Airliner Aerodynamics After Wing Loss

Yujun Liu, Shen Shen, Abdilahi Mohammed, Ming Yan
University of Toronto, Scarborough

November 28, 2024

Contents

1 Abstract

This study investigates the aerodynamic forces acting on the TU-154 airliner, with a specific focus on developing a lifting-line theory for both a symmetric and a damaged wing. The goal is to model the lift distribution and resulting torque to analyze the controllability of the aircraft following damage to approximately one-third of the left wingspan, caused by a collision with a birch tree. The theoretical framework begins with thin airfoil theory to understand lift generation for infinite wings, then progresses to finite wings using the Lanchester-Prandtl lifting-line theory to incorporate spanwise variations in circulation $\Gamma(y)$.

The lifting-line theory is formulated as an integral equation and discretized into a system of $N \times N$ linear equations, solved numerically using Python. For the damaged wing, the method incorporates geometric asymmetries and structural discontinuities. Simulations evaluate lift distribution, torque, and roll dynamics, providing insights into the stability and control of the aircraft under these extreme conditions. This approach bridges classical aerodynamic principles with modern computational methods, offering a robust framework for analyzing the effects of wing damage on flight dynamics.

2 Introduction

On April 10, 2010, a Polish presidential aircraft, a Tupolev Tu-154, tragically crashed near Smolensk, Russia, claiming the lives of all 96 passengers, including high-ranking officials.(**Harro Ranter, 2024**) While the official investigation attributed the accident to a combination of pilot error and adverse weather conditions, public and political discourse has since raised broader questions about the dynamics of aircraft stability and control under extreme conditions. One particular area of interest lies in the aerodynamic behavior of damaged wings and how they impact the lift distribution and torque generated during flight.

This report focuses on analyzing the aerodynamic forces acting on aircraft wings, drawing upon the Lanchester-Prandtl lifting-line theory and the insights from Pope's Basic Wing and Airfoil Theory (1951).(**The History of Gaming: The Evolution of GPUs, n.d.**) The study begins with a detailed theoretical exploration of infinite wing models to establish fundamental concepts. It then progresses to finite wings, examining the effects of wingspan, aspect ratio, and wing shape on lift and induced drag. The theoretical framework includes the derivation of integral equations governing lift distribution, which are crucial for understanding the aerodynamics of symmetrical and asymmetrical wings.

Following the theoretical discussion, the report implements numerical simulations using Python, C, and Fortran. These simulations discretize the lifting-line theory into a system of linear equations to compute lift distribution and torque across various wing configurations. The simulations compare results for symmetrical and asymmetrical wings, quantifying the effects of finite wingspan and damage-induced asymmetry on aerodynamic performance. This computational approach provides a robust foundation for modeling real-world aerodynamic behaviors.

Lastly, the study generalizes the method to analyze irregularly shaped wings, exploring how physical constraints such as wing damage or asymmetry influence aircraft stability and control. By simulating various scenarios, the report contributes to a deeper understanding of lift and torque characteristics under challenging conditions.

Although this study does not directly examine the Smolensk air crash, its findings offer valuable insights into the broader principles of aerodynamics and aircraft stability. By bridging foundational aerodynamic theories with modern computational methods, the study highlights the enduring relevance of classical works like those of Lanchester, Prandtl, and Pope. This report systematically expands the analysis from infinite wings to finite and damaged wings, demonstrating the versatility and robustness of classical aerodynamic tools in addressing complex real-world challenges.

3 Theory

3.1 Key Terms

- **Airfoil** is the two-dimensional cross-sectional shape of an aircraft wing, designed to generate lift and reduce drag through aerodynamic properties.
- **Infinite Wing** is a hypothetical concept that refers to a wing with infinite wingspan. Since there is no wingtip vortex, it does not produce induced drag and is a theoretical model for studying airfoil performance.
- **Lift Coefficient** is a dimensionless parameter describing the magnitude of lift and is defined as:

$$C_L = \frac{L}{\frac{1}{2}\rho V^2 S}$$

  Where L is the lift force, ρ is the air density, V is the velocity of incoming flow, S is the reference area of wings.
- **Induced drag**: due to the influence of the finite wing tip vortex, lift is generated accompanied by additional drag, which is called induced drag.
- **Circulation,** Γ is a parameter that describes the strength of air flow around a wing and is defined as the velocity integral of a closed path around the airfoil:

$$\Gamma = \oint_C \vec{v} \cdot d\vec{l}$$

- **Biot-Savart Law**: is a fundamental principle in physics that describes the magnetic field generated by a steady electric current, it is also a fundamental principle used in aerodynamics to calculate the induced velocity generated by vortex filaments. It describes how the velocity at a point in space is influenced by a vortex segment, taking into account the strength of the vortex, the relative position of the point, and the distance from the vortex. Mathematically, it is expressed as:

$$\vec{V} = \frac{\Gamma}{4\pi} \int \frac{\vec{r} \times d\vec{l}}{r^3}$$

  where $\vec{V}$ is the induced velocity at a given point, $d\vec{l}$ is the infinitesimal segment of the vortex filament, $\vec{r}$ is the position vector from the vortex filament to the point, and r is the magnitude of the position vector.
- **Angle of Attack,** α is the angle of attack is the angle between the chord of the wing and the direction of the incoming flow, which directly affects the magnitude of the lift.Denote by AOA.
- **Tip Vortices** is the rotational airflow generated at the wingtips due to pressure differences between the upper (low pressure) and lower (high pressure) surfaces of a finite wing, where air leaks around the wingtips, forming spiraling vortex structures.
- **Effective AOA** is the actual angle of attack experienced by the wing, reduced from the geometric angle of attack due to the downwash induced by tip vortices.
- **Downwash** is caused by the wingtip vortices, which alters the local flow around the wing and reduces the effective angle of attack.
- **Induced Drag** is a type of drag caused by the tilting of the lift vector due to downwash, converting part of the lift into a rearward force.

