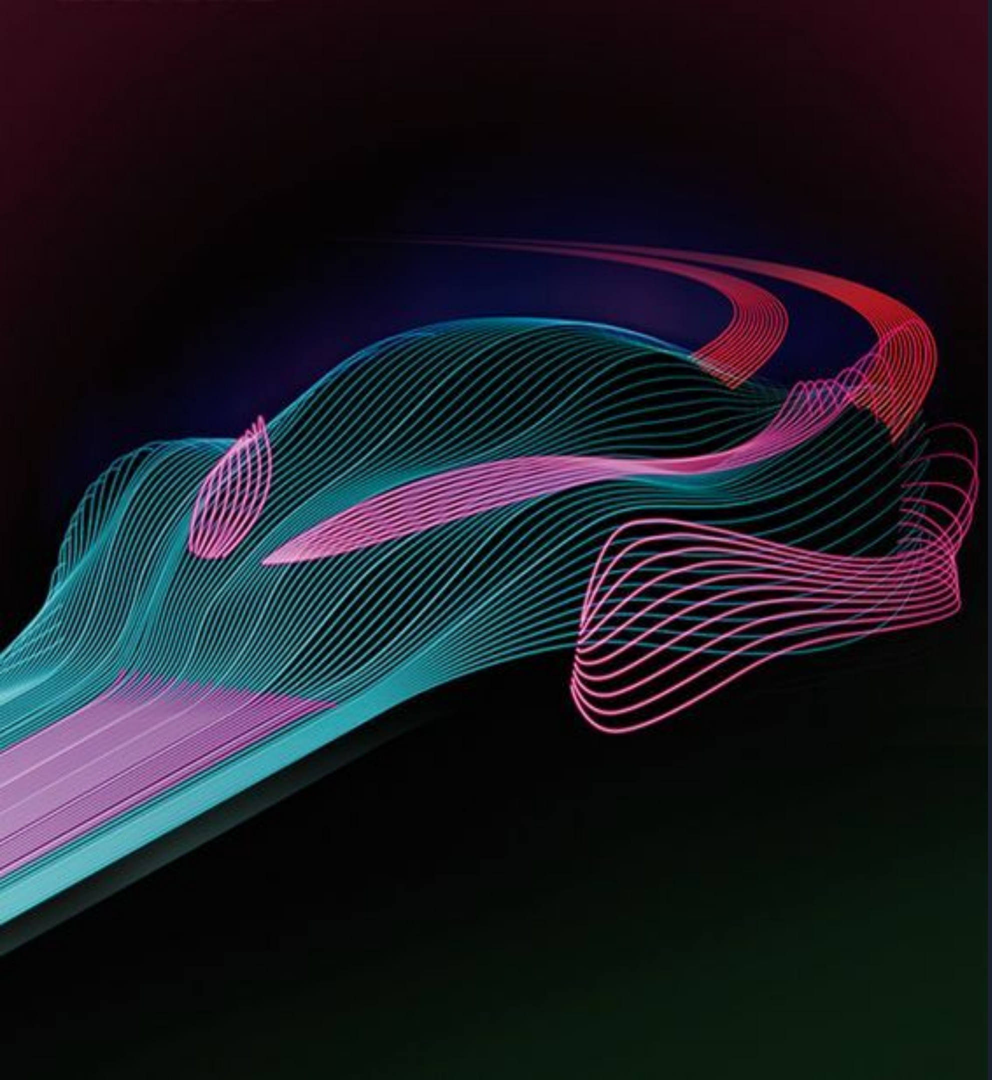
Optimising Wing Design For Improved Aerodynamic Performance

