Effect of Chamber on Joukowski Airfoil

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

#
==========================================================================
==
# FIXED PARAMETERS
#
==========================================================================
==
U_inf = 80.0       # Free-stream velocity [m/s]
Gamma = 30.0       # Circulation [m²/s]
b = 0.8            # Fixed Joukowski parameter [m]
rho = 1.225        # Air density [kg/m³]

#
==========================================================================
==
# ANALYTICAL FUNCTIONS
#
==========================================================================
==
def geometric_parameters(R, b):
    """Calculate all geometric parameters analytically"""
    a = R + b**2/R      # Semi-major axis
    h = R - b**2/R      # Semi-minor axis
    chord = 2 * a
    thickness = 2 * h
    thickness_ratio = h / a

    return {
        'R': R,
        'a': a,
        'h': h,
        'chord': chord,
        'thickness': thickness,
        'thickness_ratio': thickness_ratio
    }

def aerodynamic_parameters(R, b, U_inf, Gamma):
```

```python
    """Calculate all aerodynamic parameters analytically"""
    # Geometric parameters
    geom = geometric_parameters(R, b)
    chord = geom['chord']

    # Key velocities
    V_LE = Gamma / (2 * np.pi * R * abs(1 - b**2/R**2))
    V_TE = V_LE  # Symmetric airfoil
    V_MC = (2*U_inf + Gamma/(2*np.pi*R)) / (1 + b**2/R**2)

    # Aerodynamic coefficients
    C_L = 2 * Gamma / (U_inf * chord)
    L_prime = rho * U_inf * Gamma

    return {
        'V_LE': V_LE,
        'V_TE': V_TE,
        'V_MC': V_MC,
        'C_L': C_L,
        'L_prime': L_prime
    }

def sensitivity_derivatives(R, b, U_inf, Gamma):
    """Calculate analytical derivatives with respect to R"""
    # First derivatives
    dc_dR = 2 * (1 - b**2/R**2)
    dt_dR = 2 * (1 + b**2/R**2)
    dtc_dR = 4 * b**2 / (R**2 * (1 + b**2/R**2)**2)

    dC_L_dR = -Gamma * (1 - b**2/R**2) / (U_inf * (R + b**2/R)**2)
    dV_LE_dR = -Gamma * (1 + b**2/R**2) / (2 * np.pi * R**2 * (1 -
b**2/R**2)**2)

    # Second derivatives
    d2c_dR2 = 4 * b**2 / R**3
    d2t_dR2 = -4 * b**2 / R**3
    d2tc_dR2 = -8 * b**2 * (3*R**2 + b**2) / (R**4 * (1 + b**2/R**2)**3)

    return {
        'dc_dR': dc_dR,
        'dt_dR': dt_dR,
        'dtc_dR': dtc_dR,
        'dC_L_dR': dC_L_dR,
        'dV_LE_dR': dV_LE_dR,
        'd2c_dR2': d2c_dR2,
        'd2t_dR2': d2t_dR2,
        'd2tc_dR2': d2tc_dR2
    }

def generate_airfoil(R, b, n_points=500):
    """Generate airfoil coordinates for given R and b"""
    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 calculate_pressure_distribution(R, b, U_inf, Gamma, n_points=200):
    """Calculate pressure coefficient distribution"""
    theta = np.linspace(0, 2*np.pi, n_points)
    Cp = np.zeros_like(theta)

    for i, t in enumerate(theta):
        z = R * np.exp(1j * t)
        W_z = U_inf * (1 - R**2/z**2) + 1j * Gamma/(2*np.pi*z)
        dz_prime_dz = 1 - b**2/z**2
        W_z_prime = W_z / dz_prime_dz
        V = np.abs(W_z_prime)
        Cp[i] = 1 - (V/U_inf)**2

    # Generate airfoil coordinates for x-position
    x_airfoil, _ = generate_airfoil(R, b, n_points)
    x_norm = (x_airfoil - min(x_airfoil)) / (max(x_airfoil) -
min(x_airfoil))

    return x_norm, Cp

#
=============================================================================
==
# MAIN ANALYSIS CODE
#
=============================================================================
==
# Define range of R values (must be > b for valid transformation)
R_values = np.linspace(b * 1.01, 3.0, 15)  # R from just above b to 3.0 m