- **Potential Flow Theory**: is a simplified fluid mechanics theory used to describe the flow behavior of inviscid and incompressible fluids.
- **Panel Method**: is a numerical computational method based on potential flow theory, the Panel Method is used to analyze the flow field distribution of incompressible, inviscid fluid around objects such as wings, fuselages, or entire aircraft.
- **Kutta Condition**: is a fundamental principle in aerodynamics that ensures a smooth flow of air over an airfoil, particularly at the trailing edge. It states that the flow leaves the trailing edge of the airfoil smoothly, with the velocity on the upper and lower surfaces of the airfoil meeting at a single point.

3.2 Infinite Wing

An infinite wing is an idealized aerodynamic model that assumes an infinitely long wingspan. By eliminating wingtip vortices and induced drag, the infinite wing simplifies the analysis of lift distribution, which is uniform along the span. For an infinite wing, lift is generated solely by the two-dimensional airflow around the airfoil, governed by Bernoulli's principle and the Kutta-Joukowski theorem:

$$L' = \rho V_\infty \Gamma$$

where L' is the lift per unit span, V_∞ is the free-stream velocity. The lift coefficient (C_L) is linearly proportinal the the AOA (α) for an infinite wing:

$$C_L = 2\pi\alpha$$

In figure 2.2.1 and 2.2.2, simulations using python demonstrate the lift distribution on a small airplane wing under the assumptions.

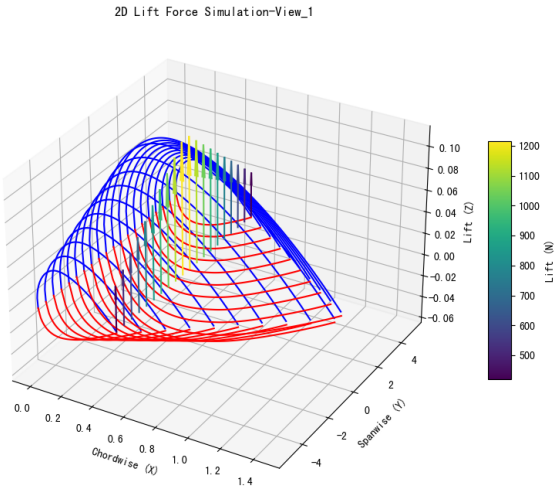

Figure 1: 2D-Simulation-View-1

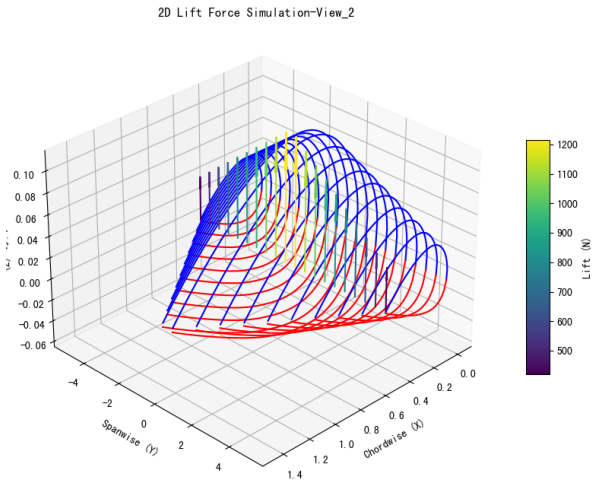

Figure 2: 2D-Simulation-View-2

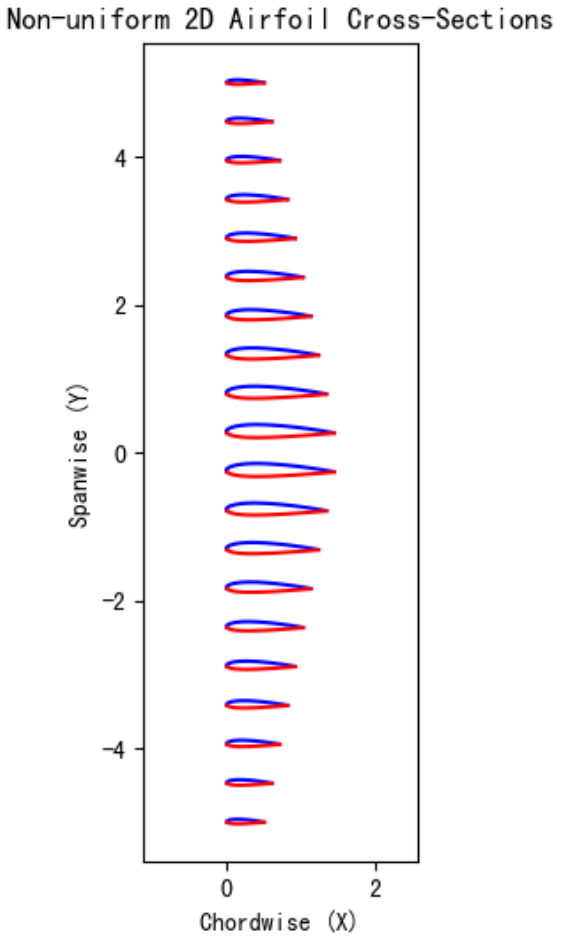

Figure 3: Cross-Section

Figure Description: The wing geometry was defined with a span of 10 meters, divided into 20 discrete sections along the spanwise direction, with a linearly varying chord length decreasing from 1.5 meters at the root to 0.5 meters at the tip. The airfoil shape was generated based on the NACA 4-digit airfoil equations, with a camber of 0.02, a camber position at 40% of the chord, and a maximum thickness of 12% of the chord length. Lift forces were calculated at each spanwise section using the lift coefficient formula $C_L = 2\pi\alpha$, where the angle of attack was set to 5° , air density was $1.225\,\mathrm{kg/m^3}$, and freestream velocity was $50\,\mathrm{m/s}$. The results, representing the spanwise lift distribution, were visualized using 3D plots, with arrows indicating lift vectors at each section, their length and color denoting the lift magnitude.

