Documentation and Validation of Py2Fly

Brian C. Vermeire and Sai Niranjan Jyothimahalingam

February 16, 2022

Contents

List of Figures

1 Thin Airfoil theory

1.1 Summary and Derivation [1]

Thin airfoil theory allows us to predict airfoil performance characteristics such as lift and moment coefficients at different angles of attacks. The airfoils are modeled as 'thin' airfoils, allowing to model the airfoil by placing a vortex sheet along its camber line. The theory then has to satisfy two conditions calculating the variation of the circulation along this camberline, $\gamma(s)$, such that the camber line is a streamline of the flow, and $\gamma(TE) = 0$; i.e. satisfying Kutta condition at the trailing edge. The total circulation Γ can then be found out by integrating $\gamma(s)$ from leading edge to trailing edge. Using Kutta-Joukowski theorem, the Lift is then derived from this circulation.

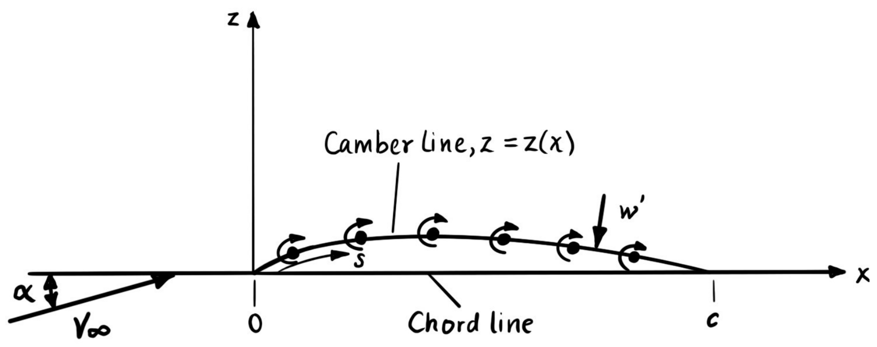

Figure 1.1: Vortex sheet placed on the camber line

Consider a *thin* airfoil in incompressible fluid flow with a freestream velocity V_∞ and angle of attack α. For *thin* airfoils, a reasonable assumption we can make is that a vortex sheet placed on the surface of the airfoil can be approximated by a vortex sheet placed on the camberline, which in turn is the same as a vortex sheet placed on the chord line for moderate camber.

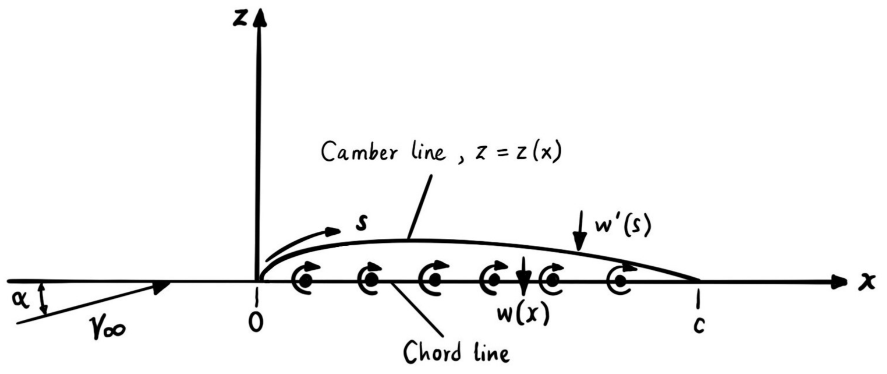

Figure 1.2: Vortex sheet placed on the chord line

As shown in figures 1.1 and 1.2, the *x axis* falls on the chord line and *z axis* is perpendicular to the chord. The chord length is c. The camberline is defined by $z = z(x)$ and the distance measured along it is denoted by s. When placed on the camberline, the vortex sheet induces velocity $\omega^{'}$ which is normal to the camber line; $\omega^{'} = \omega^{'}(s)$. Now, considering the vortex sheet is on the chord as shown in the figure 1.2, here $\gamma = \gamma(x)$. The strength is calculated such that the camber line is a streamline of the flow and also $\gamma(c) = 0$ (satisfying Kutta condition).

If the camber line has to be a streamline of the flow, it means the component of velocity normal to the camber line must be zero at all points along the camber line. If the component of the freestream velocity normal to the camber line is denoted by $V_{\infty,n}$ then, at every point on the camber line

$$V_{\infty,n} + \omega^{'}(s) = 0. \tag{1.1}$$

Now the normal component of freestream velocity can be determined simply by geometry. Refer figure 1.3. At any point where the slope is given by dz/dx,

$$V_{\infty,n} = V_{\infty} sin[\alpha + tan^{-1}(-dz/dx)]$$

. Approximating $sin\theta \approx tan\theta \approx \theta$,

$$V_{\infty,n} = V_{\infty}\left(\alpha - \frac{dz}{dx}\right). \tag{1.2}$$

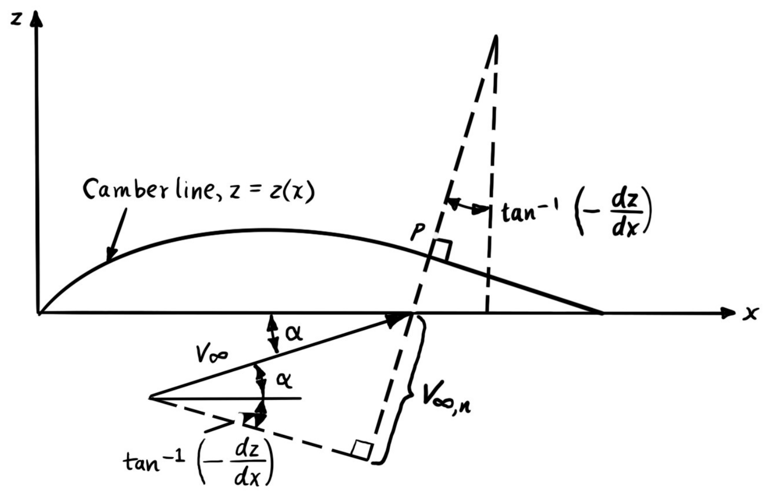

Figure 1.3: Component of Freestream velocity normal to the camberline

Now we have to determine an expression for $\omega^{'}(s)$. Again, since we are dealing with *thin* airfoils, the component of velocity (induced by the vortex sheet) normal to the camber line can be approximated to the component of the velocity normal to the chord line.

$$\omega^{'}(s) \approx \omega(x)$$

.

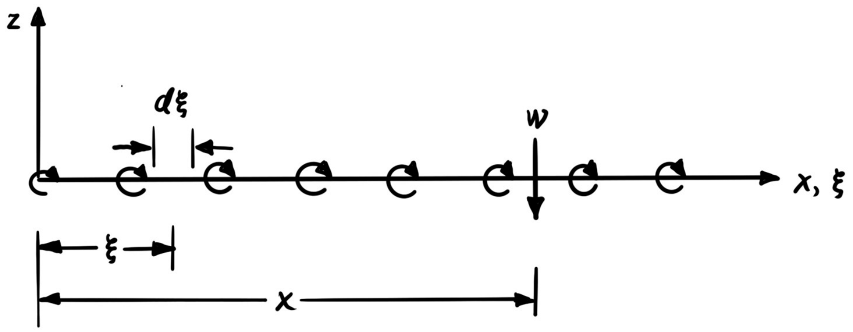

Figure 1.4: Induced velocity on the chord line

To find an expression for $\omega(x)$, consider figure 1.4 which shows the vortex sheet along the chord. An elemental vortex of strength $\gamma d\xi$ is considered at an distance ξ from the origin along the chord as shown. The strength of the vortex sheet is a function of the distance along the chord; i.e. $\gamma = \gamma(\xi)$.

The velocity $d\omega$ at point x induced by this elemental vortex is given by

$$d\omega = -\frac{\gamma(\xi)d\xi}{2\pi(x-\xi)}. \tag{1.3}$$

By integrating this expression from the leading edge ($\xi = 0$) to the trailing edge ($\xi = c$), we can obtain $\omega(x)$ as

$$\omega(x) = -\int_0^c \frac{\gamma(\xi)d(\xi)}{2\pi(x-\xi)}. \tag{1.4}$$

Substituting equations 1.2 and 1.4 in equation 1.1 we get

$$V_\infty\left(\alpha - \frac{dz}{dx}\right) - \int_0^c \frac{\gamma(\xi)d(\xi)}{2\pi(x-\xi)} = 0, \tag{1.5}$$

which is

$$\boxed{\frac{1}{2\pi}\int_0^c \frac{\gamma(\xi)d(\xi)}{x-\xi} = V_\infty\left(\alpha - \frac{dz}{dx}\right).} \tag{1.6}$$

This is the fundamental equation of thin airfoil theory.

It is imperative to note that for a given airfoil at a given angle of attack, both α and dz/dx will be known. Hence, in equation 1.6, the vortex strength $\gamma(\xi)$ will be the only unknown. The critical step now is to solve this equation for $\gamma(\xi)$ such that it satisfies the Kutta condition, i.e. $\gamma(c) = 0$.

Let's take the case of a particular arbitrary airfoil. To deal with the integral in the fundamental equation, transform the variable ξ to θ using the transformation

$$\xi = \frac{c}{2}(1 - cos\theta)$$

. Differentiating this, we get

$$d\xi = \frac{c}{2}sin\theta d\theta$$

.

The value of x corresponds to a particular θ, let it be θ_0, such that

$$x = \frac{c}{2}(1 - cos\theta_0)$$

. The limits of the integration transform to $\theta = 0$ when $\xi = 0$ and $\theta = \pi$ when $\xi = c$. Substituting all these in the equation, we get

$$\frac{1}{2\pi}\int_0^\pi \frac{\gamma(\theta)sin\theta d\theta}{cos\theta - cos\theta_0} = V_\infty\left(\alpha - \frac{dz}{dx}\right). \tag{1.7}$$

Equation 1.7 has to be solved for $\gamma(\theta)$, such that it satisfies the Kutta condition i.e. $\gamma(\pi) = 0$. Such a solution will make the camber line a streamline of the flow. This solution is directly stated below

$$\boxed{\gamma(\theta) = 2V_\infty \left(A_0 \frac{1 + cos\theta}{sin\theta} + \sum_{n=1}^{\infty} A_n sinn\theta \right).} \tag{1.8}$$

The values of the coefficients, A_n depends on the slope of the camber line dz/dx and A_0 depends on both dz/dx and α. Substituting equation 1.8 in equation 1.7

$$\frac{1}{\pi} \int_0^\pi \frac{A_0(1 + cos\theta) d\theta}{cos\theta - cos\theta_0} + \frac{1}{\pi} \sum_{n=1}^{\infty} \int_0^\pi \frac{A_n sinn\theta sin\theta d\theta}{cos\theta - cos\theta_0} = \alpha - \frac{dz}{dx}. \tag{1.9}$$

Both the integral terms appearing above are forms of standard integrals which can be reduced as follows

$$\int_0^\pi \frac{cosn\theta d\theta}{cos\theta - cos\theta_0} = \frac{\pi sinn\theta_0}{sin\theta_0}, \tag{1.10}$$

$$\int_0^\pi \frac{sinn\theta sin\theta d\theta}{cos\theta - cos\theta_0} = -\pi cosn\theta_0. \tag{1.11}$$

Hence equation 1.9 can be simplified to

$$A_0 - \sum_{n=1}^{\infty} A_n cosn\theta_0 = \alpha - \frac{dz}{dx}$$

or

$$\frac{dz}{dx} = (\alpha - A_0) + \sum_{n=1}^{\infty} A_n cosn\theta_0. \tag{1.12}$$

This equation is in the form of a Fourier cosine series. In general, Fourier cosine series of a function $f(\theta)$ over an interval $0 \leq \theta \leq \pi$ is represented as

$$f(\theta) = B_0 + \sum_{n=1}^{\infty} B_n cosn\theta, \tag{1.13}$$

where the coefficients are given by

$$B_0 = \frac{1}{\pi} \int_0^\pi f(\theta) d\theta,$$
$$B_n = \frac{2}{\pi} \int_0^\pi f(\theta) cosn\theta d\theta.$$

Thus, the coefficients in equation 1.12 can be represented as

$$(\alpha - A_0) = \frac{1}{\pi} \int \frac{dz}{dx} d\theta_0$$

or

$$A_0 = \alpha - \frac{1}{\pi}\int_0^{\pi} \frac{dz}{dx} d\theta_0, \tag{1.14}$$

$$A_n = \frac{2}{\pi}\int_0^{\pi} \frac{dz}{dx} cosn\theta_0 d\theta_0. \tag{1.15}$$

It can be seen from the expressions that A_0 depends on both α and the shape of the camberline(dz/dx), and the values of A_n depend only on the shape of the camberline. Hence, for a specific airfoil at a given angle of attack, the values of A_0 and A_n can be calculated.

Next, we have to calculate the expressions for the aerodynamic coefficients. The total circulation due to the vortex sheet from the leading edge to the trailing edge is given by

$$\Gamma = \int_0^c \gamma(\xi)d\xi = \frac{c}{2}\int_0^{\pi} \gamma(\theta) sin\theta d\theta. \tag{1.16}$$

Substitute the expression of $\gamma(\theta)$ given by equation 1.8 into equation 1.16

$$\Gamma = cV_\infty \left(A_0 \int_0^{\pi} (1 + cos\theta)d\theta + \sum_{n=1}^{\pi} A_n \int_0^{\pi} sinn\theta sin\theta d\theta \right). \tag{1.17}$$

From theory of integral calculus, we know

$$\int_0^{\pi} (1 + cos\theta)d\theta = \pi \int_0^{\pi} sinn\theta sin\theta d\theta = \begin{cases} \pi/2 & for\ n = 1 \\ 0 & for\ n \neq 1 \end{cases}$$

Hence equation 1.17 becomes

$$\Gamma = cV_\infty \left(\pi A_0 + \frac{\pi}{2} A_1 \right). \tag{1.18}$$

Therefore, lift per unit span can be expressed as

$$L^{'} = \rho_\infty V_\infty \Gamma = \rho_\infty V_\infty^2 c \left(\pi A_0 + \frac{\pi}{2} A_1 \right). \tag{1.19}$$

Thus coefficient of lift can be expressed as

$$c_l = \frac{L^{'}}{\frac{1}{2}\rho_\infty V_\infty^2 c(1)} = \pi(2A_0 + A_1). \tag{1.20}$$

which can be written as

$$\boxed{c_l = 2\pi \left[\alpha + \frac{1}{\pi}\int_0^{\pi} \frac{dz}{dx}(cos\theta_0 - 1)d\theta_0 \right],} \tag{1.21}$$

and

$$\boxed{\text{Lift slope} \equiv \frac{dc_l}{d\alpha} = 2\pi.} \tag{1.22}$$

It is important to note that, irrespective of the airfoil characteristics, thin airfoil theory predicts that the value of the lift slope will always be 2π.