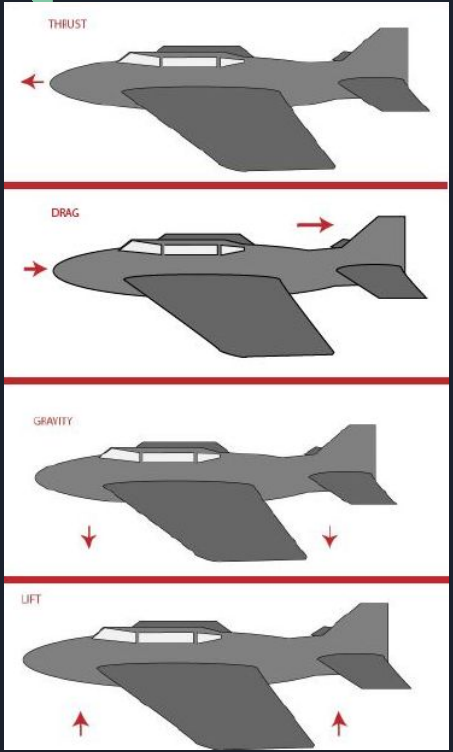
Rudejn Pepaj (31070)



Introduction

Wing design using differential equations for enhanced aerodynamic performance. The goal was to improve efficiency and reduce drag in aviation, aerospace, and wind energy applications

Aerodynamics



01

- Basic Aerodynamic Variables: Pressure, Density, Velocity, Temperature
- Elements: Acting Forces, Flow Types, Airfoil Design, Boundary Layers

