Airfoil Generation:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
import matplotlib.gridspec as gridspec

#
===========================================================================
==
# CORRECTED PARAMETERS
#
===========================================================================
==
U_inf = 80.0        # Free-stream velocity [m/s]
Gamma = 30.0        # Circulation [m²/s] - CORRECTED FROM 80 TO 30
R = 1.5            # Cylinder radius [m]
b = 0.8            # Joukowski parameter [m]
rho = 1.225        # Air density [kg/m³]

# Derived geometric parameters
a = R + b**2/R     # Semi-major axis [m]
h = R - b**2/R     # Semi-minor axis [m]
c = 2 * a          # Chord length [m]

print("="*70)
print("CORRECTED JOUKOWSKI AIRFOIL PARAMETERS")
print("="*70)
print(f"Free-stream velocity (U∞): {U_inf} m/s")
print(f"Circulation (Γ): {Gamma} m²/s")
print(f"Cylinder radius (R): {R} m")
print(f"Joukowski parameter (b): {b} m")
print(f"Semi-major axis (a): {a:.4f} m")
print(f"Semi-minor axis (h): {h:.4f} m")
print(f"Chord length (c): {c:.4f} m")
print(f"Thickness ratio (t/c): {2*h/c:.4f} ({2*h/c:.1%})")
print("="*70)

#
===========================================================================
==
# 1. AIRFOIL GEOMETRY GENERATION
#
===========================================================================
==
```

```python
def joukowski_transform(z):
    """Apply Joukowski transformation: z' = z + b²/z"""
    return z + (b**2) / z

# Generate circle in z-plane
theta = np.linspace(0, 2*np.pi, 500)
z_circle = R * np.exp(1j * theta)

# Transform to airfoil
z_airfoil = joukowski_transform(z_circle)
x_airfoil = np.real(z_airfoil)
y_airfoil = np.imag(z_airfoil)

# Identify key points
le_idx = np.argmin(x_airfoil)       # Leading edge
te_idx = np.argmax(x_airfoil)       # Trailing edge
top_idx = np.argmax(y_airfoil)      # Maximum thickness (top)
bot_idx = np.argmin(y_airfoil)      # Maximum thickness (bottom)

#
===========================================================================
==
# 2. VELOCITY CALCULATION (CORRECTED)
#
===========================================================================
==
def velocity_in_z_plane(z):
    """Complex velocity in z-plane: W(z) = dF/dz"""
    return U_inf * (1 - R**2 / z**2) + (1j * Gamma) / (2 * np.pi * z)

def velocity_on_airfoil(theta_val):
    """Velocity at airfoil surface (in z'-plane)"""
    z = R * np.exp(1j * theta_val)
    W_z = velocity_in_z_plane(z)
    dz_prime_dz = 1 - b**2 / z**2  # Derivative of transformation
    W_z_prime = W_z / dz_prime_dz
    return W_z_prime

# Calculate velocities at key points
W_LE = velocity_on_airfoil(np.pi)
V_LE = np.abs(W_LE)

W_TE = velocity_on_airfoil(0)
V_TE = np.abs(W_TE)
```

```
W_MC = velocity_on_airfoil(np.pi/2)
V_MC = np.abs(W_MC)

#
================================================================================
==
# 3. PRESSURE COEFFICIENT CALCULATION
#
================================================================================
==
def pressure_coefficient_on_airfoil(theta_val):
    """Pressure coefficient Cp = 1 - (V/U∞)²"""
    W = velocity_on_airfoil(theta_val)
    V = np.abs(W)
    return 1 - (V / U_inf)**2

Cp = pressure_coefficient_on_airfoil(theta)

#
================================================================================
==
# 4. LIFT CALCULATION (CORRECTED)
#
================================================================================
==
L_prime = rho * U_inf * Gamma
C_L = 2 * Gamma / (U_inf * c)  # Theoretical formula

#
================================================================================
==
# 5. COMPREHENSIVE VISUALIZATION
#
================================================================================
==
fig = plt.figure(figsize=(16, 12))
plt.suptitle(f'Joukowski Airfoil Analysis: U∞={U_inf} m/s, Γ={Gamma} m²/s,
R={R} m, b={b} m',
             fontsize=14, fontweight='bold', y=0.98)