Next, the moment about leading edge has to be calculated. As shown in figure 1.5, a elemental vortex of strength $\gamma(\xi)d\xi$ located at a distance of ξ from the leading edge creates a circulation whose value is given by $d\Gamma = \gamma(\xi)d\xi$. So, the incremental lift created by this elemental vortex is $dL = \rho_\infty V_\infty d\Gamma$. This creates a moment about the leading edge $dM = -\xi(dL)$.

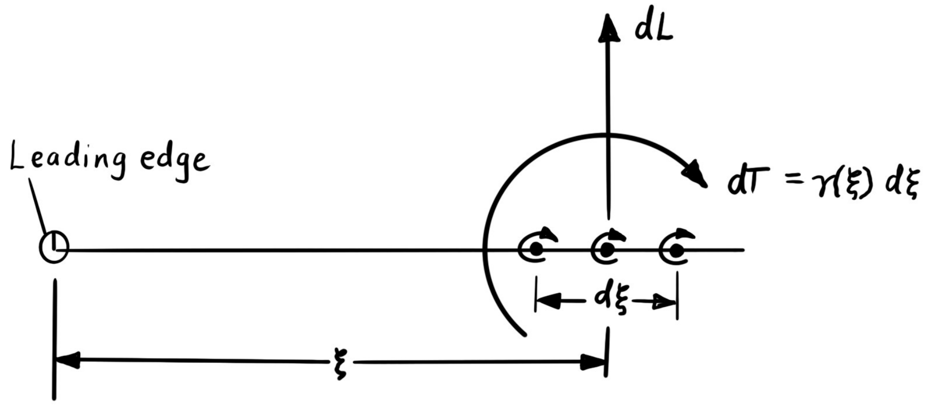

Figure 1.5: Moments about the leading edge

The total moment per unit span about the LE, therefore, can be calculated as

$$M'_{LE} = -\int_0^c \xi(dL) = -\rho_\infty V_\infty \int_0^c \xi\gamma(\xi)d\xi. \tag{1.23}$$

We also know the expression for the moment coefficient

$$c_{m,le} = \frac{M'_{LE}}{q_\infty Sc}. \tag{1.24}$$

Using similar transformation as was done before in the derivation of $\gamma(\theta)$ (equation 1.8), and performing the integration, and substituting $S = c * (1)$(considering unit depth), we get

$$c_{m,le} = -\frac{\pi}{2}\left(A_0 + A_1 - \frac{A_2}{2}\right). \tag{1.25}$$

Substituting equation 1.20 in 1.25, we obtain

$$c_{m,le} = -\left[\frac{c_l}{4} + \frac{\pi}{4}(A_1 - A_2)\right]. \tag{1.26}$$

The moment coefficient about the quarter-chord point is given by

$$c_{m,c/4} = c_{m,le} + \frac{c_l}{4}. \tag{1.27}$$

Substituting equation 1.26 in equation 1.27, we obtain

$$\boxed{c_{m,c/4} = \frac{\pi}{4}(A_2 - A_1).} \tag{1.28}$$

The location of the centre of pressure is given by

$$x_{cp} = -\frac{M'_{LE}}{L'} = -\frac{c_{m,le}c}{c_l}. \tag{1.29}$$

Substituting the expression for $c_{m,le}$ from equation 1.26, we can obtain the expression

$$\boxed{x_{cp} = \frac{c}{4}\left[1 + \frac{\pi}{c_l}(A_1 - A_2)\right].} \tag{1.30}$$

1.2 The Python Code

This section explains the crucial elements of the python code dealing with Thin Airfoi theory. The inputs given are geometry of the airfoil, specifically the coordinates of the camberline (‘camberline.txt’), and the range of values of the angle of attack. The outputs are aerodynamic characteristics and several plots. Some snippets of the code are attached here with a brief contextual explanation.

Listing 1: Input values

```
# Sweep over the desired angles of attack
minalpha = -6
maxalpha = 10
alphastep = 1
sweep(minalpha, maxalpha, alphastep, plotgam)
```

Towards the end of the code, you can find the values input for angle of attack, and the function ‘sweep’ is called. This function is defined mainly to calculate the aerodynamic characteristics at each value of angle of attack. The function is defined as follows.

Listing 2: Sweep function

```
def sweep(minalpha, maxalpha, alphastep, plotgam):
    # Initialize plotting arrays
    alpha_plot = []
    cl_plot = []
    cm_c4_plot = []
    x_cp_plot = []
```

```
# Start by loading the camber line points from disk
x_c = np.loadtxt('camberline.txt')

# Compute the corresponding theta positions
theta = np.arccos(1 - 2 * x_c[:, 0])

# Approximate the slope of the camber line
dz_dx = np.zeros(np.shape(x_c)[0])
dz_dx[0] = (x_c[1, 1] - x_c[0, 1]) / (x_c[1, 0] - x_c[0, 0])
dz_dx[-1] = (x_c[-1, 1] - x_c[-2, 1]) / (x_c[-1, 0] - x_c[-2, 0])
for i in range(1, np.shape(x_c)[0] - 1):
    dz_dx[i] = (x_c[i + 1, 1] - x_c[i, 1]) / (x_c[i + 1, 0] - x_c[i, 0])

# Sweep over a range of angles of attack
alpha = minalpha
while alpha <= maxalpha:
    [cl, cm_c4, x_cp] = thin_airfoil(alpha, plotgam, x_c, theta, dz_dx)

    # Append data to the plotting arrays
    alpha_plot.append(alpha)
    cl_plot.append(cl)
    cm_c4_plot.append(cm_c4)
    x_cp_plot.append(x_cp)

    alpha = alpha + alphastep
```

It can be noted here that the file camberline.txt is used to input the points of the camberline of the airfoil used, hence specifying the geometry. The other crucial part is calling of the function 'thin_airfoil'. This function is at the heart of the code, mainly specifying all the calculations of the aerodynamic coefficients. Some sections are discussed below.

Listing 3: A_0 and A_n Coefficients

```
# Generate the spectral coefficient array
A = np.zeros(math.floor(np.shape(x_c)[0] / 2))

# Compute the integrals for each A coefficient
A[0] = alpha - 1.0 / math.pi * integrate(theta, dz_dx)
for i in range(1, np.shape(A)[0]):
    A[i] = 2.0 / math.pi * integrate(theta, np.multiply(dz_dx, np.cos(i * theta)))
```

This section inside the function defined 'thin_airfoil' shows the calculation of each A[ ] coefficient clearly using the expressions obtained in equations 1.14 and 1.15.

Listing 4: Circulation

```
# Generate the gamma over U distribution to plot skipping the leading edge
gam = np.zeros(np.shape(x_c)[0])
for i in range(1, np.shape(x_c)[0]):
        gam[i] = A[0] * ((1 + math.cos(theta[i])) / (math.sin(theta[i])))
        for j in range(1, np.shape(A)[0]):
            gam[i] = gam[i] + A[j] * math.sin(j * theta[i])
    gam = 2 * gam
```

This section calculates the values of $\gamma(\theta)$ by applying equation 1.8. The term is split into two loops with the nested loop taking care of the Sum(Σ) expression.

Listing 5: Formulae

```
    cl = math.pi * (2 * A[0] + A[1])
    cm_c4 = math.pi / 4 * (A[2] - A[1])
    x_cp = 1 / 4 * (1 + math.pi / cl * (A[1] - A[2]))
```

These lines are simply application of the various expressions obtained for c_l, $c_{m,c/4}$, x_{cp} from the Thin airfoil theory.

1.3 Test case

The efficacy of any code can be judged by the comparison of the results from the code with experimental data. NACA 2412 airfoil has been taken as a test case, and the experimental data (obtained from reference [4]) has been compared and plotted as below. It can be observed that the results are fairly close.

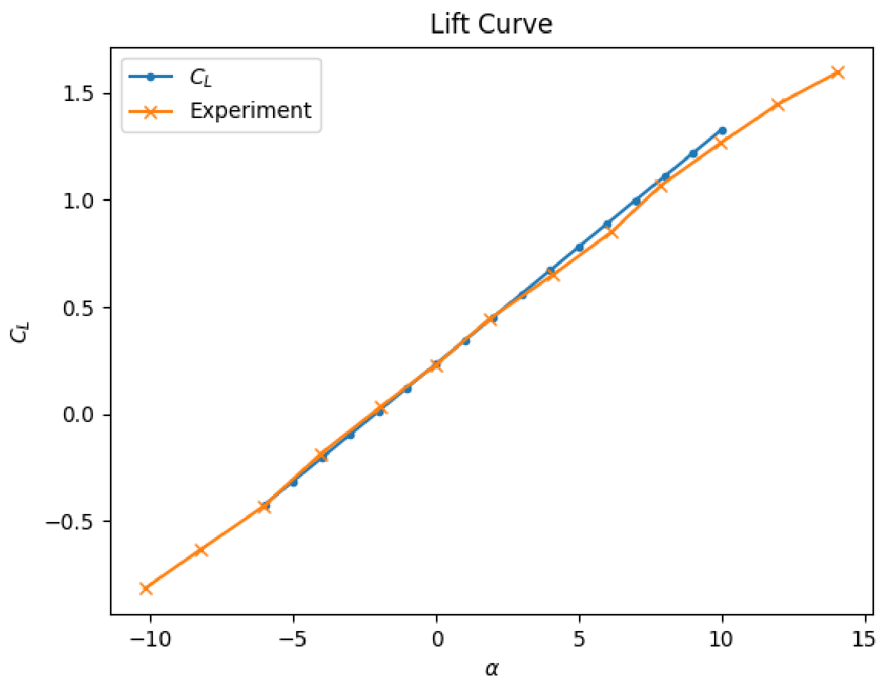

Figure 1.6: NACA2412 - C_L vs α

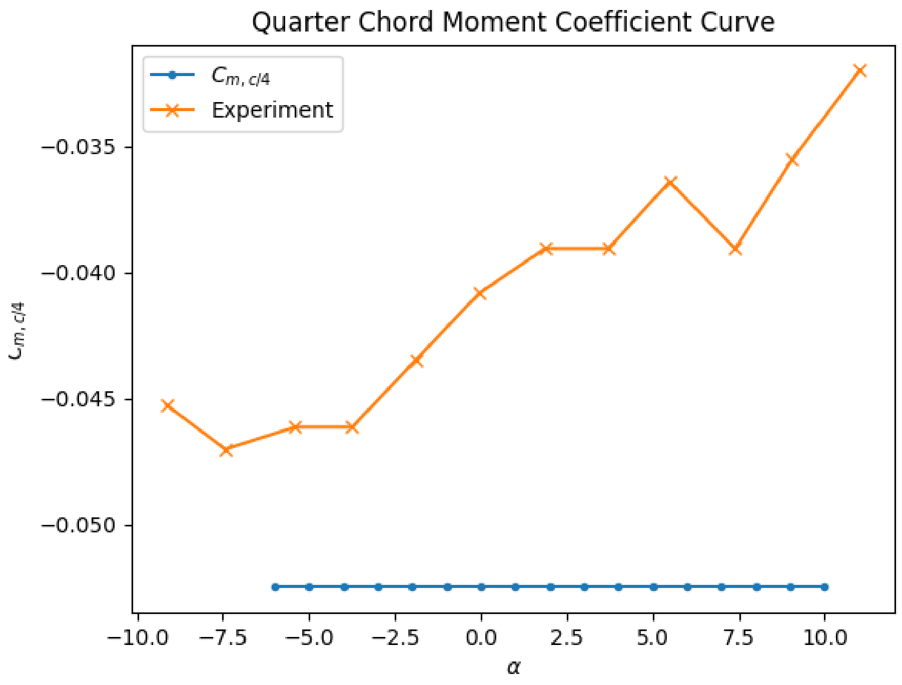

Figure 1.7: NACA2412 - Moment coefficient

2 Vortex Panel Method

2.1 Summary and Derivation [2]

In this method, the airfoil considered is represented by a closed polygon comprised of vortex panels as shown in figure 2.1. The number of panels is denoted by m, they're assumed to be planar and are named in clockwise direction starting from the trailing edge as shown in the figure. On each panel, the circulation density varies from one corner to the other in a linear fashion and is continuous across the corner.

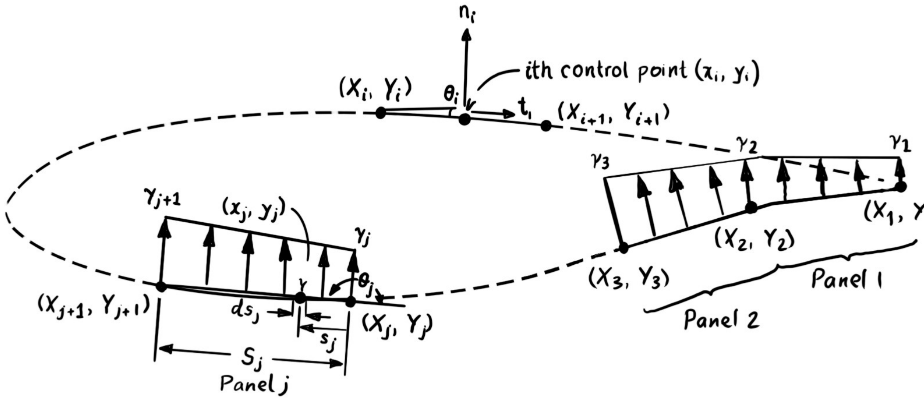

Figure 2.1: Airfoil represented by Vortex panels

The condition that the surface of the airfoil should be a streamline of the flow is satisfied by equating the normal component of velocity to zero at *control points*, which are specified as the mid points of the panels. The end points of each panel are termed as the *boundary points*.

The velocity potential ϕ at the ith control point (x_i, y_i) is given by equation 2.1, where V_∞ is the velocity of the uniform flow, α is the angle of attack, m is the number of panels

$$\phi(x_i, y_i) = V_\infty(x_i cos\alpha + y_i sin\alpha) - \sum_{j=1}^{m} \int_j \frac{\gamma(s_j)}{2\pi} tan^{-1}\left(\frac{y_i - y_j}{x_i - x_j}\right) ds_j, \tag{2.1}$$

where

$$\gamma(s_j) = \gamma_j + (\gamma_{j+1} - \gamma_j)\frac{s_j}{S_j}. \tag{2.2}$$

Now, consider the boundary condition represented in figure 2.1. It requires that the velocity in the direction of n_i, which is the unit vector directed outwards normal to the ith control panel, be equal to zero. So

$$\frac{\partial}{\partial n_i}\phi(x_i, y_i) = 0; \qquad i = 1, 2, ..., m$$

.