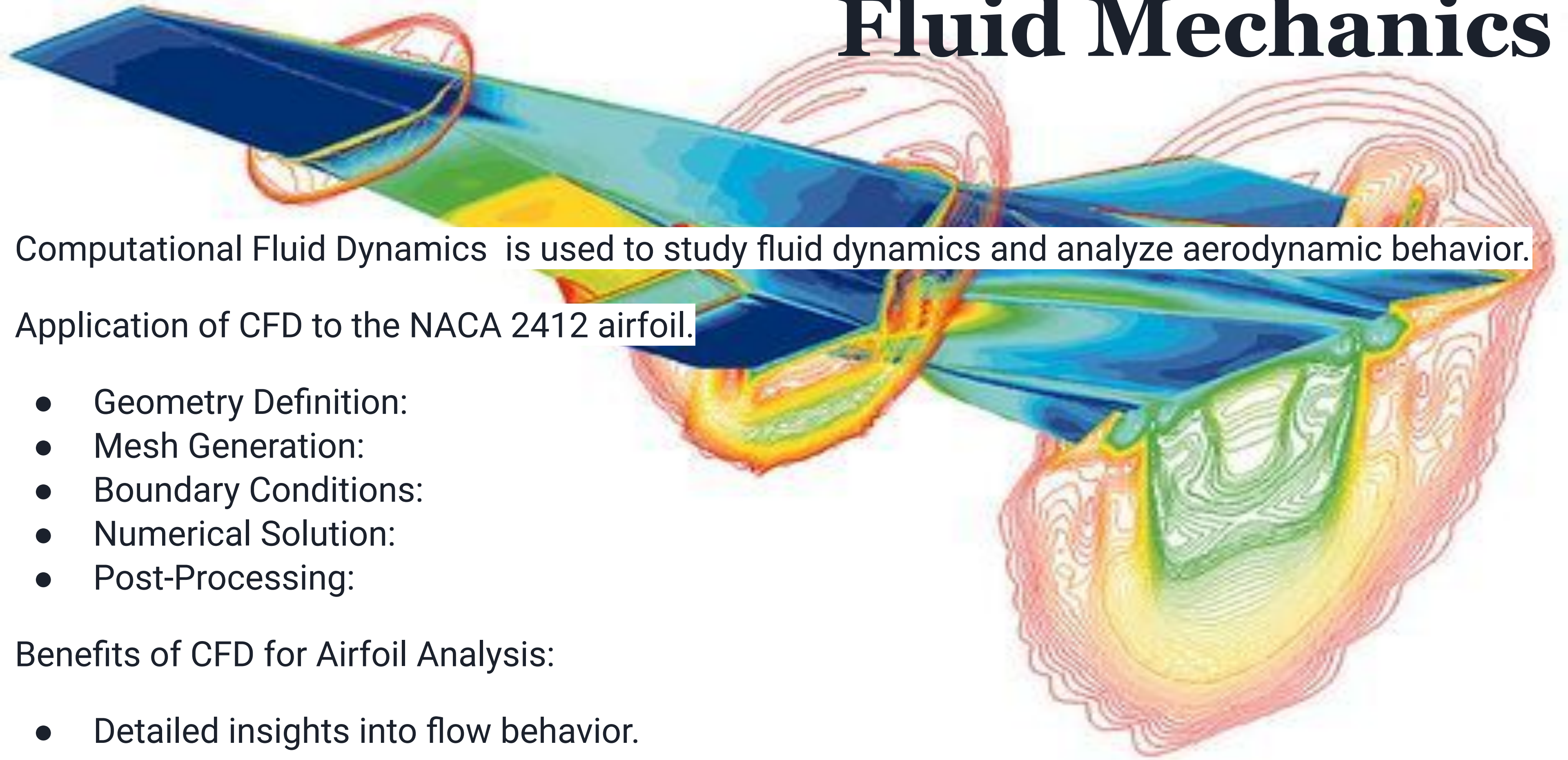
02

- Lift: Generated by fluid-airfoil interaction, counteracts weight
- Drag: Resistance force opposing motion, reduces velocity
- Thrust: Forward propelling force, counters drag
- Weight: Force of gravity pulling aircraft down, requires lift for flight

03

- Safety and Efficiency: Essential for safe flight operations
- Balance and Control: Critical for stability and maneuverability

Fluid Mechanics



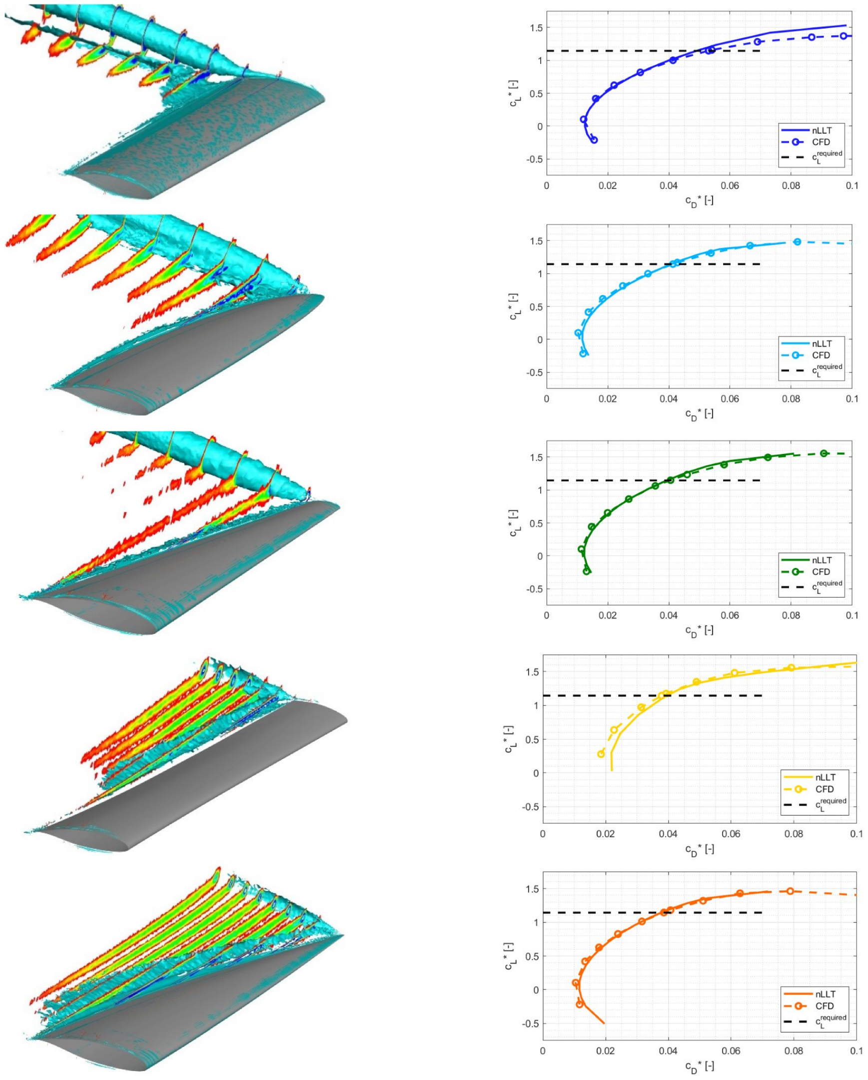
Computational Fluid Dynamics is used to study fluid dynamics and analyze aerodynamic behavior.