# Create custom layout
gs = gridspec.GridSpec(3, 3, height_ratios=[1, 1, 1], hspace=0.3,
wspace=0.3)

# Plot 1: Airfoil Geometry
```

```
ax1 = plt.subplot(gs[0, 0])
ax1.plot(x_airfoil, y_airfoil, 'b-', linewidth=2.5)
ax1.fill(x_airfoil, y_airfoil, 'lightblue', alpha=0.5)
ax1.plot(x_airfoil[le_idx], y_airfoil[le_idx], 'ro', markersize=10,
label='LE')
ax1.plot(x_airfoil[te_idx], y_airfoil[te_idx], 'go', markersize=10,
label='TE')
ax1.plot(x_airfoil[top_idx], y_airfoil[top_idx], 'mo', markersize=8,
label='Max thickness')
ax1.set_xlabel('x [m]')
ax1.set_ylabel('y [m]')
ax1.set_title('Airfoil Geometry')
ax1.axis('equal')
ax1.grid(True, alpha=0.3)
ax1.legend(loc='best')
# Add geometry info
geo_text = f'Chord: {c:.3f} m\nThickness: {2*h:.3f} m\nt/c: {2*h/c:.3f}'
ax1.text(0.02, 0.98, geo_text, transform=ax1.transAxes,
verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.7),
fontsize=9)

# Plot 2: Pressure Coefficient Distribution
ax2 = plt.subplot(gs[0, 1])
# Sort Cp by x for proper plotting
x_points = np.real(z_airfoil)
x_norm = (x_points - x_points.min()) / (x_points.max() - x_points.min())
sort_idx = np.argsort(x_norm)
ax2.plot(x_norm[sort_idx], Cp[sort_idx], 'r-', linewidth=2.5)
ax2.fill_between(x_norm[sort_idx], Cp[sort_idx], 0, where=Cp[sort_idx]>0,
                 alpha=0.3, color='red', label='Pressure (Cp > 0)')
ax2.fill_between(x_norm[sort_idx], Cp[sort_idx], 0, where=Cp[sort_idx]<0,
                 alpha=0.3, color='blue', label='Suction (Cp < 0)')
ax2.axhline(y=0, color='k', linestyle='-', alpha=0.3)
ax2.set_xlabel('x/c (normalized)')
ax2.set_ylabel(r'$C_p$')
ax2.set_title('Pressure Coefficient Distribution')
ax2.grid(True, alpha=0.3)
ax2.invert_yaxis()
ax2.legend(loc='best')
# Add Cp values
cp_text = f'Min Cp: {Cp.min():.3f}\nMax Cp: {Cp.max():.3f}\nΔCp:
{Cp.max()-Cp.min():.3f}'
ax2.text(0.02, 0.98, cp_text, transform=ax2.transAxes,
verticalalignment='top',
```

```
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.7),
fontsize=9)

# Plot 3: Surface Velocity Distribution
ax3 = plt.subplot(gs[0, 2])
# Calculate velocity magnitude on airfoil surface
V_surface = np.array([np.abs(velocity_on_airfoil(t)) for t in theta])
# Sort by x position
x_surf = np.real(z_airfoil)
x_surf_norm = (x_surf - x_surf.min()) / (x_surf.max() - x_surf.min())
sort_idx_surf = np.argsort(x_surf_norm)
ax3.plot(x_surf_norm[sort_idx_surf], V_surface[sort_idx_surf], 'b-',
linewidth=2.5)
ax3.axhline(y=U_inf, color='r', linestyle='--', alpha=0.7, linewidth=2,
label=r'$U_\infty$ = 80 m/s')
# Mark key points
for idx, label, color in [(le_idx, 'LE', 'red'), (te_idx, 'TE', 'green'),
(top_idx, 'Max t', 'magenta')]:
    ax3.plot(x_surf_norm[idx], V_surface[idx], 'o', color=color,
markersize=8)
    ax3.annotate(f'{label}\n{V_surface[idx]:.1f} m/s', (x_surf_norm[idx],
V_surface[idx]),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax3.set_xlabel('x/c (normalized)')
ax3.set_ylabel('Velocity [m/s]')
ax3.set_title('Surface Velocity Distribution')
ax3.grid(True, alpha=0.3)
ax3.legend(loc='best')
ax3.set_xlim(0, 1)
ax3.set_ylim(0, max(V_surface)*1.1)