While the infinite wing model provides a fundamental understanding of airfoil performance and aerodynamic principles, it has inherent limitations when applied to real-world aircraft. By transitioning from the idealized infinite wing to the finite wing, we can bridge the gap between theoretical analysis and practical application, enabling a comprehensive study of lift, drag, and aerodynamic efficiency under realistic conditions.

3.3 Finite Wing

In an idealized two-dimensional model, such as infinite wing theory, aerodynamic performance is simplified by neglecting three-dimensional effects. However, when comparing this ideal model to a real three-dimensional wing, significant differences arise due to finite wingspan effects. These effects primarily stem from wingtip phenomena, where a pressure difference between the upper and lower wing surfaces causes air to leak from the high-pressure region below the wing to the low-pressure region above it, forming wingtip vortices.

As illustrated in Figure 2.1, the image provides a rearward view of one side of the wing, highlighting the blue-colored wingtip vortices. These vortices weaken in intensity as the distance from the wingtip increases. The wingtip vortices induce a downward velocity, known as downwash, which varies along the spanwise direction and is most pronounced near the wingtips. This downwash modifies the local flow field, impacting the aerodynamic performance of the wing.

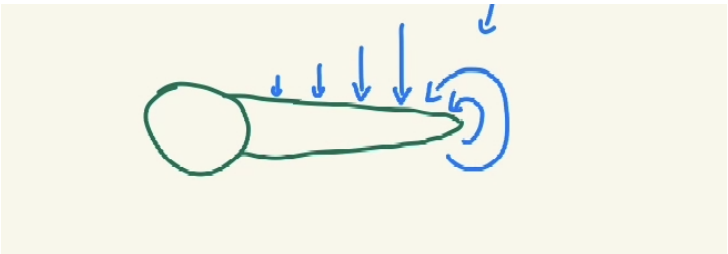

Figure 4: Tip Vortex

Downwash affects both the angle of attack (AOA) and the lift of an aircraft. As shown in Figure 2.2, the green line represents the cross-section of the wing, while the blue arrow pointing to the right indicates the free airflow (U_∞). The red downward arrow represents the downwash, which generates a downward induced velocity (U_{induced}), a velocity component perpendicular to U_∞ .This induced velocity reduces the AOA of the wing, resulting in an effective AOA (α_{eff}), calculated as: $\alpha_{\text{eff}} = \alpha - \alpha_i$. Where α_i is the induced angle of attack caused by U_{induced} .

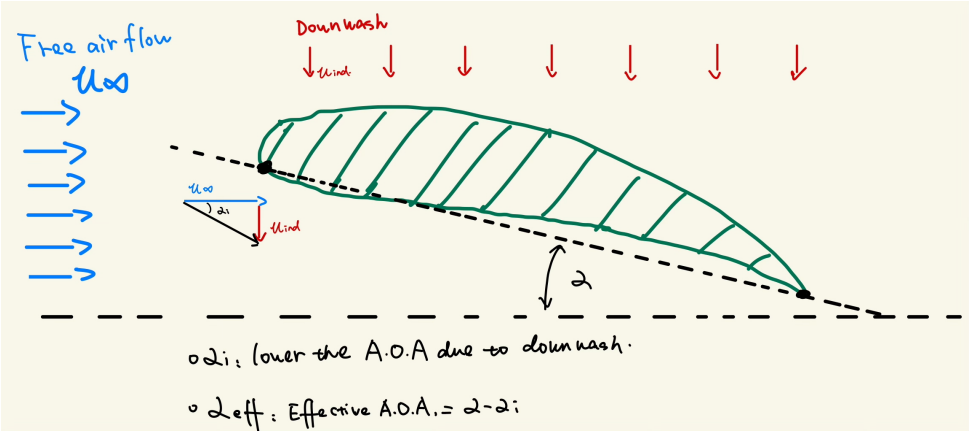

Figure 5: Downwash

Furthermore, downwash changes the lift force's direction, making it perpendicular to the total velocity vector rather than aligned with the flight direction. This adjustment reduces the vertical component of lift, as part of the lift is converted into induced drag.

The finite wingspan and downwash effects discussed earlier play a critical role in shaping the aerodynamic performance of a wing. These phenomena alter the effective angle of attack along the spanwise direction and lead to a non-uniform lift distribution, especially near the wingtips. To address these challenges, lifting line theory was developed as a mathematical framework to model the lift distribution and account for the influence of downwash on finite wings.

3.4 Lifting-Body Panel method

The Panel Method is a computational technique based on Potential Flow Theory used to calculate the lift of an object. However, the assumptions of Potential Flow Theory can lead to issues such as infinite velocity and indeterminate circulation at the trailing edge of an airfoil.

To resolve these problems, the Kutta Condition is introduced. It serves two main purposes: first, it determines the circulation of the fluid, ensuring the lift can be accurately calculated,

and second, it enforces smooth separation of the fluid at the trailing edge by ensuring that the velocities on the upper and lower surfaces meet with equal magnitude and direction.The Kutta Condition also states that the pressure on the lower and upper surfaces of the airfoil at the trailing edge must be equal. This ensures that the airflow does not curve around the trailing edge but instead leaves tangentially. Additionally, the rear stagnation point must be located precisely at the trailing edge.

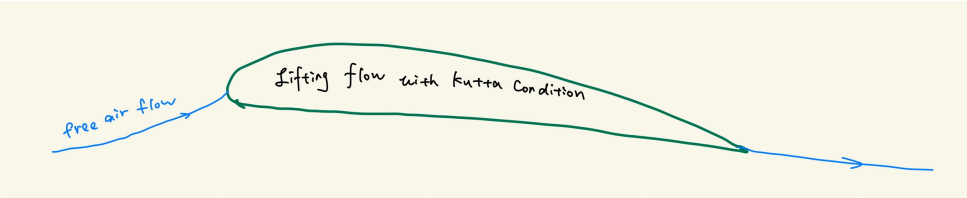

Figure 6: Kutta Condition

3.4.1 Integration Formula

Building on this theoretical foundation, the integral equation governing the velocity distribution on the airfoil is expressed as (**Katz & Plotkin, 2001**):

$$-\frac{1}{4\pi}\int_{\text{wing+wake}} \frac{\gamma_y(x-x_0)-\gamma_x(y-y_0)}{[(x-x_0)^2+(y-y_0)^2]^{3/2}}dx_0dy_0 = Q_\infty\left(\frac{\partial \eta_c}{\partial x}-\alpha\right)$$