Carrying out the differentiation steps, we finally obtain

$$\sum_{j=1}^{m}(C_{n1_{ij}}\gamma_j^{'} + C_{n2_{ij}}\gamma_{j+1}^{'}) = sin(\theta_i - \alpha); \qquad i = 1, 2, ..., m, \tag{2.3}$$

where $\gamma^{'} = \gamma/2\pi V_\infty$ is a dimensionless quantity representing circulation density, θ_i is the orientation angle of the ith panel which is the angle between the panel surface and x axis and the coefficients are expressed as

$$\begin{aligned} C_{n1_{ij}} &= 0.5DF + CG - C_{n2_{ij}} \\ C_{n2_{ij}} &= D + 0.5QF/S_j - (AC + DE)G/S_j \end{aligned}$$

.

These new constants in the above expressions are defined as

$$\begin{aligned} A &= -(x_i - X_j)cos\theta_j - (y_j - Y_j)sin\theta_j \\ B &= (x_i - X_j)^2 + (y_i - Y_j)^2 \\ C &= sin(\theta_i - \theta_j) \\ D &= cos(\theta_i - \theta_j) \\ E &= (x_i - X_j)sin\theta_j - (y_i - Y_j)cos\theta_j \\ F &= ln\left(1 + \frac{S_j^2 + 2AS_j}{B}\right) \\ G &= tan^{-1}\left(\frac{ES_j}{B + AS_j}\right) \\ P &= (x_i - X_j)sin(\theta_i - 2\theta_j) + (y_i - Y_j)cos(\theta_i - 2\theta_j) \\ Q &= (x_i - X_j)cos(\theta_i - 2\theta_j) - (y_i - Y_j)sin(\theta_i - 2\theta_j) \end{aligned}$$

.

These constants can be calculated for all possible values of i and j once the panel geometry is specified.

The expression on the left hand side of equation 2.3 enclosed in the parantheses denotes the normal velocity at the ith control point induced by the vortices on the jth panel. If we equate $i = j$, we get the self-induced normal velocity at the ith control point, the coefficients are simplied

$$C_{n1_{ij}} = -1 \qquad and \qquad C_{n2_{ij}} = 1.$$

Now, the Kutta condition has to be applied to ensure a smooth flow at the trailing edge. In the present context, this condition (strength of the vorticity at the trailing edge equals zero) can be represented as

$$\gamma_1^{'} + \gamma_{m+1}^{'} = 0. \tag{2.4}$$

Combining the equations 2.3 and 2.4, we have $(m+1)$ number of equations which are sufficient to calculate $(m+1)$ number of unknown values of $\gamma_j^{'}$.

To facilitate in writing the code to implement this method, this system of simultaneous equations can be written in a more convenient form as

$$\sum_{j=1}^{m+1} A_{n_{ij}} \gamma_j^{'} = RHS_i; \qquad i = 1, 2, ..., m+1. \tag{2.5}$$

Here, for $i \leq m+1$

$$\begin{aligned} A_{n_{i1}} &= C_{n1_{i1}} \\ A_{n_{ij}} &= C_{n1_{ij}} + C_{n2_{ij-1}}; \qquad j = 2, 3, ..., m \\ A_{n_{im+1}} &= C_{n2_{im+1}} \\ RHS_i &= sin(\theta_i - \alpha) \end{aligned}$$

, and, for $i = m+1$

$$\begin{aligned} A_{n_{i1}} &= A_{n_{im+1}} = 1 \\ A_{n_{ij}} &= 0; \qquad j = 2, 3, ..., m \\ RHS_i &= 0 \end{aligned}$$

.

2.2 Explanation of the code

This section deals with a brief explanation of the code associated with the Vortex panel method. The input parameters are the geometry of the airfoil (via a .txt file containing the coordinates) and the range of values of angle of attack.

Listing 6: Input values

```
# Load the airfoil geometry points
[n, points, centroids] = load_points(plotgeo)

# Sweep over all desired angles
minalpha = 0
maxalpha = 10
alphastep = 2.5
sweep(minalpha, maxalpha, alphastep, n, points, centroids, plotcl, plotcp)
```

Towards the end of the code, this section specifying the input parameters cna be found. It can be observed that loading the geometry here is performed by a function 'load_points'. The function can be found at the start of the code.

Listing 7: Specifying the geometry

```
def load_points(plot):
    # Load the points from airfoil.txt (must be a closed trailing edge with first/last point
        duplicated)
    points = np.loadtxt('airfoil.txt')

    # Flip the array upside down if needed (depending on which way the nodes are ordered)
    points = np.flipud(points)

    # Get the number of panels
    n = np.shape(points)[0] − 1

    # Get the centroids of the panels
    centroids = np.zeros((n, 2))
    for i in range(0, n):
        centroids[i, 0] = 0.5 * (points[i, 0] + points[i + 1, 0])
        centroids[i, 1] = 0.5 * (points[i, 1] + points[i + 1, 1])

    # Plot the airfoil shape, endpoints, and centroids
    plot.plot(points[:, 0], points[:, 1])
    plot.set_xlabel('x/c')
    plot.set_ylabel('y/c')
    plot.set_title('Airfoil')
    plot.set_ylim([−0.5, 0.5])
    figcp.savefig('plots/airfoil.pdf')

    return n, points, centroids
```

As shown, this function is defined to basically take the input text file and calculate the number of panels, centroids (mid points of the panels).

The function 'sweep' which can be seen in the section dealing with input parameters is defined basically to calculate the values of C_l for the different values of α .

Listing 8: Calculating C_l for the range of angle of attack values

```
def sweep(minalpha, maxalpha, alphastep, n, points, centroids, plotcl, plotcp):
    # Initialize arrays for plot lift curve
    alpha_plot = []
    cl_plot = []

    # Sweep over a range of angles of attack
    alpha = minalpha
    while alpha <= maxalpha:
        # Solve for all panels at angle of attack alpha
        cl = solve_panel(points, centroids, n, math.pi * alpha / 180, plotcp)

        # Print the angle and lift coefficient
        if cl < 1.3:
            print('Angle:␣', alpha, '␣Cl:␣', cl)
        else:
            print('Angle:␣', alpha, '␣Cl:␣', cl, '␣WARNING:␣Stall␣Likely!')

        # Save the values
        alpha_plot.append(alpha)
        cl_plot.append(cl)

        alpha = alpha + alphastep

    # Put the point on the lift curve
    plotcl.plot(alpha_plot, cl_plot, marker=".", markersize=6, label=r'$C_L$')
    plotcl.legend()
    plotcl.set_xlabel(r'$\alpha$')
    plotcl.set_ylabel(r'$C_L$')
    plotcl.set_title('Lift␣Curve')
    np.savetxt('data/cl.txt', np.c_[alpha_plot, cl_plot], delimiter='␣')

    return
```

It can be observed that the primary calculation of C_l is carried out by calling the function 'solve_panel'. This function is crucial and at the heart of this code, and has all the calculations discussed in the theory.

Listing 9: Calculation of C_l

```
def solve_panel(points, centroids, n, alpha, plot):
    jmat = np.zeros((n, n))
    kmat = np.zeros((n, n))
    amat = np.zeros((n + 1, n + 1))
    vvec = np.zeros((n + 1, 1))
    svec = np.zeros((n + 1, 1))

    # Get the coefficients for all panels on each other
    for i in range(0, n):
        for j in range(0, n):
            if j != i:
                # Get the coefficients for panel i and panel j
                [a, d, e, f, g, sj, c1, c2] = coefficients(points, centroids, i, j)

                # Build the matrices
                jmat[i, j] = -c2 / 2 * f - c1 * g
                kmat[i, j] = -c2 + 1 / sj * ((a * c2 + d / 2) * f + (a * c1 + e * c2) * g)
                svec[j] = sj
            else:
                kmat[i, j] = -1
    # Build the amat matrix, angle, and velocity
    for i in range(0, n):
        for j in range(0, n):
            amat[i, j] = amat[i, j] + jmat[i, j] - kmat[i, j]
            amat[i, j + 1] = amat[i, j + 1] + kmat[i, j]

        angle = math.atan2((points[i + 1, 1] - points[i, 1]), (points[i + 1, 0] - points[i, 0]))
        vvec[i] = 2 * math.pi * math.sin(alpha - angle)

    # Add the Kutta condition constraint
    amat[n, 0] = 1.0
    amat[n, n] = 1.0

    # Solve for the circulation
    gam = np.linalg.solve(amat, vvec)

    # Get the pressure coefficient
    cp = 1 - (gam ** 2)

    # Get the lift coefficient by integrating the circulation
    cl = 0
    for i in range(0, n):
        cl = cl + 0.5 * (gam[i] + gam[i + 1]) * svec[i]
    cl = cl[0] / 0.5

    # Plot the pressure coefficient
    plot.plot(points[2:-2, 0], -cp[2:-2, 0], label=r'$\alpha = $' + str(alpha / math.pi * 180))
    plot.legend()
    plot.set_xlabel('x/c')
    plot.set_ylabel(r'$-C_P$')
    plot.set_title('Pressure Coefficient')
    np.savetxt('data/cp_' + str(alpha / math.pi * 180) + '.txt', np.c_[points[2:-2, 0], cp[2:-2, 0]],
         delimiter=' ')

    return cl
```

The various coefficients are calculated here calling yet another function ‘coefficients’. This is defined before the ‘solve_panel’ function, and is basically the application of various expressions described after equation 2.3.

Listing 10: Coefficients

```
def coefficients(points, centroids, i, j):
    # Get the angles of both panels
    phii = math.atan2((points[i + 1, 1] - points[i, 1]), (points[i + 1, 0] - points[i, 0]))
    phij = math.atan2((points[j + 1, 1] - points[j, 1]), (points[j + 1, 0] - points[j, 0]))
```

```
# Get all of the terms
a = −(centroids[i, 0] − points[j, 0]) ∗ math.cos(phij) − (centroids[i, 1] − points[j, 1]) ∗ math.
    sin(phij)
b = (centroids[i, 0] − points[j, 0]) ∗∗ 2 + (centroids[i, 1] − points[j, 1]) ∗∗ 2
c1 = math.sin(phii − phij)
c2 = math.cos(phii − phij)
d = (centroids[i, 0] − points[j, 0]) ∗ math.cos(phii) + (centroids[i, 1] − points[j, 1]) ∗ math.
    sin(phii)
sj = math.sqrt((points[j + 1, 0] − points[j, 0]) ∗∗ 2 + (points[j + 1, 1] − points[j, 1]) ∗∗ 2)
e = math.sqrt(b − a ∗∗ 2)
f = math.log((sj ∗∗ 2 + 2 ∗ a ∗ sj + b) / b)
g = math.atan((sj + a) / e) − math.atan(a / e)

return a, d, e, f, g, sj, c1, c2
```

2.3 Test case

The results from the code agree well with the experimental data of NACA 2412 as shown in the plots below.

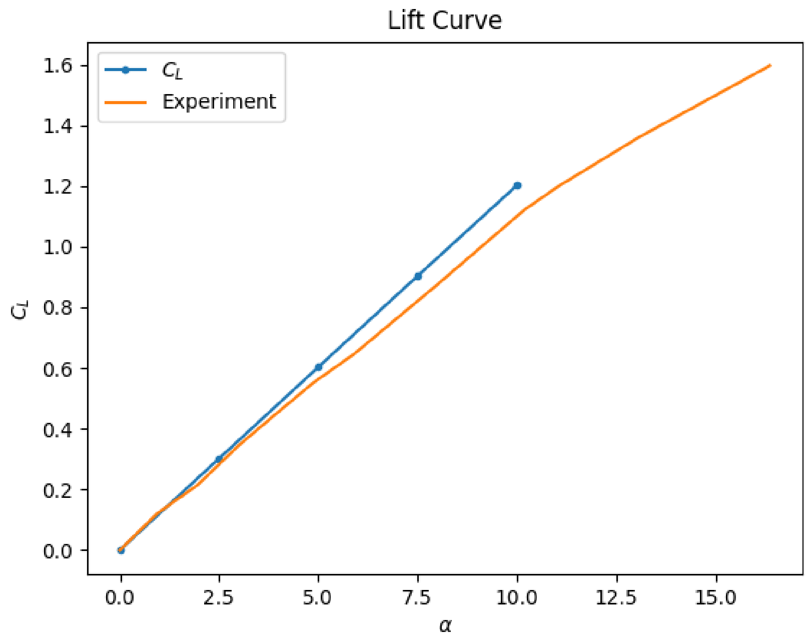

Figure 2.2: NACA2412 - Lift curve

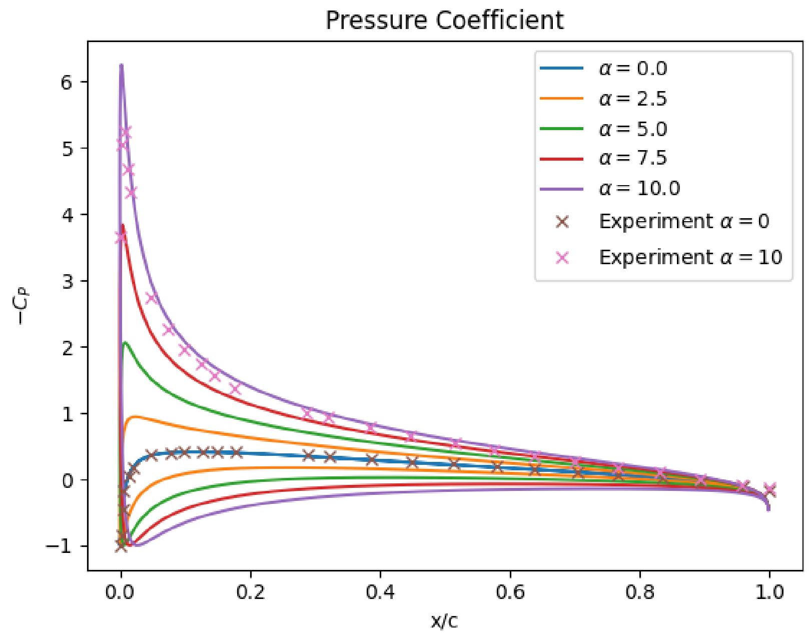

Figure 2.3: NACA2412 - Pressure coefficient

3 Finite Wing - Prandtl's Classical Lifting-line Theory

3.1 Summary of the theory [1]

The previous theories dealt with airfoils, generally considered to have an 'infinite' span. In practical applications however, all wings have a finite span. The finite wing theory analyses the characteristics differing from airfoil characteristics, like downwash, effective angle of attack, and induced drag. The first successful theory to predict the aerodynamic properties of a wing is Prandtl's Lifting-line theory. It is explained briefly in this section with emphasis on obtaining the final expressions of the coefficients.