# Plot 4: Streamlines
ax4 = plt.subplot(gs[1, 0])
# Create grid for stream function
x_min, x_max = -3, 3
y_min, y_max = -2, 2
x_grid = np.linspace(x_min, x_max, 80)
y_grid = np.linspace(y_min, y_max, 60)
X, Y = np.meshgrid(x_grid, y_grid)

# Stream function for flow around cylinder with circulation
def stream_function(x, y):
    r = np.sqrt(x**2 + y**2)
    theta_pos = np.arctan2(y, x)
```

```
    return U_inf * (r - R**2/r) * np.sin(theta_pos) + (Gamma/(2*np.pi)) *
np.log(r + 1e-10)

Psi = stream_function(X, Y)

# Plot airfoil
ax4.fill(x_airfoil, y_airfoil, 'gray', alpha=0.7, edgecolor='black',
linewidth=1.5)
# Plot streamlines
ax4.contour(X, Y, Psi, levels=30, colors='blue', alpha=0.6,
linewidths=0.8)
ax4.set_xlabel('x [m]')
ax4.set_ylabel('y [m]')
ax4.set_title('Streamlines Around Airfoil')
ax4.axis('equal')
ax4.grid(True, alpha=0.2)
ax4.set_xlim(x_min, x_max)
ax4.set_ylim(y_min, y_max)

# Plot 5: Velocity Vectors
ax5 = plt.subplot(gs[1, 1])
# Create coarser grid for vectors
x_vec = np.linspace(x_min, x_max, 20)
y_vec = np.linspace(y_min, y_max, 15)
Xv, Yv = np.meshgrid(x_vec, y_vec)

# Velocity field function
def velocity_field(x, y):
    z = x + 1j*y
    mask = np.abs(z) > 0.01
    W = np.zeros_like(z, dtype=complex)
    W[mask] = velocity_in_z_plane(z[mask])

    dz_prime_dz = 1 - b**2 / z**2
    W_prime = np.zeros_like(z, dtype=complex)
    valid = np.abs(dz_prime_dz) > 0.01
    W_prime[valid] = W[valid] / dz_prime_dz[valid]

    return np.real(W_prime), np.imag(W_prime)

Vx, Vy = velocity_field(Xv, Yv)
V_mag = np.sqrt(Vx**2 + Vy**2)

# Plot airfoil
```

```
ax5.fill(x_airfoil, y_airfoil, 'gray', alpha=0.7, edgecolor='black',
linewidth=1.5)
# Plot vectors
scale = 0.15 * (x_max - x_min) / np.max(V_mag)
ax5.quiver(Xv, Yv, Vx, Vy, V_mag, scale=1.0/scale, width=0.003,
           cmap='viridis', alpha=0.8)
ax5.set_xlabel('x [m]')
ax5.set_ylabel('y [m]')
ax5.set_title('Velocity Vectors (Colored by Magnitude)')
ax5.axis('equal')
ax5.grid(True, alpha=0.2)
ax5.set_xlim(x_min, x_max)
ax5.set_ylim(y_min, y_max)

# Plot 6: Original Circle and Transformation
ax6 = plt.subplot(gs[1, 2])
circle = Circle((0, 0), R, fill=False, edgecolor='r', linewidth=2,
linestyle='--', alpha=0.8)
ax6.add_patch(circle)
ax6.plot(np.real(z_circle), np.imag(z_circle), 'r-', alpha=0.7,
linewidth=1.5, label='Original Circle')
ax6.plot(x_airfoil, y_airfoil, 'b-', alpha=0.7, linewidth=1.5,
label='Joukowski Airfoil')
ax6.plot([b, -b], [0, 0], 'kx', markersize=10, label='Critical points ±b')
ax6.set_xlabel('x [m]')
ax6.set_ylabel('y [m]')
ax6.set_title('Joukowski Transformation (R=1.5m)')
ax6.axis('equal')
ax6.grid(True, alpha=0.3)
ax6.legend(loc='best')
ax6.set_xlim(-2.5, 2.5)
ax6.set_ylim(-2, 2)