This equation is based on the no-penetration boundary condition, which ensures that the induced velocity matches the free-stream's normal velocity component with opposite sign. The integral $\int_{\text{wing+wake}}$ covers the wing and wake surfaces, where $\gamma_y(x-x_0)$ and $\gamma_x(y-y_0)$ represent vortex strength distributions in the y- and x-directions, respectively. The term $[(x-x_0)^2+(y-y_0)^2]^{3/2}$ gives the cubed distance between two points on the surface. On the right, Q_∞ is the free-stream velocity, $\frac{\partial \eta_c}{\partial x}$ is the camber slope, and $-\alpha$ accounts for the angle of attack. Together, $Q_\infty\left(\frac{\partial \eta_c}{\partial x}-\alpha\right)$ defines the total normal velocity at the wing surface.

3.4.2 Numerical Solution

Directly solving the integral equation for an analytical solution is highly challenging. To simplify the problem, the lifting characteristics of the wing can be approximated by a single lifting line. Several other conditions also need to be considered.

Consider a thin, finite lifting wing moving at a constant speed in an undisturbed flow field. The free-stream velocity forms a small angle of attack, α, with respect to the wing. The flow around the wing can be described using a perturbation potential field Φ, which satisfies Laplace's equation:

$$\nabla^2\Phi = 0$$

On the surface of the wing, the following no-penetration boundary condition must hold:

$$\frac{\partial \Phi}{\partial z}(x,y,0^\pm) = Q_\infty\left(\frac{\partial \eta}{\partial x}-\alpha\right)$$

Where Q_∞ is the free-stream velocity, and $\eta(x,y)$ represents the camber surface of the wing.

To further develop the lifting line model and establish a linear system to solve for the circulation distribution $\Gamma(y)$, we start by considering Prandtl's lifting line theory. This theory simplifies the complex three-dimensional flow around a wing into a two-dimensional problem by assuming that the wing can be represented as a single bound vortex line (the lifting line) along the spanwise direction y.

The circulation distribution $\Gamma(y)$ varies along the span of the wing and is responsible for generating lift. Due to this varying circulation, trailing vortices are shed into the wake, inducing a downward velocity (downwash) w(y) at the wing. The induced downwash can be calculated using the Biot-Savart law:

$$w(y) = -\frac{1}{4\pi}\int_{-b/2}^{b/2}\frac{\frac{d\Gamma}{dy}}{y'-y}dy$$

The downwash also results in a change in the effective angle of attack (α_{eff}). The relationship between the induced velocity and the free-stream velocity u_∞ is expressed as:

$$\tan(\alpha_i) = \frac{-w(y)}{u_\infty}$$

Assuming α_i is small, the change in the angle of attack can be approximated as:

$$\alpha_i = \frac{-w(y)}{u_\infty}$$

Substituting $w(y)$ into the equation, the induced angle of attack α_i becomes:

$$\alpha_i = \frac{1}{4\pi}\int_{-b/2}^{b/2}\frac{\frac{d\Gamma}{dy}}{y'-y}dy$$

According to the Thin Airfoil Theory, the lift coefficient c_L is given by:

$$c_L = \frac{2\Gamma}{u_\infty c} = 2\pi(\alpha_{\text{eff}} - \alpha_{L=0}),$$

where Γ is the circulation, u_∞ is the free-stream velocity, c is the chord length, α_{eff} is the effective angle of attack, and $\alpha_{L=0}$ is the zero-lift angle of attack. Rearranging this equation, the effective angle of attack could be expressed as:

$$\alpha_{\text{eff}} = \frac{\Gamma}{\pi u_\infty c} + \alpha_{L=0}.$$

In general, the above relationship leads to the Fundamental Equation of Lifting Line Theory:

$$\alpha(y) = \frac{\Gamma(y)}{\pi c(y) u_\infty} + \alpha_{L=0} + \frac{1}{4\pi u_\infty}\int_{-b/2}^{b/2}\frac{\frac{d\Gamma(y')}{dy'}}{y-y'}dy'$$

The remaining task involves determining the circulation Γ. Based on lifting line theory, the wing span is discretized into N small segments, with corresponding control points and discrete vortex elements defined for each segment. At each control point, the no-penetration boundary condition is imposed, requiring that the sum of the normal velocity components induced by all discrete vortices equals the normal component of the free-stream velocity. The Biot-Savart law is utilized to calculate the induced velocity at each control point, linking the unknown circulation to the known free-stream conditions and angle of attack. These equations are formulated into a matrix system:

$$\mathbf{A\Gamma} = \mathbf{b},$$

where:

- $\mathbf{A}$ is the influence coefficient matrix, representing the contributions of discrete vortices to the induced velocity at each control point.
- $\mathbf{\Gamma}$ is the vector of unknown circulation values for the discrete vortex elements.
- $\mathbf{b}$ is the known vector determined by the free-stream velocity and angle of attack.

To ensure that the solution satisfies the physical boundary conditions, the Kutta condition is applied. Discrete vortices are typically placed at the 3/4-chord position, while the control points are located at the 1/4-chord position. At the wing tips, the circulation is constrained to be zero, satisfying the requirement of no lift at the wingtips.

4 Methods

For the wing, more detailed theoretical considerations are necessary, as discussed in Katz's book. The lifting line, located at a quarter chord length from the leading edge, is first discretized. Boundary conditions are imposed by evaluating the induced velocity and ensuring that air does not penetrate through the wing surface. To avoid numerical issues associated with a zero denominator, the circulation $\Gamma(y_i)$ is evaluated at the endpoints of each segment, while the boundary conditions are evaluated at the midpoints. This configuration prevents zero differences in the y-coordinate, ensuring numerical stability.