# Calculate all parameters for each R
results = []
for R in R_values:
    geom = geometric_parameters(R, b)
    aero = aerodynamic_parameters(R, b, U_inf, Gamma)
    sens = sensitivity_derivatives(R, b, U_inf, Gamma)

    # Generate airfoil coordinates
    x_airfoil, y_airfoil = generate_airfoil(R, b)

    # Calculate pressure distribution
    x_norm, Cp = calculate_pressure_distribution(R, b, U_inf, Gamma)

    results.append({
        'R': R,
        'geom': geom,
        'aero': aero,
        'sens': sens,
        'x_airfoil': x_airfoil,
        'y_airfoil': y_airfoil,
        'x_norm': x_norm,
        'Cp': Cp
    })

#
============================================================================
==
# VISUALIZATION - COMPREHENSIVE PLOTS
#
============================================================================
==
fig = plt.figure(figsize=(20, 16))
fig.suptitle(f'Impact of Circle Radius (R) on Joukowski Airfoil
Characteristics\n(b={b} m, U∞={U_inf} m/s, Γ={Gamma} m²/s)',
             fontsize=18, fontweight='bold', y=0.98)

# Create custom layout
gs = GridSpec(4, 4, height_ratios=[1.2, 1, 1, 1], hspace=0.35,
wspace=0.35)
colors = cm.plasma(np.linspace(0, 1, len(R_values)))

# Plot 1: Airfoil Geometry Evolution
ax1 = plt.subplot(gs[0, :2])
for idx, (res, color) in enumerate(zip(results, colors)):
    # Normalize coordinates for comparison
    chord = res['geom']['chord']
    x_norm = (res['x_airfoil'] - min(res['x_airfoil'])) / chord
    y_norm = res['y_airfoil'] / chord
    ax1.plot(x_norm, y_norm, color=color, linewidth=1.5, alpha=0.8)

    # Label selected airfoils
    if idx % 3 == 0:
        label =
f'R={res["R"]:.2f}m\nc={chord:.2f}m\nt/c={res["geom"]["thickness_ratio"]:.
3f}'
        ax1.text(x_norm[100], y_norm[100], f'R={res["R"]:.2f}',
                fontsize=8, ha='center', va='center',
                bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))

ax1.set_xlabel('x/c (normalized)', fontsize=12, fontweight='bold')
ax1.set_ylabel('y/c (normalized)', fontsize=12, fontweight='bold')
ax1.set_title('Airfoil Geometry Evolution with Increasing R', fontsize=13,
fontweight='bold')
ax1.grid(True, alpha=0.3, linestyle='--')
ax1.axis('equal')
ax1.set_xlim(-0.1, 1.1)
ax1.set_ylim(-0.6, 0.6)
ax1.axhline(y=0, color='k', linestyle=':', alpha=0.3)

# Plot 2: Geometric Parameters vs R
ax2 = plt.subplot(gs[0, 2:])
# Extract data
R_list = [res['R'] for res in results]
chord_list = [res['geom']['chord'] for res in results]
thickness_list = [res['geom']['thickness'] for res in results]
thickness_ratio_list = [res['geom']['thickness_ratio'] for res in results]

# Plot primary parameters
ax2.plot(R_list, chord_list, 'b-o', linewidth=2.5, markersize=8,
         label='Chord (c)', markerfacecolor='white', markeredgewidth=1.5)
ax2.plot(R_list, thickness_list, 'r-s', linewidth=2.5, markersize=8,
         label='Thickness (t)', markerfacecolor='white',
markeredgewidth=1.5)

# Add theoretical asymptotes
R_fine = np.linspace(b*1.01, 3.0, 100)
c_asymptote = 2 * R_fine  # As R → ∞, c ~ 2R
t_asymptote = 2 * R_fine  # As R → ∞, t ~ 2R
ax2.plot(R_fine, c_asymptote, 'b--', alpha=0.5, linewidth=1, label='c ~ 2R
(asymptote)')
ax2.plot(R_fine, t_asymptote, 'r--', alpha=0.5, linewidth=1, label='t ~ 2R
(asymptote)')
```

```
ax2.set_xlabel('Circle Radius (R) [m]', fontsize=12, fontweight='bold')
ax2.set_ylabel('Length [m]', fontsize=12, fontweight='bold')
ax2.set_title('Geometric Parameters vs Circle Radius', fontsize=13,
fontweight='bold')
ax2.grid(True, alpha=0.3, linestyle='--')
ax2.legend(loc='upper left', fontsize=10)