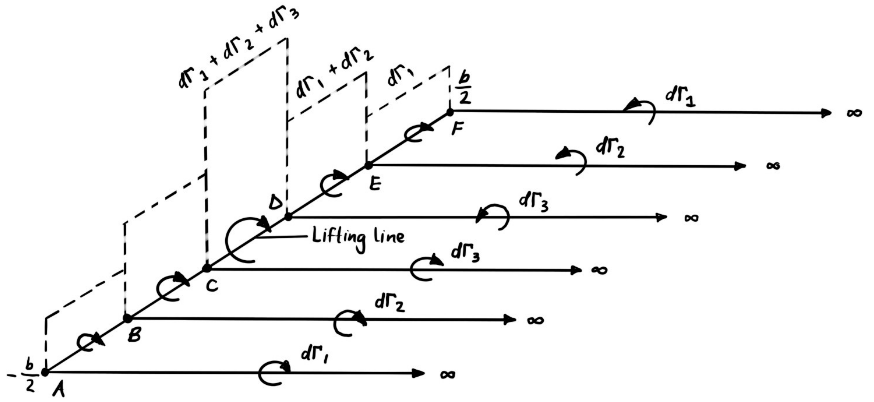

Figure 3.1: Finite number of horseshoe vortices superimposed on lifting line

The wing is represented with a large number of horseshoe vortices, superimposed in such a way that all their bound vortices coincide along a single line, termed the *lifting line*. The figure 3.1 illustrates the concept showing just three vortices for clarity. It can be noted that there are trailing vortices distributed along the span and the strength of each trailing vortex is equal to the *change in circulation* along the lifting line.

Now, assuming an infinite number of horseshoe vortices superimposed along the lifting line, each with infinitesimal strength $d\Gamma$. Here, the strength along the lifing line becomes a smooth distribution $\Gamma(y)$ with Γ_0 being the circulation at the origin. Also, trailing vortices here is a continuous vortex sheet trailing downstream. The total strength integrated across the span of the wing will be zero, because it consists of pairs of opposing vortices of equal strength.

Consider a small segment of the lifting line dy located at a distance y from the origin where the circulation is given by $\Gamma(y)$ and change in circulation over the segment is $d\Gamma = (d\Gamma/dy)dy$. Also, the strength of the trailing vortex must equal the change in the circulation $d\Gamma$ (as discussed in the case above). As shown in figure 3.2, consider a particular trailing vortex that intersects the lifting line at coordinate y, any segment dx of this vortex will induce a

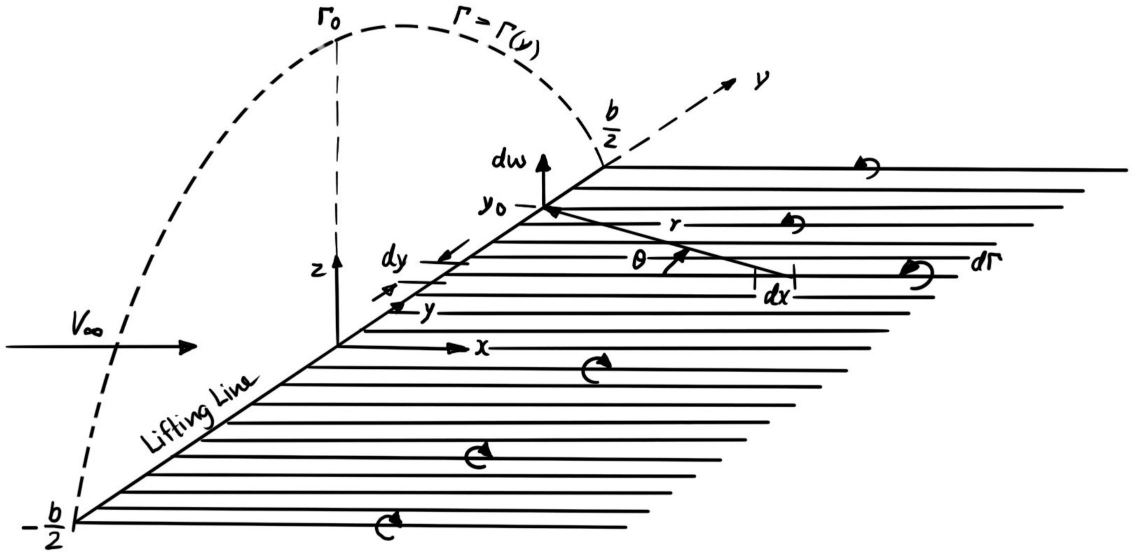

Figure 3.2: Infinite number of horseshoe vortices superimposed

velocity $d\omega$ at an arbitrary location on the lifting line y_0 such that

$$d\omega = -\frac{(d\Gamma/dy)dy}{4\pi(y_0 - y)}. \tag{3.1}$$

Integrating this equation from $-b/2$ to $b/2$, we can get the total velocity ω induced at y_0 by the entire traiing vortex sheet as

$$\omega(y_0) = -\frac{1}{4\pi}\int_{-b/2}^{b/2} \frac{(d\Gamma/dy)dy}{y_0 - y}. \tag{3.2}$$

The induced angle of attack (due to the downwash) at y_0 is given by

$$\alpha_i(y_0) = tan^{-1}\left(\frac{-\omega(y_0)}{V_\infty}\right). \tag{3.3}$$

Usually, ω is much smaller than V_∞ which means α_i is small enough to approximate

$$\alpha_i(y_0) = \frac{-\omega(y_0)}{V_\infty}. \tag{3.4}$$

Substituting the value of $\omega(y_0)$, we get an expression for the induced angle of attack in terms of the circulation distribution along the wing as

$$\boxed{\alpha_i(y_0) = \frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2} \frac{(d\Gamma/dy)dy}{y_0 - y}.} \tag{3.5}$$

We know the relation between sectional coefficient of lift and effective angle of attack is given by

$$c_l = a_0[\alpha_{\text{eff}}(y_0) - \alpha_{L=0}] = 2\pi[\alpha_{\text{eff}}(y_0) - \alpha_{L=0}]. \tag{3.6}$$

Two different expressions for Lift (per unit span) are considered, one by definition of lift coefficient and second from Kutta-Joukowski theorem

$$L^{'} = 1/2\rho_\infty V_\infty^2 c(y_0)c_l = \rho_\infty V_\infty \Gamma(y_0). \tag{3.7}$$

This gives us an expression for c_l

$$c_l = \frac{2\Gamma(y_0)}{V_\infty c(y_0)}. \tag{3.8}$$

Solving for α_{eff}, we get

$$\alpha_{\text{eff}} = \frac{\Gamma(y_0)}{\pi V_\infty c(y_0)} + \alpha_{L=0}. \tag{3.9}$$

We also know, from definition

$$\alpha_{\text{eff}} = \alpha - \alpha_i. \tag{3.10}$$

Therefore, solving for geometric angle of attack at y_0, we obtain

$$\boxed{\alpha(y_0) = \frac{\Gamma(y_0)}{\pi V_\infty c(y_0)} + \alpha_{L=0}(y_0) + \frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2} \frac{(d\Gamma/dy)dy}{y_0 - y}.} \tag{3.11}$$

This is the *fundamental equation of Prandtl's lifting-line theory.* It should be noted that the only unknown in this equation is Γ; all the other terms are known for a given wing design at a given geometric angle of attack in a flow with a given freestream velocity. Thus solution of Equation 3.11 gives us the circulation function $\Gamma = \Gamma(y_0)$, where y_0 ranges from $-b/2$ to $b/2$. This, in turn, gives us the Lift, Coefficient of lift and Induced drag.

We proceed by assuming a general circulation distribution in the form of a Fourier sine series considering the following transformation of coordinates

$$y = -\frac{b}{2}cos\theta. \tag{3.12}$$

So, the coordinate along the wing span is now θ, $0 \leq \theta \leq \pi$; the circulation distribution is assumed as

$$\Gamma(\theta) = 2bV_\infty \sum_1^N A_n sinn\theta. \tag{3.13}$$

where N is the number of terms in the Fourier series, higher number of terms gives higher accuracy. Differentiating this equation, we obtain

$$\frac{d\Gamma}{dy} = \frac{d\Gamma}{d\theta}\frac{d\theta}{dy} = 2bV_\infty \sum_1^N nA_n cosn\theta \frac{d\theta}{dy}. \tag{3.14}$$

Substituting these expressions of Γ and $d\Gamma/dy$ in the fundamental equation, we obtain

$$\boxed{\alpha(\theta_0) = \frac{2b}{\pi c(\theta_0)} \sum_1^N A_n sinn\theta_0 + \alpha_{L=0}(\theta_0) + \sum_1^N nA_n \frac{sinn\theta_0}{sin\theta_0}.} \tag{3.15}$$

It can be observed that, for a given span location (θ_0 is specified), the only unknowns in equation 3.15 are the N number of coefficients A_ns. So, if we take N different locations along the span and evaluate Equation 3.15 for each of these N locations, we end up with a system of N independent algebraic equations with N unknowns (the coefficients). Solving this system, we can obtain all the values for the A_ns. Hence, the general circulation distribution $\Gamma(\theta)$ can be obtained. The lift coefficient follows.

Lift per span distribution is given by Kutta-Joukowski theorem (in terms of span length coordinate y) as

$$L^{'}(y_0) = \rho_\infty V_\infty \Gamma(y). \tag{3.16}$$

The total lift can be obtained by integrating this equation over the span, hence

$$L = \rho_\infty V_\infty \int_{-b/2}^{b/2} \Gamma(y)dy. \tag{3.17}$$

Then the lift coefficient, by definition, will be given by

$$C_L = \frac{L}{q_\infty S} = \frac{2}{V_\infty} \int_{-b/2}^{b/2} \Gamma(y)dy. \tag{3.18}$$

Substituting the expression we assumed for the general circulation distribution, equation 3.13, we get

$$C_L = \frac{2b^2}{S} \sum_1^N A_n \int_0^\pi sinn\theta sin\theta d\theta. \tag{3.19}$$

Here, the integral can be evaluated as

$$\int_0^\pi sinn\theta sin\theta d\theta = \begin{cases} \pi/2 & for\ n = 1 \\ 0 & for\ n \neq 1 \end{cases}$$

.

Hence, the lift coefficient simplifies to

$$C_L = A_1 \pi \frac{b^2}{S} = A_1 \pi AR. \tag{3.20}$$

Next, moving on to induced drag coefficient. It follows from definition of Induced drag (assuming induced angle is small)

$$D_i^{'} = L^{'} \alpha_i. \tag{3.21}$$

Total induced drag is obtained by integrating this expression over the span, hence

$$D_i = \int_{-b/2}^{b/2} L^{'}(y)\alpha_i(y)dy. \tag{3.22}$$

The induced drag coefficient is given by

$$C_{D,i} = \frac{D_i}{q_\infty S} = \frac{2}{V_\infty S}\int_{-b/2}^{b/2} \Gamma(y)\alpha_i(y)dy. \tag{3.23}$$

Now, substituting our assumed expression for the circulation, i.e. equation 3.13, we get

$$C_{D,i} = \frac{2b^2}{S}\int_0^\pi \left(\sum_1^N A_n sinn\theta\right)\alpha_i(\theta)sin\theta d\theta. \tag{3.24}$$

The expression for induced angle of attack can be obtained from equation 3.5

$$\alpha_i(y_0) = \frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2}\frac{(d\Gamma/dy)dy}{y_0 - y} = \frac{1}{\pi}\sum_1^N nA_n\int_0^\pi \frac{cosn\theta}{cos\theta - cos\theta_0}d\theta. \tag{3.25}$$

Solving the integral using its standard form, we get

$$\alpha_i(\theta_0) = \sum_1^N nA_n\frac{sinn\theta_0}{sinn\theta_0}, \tag{3.26}$$

which can also be written as

$$\alpha_i(\theta) = \sum_1^N nA_n\frac{sinn\theta}{sinn\theta}. \tag{3.27}$$

Hence, equation 3.24 becomes

$$C_{D,i} = \frac{2b^2}{S}\int_0^\pi \left(\sum_1^N A_n sinn\theta\right)\left(\sum_1^N nA_n sinn\theta\right)d\theta. \tag{3.28}$$

This equation involves the *product* of two summations. From the standard integral

$$\int_0^\pi sinm\theta sink\theta = \begin{cases} 0 & for\ m \neq k \\ \pi/2 & for\ m = k \end{cases}. \tag{3.29}$$

This means all the mixed product terms having unequal subscripts (like A_1A_2,A_2A_4 etc) in equation 3.28 reduce to zero, simplifying it to

$$\begin{aligned} C_{D,i} &= \frac{2b^2}{S}\left(\sum_1^N nA_n^2\right)\frac{\pi}{2} = \pi AR\sum_1^N nA_n^2 \\ &= \pi AR\left(A_!^2 + \sum_2^N nA_n^2\right) \end{aligned}$$

,

$$= \pi AR A_1^2 \left[1 + \sum_2^N n \left(\frac{A_n}{A_1} \right)^2 \right]. \tag{3.30}$$

Substituting the value of C_L obtained in equation 3.20, we get

$$\boxed{C_{D,i} = \frac{C_L^2}{\pi AR}(1+\delta),} \tag{3.31}$$

where $\delta = \sum_2^N n(A_n/A_1)^2$. Now, defining a *span efficiency factor* as $e = (1+\delta)^{-1}$, equation 3.31 becomes

$$\boxed{C_{D,i} = \frac{C_L^2}{\pi e AR}.} \tag{3.32}$$

3.2 Explanation of the code

The input parameters here are Wingspan, Chord length in terms of spanwise location, Geometric angle of attack, Zero-lift angle of attack, Number of terms in the expansion(affects accuracy of the results).

