottom'] for res in results]

ax4.plot(alpha_list, V_LE_list, 'r-o', linewidth=2, markersize=5,
         label='Leading Edge (V_LE)', markerfacecolor='white')
ax4.plot(alpha_list, V_top_list, 'b-s', linewidth=2, markersize=5,
         label='Top Surface (V_top)', markerfacecolor='white')
ax4.plot(alpha_list, V_bottom_list, 'g-^', linewidth=2, markersize=5,
         label='Bottom Surface (V_bottom)', markerfacecolor='white')

ax4.axhline(y=U_inf, color='k', linestyle='--', alpha=0.5,
            label=f'Free Stream (U∞ = {U_inf} m/s)')

ax4.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax4.set_ylabel('Velocity [m/s]', fontsize=11)
ax4.set_title('Surface Velocity at Key Points vs α', fontsize=12)
ax4.grid(True, alpha=0.2)
ax4.legend(loc='upper left', fontsize=8)

# Add maximum velocity ratio
if len(V_top_list) > 0:
    max_V_top = max(V_top_list)
    max_V_idx = V_top_list.index(max_V_top)
    max_V_alpha = alpha_list[max_V_idx]
    ax4.axvline(x=max_V_alpha, color='purple', linestyle=':', alpha=0.5,
                label=f'Max top velocity at α={max_V_alpha:.1f}°')

# Plot 5: Pressure Distribution Comparison for Different α
ax5 = plt.subplot(gs[1, 1])
# Select 3 representative angles
rep_indices = [np.argmin(np.abs(np.array(alpha_list) - a)) for a in [-5,
5, 15]]
line_styles = ['-', '--', '-.']
line_colors = ['blue', 'red', 'green']

for idx, style, color in zip(rep_indices, line_styles, line_colors):
    res = results[idx]

    # Get airfoil coordinates for x-position
    x_airfoil, _ = generate_symmetric_airfoil(R, b, 200)
    x_norm = (x_airfoil - np.min(x_airfoil)) / chord
```

```
    # Sort for plotting
    sort_idx = np.argsort(x_norm)

    label = f'α = {res["alpha_deg"]:.1f}°, C_L = {res["C_L_KJ"]:.3f}'
    ax5.plot(x_norm[sort_idx], res['Cp_full'][sort_idx], style,
             color=color, linewidth=2, label=label)

ax5.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
ax5.set_xlabel('x/c (normalized)', fontsize=11)
ax5.set_ylabel('Pressure Coefficient, C_p', fontsize=11)
ax5.set_title('Pressure Distribution for Different Angles of Attack',
fontsize=12)
ax5.grid(True, alpha=0.2)
ax5.invert_yaxis()
ax5.legend(loc='best', fontsize=8)
ax5.set_xlim(0, 1)

# Plot 6: Airfoil Orientation at Different α
ax6 = plt.subplot(gs[1, 2])
# Show airfoil at different orientations
angles_to_show = [-10, 0, 10, 20]  # Degrees

# Create rotated coordinate systems
for i, alpha_show in enumerate(angles_to_show):
    # Original airfoil coordinates
    x_norm = (x_airfoil_base - np.min(x_airfoil_base)) / chord - 0.5  #
Center at 0
    y_norm = y_airfoil_base / chord

    # Rotate airfoil
    alpha_rad_show = np.deg2rad(alpha_show)
    x_rot = x_norm * np.cos(alpha_rad_show) - y_norm *
np.sin(alpha_rad_show)
    y_rot = x_norm * np.sin(alpha_rad_show) + y_norm *
np.cos(alpha_rad_show)

    # Offset for visualization
    x_offset = i * 0.6
    ax6.plot(x_rot + x_offset, y_rot, 'b-', linewidth=1.5, alpha=0.8)
    ax6.fill(x_rot + x_offset, y_rot, 'lightblue', alpha=0.3)

    # Add angle indicator
    ax6.text(x_offset, 0.15, f'α={alpha_show}°', fontsize=9,
             ha='center', va='center',
             bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))
```

```
    # Add free stream vector
    ax6.arrow(x_offset - 0.3, 0, 0.2, 0, head_width=0.03,
head_length=0.05,
              fc='red', ec='red', alpha=0.7)

ax6.set_xlabel('Position', fontsize=11)
ax6.set_ylabel('Position', fontsize=11)
ax6.set_title('Airfoil Orientation at Different Angles of Attack',
fontsize=12)
ax6.grid(True, alpha=0.2)
ax6.axis('equal')
ax6.set_xlim(-1, len(angles_to_show)*0.6)
ax6.set_ylim(-0.3, 0.3)