# Add thickness ratio on secondary axis
ax2b = ax2.twinx()
ax2b.plot(R_list, thickness_ratio_list, 'g-^', linewidth=2.5,
markersize=8,
          label='Thickness/Chord (t/c)', markerfacecolor='white',
markeredgewidth=1.5)
ax2b.set_ylabel('Thickness/Chord Ratio', fontsize=12, fontweight='bold',
color='g')
ax2b.tick_params(axis='y', labelcolor='g')
ax2b.set_ylim(0, 1)

# Add combined legend
lines1, labels1 = ax2.get_legend_handles_labels()
lines2, labels2 = ax2b.get_legend_handles_labels()
ax2.legend(lines1 + lines2, labels1 + labels2, loc='upper left',
fontsize=9)

# Plot 3: Aerodynamic Parameters vs R
ax3 = plt.subplot(gs[1, :2])
# Extract aerodynamic data
C_L_list = [res['aero']['C_L'] for res in results]
V_LE_list = [res['aero']['V_LE'] for res in results]
V_MC_list = [res['aero']['V_MC'] for res in results]

# Plot C_L
ax3.plot(R_list, C_L_list, 'b-o', linewidth=3, markersize=8,
         label=r'Lift Coefficient ($C_L$)', markerfacecolor='white')

# Add theoretical asymptote for C_L
C_L_asymptote = Gamma / (U_inf * R_fine)  # As R → ∞, C_L ~ Γ/(U∞R)
ax3.plot(R_fine, C_L_asymptote, 'b--', alpha=0.5, linewidth=1,
label=r'$C_L \sim \Gamma/(U_\infty R)$')

ax3.set_xlabel('Circle Radius (R) [m]', fontsize=12, fontweight='bold')
ax3.set_ylabel(r'Lift Coefficient ($C_L$)', fontsize=12,
fontweight='bold', color='b')
ax3.set_title('Aerodynamic Coefficients vs Circle Radius', fontsize=13,
fontweight='bold')
```

```
ax3.grid(True, alpha=0.3, linestyle='--')
ax3.tick_params(axis='y', labelcolor='b')

# Add velocities on secondary axis
ax3b = ax3.twinx()
ax3b.plot(R_list, V_LE_list, 'r-s', linewidth=2.5, markersize=8,
          label='V_LE (Leading Edge)', markerfacecolor='white')
ax3b.plot(R_list, V_MC_list, 'g-^', linewidth=2.5, markersize=8,
          label='V_MC (Mid-chord)', markerfacecolor='white')
ax3b.set_ylabel('Velocity [m/s]', fontsize=12, fontweight='bold',
color='r')
ax3b.tick_params(axis='y', labelcolor='r')

# Add theoretical asymptote for V_LE
V_LE_asymptote = Gamma / (2 * np.pi * R_fine)  # As R → ∞, V_LE ~ Γ/(2πR)
ax3b.plot(R_fine, V_LE_asymptote, 'r--', alpha=0.5, linewidth=1,
label=r'$V_{LE} \sim \Gamma/(2\pi R)$')

lines3, labels3 = ax3.get_legend_handles_labels()
lines3b, labels3b = ax3b.get_legend_handles_labels()
ax3.legend(lines3 + lines3b, labels3 + labels3b, loc='upper right',
fontsize=9)

# Plot 4: Sensitivity Derivatives
ax4 = plt.subplot(gs[1, 2:])
# Extract sensitivity data
dc_dR_list = [res['sens']['dc_dR'] for res in results]
dt_dR_list = [res['sens']['dt_dR'] for res in results]
dC_L_dR_list = [res['sens']['dC_L_dR'] for res in results]

# Plot derivatives
ax4.plot(R_list, dc_dR_list, 'b-', linewidth=2.5, label='dc/dR')
ax4.plot(R_list, dt_dR_list, 'r-', linewidth=2.5, label='dt/dR')
ax4.plot(R_list, dC_L_dR_list, 'g-', linewidth=2.5, label='dC_L/dR')

# Add critical points
# Find where dc/dR changes sign (minimum chord occurs at R = b)
R_critical = b
ax4.axvline(x=R_critical, color='k', linestyle='--', alpha=0.5,
label=f'Critical R = b = {b} m')