Listing 11: Input values

```
# Define the input parameters
b = 2 # Wingspan (m)
c = 1 − abs(y) / 4 # Chord length as a function of spanwise location (m)
alpha = 1 # Geometric angle of attack of the wing (deg)
alpha_0 = −1 # Zero−lift angle of attack as a function of spanwise location (deg)
N = 20 # Number of terms in the expansion

# Call the main function
finite_wing(b, c, alpha_0, alpha, N, true)

# Call the drag polar plotter
drag_polar(b, c, alpha_0, N)
```

Towards the end of the code, you can find the input parameters defined. It can be seen that two functions have been called here. The 'drag_polar' function simply does the function of calculating C_L and C_{D_i} for the range of angle of attack values and plotting them. The crucial function which performs these calculations following the theory discussed above is the 'finite_wing' function. A few sections are discussed below.

Listing 12: A_n Coefficients

```
    # Generate the entries in the linear system
    for i in range(0, N):
        # Get the theta_0 position of this point
        theta_0 = math.pi ∗ (i + 0.5) / N
        beta[i] = (alpha - alpha_0).subs(theta, theta_0)

        # Generate the coefficient matrix
        for j in range(0, N):
            # First term
            A[i, j] = 2 * b / (math.pi * c.subs(theta, theta_0)) * sin((j + 1) * theta_0)

            # Second term
            A[i, j] += (j + 1) * sin((j + 1) * theta_0) / sin(theta_0)

    # Solve the linear system for the An coefficients
    An = np.linalg.solve(A, beta)
```

The equations written here to generate the coefficient matrix is obtained from Equation 3.15, solving a system of N number of independent algebraic equations to get the N number of coefficients. These values are stored in the 'An' array.

Listing 13: Calculating Lift and Drag Coefficients

```
    # Get the lift coefficient
    Cl = An[0] * math.pi * b ** 2 / S

    # Get the delta coefficient for the drag coefficient
    delta = 0
    for i in range(1, N):
        delta += (i + 1) * (An[i] / An[0]) ** 2

    # Compute the induced drag coefficient
    Cdi = Cl ** 2 / (math.pi * AR) * (1 + delta)

    # Get the span efficiency factor
    eps = 1 / (1 + delta)
```

The computation of lift and drag coefficients here are straighforward applications of the expressions obtained in the theory, equations 3.20 and 3.31.

Listing 14: Calculating Circulation

```
    # Compute the circulation distribution
    gamma = 0
    for i in range(0, N):
        gamma += 2 * b * An[i] * sin((i + 1) * theta)

    gamma = gamma.subs(theta, acos(y * - 2 / b))
```

The computation of circulation is a direct application of the general expression we assumed, equation 3.13.

3.3 Validation

For validating the Finite wing code, two different plots were used from the relevant section of reference [1]. First, circulation distribution generated using a *Numerical Non-linear method* for rectangular wings of aspect ratios 4 and 8 (NACA 1412) are compared with the values obtained from the code. Excellent agreement can be observed, as shown in figures 3.3 and 3.4. Further, figure 3.5 shows comparison of values of C_L for a rectangular wing of aspect ratio 5.536 (NACA 0015).

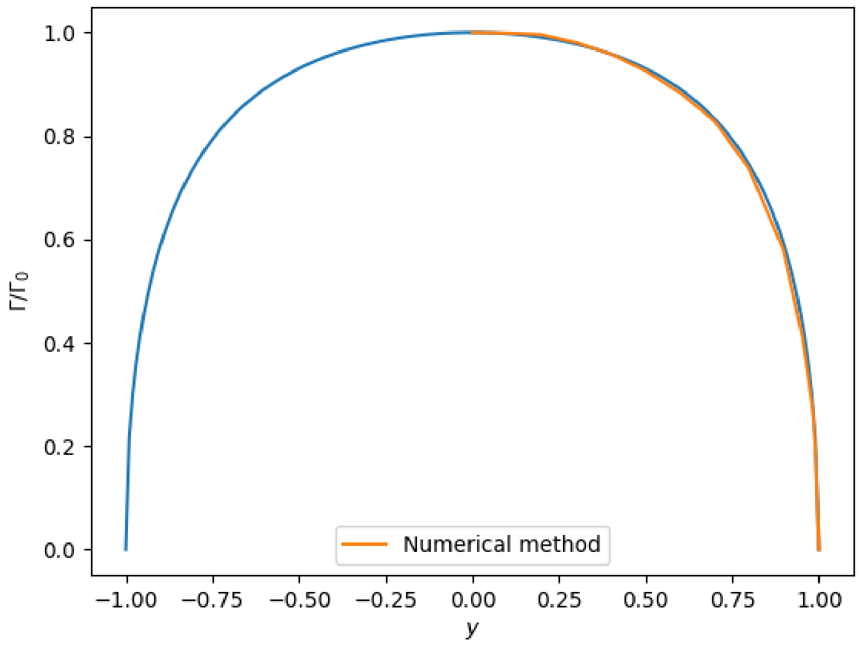

Figure 3.3: Comparison of Γ/Γ_0 for a rectangular wing of AR 4 (NACA1412)

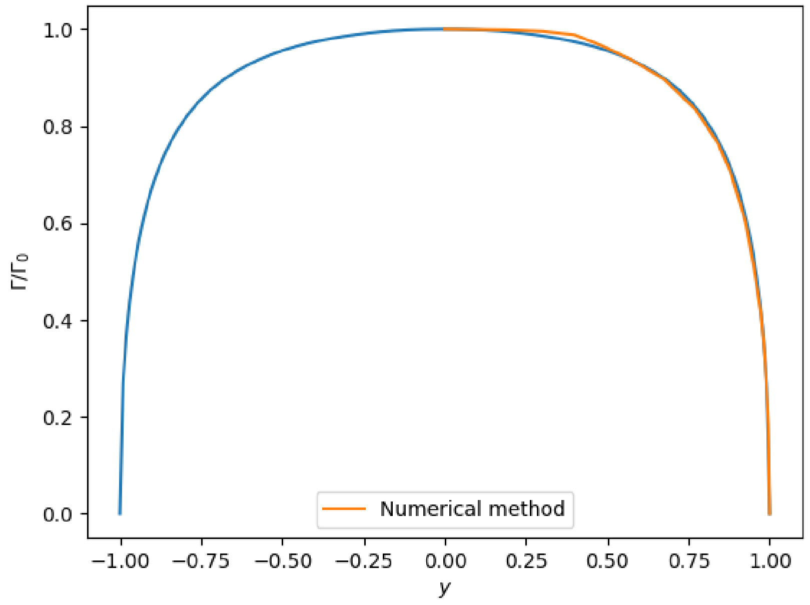

Figure 3.4: Comparison of Γ/Γ_0 for a rectangular wing of AR 8 (NACA1412)

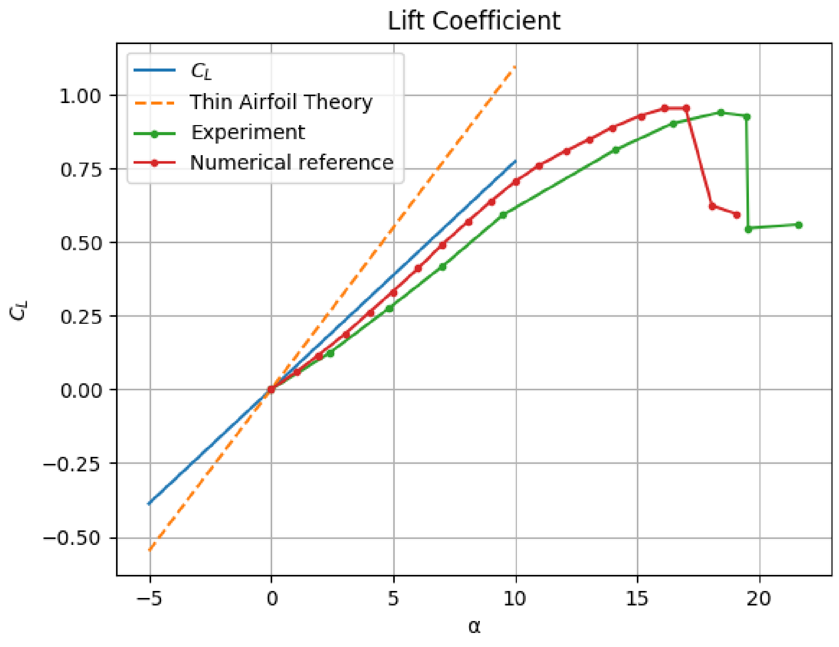

Figure 3.5: Comparison of C_L for a rectangular wing of AR 5.536 (NACA0015)

4 Performance

4.1 Summary of the theory involved [3]

4.1.1 General performance parameters

Thrust required First we are going to discuss the thrust required to maintain *steady*, *level* flight. *Steady* flight means the aircraft is at a constant speed i.e. no acceleration and *level* flight means both climb angle and bank angle are zero i.e. there is no change in altitude. The thrust required is a function of airspeed, altitude, size, shape, and weight of the aircraft. The forces acting on an aircraft in steady, level flight is shown in figure 4.1.

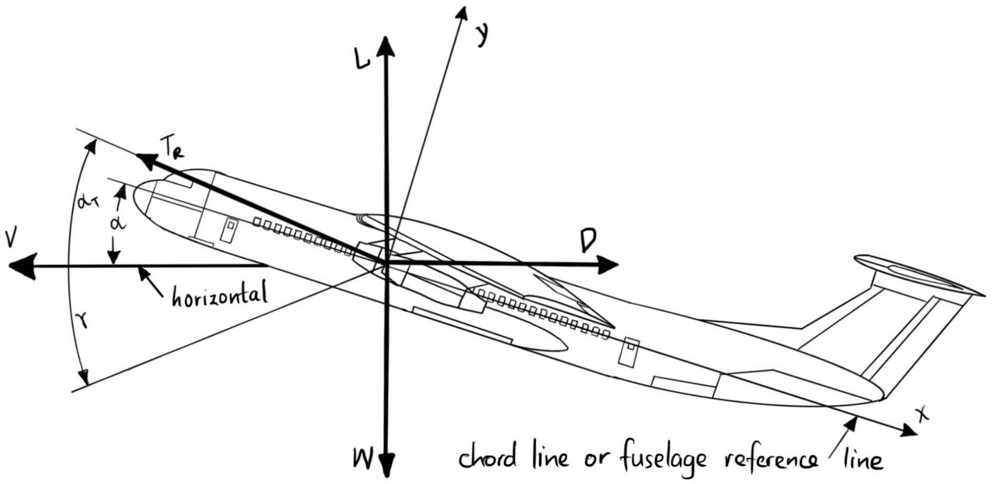

Figure 4.1: Forces acting on an aircraft in level flight

Since there is no acceleration, the drag force equals the thrust component in the direction of flight

$$D = T_R cos\alpha_T. \tag{4.1}$$

The lift must equal the weight minus the thrust component perpendicular to the direction of flight

$$L = W - T_R sin\alpha_T. \tag{4.2}$$

From these two equations, it is apparent that

$$\frac{L}{D} = \frac{W - T_R sin\alpha_T}{T_R cos\alpha_T}. \tag{4.3}$$

or, solving for the expression for Thrust required

$$T_R = \frac{W}{(L/D)cos\alpha_T + sin\alpha_T}. \tag{4.4}$$

Now, the total drag force, which is the sum of parasitic drag and induced drag, can be written as

$$D = \frac{1}{2}\rho V^2 S_w C_D = \frac{1}{2}\rho V^2 S_w \left(C_{D_p} + \frac{C_L^2}{\pi e_s R_A} \right), \tag{4.5}$$

where V is the airspeed, S_w is the area of the main wing, e_s is the span efficiency factor, R_A is the aspect ratio, and C_{D_p} represents the parasitic drag coefficient. C_{D_p} can be written as a parabolic function of C_L

$$C_{D_p} = C_{D_0} + C_{D_0,L} C_L + C_{D_0,L^2} C_L^2. \tag{4.6}$$

Substituting equation 4.6 in equation 4.5 and combining the quadratic term from the expression for the parasitic drag with the induced drag term, the total drag can be expressed as

$$D = \frac{1}{2}\rho V^2 S_w \left(C_{D_0} + C_{D_0,L} C_L + \frac{C_L^2}{\pi e R_A} \right). \tag{4.7}$$

Here e is the Oswald efficiency factor. Therefore

$$\boxed{\frac{L}{D} = \frac{C_L}{C_D} = \frac{C_L}{C_{D_0} + C_{D_0,L} C_L + \frac{C_L^2}{\pi e R_A}}.} \tag{4.8}$$

Substituting equation 4.2 in the definition of C_L, we have

$$C_L = \frac{L}{\frac{1}{2}\rho V^2 S_w} = \frac{W - T_R sin\alpha_T}{\frac{1}{2}\rho V^2 S_w}. \tag{4.9}$$

Using the expression for Thrust required from 4.4

$$\boxed{C_L = \frac{W}{\frac{1}{2}\rho V^2 S_w} \left[1 - \frac{sin\alpha_T}{(L/D)cos\alpha_T + sin\alpha_T} \right] = \frac{W}{\frac{1}{2}\rho V^2 S_w} \left[\frac{1}{1 + (D/L)tan\alpha_T} \right].} \tag{4.10}$$

Using the equations 4.8 and 4.10, both C_L and L/D as a function of airspeed can be determined numerically. A typical example of a plot of lift-drag ratio versus airspeed is shown in figure 4.2.

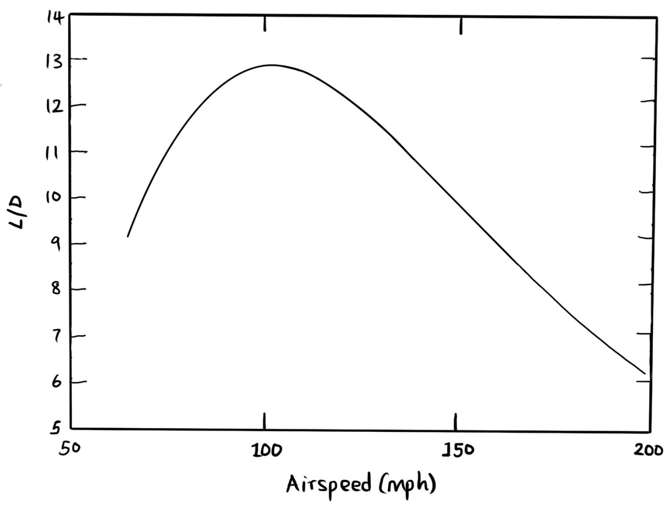

Figure 4.2: Typical variation of L/D ratio vs Airspeed at sea level

When the thrust required is plotted versus airspeed, for a typical general aviation aircraft, it looks as shown in figure 4.3.