In the test case, the induced velocity is calculated for each segment from $y_i \rightarrow y_{i+1}$, and at each y_i, the contributions from the lifting line to infinity are considered. At each point y_i, the system solves for $\Gamma'(y_i)$, resulting in an $N \times N$ system of equations, where N represents the number of boundary condition points. Using Equation 2.69 from Katz's book, a system is constructed for the test case, and the cumulative sum of Γ' provides the circulation Γ.

Using formula 2.69 in Katz's book, a system was setup for the test case. The cumulative sum of Γ' will be Γ.

For the damaged wing, a similar set up was used, considering anhedral angle of the wing about 3 degrees and -1 degree at wing tip, and uniform spacing of the wing.

5 Results and Discussion

The results are presented in two figures, showing the spanwise distribution of vortex strength Γ.

- **Figure 1:** This figure simulates the vortex strength distribution $\Gamma(y)$ for a symmetric wing. The circulation shows a general symmetry along the span, with the highest Γ occurring near the center. However, on the right end of the wing, the circulation does not reach zero, which may indicate numerical or modeling inconsistencies at the wingtip.

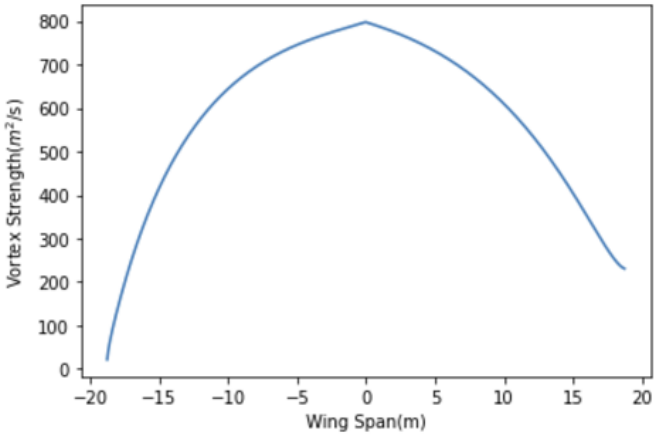

Figure 7: The distribution of vortex strength

- **Figure 2:** This figure represents the vortex strength distribution for a damaged wing, specifically with the right wing bent upward. As expected, the left wing generates significantly more lift compared to the damaged right wing. The asymmetry in $\Gamma(y)$ aligns with the physical expectation that the structural damage would reduce the lift on the right side, demonstrating the validity of the model's directional response. However, near the location of the structural damage, the vortex strength $\Gamma(y)$ shows an unexpected upward trend. This anomaly could be attributed to several factors, including the sudden geometric

discontinuity at the break, which alters local airflow patterns and induces additional vortex structures. Another potential cause is the numerical instability in calculating induced velocities at points close to the discontinuity, leading to localized inaccuracies. Future refinement of the model, such as increasing the discretization resolution or incorporating more robust handling of geometric discontinuities, may help address this behavior.

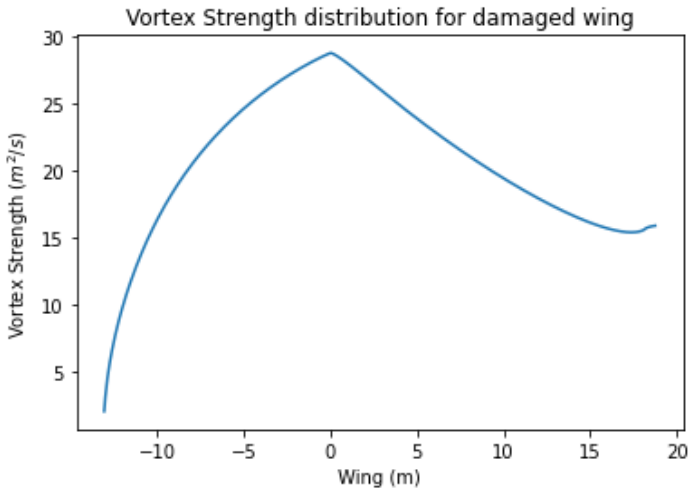

Figure 8: Vortex Strength distribution for damaged wing

5.1 Discussion and Further Improvements

- **Boundary Condition Optimization:** To address the issue of $\Gamma(y)$ not reaching zero at the wingtip, improvements to the numerical model, particularly the Kutta condition, are recommended.
- **Higher Resolution Discretization:** Increasing the number of discretized points along the spanwise direction may enhance accuracy.
- **Experimental Validation:** Wind tunnel experiments could be conducted to measure the vortex strength distribution and validate the model's predictions.

6 Conclusion

This study investigated the aerodynamic behavior of finite and damaged wings using the lifting line theory and numerical methods. The theoretical framework was based on discretizing the wing span and solving for the spanwise distribution of circulation $\Gamma(y)$ through a linear system of equations. For a symmetric wing, the simulation results showed a nearly symmetrical distribution of $\Gamma(y)$ with a peak near the center of the wing. However, discrepancies were observed near the wingtips, where the circulation did not fully decay to zero, suggesting potential refinements in the boundary conditions or numerical resolution.

For the damaged wing, the results accurately captured the expected asymmetry in lift distribution, with the left wing generating significantly more lift than the damaged right wing. However, an unexpected upward trend in $\Gamma(y)$ was observed near the structural damage point, likely caused by geometric discontinuities or numerical instabilities in induced velocity calculations.

These findings demonstrate the effectiveness of the lifting line theory in modeling aerodynamic performance, while highlighting the need for further refinement in handling geometric discontinuities and wingtip effects. Future work could focus on improving numerical stability,

increasing discretization resolution, and validating results with experimental data. This approach provides a robust framework for understanding the aerodynamic characteristics of finite and damaged wings, contributing to advancements in aircraft stability and design.

7 References

References

[1] Anderson, M. (2018, July 18). Famous Graphics Chips: NEC µPD7220 Graphics Display Controller. *IEEE Computer Society*. Retrieved from `https://www.computer.org/publications/tech-news/chasing-pixels/famous-graphics-chips`

[2] AWS. (n.d.). GPU vs CPU - Difference Between Processing Units. *Amazon Web Services, Inc.*. Retrieved from `https://aws.amazon.com/compare/the-difference-between-gpus-cpus/?nc1=h_ls`

[3] Henne, P. A. (1990). *Applied Computational Aerodynamics*. American Institute Of Aeronautics And Astronautics.