# Find where dC_L/dR has maximum magnitude
if len(dC_L_dR_list) > 0:
    max_sens_idx = np.argmin(dC_L_dR_list)  # Most negative (steepest
descent)
```

```
    R_max_sens = R_list[max_sens_idx]
    ax4.axvline(x=R_max_sens, color='purple', linestyle=':', alpha=0.7,
                label=f'Max C_L sensitivity: R={R_max_sens:.2f} m')

ax4.set_xlabel('Circle Radius (R) [m]', fontsize=12, fontweight='bold')
ax4.set_ylabel('Sensitivity Derivative Value', fontsize=12,
fontweight='bold')
ax4.set_title('Sensitivity Analysis: Derivatives with Respect to R',
fontsize=13, fontweight='bold')
ax4.grid(True, alpha=0.3, linestyle='--')
ax4.legend(loc='upper right', fontsize=9)
ax4.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=0.5)

# Plot 5: Pressure Distribution Comparison
ax5 = plt.subplot(gs[2, :2])
# Select 3 representative R values
rep_indices = [0, len(R_list)//2, -1]  # Small, medium, large R
line_styles = ['-', '--', '-.']

for idx, style in zip(rep_indices, line_styles):
    res = results[idx]
    label = f'R={res["R"]:.2f}m, t/c={res["geom"]["thickness_ratio"]:.3f}'
    ax5.plot(res['x_norm'], res['Cp'], style, linewidth=2, label=label)

ax5.axhline(y=0, color='k', linestyle='-', alpha=0.3, linewidth=1)
ax5.set_xlabel('x/c (normalized)', fontsize=12, fontweight='bold')
ax5.set_ylabel(r'Pressure Coefficient ($C_p$)', fontsize=12,
fontweight='bold')
ax5.set_title('Pressure Distribution for Different R Values', fontsize=13,
fontweight='bold')
ax5.grid(True, alpha=0.3, linestyle='--')
ax5.invert_yaxis()  # Negative Cp (suction) upwards
ax5.legend(loc='best', fontsize=9)
ax5.set_xlim(0, 1)

# Plot 6: Normalized Airfoil Comparison (Same Absolute Size)
ax6 = plt.subplot(gs[2, 2:])
# Scale all airfoils to same chord for shape comparison
for idx, (res, color) in enumerate(zip(results, colors)):
    if idx % 2 == 0:  # Plot every other for clarity
        chord = res['geom']['chord']
        x_norm = (res['x_airfoil'] - min(res['x_airfoil'])) / chord
        y_norm = res['y_airfoil'] / chord
        ax6.plot(x_norm, y_norm, color=color, linewidth=1, alpha=0.7,
                label=f'R={res["R"]:.2f}m' if idx in [0, len(results)//2,
-1] else None)

ax6.set_xlabel('x/c (normalized)', fontsize=12, fontweight='bold')
ax6.set_ylabel('y/c (normalized)', fontsize=12, fontweight='bold')
ax6.set_title('Normalized Airfoil Shapes (Same Chord Length)',
fontsize=13, fontweight='bold')
ax6.grid(True, alpha=0.3, linestyle='--')
ax6.axis('equal')
ax6.set_xlim(-0.1, 1.1)
ax6.set_ylim(-0.6, 0.6)
ax6.legend(loc='upper right', fontsize=9)

# Plot 7: Design Space Exploration
ax7 = plt.subplot(gs[3, :2])
# Create contour/heatmap of key relationships
R_grid = np.linspace(b*1.01, 3.0, 50)
b_over_R = b / R_grid

# Calculate key metrics
t_c_ratio = (1 - b_over_R**2) / (1 + b_over_R**2)
C_L_grid = Gamma / (U_inf * R_grid * (1 + b_over_R**2))
V_LE_grid = Gamma / (2 * np.pi * R_grid * (1 - b_over_R**2))

# Plot trade-off curves
scatter = ax7.scatter(t_c_ratio, C_L_grid, c=R_grid, cmap='viridis',
                      s=50, alpha=0.8, edgecolors='k', linewidths=0.5)