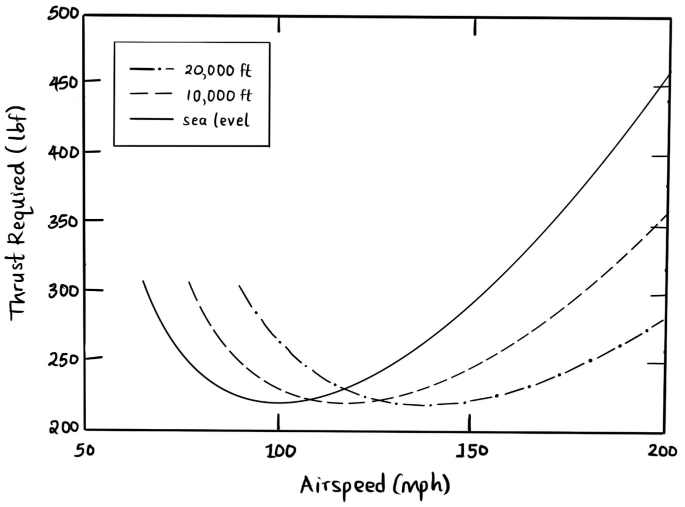

Figure 4.3: Typical variation of Thrust Required vs Airspeed

From equation 4.4, we can obtain the optimum thrust angle, i.e. the value of α_T which minimizes the thrust required. This is done by differentiating the equation with respect to α_T and equating the result to zero. Thus, we get

$$\boxed{\alpha_T = tan^{-1}(D/L).} \tag{4.11}$$

For conventional airplanes, in most cases, both α_T and (D/L) are small values. Thus, we can safely use the following approximations for preliminary performance calculations

$$cos\alpha_T \approx 1$$
$$sin\alpha_T \approx \alpha_T$$
$$(D/L)sin\alpha_T \approx \alpha_T^2 \approx 0$$

. This is termed *small-thrust-angle approximation.* Applying this, expressions for Thrust required and Lift become simplified and equation 4.4 reduces to

$$T_R = D, \tag{4.12}$$

$$L = W, \tag{4.13}$$

$$T_R = \frac{W}{L/D}. \tag{4.14}$$

Power required The power required can be obtained by multiplying the thrust required by the airspeed and cosine of the thrust angle.

$$\boxed{P_R = T_R V cos\alpha_T = DV,} \tag{4.15}$$

where P_R is the power required for steady level flight. Figure 4.4 shows a typical plot of power required versus airspeed for a general aviation aircraft.

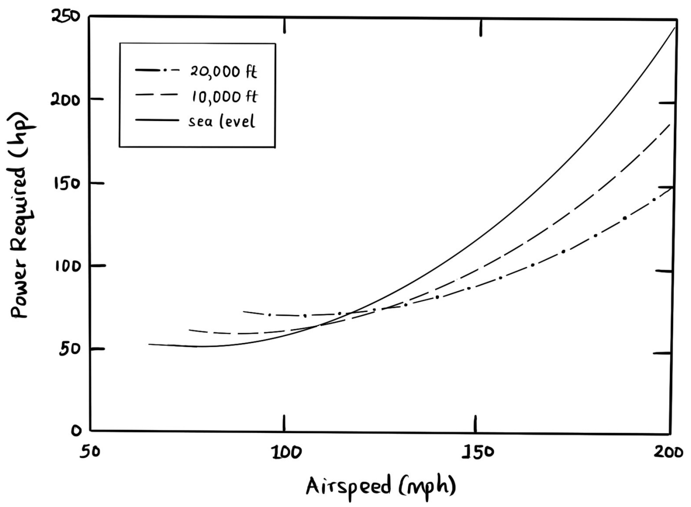

Figure 4.4: Typical variation of Power Required vs Airspeed

Rate of Climb The rate of climb is a function of the aerodynamic design, weight of the aircraft and the power available from the engines. To find out the *steady* rate of climb, consider an airplane in unaccelerated climbing flight as shown in figure 4.5. T_A denotes the thrust available, and γ is the angle between the flight path relative to the ambient air and the horizontal.

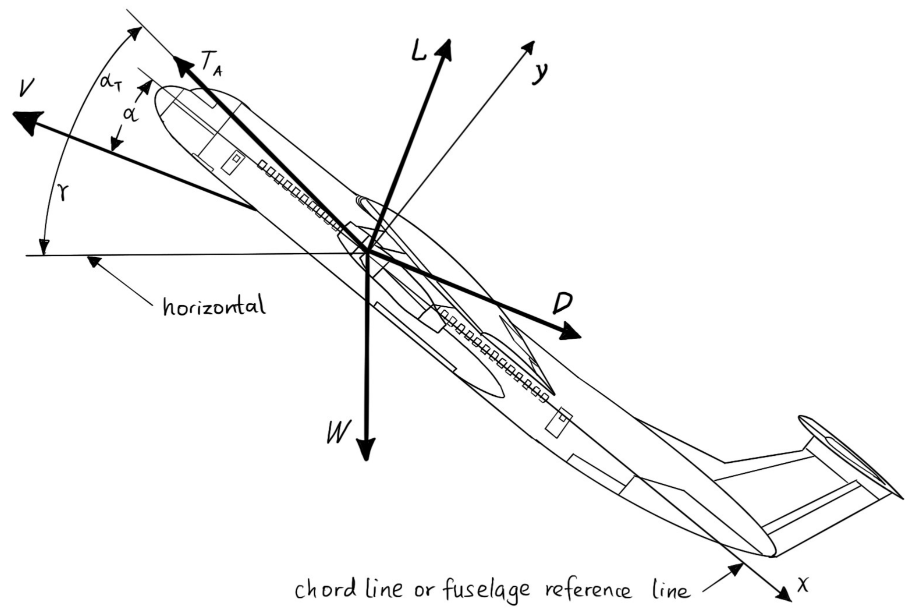

Figure 4.5: Forces acting on an aircraft in climb

The drag must be equal to the component of thrust parallel to the flight path minus the component of weight aligned with the flight path

$$D = T_A cos\alpha_T - W sin\gamma. \tag{4.16}$$

The lift can be computed as the weight component perpendicular to the flight path minus the thrust component normal to the flight path

$$L = W cos\gamma - T_A sin\alpha_T. \tag{4.17}$$

Applying these expressions to equation 4.7 and definition of lift coefficient, we can obtain the *general formulation for steady climbing flight*

$$\frac{1}{2}\rho V^2 S_w \left(C_{D_0} + C_{D_0,L} C_L + \frac{C_L^2}{\pi e R_A} \right) = T_A cos\alpha_T - W sin\gamma. \tag{4.18}$$

$$\frac{1}{2}\rho V^2 S_w C_L = W cos\gamma - T_A sin\alpha_T. \tag{4.19}$$

The rate of climb, V_c, is basically the vertical component of the airplane's velocity, i.e. airspeed multiplied by the sine of flight path angle

$$V_c = V sin\gamma.$$

From equation 4.16, we can obtain

$$sin\gamma = \frac{T_A cos\alpha_T - D}{W}. \tag{4.20}$$

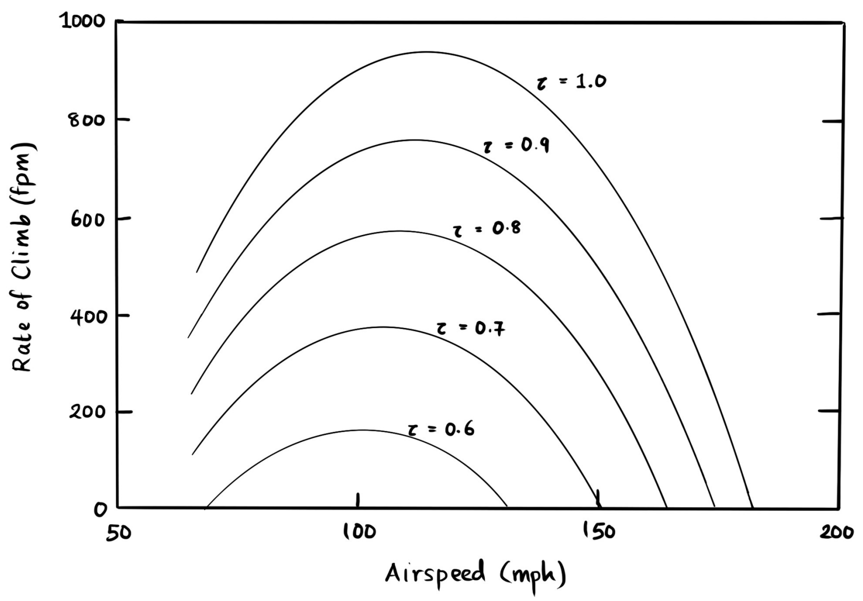

Figure 4.6: Typical variation of Rate of Climb vs Airspeed at sea level

Thus, rate of climb can be written as

$$V_c = \frac{VT_A cos\alpha_T - VD}{W}. \tag{4.21}$$

Now, since the climb angle is mostly small, we can use the value of drag for level flight here as an approximate estimate. Substituting the expression we discussed for level flight drag, the rate of climb can be written as

$$V_c = \frac{VT_A cos\alpha_T - VT_R cos\alpha_T}{W}. \tag{4.22}$$

Further, the product of airspeed with thrust available and cosine of the thrust angle, is the dot product of the thrust vector with the velocity vector which is the definition of Power available, P_A. Also, the airspeed multiplied by thrust required for level flight and cosine of the thrust angle is the definition of power required for level flight, P_R. Thus, the rate of climb can be further written as

$$\boxed{V_c = \frac{P_A - P_R}{W}.} \tag{4.23}$$

Figure 4.6 shows a typical plot of Rate of Climb versus Airspeed for a general aviation aircraft at sea level for different throttle settings, τ defined as a fraction of the full-throttle power available.

Fuel consumption Fuel consumption directly affects the endurance, defined as the total time an aircraft can fly on a given tank of fuel. To maximize endurance for a given aircraft and fuel capacity, we need to minimize fuel consumption per unit time. The power-specific fuel consumption for an engine, q_P, can be defined as the ratio between the weight of fuel consumption per unit time and the available power.

$$q_P = \frac{\dot{Q}}{P_A}. \tag{4.24}$$

For steady level flight, the power available will be equal to the power required, $P_A = P_R$. Hence, we can find fuel consumption per unit time, represented by q in the code, by simply multiplying q_P and P_R

$$\boxed{q = q_P.P_R.} \tag{4.25}$$

The fuel consumption varies with airspeed for steady level flight as shown in figure 4.7.

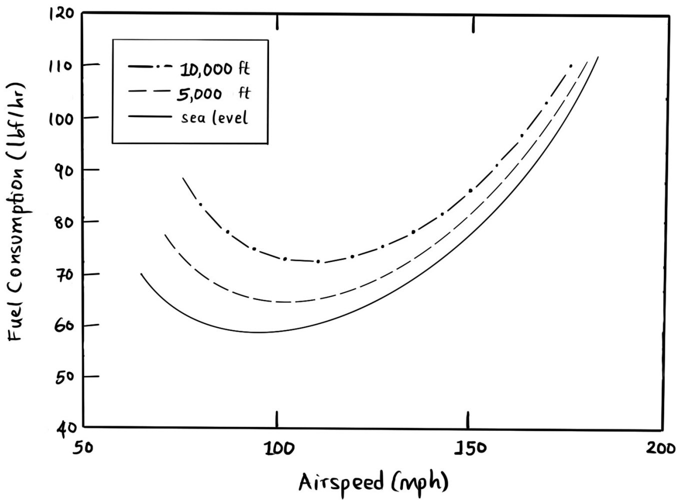

Figure 4.7: Typical variation of Fuel Consumption vs Airspeed for steady level flight

Specific range Range is defined as the total distance an aircraft can fly for a given tank of fuel. Range can also be maximized by minimizing fuel consumption. It is helpful to define Specific range as the fuel consumed per unit distance.

$$\boxed{r_s = \frac{q}{v}.} \tag{4.26}$$

A typical plot of variation of Specific range with airspeed is shown in the figure 4.8.

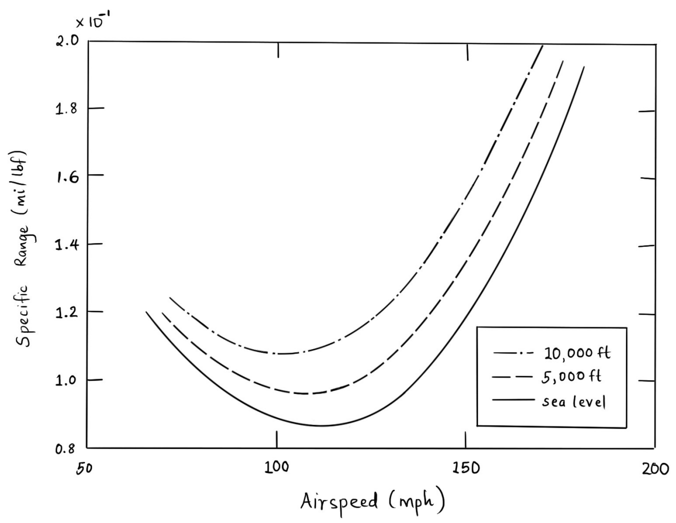

Figure 4.8: Typical variation of Specific Range vs Airspeed at sea level

Stall speed The minimum airspeed at which the airplane can fly is termed the stall speed. The level flight stall speed is an important parameter and can be obtained as follows. Consider the expression for C_L applying the small-thrust-angle approximation

$$C_L = \frac{W}{\frac{1}{2}\rho V^2 S_w}. \tag{4.27}$$

Solving for the expression for airspeed

$$V = \sqrt{\frac{2(W/S_w)}{\rho C_L}}. \tag{4.28}$$

It can be seen that the value of velocity will be minimum when C_L is at the maximum value. Therefore

$$\boxed{V_{min} = \sqrt{\frac{2}{C_{L_{max}}}}\sqrt{\frac{(W/S_w)}{\rho}}.} \tag{4.29}$$

4.1.2 Static stability

Static margin Similar to the aerodynamic center of an airfoil, the whole aircraft has a point about which the total pitching moment is independent of the angle of attack. This point is termed *stick-fixed neutral point* as the airplane will be neutrally stable in pitch if the center of gravity is located at this point. For an aircraft to be statically stable, it can be proven that the center of gravity must be forward of the stick-fixed neutral point. The distance of the CG from the neutral point, expressed as a fraction of the mean chord length, is termed *stick-fixed static margin*. The expression is given in the equation 4.30.