# Plot 7: Flow Regimes vs Angle of Attack
ax7 = plt.subplot(gs[2, 0])
# Create flow regime map
flow_regimes = {
    'Deep Stall (α > 20°)': (20, 25, 'red'),
    'Stall (15° < α ≤ 20°)': (15, 20, 'orange'),
    'Post-Stall Buffer (12° < α ≤ 15°)': (12, 15, 'yellow'),
    'Maximum CL (8° < α ≤ 12°)': (8, 12, 'lightgreen'),
    'Linear Region (0° < α ≤ 8°)': (0, 8, 'green'),
    'Zero Lift (α = 0°)': (-0.5, 0.5, 'gray'),
    'Negative Lift Linear (-8° ≤ α < 0°)': (-8, 0, 'lightblue'),
    'Negative Stall (α < -8°)': (-15, -8, 'blue')
}

y_pos = 1
for regime, (alpha_min, alpha_max, color) in flow_regimes.items():
    ax7.barh(y_pos, alpha_max - alpha_min, left=alpha_min, height=0.6,
             color=color, edgecolor='black', alpha=0.7)
    ax7.text((alpha_min + alpha_max)/2, y_pos, regime,
             ha='center', va='center', fontsize=7)
    y_pos -= 1

ax7.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax7.set_title('Flow Regimes vs Angle of Attack', fontsize=12)
ax7.grid(True, alpha=0.2, axis='x')
ax7.set_yticks([])
ax7.set_xlim(-16, 26)
ax7.axvline(x=0, color='k', linestyle='-', alpha=0.3, linewidth=1)

# Plot 8: Aerodynamic Efficiency
```

```
ax8 = plt.subplot(gs[2, 1])
# Simple drag model: C_D = C_D0 + k·C_L²
C_D0 = 0.01  # Zero-lift drag coefficient
k = 0.05     # Induced drag factor

# Calculate drag and L/D ratio
C_L_values = np.array(C_L_KJ_list)
C_D_values = C_D0 + k * C_L_values**2
L_over_D = C_L_values / C_D_values

ax8.plot(alpha_list, L_over_D, 's-', color='purple', linewidth=2.5,
markersize=6,
         label='Lift-to-Drag Ratio (L/D)', markerfacecolor='white')

# Find maximum L/D
max_LD_idx = np.argmax(L_over_D)
max_LD_alpha = alpha_list[max_LD_idx]
max_LD_value = L_over_D[max_LD_idx]

ax8.axvline(x=max_LD_alpha, color='purple', linestyle='--', alpha=0.5,
            label=f'Max L/D at α={max_LD_alpha:.1f}°')

ax8.set_xlabel('Angle of Attack, α [degrees]', fontsize=11)
ax8.set_ylabel('Lift-to-Drag Ratio (L/D)', fontsize=11)
ax8.set_title('Aerodynamic Efficiency vs Angle of Attack', fontsize=12)
ax8.grid(True, alpha=0.2)
ax8.legend(loc='upper right', fontsize=8)
ax8.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)

# Add drag coefficient on secondary axis
ax8b = ax8.twinx()
ax8b.plot(alpha_list, C_D_values, '^-', color='gray', linewidth=1.5,
markersize=4,
          label='Drag Coefficient (C_D)', alpha=0.7)
ax8b.set_ylabel('Drag Coefficient, C_D', fontsize=11, color='gray')
ax8b.tick_params(axis='y', labelcolor='gray')

# Plot 9: Design Guidelines and Summary
ax9 = plt.subplot(gs[2, 2])
ax9.axis('off')

# Extract key performance metrics
# Find key angles
zero_lift_idx = np.argmin(np.abs(C_L_KJ_list))
zero_lift_alpha = alpha_list[zero_lift_idx]
```

```
stall_start_idx = np.argmax(C_L_KJ_list)  # Simple stall detection (max
CL)
stall_alpha = alpha_list[stall_start_idx]
max_CL = C_L_KJ_list[stall_start_idx]

summary_text = f"""
DESIGN GUIDELINES: ANGLE OF ATTACK

Key Angles for Symmetric Airfoil:
• Zero-lift angle: α = {zero_lift_alpha:.1f}° (C_L =
{C_L_KJ_list[zero_lift_idx]:.3f})
• Max L/D angle: α = {max_LD_alpha:.1f}° (L/D = {max_LD_value:.1f})
• Stall onset: α ≈ {stall_alpha:.1f}° (C_L,max = {max_CL:.3f})

Performance Metrics at Key α:
1. Cruise (α = {max_LD_alpha:.1f}°):
   • C_L = {C_L_KJ_list[max_LD_idx]:.3f}, L/D = {max_LD_value:.1f}
   • Optimal efficiency