# Plot 7: Comparison Table
ax7 = plt.subplot(gs[2, 0])
ax7.axis('off')
comparison_text = f"""
PARAMETER COMPARISON

CURRENT (Corrected):
• U∞ = 80 m/s, Γ = 30 m²/s, R = 1.5 m
• Chord: {c:.3f} m, Thickness: {2*h:.3f} m
• t/c ratio: {2*h/c:.3f} ({2*h/c:.1%})
• C_L = {C_L:.4f}, L' = {L_prime:.0f} N/m
• V_LE/TE = {V_LE:.2f} m/s
```

```
• V_Mid-chord = {V_MC:.2f} m/s

ORIGINAL (From Problem):
• U∞ = 20 m/s, Γ = 10 m²/s, R = 1 m
• Chord: 3.28 m, Thickness: 0.72 m
• t/c ratio: 0.2195 (21.9%)
• C_L = 0.3049, L' = 245 N/m
• V_LE/TE = 4.42 m/s
• V_Mid-chord = 25.36 m/s

CHANGE FACTORS:
• U∞: ×4.0 (80/20)
• Γ: ×3.0 (30/10)
• R: ×1.5 (1.5/1)
• C_L: ×{C_L/0.3049:.2f} ({C_L:.4f}/0.3049)
• L': ×{L_prime/245:.2f} ({L_prime:.0f}/245)
"""
ax7.text(0.05, 0.95, comparison_text, transform=ax7.transAxes,
         verticalalignment='top', fontsize=9,
         bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.3))

# Plot 8: Lift Calculation Details
ax8 = plt.subplot(gs[2, 1])
ax8.axis('off')
lift_text = f"""
LIFT CALCULATION DETAILS

Given:
• ρ = {rho} kg/m³
• U∞ = {U_inf} m/s
• Γ = {Gamma} m²/s
• c = {c:.4f} m

Kutta-Joukowski Theorem:
L' = ρU∞Γ = {rho} × {U_inf} × {Gamma}
   = {L_prime:.2f} N/m

Lift Coefficient:
C_L = L' / (½ρU∞²c)
   = {L_prime:.2f} / (0.5 × {rho} × {U_inf**2} × {c:.4f})
   = {L_prime:.2f} / {0.5*rho*U_inf**2*c:.2f}
   = {C_L:.4f}

Alternative Formula:
C_L = 2Γ/(U∞c) = 2×{Gamma}/({U_inf}×{c:.4f})
```

```
    = {2*Gamma} / {U_inf*c:.4f}
    = {C_L:.4f}

VELOCITY CALCULATIONS:
Leading Edge (θ=π):
W(z) = U∞(1 - R²/z²) + iΓ/(2πz)
     = {U_inf}×(1 - {R**2}/{R**2}) + i×{Gamma}/(2π×{R})
     = -i×{Gamma/(2*np.pi*R):.3f} m/s
W'(L.E.) = {-Gamma/(2*np.pi*R):.3f}i / (1 - {b**2}/{R**2})
         = {-Gamma/(2*np.pi*R):.3f}i / {1-b**2/R**2:.4f}
         = -i×{V_LE:.3f} m/s
"""
ax8.text(0.05, 0.95, lift_text, transform=ax8.transAxes,
         verticalalignment='top', fontsize=8.5,
         bbox=dict(boxstyle='round', facecolor='lightyellow', alpha=0.3))

# Plot 9: Key Results Summary
ax9 = plt.subplot(gs[2, 2])
ax9.axis('off')
summary_text = f"""
KEY RESULTS SUMMARY

AIRFOIL GEOMETRY:
• Type: Symmetric elliptical airfoil
• Chord length: {c:.3f} m
• Maximum thickness: {2*h:.3f} m
• Thickness ratio: {2*h/c:.3f} ({2*h/c:.1%})
• Semi-major axis: {a:.3f} m
• Semi-minor axis: {h:.3f} m