[4] IEEE Computer Society. (n.d.). IBM's PGC and 8514/A. *IEEE Computer Society*. Retrieved from `https://www.computer.org/publications/tech-news/chasing-pixels/Famous-Graphics-Chips-IBMs-professional-graphics-the-PGC-and-8514A`

[5] Katz, J., & Plotkin, A. (2001). *Low-Speed Aerodynamics: From Wing Theory to Panel Methods*. Cambridge University Press.

[6] Olena. (2018, February 22). A Brief History of GPU. *Medium*. Retrieved from `https://medium.com/altumea/a-brief-history-of-gpu-47d98d6a0f8a`

[7] Pope, A. (1951). *Basic Wing and Airfoil Theory*. Dover Publications.

[8] Shadow.tech. (n.d.). The History of Gaming: The Evolution of GPUs. *Shadow.tech*. Retrieved from `https://shadow.tech/en-GB/blog/history-of-gaming-gpus`

8 Appendix

8.1 Infinite Wing Lift Force Simulation

The following Python code was used to simulate the lift force distribution along the wing span based on infinite wing theory.

```
##Required libraries
import numpy as np
import matplotlib.pyplot as plt


# Define the parameters of wing
span = 10
num_sections = 40  \
# Number of sections the span is divided into

# Define the root and tip chord lengths
chord_root = 1.5
# Chord length at the root (meters)
chord_tip = 0.5
# Chord length at the tip (meters)

# Create an array of spanwise positions from -span/2 to span/2
y_positions = np.linspace(-span / 2, span / 2, num_sections)

# Function to calculate chord length at a given spanwise position
def linear_chord(y, span, chord_root, chord_tip):
    #Compute the chord based one liear relation
    return chord_root - (chord_root - chord_tip) * (2 * np.abs(y) / span)

# Function to generate airfoil coordinates based on given chord length
def generate_airfoil(chord, points=100):

    #Generates the upper and lower surface coordinates of an airfoil.
    m, p, t = 0.02, 0.4, 0.12  # Camber, camber position, and thickness
    x = np.linspace(0, 1, points)  # Non-dimensional x-coordinates

    # Camber line calculation
    yc = np.where(
        x < p,
        m / (p**2) * (2 * p * x - x**2),
        m / ((1 - p)**2) * ((1 - 2 * p) + 2 * p * x - x**2),  # Max camber to
            trailing edge
    )

    # Thickness distribution
    yt = 5 * t * (0.2969 * np.sqrt(x) - 0.126 * x - 0.3516 * x**2 + 0.2843 * x
        **3 - 0.1015 * x**4)

    xu = x - yt  # Upper x
    xl = x + yt  # Lower x
    zu = yc + yt  # Upper z
    zl = yc - yt  # Lower z

    return chord * xu, chord * zu, chord * xl, chord * zl

# Function to calculate lift force for a given chord length and flow conditions
def calculate_lift(chord, angle_of_attack, air_density, velocity):

    Cl = 2 * np.pi * np.radians(angle_of_attack)
    #lift coeeficient
    area = chord * 1
    # Reference area (chord x unit span)
```

```
    lift = 0.5 * air_density * velocity**2 * area * Cl
    # Lift force
    return lift


angle_of_attack = 5  # AOA
air_density = 1.225  # Air density (kg/m^3)
velocity = 65  # Free-stream velocity (m/s)

# Calculate chord lengths and lift forces for all spanwise sections
chord_lengths = [linear_chord(y, span, chord_root, chord_tip) for y in
    y_positions]
lifts = [calculate_lift(chord, angle_of_attack, air_density, velocity) for chord
     in chord_lengths]

# Function to visualize the wing geometry and lift distribution
def plot_wing_view(elev, azim, view_title):
    """
    Creates a 3D plot of the wing geometry and visualizes the lift distribution.
    elev: Elevation angle for the 3D view.
    azim: Azimuthal angle for the 3D view.
    view_title: Title of the plot.
    """
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # visualize cross section
    for y, chord_length in zip(y_positions, chord_lengths):
        xu, zu, xl, zl = generate_airfoil(chord_length)
        ax.plot(xu, [y] * len(xu), zu, 'b')  # Upper surface
        ax.plot(xl, [y] * len(xl), zl, 'r')  # Lower surface

    # Normalize lift values
    norm = plt.Normalize(min(lifts), max(lifts))
    colors = plt.cm.viridis(norm(lifts))

    # lift vectors
    for i, (y, lift) in enumerate(zip(y_positions, lifts)):
        ax.quiver(0.5, y, 0, 0, 0, lift / 1000, color=colors[i], length=0.1)  #
            Scaled lift vector


    sm = plt.cm.ScalarMappable(cmap="viridis", norm=norm)
    sm.set_array([])
    cbar = plt.colorbar(sm, ax=ax, shrink=0.5, aspect=10)
    cbar.set_label("Lift (N)")
    ax.set_xlabel("Chordwise (X)")
    ax.set_ylabel("Spanwise (Y)")
    ax.set_zlabel("Lift (Z)")
    ax.set_title(view_title)
    ax.view_init(elev=elev, azim=azim)
    plt.show()

# Visualize the wing and lift distribution from two different angles
plot_wing_view(elev=30, azim=-40, view_title="2D Lift Force Simulation - View 1"
    )
plot_wing_view(elev=30, azim=45, view_title="2D Lift Force Simulation - View 2")
```