2. Takeoff/Climb (α = 8-12°):
   • Higher C_L for lower speed
   • Moderate L/D penalty

3. Stall (α > {stall_alpha:.1f}°):
   • Flow separation begins
   • Rapid drag increase
   • Lift decrease or plateau

Theoretical Relationships:
• Thin airfoil theory: C_L = 2πα (radians)
• Actual: C_L = {dC_L_dalpha:.3f}α + C_L0 (linear region)
• Circulation: Γ = 4πRU∞sin(α)

Flow States by α:
• α < 0°: Negative lift, inverted flight
• 0-8°: Linear lift region
• 8-12°: Pre-stall, high lift
• 12-15°: Initial separation
• >15°: Fully separated (stall)
"""

ax9.text(0.05, 0.95, summary_text, transform=ax9.transAxes,
         verticalalignment='top', fontsize=8,
         bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.3))
```

```
plt.tight_layout()
plt.show()

#
==========================================================================
==
# ADDITIONAL ANALYSIS: DETAILED FLOW VISUALIZATION
#
==========================================================================
==
print("="*80)
print("DETAILED ANGLE OF ATTACK ANALYSIS")
print("="*80)

print("\nFixed Parameters:")
print(f"  • Circle radius (R): {R} m")
print(f"  • Joukowski parameter (b): {b} m")
print(f"  • Free stream velocity (U∞): {U_inf} m/s")
print(f"  • Chord length: {chord:.3f} m")

print(f"\nAngle of Attack Range: α = [{min(alpha_deg_values):.1f}°,
{max(alpha_deg_values):.1f}°]")

print("\nKey Performance Metrics:")
print("-"*50)

# Find and report key angles
print(f"\n1. Zero-Lift Angle:")
print(f"   • Theoretical for symmetric airfoil: α = 0°")
print(f"   • Calculated zero-lift: α = {zero_lift_alpha:.2f}°")
print(f"   • C_L at α=0°: {C_L_KJ_list[zero_lift_idx]:.4f}")

print(f"\n2. Lift Curve Slope:")
print(f"   • Near α=0°: dC_L/dα = {dC_L_dalpha:.3f} per degree")
print(f"   • In radians: dC_L/dα = {dC_L_dalpha_rad:.2f} per radian")
print(f"   • Theoretical (thin airfoil): dC_L/dα = 2π = {2*np.pi:.2f} per
radian")

print(f"\n3. Maximum Performance:")
print(f"   • Maximum C_L: {max_CL:.3f} at α = {stall_alpha:.1f}°")
print(f"   • Maximum L/D: {max_LD_value:.1f} at α = {max_LD_alpha:.1f}°")
print(f"   • Corresponding C_L at max L/D: {C_L_KJ_list[max_LD_idx]:.3f}")

print(f"\n4. Flow Regime Boundaries:")
```

```
for regime, (alpha_min, alpha_max, _) in flow_regimes.items():
    if alpha_min <= 0 <= alpha_max or "Zero" in regime:
        print(f"   • {regime}: α ∈ [{alpha_min}, {alpha_max}]°")

print("\n" + "-"*50)
print("PHYSICAL INTERPRETATION:")
print("-"*50)

print("\n1. HOW ANGLE OF ATTACK WORKS:")
print("   • Changes effective camber seen by flow")
print("   • Alters stagnation point location")
print("   • Modifies pressure distribution asymmetry")
print("   • Controls circulation strength (via Kutta condition)")

print("\n2. AERODYNAMIC MECHANISMS:")
print("   • Positive α: Increases upper surface suction")
print("   • Positive α: Decreases lower surface pressure")
print("   • Net effect: Increased pressure difference → Increased lift")
print("   • At high α: Flow separation reduces lift, increases drag")

print("\n3. CRITICAL ANGLES:")
print("   • α = 0°: Symmetric flow, zero lift (for symmetric airfoil)")
print("   • α = α_max L/D: Optimal cruise efficiency")
print("   • α = α_stall: Maximum lift before separation")
print("   • α > α_stall: Separated flow, high drag, reduced lift")

print("\n" + "-"*50)
print("DESIGN IMPLICATIONS:")
print("-"*50)

print("\n1. OPERATIONAL ANGLES OF ATTACK:")
print("   • Cruise: α = 2-4° (maximum efficiency)")
print("   • Takeoff: α = 8-12° (high lift)")
print("   • Landing: α = 10-14° (maximum lift at low speed)")
print("   • Stall recovery: Reduce α below 10°")

print("\n2. AIRCRAFT DESIGN CONSIDERATIONS:")
print("   • Wing incidence angle sets cruise α")
print("   • High-lift devices increase effective α")
print("   • Stall characteristics depend on α at stall")
print("   • Control surfaces modify local α")

print("\n3. SAFETY MARGINS:")
print("   • Stall margin: Typically 3-5° below stall α")
print("   • Buffet onset: Usually 2-3° below stall")
```