VELOCITY MAGNITUDES:
• Leading edge: {V_LE:.3f} m/s
• Trailing edge: {V_TE:.3f} m/s
• Mid-chord (max thickness): {V_MC:.3f} m/s
• Free stream: {U_inf} m/s

PRESSURE DISTRIBUTION:
• Minimum Cp (max suction): {Cp.min():.3f}
• Maximum Cp: {Cp.max():.3f}
• Cp difference (ΔCp): {Cp.max()-Cp.min():.3f}

LIFT CHARACTERISTICS:
• Lift coefficient (C_L): {C_L:.4f}
• Lift per unit span (L'): {L_prime:.0f} N/m
• Using Kutta condition at trailing edge
```

```
FLOW FEATURES:
• Symmetric flow at 0° angle of attack
• Stagnation points shifted by circulation
• Smooth flow meeting at trailing edge
• Higher velocities on upper surface
"""
ax9.text(0.05, 0.95, summary_text, transform=ax9.transAxes,
         verticalalignment='top', fontsize=8.5,
         bbox=dict(boxstyle='round', facecolor='lightgreen', alpha=0.3))

plt.tight_layout()
plt.show()

#
==========================================================================
==
# 6. DETAILED CALCULATION OUTPUT
#
==========================================================================
==
print("\n" + "="*70)
print("DETAILED CALCULATIONS WITH CORRECTED CIRCULATION (Γ = 30 m²/s)")
print("="*70)

print(f"\n1. GEOMETRY (unchanged):")
print(f"   Semi-major axis: a = R + b²/R = {R} + {b**2}/{R} = {a:.4f} m")
print(f"   Semi-minor axis: h = R - b²/R = {R} - {b**2}/{R} = {h:.4f} m")
print(f"   Chord length: c = 2a = 2 × {a:.4f} = {c:.4f} m")
print(f"   Thickness ratio: t/c = {2*h:.4f}/{c:.4f} = {2*h/c:.4f}
({2*h/c:.1%})")

print(f"\n2. VELOCITY CALCULATIONS (Corrected with Γ=30):")
print(f"   Leading edge (θ=π, z=-R):")
print(f"     W(z) = -i×Γ/(2πR) = -i×{Gamma}/(2π×{R}) = -
i×{Gamma/(2*np.pi*R):.3f} m/s")
print(f"     dz'/dz = 1 - b²/R² = 1 - {b**2}/{R**2} = {1-b**2/R**2:.4f}")
print(f"     W'(L.E.) = (-i×{Gamma/(2*np.pi*R):.3f})/{1-b**2/R**2:.4f} = -
i×{V_LE:.3f} m/s")
print(f"     Magnitude: {V_LE:.3f} m/s")

print(f"\n   Trailing edge (θ=0, z=R):")
print(f"     W'(T.E.) = i×{V_TE:.3f} m/s, Magnitude: {V_TE:.3f} m/s")

print(f"\n   Mid-chord (θ=π/2, z=iR):")
```

```
print(f"     W(z) = 2U∞ + Γ/(2πR) = 2×{U_inf} + {Gamma}/(2π×{R})")
print(f"           = {2*U_inf} + {Gamma/(2*np.pi*R):.3f} =
{2*U_inf+Gamma/(2*np.pi*R):.3f} m/s")
print(f"     dz'/dz = 1 - b²/(iR)² = 1 - {b**2}/(-{R**2}) = 1 +
{b**2/R**2:.4f} = {1+b**2/R**2:.4f}")
print(f"     W'(mid) = {2*U_inf+Gamma/(2*np.pi*R):.3f}/{1+b**2/R**2:.4f} =
{V_MC:.3f} m/s")

print(f"\n3. LIFT CALCULATION (Corrected):")
print(f"   L' = ρU∞Γ = {rho} × {U_inf} × {Gamma} = {L_prime:.2f} N/m")
print(f"   C_L = 2Γ/(U∞c) = 2×{Gamma}/({U_inf}×{c:.4f}) = {C_L:.4f}")



print("="*70)
```