Application of CFD to the NACA 2412 airfoil.

- Geometry Definition:
- Mesh Generation:
- Boundary Conditions:
- Numerical Solution:
- Post-Processing:

Benefits of CFD for Airfoil Analysis:

- Detailed insights into flow behavior.
- Optimization of airfoil designs.



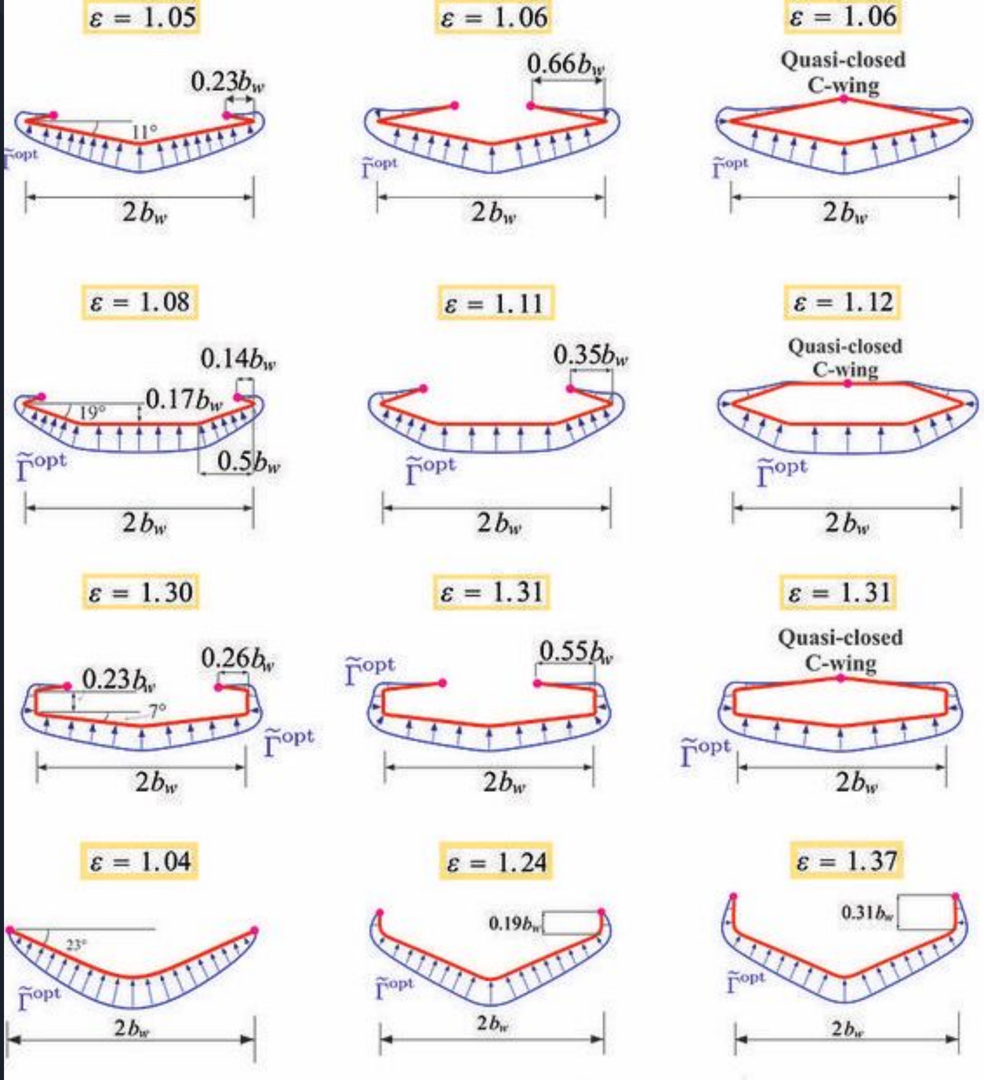
Differential Equations

Aerodynamic Modelling

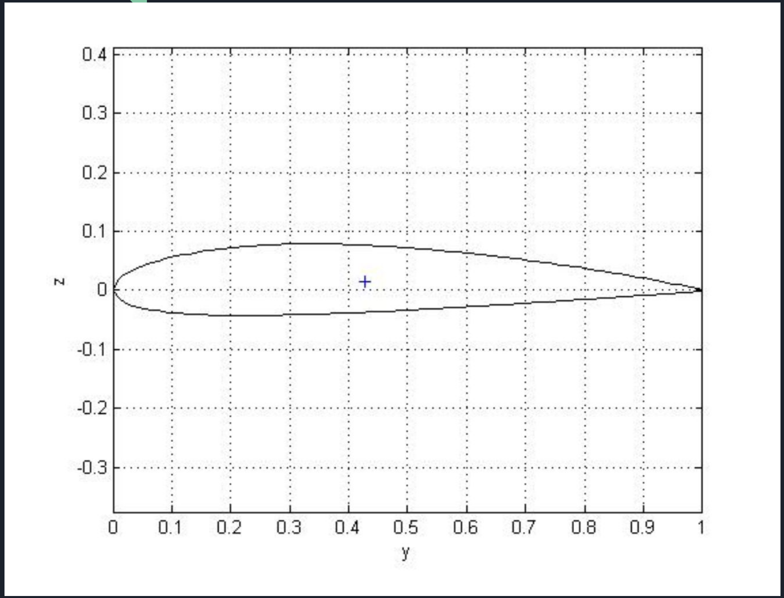
- The Navier-Stokes equations are fundamental equations in fluid dynamics that describe fluid motion.
- They consider the effects of viscosity and compressibility.
- Derived from Newton's second law of motion applied to a fluid element.
- Expresses conservation of mass for incompressible fluids.