```
print("   • Maneuver limits: Based on α limits")

print("="*80)

#
==========================================================================
==
# VISUALIZATION 2: DETAILED FLOW FIELD AT KEY ANGLES
#
==========================================================================
==
fig2, axes2 = plt.subplots(2, 3, figsize=(15, 10))
fig2.suptitle('Detailed Flow Analysis at Key Angles of Attack',
              fontsize=14, fontweight='bold')

# Select key angles for detailed analysis
key_angles = [-5, 0, 5, 10, 15, 20]  # Degrees
key_titles = ['Negative α (-5°)', 'Zero α (0°)', 'Low Positive α (5°)',
              'Moderate α (10°)', 'High α (15°)', 'Near Stall (20°)']

for i, (alpha_deg, title) in enumerate(zip(key_angles, key_titles)):
    ax = axes2[i//3, i%3]

    # Find closest result
    alpha_idx = np.argmin(np.abs(np.array(alpha_list) - alpha_deg))
    res = results[alpha_idx]

    # Generate airfoil coordinates
    x_airfoil, y_airfoil = generate_symmetric_airfoil(R, b, 200)
    x_norm = (x_airfoil - np.min(x_airfoil)) / chord

    # Sort for plotting
    sort_idx = np.argsort(x_norm)

    # Plot pressure distribution
    ax.plot(x_norm[sort_idx], res['Cp_full'][sort_idx], 'b-', linewidth=2)

    # Fill pressure areas
    # Upper surface (typically lower pressure)
    upper_idx = np.where(res['Cp_full'][sort_idx] < 0)[0]
    if len(upper_idx) > 0:
        ax.fill_between(x_norm[sort_idx][upper_idx],
                        res['Cp_full'][sort_idx][upper_idx], 0,
                        alpha=0.3, color='blue', label='Suction')
```

```
    # Lower surface (typically higher pressure)
    lower_idx = np.where(res['Cp_full'][sort_idx] > 0)[0]
    if len(lower_idx) > 0:
        ax.fill_between(x_norm[sort_idx][lower_idx],
                        res['Cp_full'][sort_idx][lower_idx], 0,
                        alpha=0.3, color='red', label='Pressure')

    ax.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)
    ax.set_xlabel('x/c', fontsize=9)
    ax.set_ylabel('C_p', fontsize=9)
    ax.set_title(f'{title}\nC_L = {res["C_L_KJ"]:.3f}, Γ =
{res["Gamma"]:.1f} m²/s', fontsize=10)
    ax.grid(True, alpha=0.2)
    ax.invert_yaxis()
    ax.set_xlim(0, 1)

    # Add flow state annotation
    ax.text(0.05, 0.05, res['flow_state'], transform=ax.transAxes,
            fontsize=8, bbox=dict(boxstyle='round', facecolor='white',
alpha=0.8))

plt.tight_layout()
plt.show()

#
==========================================================================
==
# FINAL SUMMARY
#
==========================================================================
==
print("\n" + "="*80)
print("FINAL SUMMARY: ANGLE OF ATTACK EFFECTS")
print("="*80)

print("\n1. KEY FINDINGS:")
print("   • Angle of attack is the PRIMARY CONTROL for lift generation")
print("   • Linear relationship between α and C_L for α < 8-10°")
print("   • Maximum lift occurs at stall angle (typically 12-16°)")
print("   • Optimal efficiency at lower α (typically 2-6°)")

print("\n2. MATHEMATICAL RELATIONSHIPS:")
print("   • Thin airfoil theory: C_L = 2πα (radians)")
print("   • Kutta condition: Γ = 4πRU∞sin(α)")
print("   • Joukowski lift: C_L = 2Γ/(U∞c) = 8πR sin(α)/c")
```

```
print("\n3. PRACTICAL DESIGN GUIDELINES:")
print("   • Design cruise α for maximum L/D")
print("   • Provide sufficient stall margin (3-5° below stall)")
print("   • Consider α limits for different flight phases")
print("   • Account for α variation along wing span")

print("\n4. SAFETY CONSIDERATIONS:")
print("   • Stall occurs at critical α, not critical speed")
print("   • Recovery requires reducing α")
print("   • High α during takeoff/landing requires careful control")
print("   • Gusts can cause sudden α changes")

print("\nThis completes the comprehensive analysis of angle of attack
effects")
print("on Joukowski airfoil aerodynamics.")
print("="*80)
```