Listing 1: Lift Force Simulation Code

8.2 Lifting force based on Lifting Line Theory-Undamaged Wings

The following Python code implements the lifting line theory by constructing a linear system to solve for the spanwise circulation distribution $\Gamma(y)$. The system uses the Biot-Savart law to compute the influence of discrete vortices and enforces the no-penetration boundary condition at control points. The resulting system of equations is represented in matrix form:

$$\mathbf{A\Gamma} = \mathbf{b},$$

where $\mathbf{A}$ is the influence coefficient matrix, $\mathbf{\Gamma}$ is the vector of unknown circulation values, and $\mathbf{b}$ is the known right-hand side vector determined by flow parameters.

```
import numpy as np
import matplotlib.pyplot as plt

import numpy as np
from scipy.linalg import solve
import matplotlib.pyplot as plt
N=1000
l_wing_range=int(1055/3755*N)
fule_range=int(1385*N/3755)
half=int(N/2)
print(l_wing_range)
print(fule_range)
wing_span=37.5
aoa=2/180*np.pi
Uinf=150
dy=(wing_span)/N
BC=np.zeros((N,2),dtype=np.float64)
LL=np.zeros((N,2),dtype=np.float64)
b=np.zeros(N,dtype=np.float64)
chord=np.zeros(N,dtype=np.float64)
LE_slope=np.tan(50*np.pi/180)
LL_slope=np.tan(27*np.pi/180)+0.75*(np.tan(50*np.pi/180)-np.tan(27*np.pi/180))
BC_slope=np.tan(27*np.pi/180)+0.25*(np.tan(50*np.pi/180)-np.tan(27*np.pi/180))
for i in range(half):
    chord[i]=2+i*2*(LL_slope-BC_slope)*dy
    chord[N-i-1]=chord[i]+2*(LL_slope-BC_slope)*dy
    BC[i,0]=-wing_span/2+(0.5+i)*dy
    BC[i,1]=(0.5+i)*dy*BC_slope
    BC[N-i-1,0]=-wing_span/2+(0.5+N-i-1)*dy
    BC[N-i-1,1]=BC[i,1]+dy*BC_slope
    LL[i,0]=BC[i,0]-dy/2
    LL[i,1]=i*dy*LL_slope+chord[i]/2
    LL[N-i-1,0]=-wing_span/2+(N-i-1)*dy
    LL[N-i-1,1]=LL[i,1]+dy*LL_slope
    #print(BC[i,0])
    #print(LL[i,0])
    b[i]=Uinf*aoa
    b[N-i-1]=Uinf*aoa
#Bound
A=np.zeros((N,N),dtype=np.float64)
#Induced
for j in range(N):
    for i in range(half):
        d_l=BC[j,0]-LL[i,0]
        d_r=BC[j,0]-LL[N-i-1,0]
        cb_l=(LL[i,1]-BC[j,1])/((d_l*d_l+(LL[i,1]-BC[j,1])**2)**0.5)
        #cb_2=(LL[i,1]-0.75*chord[i]-BC[j,1])/((d*d+(LL[i,1]-0.75*chord[i]-BC[j
            ,1])**2)**0.5)
        cb_2=-1
        cb_r=(LL[N-i-1,1]-BC[j,1])/((d_r*d_r+(LL[N-i-1,1]-BC[j,1])**2)**0.5)
        A[j,i]-=(cb_l-cb_2)*dy/(4*np.pi*d_l)
```

```
        A[j,N-i-1]+=(cb_r-cb_2)*dy/(4*np.pi*d_r)
for j in range(N):
    for i in range(half):

        dl=(pow(BC[j,0]-LL[i,0],2)+pow(BC[j,1]-LL[i,1],2))**0.5
        dr=(pow(BC[j,0]-LL[N-i-1,0],2)+pow(BC[j,1]-LL[N-1-i,1],2))**0.5
        dm_l=(pow(BC[j,0]-LL[half-1,0],2)+pow(BC[j,1]-LL[half-1,1],2))**0.5
        dm_r=(pow(BC[j,0]-LL[half,0],2)+pow(BC[j,1]-LL[half,1],2))**0.5
        dp_middle_l=(BC[j,0]-LL[half-1,0])*dy+(BC[j,1]-LL[half-1,1])*dy*LL_slope
        dp_middle_r=(BC[j,0]-LL[half,0])*dy-(BC[j,1]-LL[half,1])*dy*LL_slope
        dp_left=(BC[j,0]-LL[i,0])*dy+(BC[j,1]-LL[i,1])*dy*LL_slope
        dp_right=(BC[j,0]-LL[N-i-1,0])*dy-(BC[j,1]-LL[N-i-1,1])*dy*LL_slope
        cb_l=dp_left/(dy*((1+LL_slope*LL_slope)**0.5)*dl)
        cb_r=dp_right/(dy*((1+LL_slope*LL_slope)**0.5)*dr)
        cb_ml=dp_middle_l/(dy*((1+LL_slope*LL_slope)**0.5)*dm_l)
        cb_mr=dp_middle_r/(dy*((1+LL_slope*LL_slope)**0.5)*dm_r)
        A[j,i]+=dy*(cb_l-cb_ml)/(2*np.pi*chord[j])
        A[j,N-i-1]-=dy*(cb_r-cb_mr)/(2*np.pi*chord[j])
b=np.asarray(b)
gamma1=np.linalg.solve(A,b)
np.linalg.cond(A)
plt.plot(np.cumsum(gamma1)*dy)
```

8.3 Lifting force based on Lifting Line Theory-Damaged Wings

The following Python code implements the lifting line theory by constructing a linear system to solve for the spanwise circulation distribution $\Gamma(y)$, this time the wing is damaged.

```
import numpy as np
import matplotlib.pyplot as plt



LE_slope=np.tan(50*np.pi/180)
LL_slope=np.tan(27*np.pi/180)+0.75*(np.tan(50*np.pi/180)-np.tan(27*np.pi/180))
BC_slope=np.tan(27*np.pi/180)+0.25*(np.tan(50*np.pi/180)-np.tan(27*np.pi/180))


N=500
dy = (37.5-5.7) / N
dz = dy * 3 * np.pi/ 180
dz_t = -dy * 1 * np.pi/ 180
dt = 0.01
f_frac=int(18.75*N/(13.05+18.75))
tip_frac=int(N*0.02)
chord=np.zeros(N)
wing_span = 37.5
BC = np.zeros((N, 3))
LL = np.zeros((N, 3))
for i in range(int(tip_frac)):
    idx = N - i - 1
    chord[idx]=1.7+2* i * (LL_slope-BC_slope) * dy
    BC[idx,0] = (i + 0.5) * dy*BC_slope
    BC[idx,1] = wing_span / 2 - (i + 0.5) * dy
    BC[idx,2] = i * dz_t + 0.5 * dz_t
    LL[idx,0] = (i + 1) * dy * LL_slope
    LL[idx,1] = wing_span / 2 - (i + 1) * dy
    LL[idx,2] = i * dz_t

for i in range(int(tip_frac ), f_frac):
    idx = N - i - 1