# Add contour lines for constant R
for R_marker in [b*1.1, b*1.5, b*2, b*2.5, b*3]:
    if R_marker <= max(R_grid):
        idx = np.argmin(np.abs(R_grid - R_marker))
        ax7.plot(t_c_ratio[idx], C_L_grid[idx], 'ro', markersize=8,
markeredgecolor='k')
        ax7.annotate(f'R={R_marker:.2f}m', (t_c_ratio[idx],
C_L_grid[idx]),
                    xytext=(5, 5), textcoords='offset points', fontsize=8)

ax7.set_xlabel('Thickness/Chord Ratio (t/c)', fontsize=12,
fontweight='bold')
ax7.set_ylabel(r'Lift Coefficient ($C_L$)', fontsize=12,
fontweight='bold')
ax7.set_title('Design Space: C_L vs t/c (Colored by R)', fontsize=13,
fontweight='bold')
ax7.grid(True, alpha=0.3, linestyle='--')
```

```python
# Add colorbar
cbar = plt.colorbar(scatter, ax=ax7)
cbar.set_label('Circle Radius (R) [m]', fontsize=10, fontweight='bold')

# Plot 8: Performance Metrics Summary
ax8 = plt.subplot(gs[3, 2:])
ax8.axis('off')

# Calculate key metrics for summary
summary_data = []
for R_rep in [R_list[0], R_list[len(R_list)//2], R_list[-1]]:
    idx = np.argmin(np.abs(np.array(R_list) - R_rep))
    res = results[idx]

    summary_data.append({
        'R': res['R'],
        'Chord': res['geom']['chord'],
        't/c': res['geom']['thickness_ratio'],
        'C_L': res['aero']['C_L'],
        'V_LE': res['aero']['V_LE'],
        'V_MC': res['aero']['V_MC']
    })

# Create summary table
table_data = []
headers = ['R [m]', 'Chord [m]', 't/c', 'C_L', 'V_LE [m/s]', 'V_MC [m/s]']
for data in summary_data:
    table_data.append([
        f"{data['R']:.2f}",
        f"{data['Chord']:.2f}",
        f"{data['t/c']:.3f}",
        f"{data['C_L']:.4f}",
        f"{data['V_LE']:.2f}",
        f"{data['V_MC']:.2f}"
    ])

# Create table
table = ax8.table(cellText=table_data, colLabels=headers,
                  cellLoc='center', loc='center',
                  colWidths=[0.12, 0.15, 0.12, 0.12, 0.15, 0.15])

# Style table
table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1.2, 1.8)

# Add title
ax8.set_title('Key Performance Metrics for Representative R Values',
              fontsize=13, fontweight='bold', y=0.95)

# Add analysis text
analysis_text = f"""
Key Trends:
1. As R increases: Chord ↗, Thickness ↗, t/c ↗, C_L ↘
2. Minimum R = b = {b} m (singularity at t=0)
3. Optimal R for max t/c sensitivity: R ≈ {b*np.sqrt(3):.3f} m
4. As R → ∞: Airfoil → Circle (t/c → 1), C_L → 0
"""
ax8.text(0.05, -0.1, analysis_text, transform=ax8.transAxes,
         verticalalignment='top', fontsize=9,
         bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.3))

plt.tight_layout()
plt.show()

#
==========================================================================
==
# ADDITIONAL ANALYSIS: OPTIMIZATION
#
==========================================================================
==
print("="*80)
print("OPTIMIZATION ANALYSIS")
print("="*80)

# Find R that maximizes thickness ratio sensitivity
def thickness_ratio(R):
    return (R - b**2/R) / (R + b**2/R)

# Find inflection point of thickness ratio
R_range = np.linspace(b*1.01, 3.0, 1000)
t_c_vals = thickness_ratio(R_range)

# Calculate second derivative numerically for inflection point
dtc_dR = np.gradient(t_c_vals, R_range)
d2tc_dR2 = np.gradient(dtc_dR, R_range)

# Find where second derivative is maximum (inflection point)
```

```
inflection_idx = np.argmax(d2tc_dR2)
R_inflection = R_range[inflection_idx]

print(f"\n1. Geometric Optimization:")
print(f"   • Minimum valid R: {b:.3f} m (singularity)")
print(f"   • Inflection point (max curvature change): R =
{R_inflection:.3f} m")
print(f"   • Theoretical optimal: R = b√3 = {b*np.sqrt(3):.3f} m")

# Find R for maximum C_L efficiency (C_L per unit thickness)
def C_L_efficiency(R):
    C_L_val = Gamma / (U_inf * (R + b**2/R))
    thickness_val = 2 * (R - b**2/R)
    return C_L_val / thickness_val if thickness_val > 0 else 0