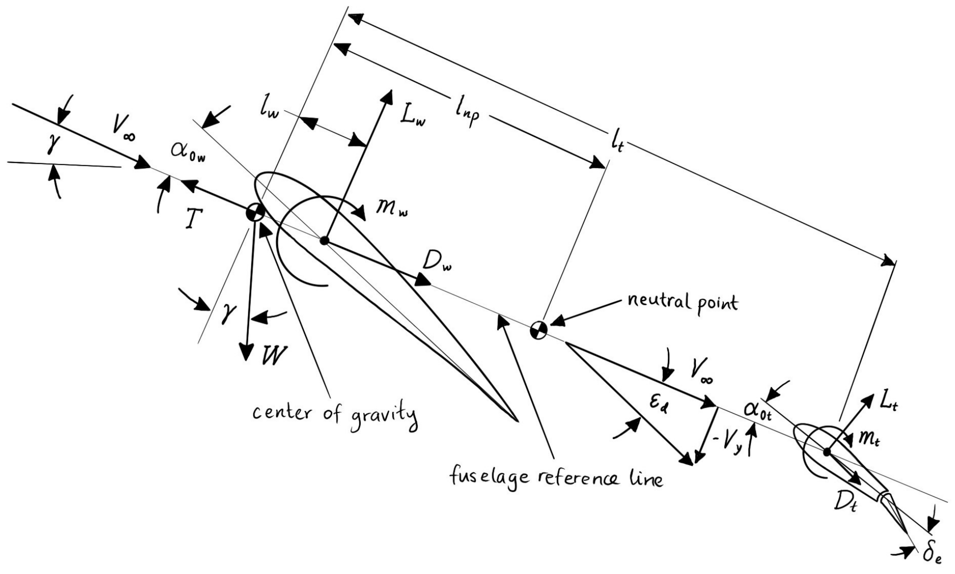

Figure 4.9: Understanding the location of the stick-fixed neutral point

$$\boxed{\frac{l_{np}}{\bar{c}_w} = \frac{l_w C_{L_w,\alpha} + \frac{S_t l_t}{S_w}\eta_t C_{L_t,\alpha}(1-\varepsilon_{d,\alpha})}{\bar{c}_w\left[C_{L_w,\alpha} + \frac{S_t}{S_w}\eta_t C_{L_t,\alpha}(1-\varepsilon_{d,\alpha})\right]}}. \tag{4.30}$$

The variables seen here denote

l_{np} = Distance between the stick-fixed neutral point and the CG
$\bar{c}_w$ = Mean chord length of the wing
l_w = Distance from CG to the aerodynamic center of the wing
l_t = Distance from CG to the aerodynamic center of the tail
$C_{L_w,\alpha}$ = Lift slope for the main wing
$C_{L_t,\alpha}$ = Lift slope for the isolated tail
S_w, S_t = Planform areas of the wing and tail respectively
η_t = Tail efficiency(can vary between 0.8 and 1.2 but usually assumed to be 1)
$\varepsilon_{d,\alpha}$ = Change in the downwash angle with respect to the angle of attack

.

Yaw Stability Derivative Static yaw stability, which is the ability to naturally produce a restorative yawing moment opposing any disturbance in yaw faced by the airplane, is provided by the vertical tail mainly. The general criterion for static yaw stability mathematically is given by

$$\frac{\partial C_n}{\partial \beta} \equiv C_{n,\beta} > 0. \tag{4.31}$$

$C_{n,\beta}$ represents the *yaw stability derivative* or *yaw stiffness.* The contribution of the vertical tail to the yaw stability derivative, which we are interested here in this section, is given by the equation 4.32. The various terms involved in that expression can be understood better through the figure 4.10 which shows the various yawing moments experienced by an aircraft in sideslip.

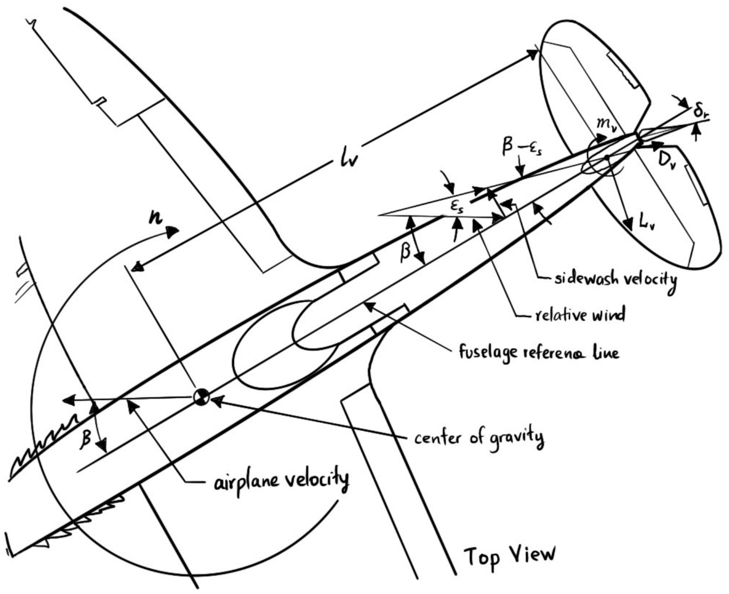

Figure 4.10: Top view of an airplane experiencing sideslip

$$\boxed{(\Delta C_{n,\beta})_v = \eta_v \frac{S_v l_v}{S_w b_w} C_{L_v,\alpha} (1 - \varepsilon_{s,\beta})_v.} \tag{4.32}$$

The variables seen in this equation denote

$$
\begin{aligned}
\eta_v &= \text{Dynamic pressure ratio for the vertical tail} \\
l_v &= \text{Distance from CG to the aerodynamic center of the vertical tail} \\
b_w &= \text{Wingspan} \\
C_{L_v,\alpha} &= \text{Lift slope for the vertical tail} \\
S_w, S_v &= \text{Planform areas of the wing and vertical tail respectively} \\
\varepsilon_{s,\beta} &= \text{Sidewash gradient} \\
\varepsilon_{s,\beta} &\equiv \frac{\partial \varepsilon_s}{\partial \beta} \\
\varepsilon_s &= \text{Sidewash angle} \\
\beta &= \text{Sideslip angle}
\end{aligned}
$$

.

Roll Stability Derivative If any disturbance in bank angle naturally produces a restoring rolling moment, the aircraft is said to have static roll stability. The general criterion for an aircraft to be statically stable in roll is given by

$$\frac{\partial C_l}{\partial \beta} \equiv C_{l,\beta}<0, \tag{4.33}$$

$C_{l,\beta}$ represents the *roll stability derivative.* The contribution of the wing dihedral to the roll stability derivative is given by the expression 4.34. The effect can be understood better from the figure 4.11 which shows what happens to the rolling moment when an airplane is experiencing sideslip.

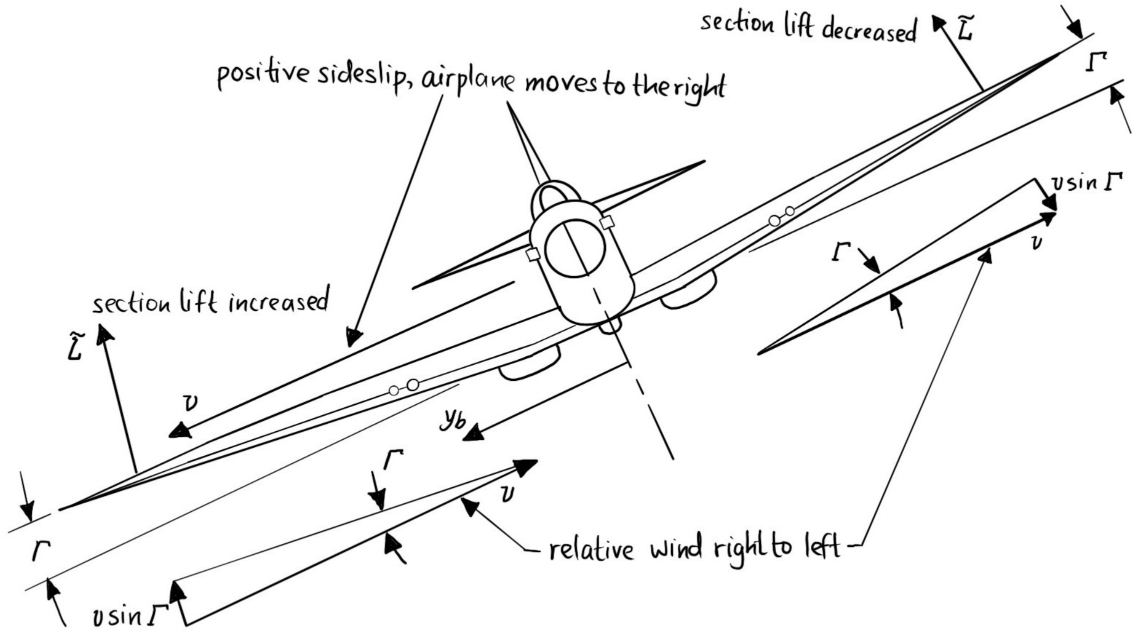

Figure 4.11: Front view of airplane showing effect of dihedral

$$\boxed{(\Delta C_{l,\beta})_{\Gamma_w} = -\frac{2\Gamma}{3\pi}\kappa_l C_{L_w,\alpha}.} \tag{4.34}$$

The contribution of vertical tail to the roll stability derivative can be obtained by

$$\boxed{(\Delta C_{l,\beta})_v = -\eta_v \frac{S_v h_v}{S_w b_w}(1-\varepsilon_{s,\beta})_v C_{L_v,\alpha}.} \tag{4.35}$$

A rough estimation of the contribution of the horizontal stabilizer to the rolling moment is calculated by the equation 4.36, where the positive sign is used when the stabilizer is mounted below the vertical tail and the negative sign is used when it is mounted above the vertical tail.

$$\boxed{(\Delta C_{l,\beta})_h = \pm 0.08\eta_v \frac{S_v b_h}{S_w b_w}(1-\varepsilon_{s,\beta})_v C_{L_v,\alpha}.} \tag{4.36}$$

The total roll derivative can be found by simply adding the individual contributions.

4.2 The python code

4.2.1 Performance

This code calculates the general performance parameters using the equations discussed in the theory above. There are a wide range of inputs needed including the environmental values, the aircraft parameters (Cessna 172 in this example), the engine parameters. The code snippet is included here, the variables defined are explained sufficiently in the comments.

Listing 15: Input values

```
# Environmental Parameters
rho_0 = 1.22 # Air density at sea level (kg/m^3)
rho = 1.22 # Air density at current altitude (kg/m^3)
g = 9.81 # Acceleration due to gravity (m/s^2)

# Aircraft Parameters (Cessna 172)
m = 1000 # Aircraft mass (kg)
s_w = 16.2 # Wing area (m^2)
c_d0 = 0.035 # Parasitic drag coefficient
c_d0l = 0 # Linear drag coefficient term (typically 0)
e = 0.7 # Oswald efficiency factor
ar = 7.32 # Wing aspect ratio
c_lmax_nf = 1.6 # Maximum lift coefficient with no flaps
c_lmin_nf = -1.3 # Minimum lift coefficient without flaps
c_lmax_f = 2.1 # Maximum lift coefficient with flaps
n_pll = 3.8 # Maximum design g-loading
n_nll = -1.5 # Minimum design g-loading

# Propulsion Parameters
p_a0 = 119000 # Engine power at sea level (W)
q_p = 8.5E-7 # Specific power fuel consumption ((N/s)/W)
alpha_t = 0.08172675000993017 # Offset between thrust and angle of attack (radians)
```

The function stall_speeds is defined to calculate and print the stall speeds both with and without flaps using the corresponding $C_{L,max}$ value, using equation 4.29.

Listing 16: Stall speeds

```
def stall_speeds(w, s_w, rho, c_lmax_nf, c_lmax_f):
    # Computes the aircraft stall speeds with and without flaps

    # Equation 3.8.3 of Phillips
    v_min_nf = math.sqrt(2 / c_lmax_nf) * math.sqrt(w / s_w / rho)
    v_min_f = math.sqrt(2 / c_lmax_f) * math.sqrt(w / s_w / rho)

    # Print the stall speeds
    print('\nAircraft Stall Speeds')
    print('Without Flaps: ', v_min_nf, ' (m/s)')
    print('With Flaps:    ', v_min_f, ' (m/s)\n')

    return v_min_nf, v_min_f
```

The lift_to_drag function is defined to find the Lift to drag ratio and optimize the thrust offset. We start out by assuming an initial value for L/D ratio as 5, as an input to find the initial C_L value using equation 4.10, and then iterate using equation 4.8. Then, the angle between the thrust vector and the direction of flight is calculated using equation 4.11.

Listing 17: L/D ratio

```
def lift_to_drag(alpha_t, w, s_w, rho, c_d0, c_d0l, e, ar):
    # Plots the lift to drag ratio as a function of airspeed
    figure, plot = plt.subplots()

    vmin = 20 # Minimum airspeed to plot
    vmax = 80 # Maximum airspeed to plot

    v = np.arange(vmin, vmax, 0.1)

    # Initially guess that lift to drag ratio is 5
    lod = 0 * v + 5

    # Iterate to find the lift to drag ratio using Equations 3.2.10, 3.2.8 of Phillips
    for i in range(0, 10):
        cl = w / (0.5 * rho * v ** 2 * s_w) * (1 / (1 + 1 / lod * math.tan(alpha_t)))
        lod = cl / (c_d0 + c_d0l * cl + cl ** 2 / (math.pi * e * ar))

    # Report a better thrust offset using Equation 3.2.16 of Phillips
    print('Optimization of Thrust Vector')
    print('A Better Thrust Offset: ', math.atan(1 / np.max(lod)), '\n')

    # Plot the results
    plot.plot(v, lod)
    plot.set_xlabel('Airspeed (m/s)')
    plot.set_ylabel('L/D Ratio')
    plot.set_title('Lift to Drag Ratio vs. Airspeed')
    figure.savefig('plots/lift_to_drag.pdf')
    np.savetxt('data/lift_to_drag.txt', np.c_[v, lod], delimiter=' ')

    return v, lod
```

The rest of the functions are straightforward applications of the concepts discussed above, specifically equations 4.14, 4.15, 4.23, 4.25, 4.26, to calculate the corresponding parameters and plot the results.