- Mathematical form: $\partial\rho/\partial t + \nabla\cdot(\rho V) = 0$.
- ρ: fluid density, t: time, V: velocity vector, $\nabla\cdot$: divergence operator.
- Represent conservation of momentum in each spatial dimension.
- Consider pressure, viscous forces, and body forces.
- Mathematical form: $\partial(\rho V)/\partial t + \nabla\cdot(\rho VV) = -\nabla P + \nabla\cdot\tau + \rho g$.
- P: pressure, τ: stress tensor, g: acceleration due to gravity.

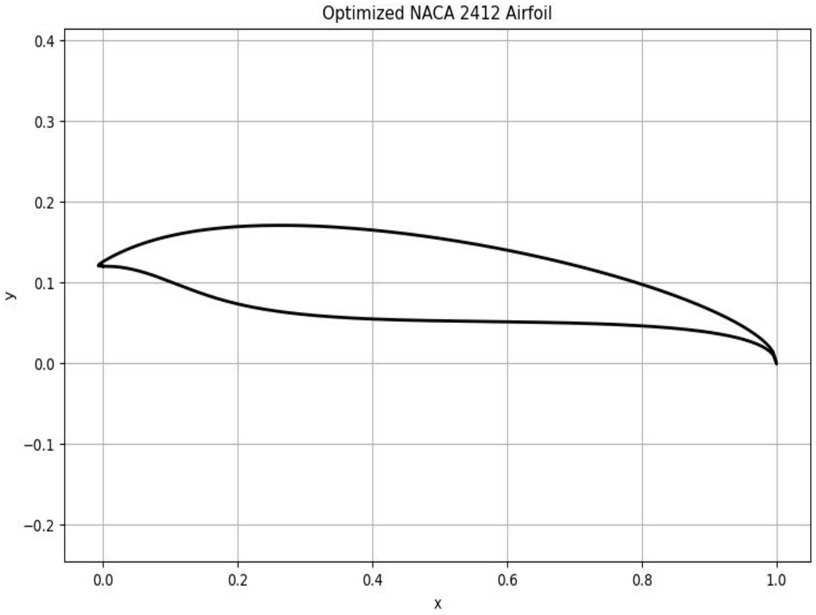
Optimisation Techniques

- Initialisation
- Fitness Evaluation
- Selection
- Reproduction
- Replacement
- Termination
- Convergence and Results
- Implementation Considerations