# Optimize for maximum C_L efficiency
R_opt_efficiency = minimize_scalar(lambda x: -C_L_efficiency(x),
                                   bounds=(b*1.01, 3.0),
                                   method='bounded').x

print(f"\n2. Aerodynamic Optimization:")
print(f"   • Maximum C_L/thickness efficiency: R = {R_opt_efficiency:.3f}
m")
print(f"   • Corresponding C_L:
{C_L_efficiency(R_opt_efficiency)*thickness_ratio(R_opt_efficiency):.4f}")
print(f"   • Corresponding t/c: {thickness_ratio(R_opt_efficiency):.3f}")

# Calculate design bounds for specific applications
print(f"\n3. Design Bounds for Applications:")
print(f"   • High-speed aircraft (thin wings, t/c < 0.12):")
for R_test in R_range:
    if thickness_ratio(R_test) < 0.12:
        print(f"       R > {R_test:.2f} m")
        break

print(f"   • General aviation (t/c ~ 0.15-0.18):")
valid_R = []
for R_test in R_range:
    if 0.15 <= thickness_ratio(R_test) <= 0.18:
        valid_R.append(R_test)
if valid_R:
    print(f"       R ∈ [{min(valid_R):.2f}, {max(valid_R):.2f}] m")

print(f"   • High-lift applications (t/c > 0.20):")
for R_test in R_range:
    if thickness_ratio(R_test) > 0.20:
        print(f"       R < {R_test:.2f} m")
        break

print(f"\n4. Practical Design Guidelines:")
print(f"   • Small R ({b:.2f}-{b*1.5:.2f} m): Thin airfoils, high
velocities")
print(f"   • Medium R ({b*1.5:.2f}-{b*2.5:.2f} m): Balanced performance")
print(f"   • Large R (> {b*2.5:.2f} m): Thick airfoils, low velocities,
low C_L")

print("="*80)

#
===========================================================================
==
# DERIVATION SUMMARY OUTPUT
#
===========================================================================
==
print("\n" + "="*80)
print("MATHEMATICAL DERIVATION SUMMARY")
print("="*80)

print(f"\nGiven: Joukowski transformation z' = z + b²/z")
print(f"Circle: |z| = R, with b = {b} m fixed")
print(f"\n1. Geometric Parameters:")
print(f"   • Chord: c(R) = 2(R + b²/R)")
print(f"     Derivative: dc/dR = 2(1 - b²/R²)")
print(f"     Sign: Positive for R > b, zero at R = b, → 2 as R → ∞")
print(f"   • Thickness: t(R) = 2(R - b²/R)")
print(f"     Derivative: dt/dR = 2(1 + b²/R²)")
print(f"     Sign: Always positive, → 2 as R → ∞")
print(f"   • Thickness ratio: t/c = (R - b²/R)/(R + b²/R)")
print(f"     Derivative: d(t/c)/dR = 4b²/[R²(R + b²/R)²]")
print(f"     Sign: Always positive, → 1 as R → ∞")

print(f"\n2. Aerodynamic Parameters:")
print(f"   • Lift coefficient: C_L(R) = 2Γ/(U∞c) = Γ/[U∞(R + b²/R)]")
print(f"     Derivative: dC_L/dR = -Γ(1 - b²/R²)/[U∞(R + b²/R)²]")
print(f"     Sign: Negative for R > b, → 0 as R → ∞")
print(f"   • Leading edge velocity: V_LE(R) = Γ/[2πR(1 - b²/R²)]")
print(f"     Derivative: dV_LE/dR = -Γ(1 + b²/R²)/[2πR²(1 - b²/R²)²]")
print(f"     Sign: Negative for R > b, → 0 as R → ∞")
print(f"\n3. Critical Points:")
print(f"   • R = b: Singularity (dz'/dz = 0), t = 0, V_LE → ∞")
print(f"   • R = b√3 ≈ {b*np.sqrt(3):.3f} m: Maximum t/c curvature")
print(f"   • R → ∞: Airfoil → circle, t/c → 1, C_L → 0")

print(f"\n4. Design Implications:")
print(f"   • Small R → Thin airfoils with high edge velocities")
print(f"   • Large R → Thick airfoils with low lift coefficients")
print(f"   • Optimal R balances structural and aerodynamic requirements")

print("="*80)
```