```

```
    chord[idx]=1.7+2*i*(LL_slope-BC_slope)*dy
    BC[idx,0] = (i + 0.5) * dy * BC_slope
    BC[idx,1] = wing_span / 2 - (i + 0.5) * dy
    BC[idx,2] = BC[N - int(tip_frac ) ][2] - (i - int(tip_frac )) * dz
    LL[idx,0] = (i + 1) * dy * LL_slope
    LL[idx,1] = wing_span / 2 - (i + 1) * dy
    LL[idx,2] = LL[N - int(tip_frac ) ][2] - (i - int(tip_frac )) * dz

    #
for i in range(f_frac, N):
    idx = N - i - 1

    chord[idx]=chord[N-f_frac]-2*(i-f_frac+1)*(LL_slope-BC_slope)*dy
    BC[idx][0] = BC[N-f_frac][0] - (i - f_frac + 1) * dy*BC_slope
    BC[idx][1] = BC[N-f_frac][1] - (i - f_frac + 1) * dy
    BC[idx][2] = BC[N-f_frac][2] - (i - f_frac + 1) * dz
    LL[idx][0] = LL[N-f_frac][0] - (i - f_frac + 1) * dy*LL_slope
    LL[idx][1] = LL[N-f_frac][1] - (i - f_frac + 1) * dy
    LL[idx][2] = LL[N-f_frac][2] - (i - f_frac + 1) * dz

A = np.zeros((N, N))
for i in range(N):
    for j in range(int(N-f_frac)):
        diff=BC[i]-LL[j]
        cb1=np.dot([-1,0,0],diff)/((np.dot(diff,diff))**0.5)
        A[i,j]-=(cb1+1)/(4*np.pi*(BC[i][1]-LL[j][1]))*dy
for i in range(N):
    for j in range(int(N-f_frac),N):
        diff=BC[i]-LL[j]
        cb1=np.dot([-1,0,0],diff)/((np.dot(diff,diff))**0.5)
        A[i,j]+=(cb1+1)/(4*np.pi*(BC[i][1]-LL[j][1]))*dy
for i in range(N):
    if (i<N-f_frac):
        for j in range(int(N-f_frac)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[N-f_frac-1]
            delta=[LL_slope,1,np.pi/60]
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
            A[i][j]-=(cb1-cb2)/(2*np.pi*chord[i])*dy
    else:
        for j in range(int(N-f_frac)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[N-f_frac-1]
            delta=[LL_slope,1,np.pi/60]
            d=(np.dot(diff_l,diff_l)*(1-cb1*cb1))**0.5
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
            A[i][j]-=(cb1-cb2)/(4*np.pi*d)*dy
for i in range(N):
    if (i<N-f_frac):
        for j in range(int(N-f_frac),int(N-tip_frac)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[int(N-f_frac)]
            delta=[-LL_slope,1,-np.pi/60]
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
```

```
            d=(np.dot(diff_l,diff_l)*(1-cb1*cb1))**0.5
            A[i][j]-=(cb1-cb2)/(4*np.pi*d)*dy
    else:
        for j in range(int(N-f_frac),int(N-tip_frac)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[int(N-f_frac)]
            delta=[-LL_slope,1,-np.pi/60]
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
            A[i][j]-=(cb1-cb2)/(2*np.pi*chord[i])*dy
for i in range(N):
    if (i<N-f_frac):
        for j in range(int(N-tip_frac),int(N)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[int(N-tip_frac)]
            delta=[-LL_slope,1,np.pi/180]
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
            d=(np.dot(diff_l,diff_l)*(1-cb1*cb1))**0.5
            A[i][j]-=(cb1-cb2)/(4*np.pi*d)*dy
    else:
        for j in range(int(N-tip_frac),int(N)):
            diff_l=BC[i]-LL[j]
            diff_r=BC[i]-LL[int(N-tip_frac)]
            delta=[-LL_slope,1,np.pi/180]
            cb1=np.dot(delta,diff_l)/(np.dot(delta,delta)*np.dot(diff_l,diff_l))
                **0.5
            cb2=np.dot(delta,diff_r)/(np.dot(delta,delta)*np.dot(diff_r,diff_r))
                **0.5
            A[i][j]-=(cb1-cb2)/(2*np.pi*chord[i])*dy
b=[]
U_inf=50
f_frac=int(18.75*N/(13.05+18.75))
aoa=12/180*np.pi
wind=[-np.cos(aoa),0,np.sin(aoa)]
for i in range(int(N-f_frac)):
    n=[0,1,3/180*np.pi]
    b.append(U_inf*np.dot(wind,n)/(np.dot(n,n))**0.5)
for i in range(int(N-f_frac),int(N-tip_frac)):
    n=[0,-1,3/180*np.pi]
    b.append(U_inf*np.dot(wind,n)/(np.dot(n,n))**0.5)
for i in range(int(N-tip_frac),N):
    n=[0,1,1/180*np.pi]
    b.append(U_inf*np.dot(wind,n)/(np.dot(n,n))**0.5)
g=np.linalg.solve(A,-np.asarray(b))

y=np.linspace(-13.05,18.75,len(np.cumsum(g)))
plt.plot(y,np.cumsum(g)*dy)
plt.title("Vortex␣Strength␣distribution␣for␣damaged␣wing")
plt.ylabel("Vortex␣Strength␣($m^2/s$)")
plt.xlabel("Wing␣(m)")
```