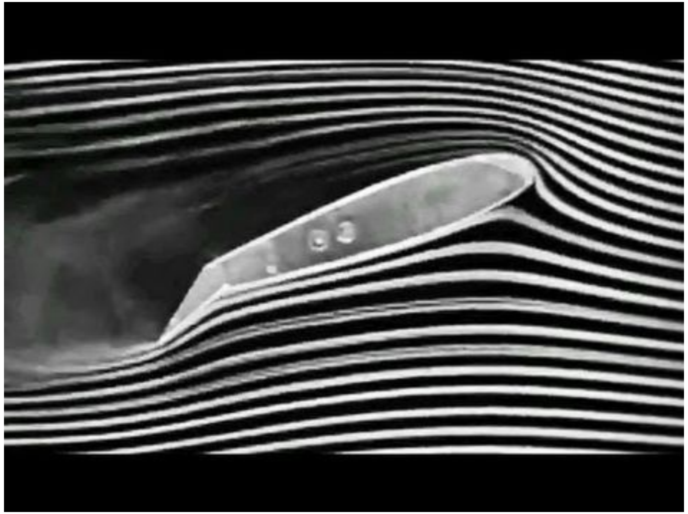


Implementation





Implementation



```python
def compute_airfoil_coordinates(c, t, num_points):
    x = np.linspace(0, c, num_points)
    yt = 5 * t * (0.2969 * np.sqrt(x) - 0.1260 * x - 0.3516 * x**2 + 0.2843 * x**3 - 0.1015 * x**4)
    yc = 0.12 * c * (1 - x**2 / c**2)**0.5

    xu = x - yt * np.sin(np.arctan2(yc, x))
    yu = yc + yt * np.cos(np.arctan2(yc, x))
    xl = x + yt * np.sin(np.arctan2(yc, x))
    yl = yc - yt * np.cos(np.arctan2(yc, x))

    x_coordinates = np.concatenate((xu, xl[::-1]))
    y_coordinates = np.concatenate((yu, yl[::-1]))

    return np.column_stack((x_coordinates, y_coordinates))
```

Definition of the airfoil

Implementation of the optimisation technique

```python
def objective_function(parameters):
    c, t, alpha = parameters
    airfoil_coordinates = compute_airfoil_coordinates(c, t, num_points=100)
    Cl, Cd = calculate_lift_drag_coefficient(airfoil_coordinates, alpha, U_inf=1.0)
    return -Cd

# Optimization
initial_parameters = [1.0, 0.12, 0.0]
bounds = [(0.5, 2.0), (0.08, 0.15), (-10.0, 10.0)]

result = minimize(objective_function, initial_parameters, bounds=bounds)
optimized_parameters = result.x
optimized_airfoil_coordinates = compute_airfoil_coordinates(optimized_parameters[0], optimized_parameters[1], num_point
```

```python
def calculate_lift_drag_coefficient(airfoil_coordinates, alpha, U_inf):
    alpha_rad = np.deg2rad(alpha)
    cos_alpha = np.cos(alpha_rad)
    sin_alpha = np.sin(alpha_rad)

    x = airfoil_coordinates[:, 0]
    y = airfoil_coordinates[:, 1]

    dx = np.diff(x)
    dy = np.diff(y)

    panel_length = np.sqrt(dx**2 + dy**2)
    nx = dy / panel_length
    ny = -dx / panel_length

    v_normal = U_inf * (cos_alpha * nx + sin_alpha * ny)
    v_tangential = U_inf * (-sin_alpha * nx + cos_alpha * ny)

    Cl = 2 * np.sum(v_normal * panel_length)
    Cd = 2 * np.sum(v_tangential * panel_length)

    return Cl, Cd
```

Calculation and improvement

Rudejn Pepaj (31070)