Listing 18: Performance calculations

```
def thrust_required(w, v, lod):
    # Computes the thrust required as a function of airspeed
    figure, plot = plt.subplots()

    # Use Equation 3.2.22 of Phillips
    t_r = (np.ones(np.size(lod)) * w) / lod
    # Plot the results
    plot.plot(v, t_r / 1000)
    plot.set_xlabel('Airspeed␣(m/s)')
    plot.set_ylabel('Thrust␣Required␣(kN)')
    plot.set_title('Thrust␣Required␣vs.␣Airspeed')
    figure.savefig('plots/thrust_required.pdf')
    np.savetxt('data/thrust_required.txt', np.c_[v, t_r / 1000], delimiter='␣')

    return t_r


def power_required(t_r, v, alpha_t):
    # Computes the power required as a function of airspeed
    figure, plot = plt.subplots()

    # Use Equation 3.3.1 of Phillips
    p_r = t_r * v * math.cos(alpha_t)

    # Plot the results
    plot.plot(v, p_r / 1000)
    plot.set_xlabel('Airspeed␣(m/s)')
    plot.set_ylabel('Power␣Required␣(kW)')
    plot.set_title('Power␣Required␣vs.␣Airspeed')
    figure.savefig('plots/power_required.pdf')
    np.savetxt('data/power_required.txt', np.c_[v, p_r / 1000], delimiter='␣')

    return p_r


def climb_rate(rho_0, rho, p_a0, p_r, w, v):
    # Computes the rate of climb for various throttle settings
    figure, plot = plt.subplots()

    # Loop over a number of different throttle settings using Equation 3.4.8 of Phillips
    for i in range(0, 11):
        tau = i / 10.0 # The percent of max power
        p_a = tau * rho / rho_0 * p_a0 # Compute the available power adjusting for altitude
        v_c = (p_a - p_r) / w # Compute the climb rate

        # Plots the results for each power setting
        plot.plot(v, v_c, label=r'$\tau␣=␣$' + str(tau))
        np.savetxt('data/climb_rate_tau=' + str(i) + '.txt', np.c_[v, v_c], delimiter='␣')

    # Plot the results
    plot.set_xlabel('Airspeed␣(m/s)')
    plot.set_ylabel('Climb␣Rate␣(m/s)')
    plot.set_title('Climb␣Rate␣vs.␣Airspeed␣for␣Various␣Power␣Settings')
    plot.legend()
    figure.savefig('plots/climb_rate.pdf')

    return


def fuel_consumption(p_r, v, q_p):
    # Computes the specific fuel consumption to maintain level flight

    # Compute the fuel consumption using Equations 3.5.1 and 3.5.4 of Phillips
    q = q_p * p_r

    # Plot the results
    figure, plot = plt.subplots()
    plot.plot(v, q / 9.81 * 3600)
    plot.set_xlabel('Airspeed␣(m/s)')
    plot.set_ylabel('Fuel␣Consumption␣(kg/hr)')
    plot.set_title('Fuel␣Consumption␣vs.␣Airspeed')
    figure.savefig('plots/fuel_consumption.pdf')
    np.savetxt('data/fuel_consumption.txt', np.c_[v, q / 9.81 * 3600], delimiter='␣')

    return q


def specific_range(q, v):
    # Plots the specific range as a function of airspeed

    # Compute the specific range
    sr = q / v

    # Plot the results
    figure, plot = plt.subplots()
    plot.plot(v, sr / 9.81 * 1000)
    plot.set_xlabel('Airspeed␣(m/s)')
    plot.set_ylabel('Specific␣Range␣(kg/km)')
    plot.set_title('Specific␣Range␣vs.␣Airspeed')
    figure.savefig('plots/specific_range.pdf')
    np.savetxt('data/fuel_consumption.txt', np.c_[v, q / 9.81 * 1000], delimiter='␣')

    return
```

4.2.2 Static Stability

This code deals with finding the values of *static margin*, *yaw derivative* and *roll derivative* for a given aircraft.

Listing 19: Input values

```
# Aircraft parameters
cbar_w = 1.66255 # Mean chord of the main wing (m)
s_w = 16.7225 # Wing area (m^2)
s_h = 3.34451 # Horizontal stabilizer area (m^2)
s_v = 1.105546 # Vertical stabilizer area (m^2)
b_w = 10.0584 # Span of the main wing (m)
b_h = 3.6576 # Span of the horizontal stabilizer (m)
h_v = 0.9144 # Distance above center of gravity to the aerodynamic center of the tail
c_lw_alpha = 4.44 # Lift slope of the main wing
c_lh_alpha = 3.97 # Lift slope of the horizontal stabilizer
c_lv_alpha = 3.40 # Lift slope of the horizontal stabilizer
l_w = -0.216408 # Distance aft of the center of gravity to aerodynamic center of the main wing
    (m)
l_h = 4.355592 # Distance aft of the center of gravity to aerodynamic center of the horizontal
    stabilizer (m)
l_v = 4.514088 # Distance aft of the center of gravity to aerodynamic center of the vertical
    stabilizer (m)
eta_h = 1.0 # Dynamic pressure ratio relative to the free stream on the horizontal stabilizer
eta_v = 1.0 # Dynamic pressure ratio relative to the free stream on the vertical stabilizer
eps_d_alpha = 0.44 # Down wash gradient, or the change in down wash with angle of attack
eps_s_beta_v = -0.10 # Side wash gradient, or the change in side wash with angle of attack
gamma = -0.1 # Wing dihedral angle (degrees)
kappa_l = 1.07 # Wing dihedral factor (See Figure 5.6.3 of Phillips)
kappa_gamma = 0.83 # Wing dihedral factor (See Figure 5.6.3 of Phillips)
```

Using equations 4.30, 4.32, 4.34, 4.35, 4.36, the required values are determined.

Listing 20: Input values

```
def static_margin(s_w, s_h, c_lw_alpha, c_lh_alpha, l_w, l_h, eta_h, eps_d_alpha, cbar_w):
    # Example 4.4.1 of Phillips
    sm = (l_w * c_lw_alpha + (s_h * l_h) / (s_w) * eta_h * c_lh_alpha * (1 - eps_d_alpha)) / (
            cbar_w * (c_lw_alpha + s_h / s_w * eta_h * c_lh_alpha * (1 - eps_d_alpha)))

    # Print the static margin
    print('Static␣Margin:␣', round(sm * 100, 2), '%␣(recommended␣5->15%)', sep='')

    return


def yaw_derivative(eta_v, s_v, s_w, l_v, l_w, b_w, c_lv_alpha, eps_s_beta_v):
    # Equation 5.2.7 of Phillips
```

```python
    deltac_n_beta_v = eta_v * (s_v * l_v) / (s_w * b_w) * c_lv_alpha * (1 - eps_s_beta_v)

    # Print the yaw stability derivative
    print('Yaw␣Stability␣Derivative:␣', round(deltac_n_beta_v, 3),
          '␣(recommended␣0.06->0.15,␣vertical␣stabilizer␣contribution␣only!)', sep='')

    return


def roll_derivative(gamma, kappa_gamma, kappa_l, c_lw_alpha, h_v):
    # Convert the dihedral angle to radians
    gamma = gamma * math.pi / 180

    # Get the contribution from the main wing, assuming negligible sweep, via Equation 5.6.13
        in Phillips
    deltac_l_beta_gammaw = -(2 * math.sin(gamma)) / (
            3 * math.pi * math.cos(gamma) ** 4) * kappa_gamma * kappa_l * c_lw_alpha

    # Get the contribution from the vertical stabilizer via Equation 5.6.22 in Phillips
    deltac_l_beta_v = -eta_v * (s_v * h_v) / (s_w * b_w) * (1 - eps_s_beta_v) * c_lv_alpha

    # Get the contribution from the horizontal stabilizer via Equation 5.6.23 in Phillips (
        assuming conventional tail)
    deltac_l_beta_h = + 0.08 * eta_v * (s_v * b_h) / (s_w * b_w) * (1 - eps_s_beta_v) * c_lv_alpha

    # Get the total roll stability derivative
    deltac_n_beta_v = deltac_l_beta_gammaw + deltac_l_beta_v + deltac_l_beta_h

    # Print the roll stability derivative
    print('Roll␣Stability␣Derivative:␣', round(deltac_n_beta_v, 3), '␣(recommended␣-0.1->0)',
        sep='')

    return
```

4.3 Validation

4.3.1 Performance code

The *performance.py* code is run using the input parameters for Cessna 172 aircraft. A preliminary validation of this code was done by comparing the general curve shapes obtained in the resulting plots with the typical plots discussed in the theory section. The plots resulting from the code are included below and it can be observed that they agree fairly well with the theory. These plots are a result of the code run at Sea level (setting density at current altitude input equal to 1.22 kg/m^3).

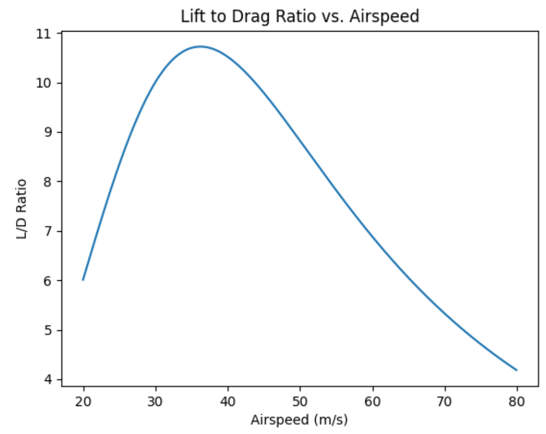

Figure 4.12: Lift-to-Drag ratio

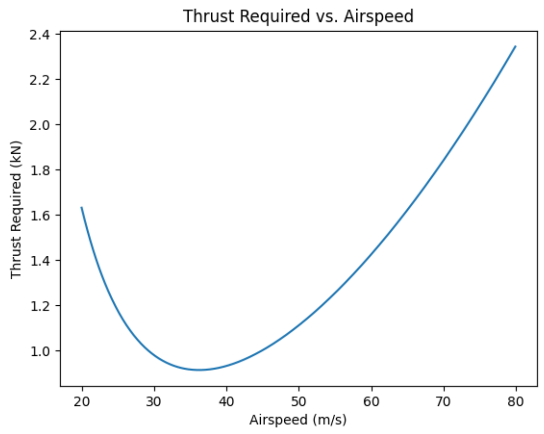

Figure 4.13: Thrust Required

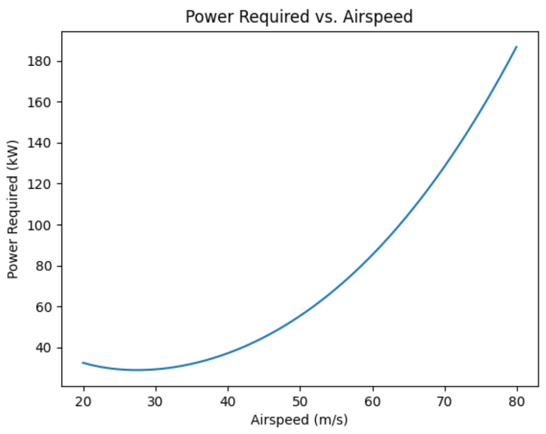

Figure 4.14: Power Required

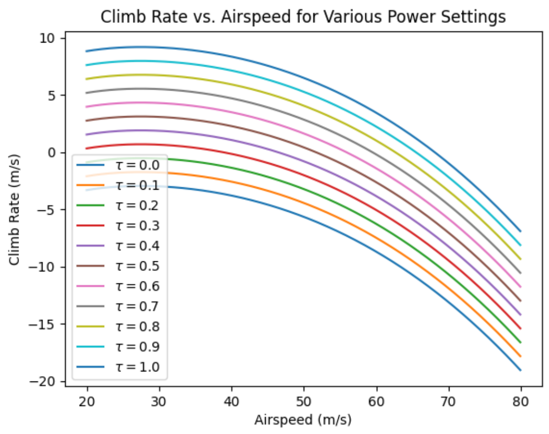

Figure 4.15: Rate of Climb

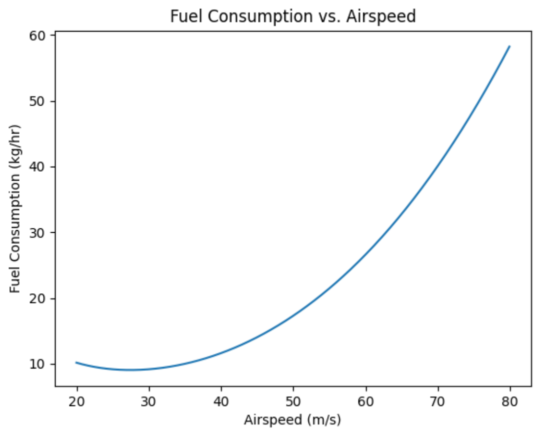

Figure 4.16: Fuel consumption

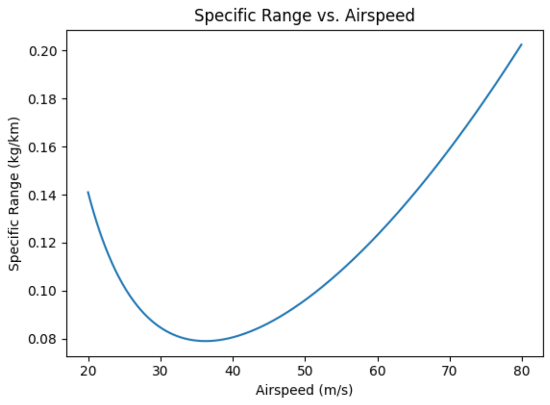

Figure 4.17: Specific Range

Furthermore, a few data points resulting from the code were compared with experimental data furnished in the aircraft's handbook [5].

1. From reference [5], the value of maximum Rate of Climb = 770 fpm = 3.9 m/s at an airspeed value of 73 kts = 37.5 m/s for full throttle ($\tau = 1$) at sea level. In comparison, the code was run at sea level (i.e. setting the input density value to 1.22 kg/m^3), and the value of maximum rate of climb was found to be 9.1 m/s at an airspeed value of 27.5 m/s. At a speed of 37.5 m/s, the value of climb rate was found to be 8.6 m/s.

2. Next, we attempt to compare Specific Range values. From the handbook [5], under cruise performance, for 75% power at 8000 feet, Range value is indicated as 485 nm = 898.2 kms, for 40 gallons usable fuel. Considering density of the fuel (Aviation grade straight mineral oil) as 0.88 kg/ltr, the weight of the fuel corresponding to 40 gallons (= 151.4 ltrs) would be around 133 kgs. Dividing the value of range with the weight of fuel consumed, the specific range value = 0.15 kg/km. Also, maximum cruise speed, 75% power at 8000 feet, is equal to 122 kts = 63 m/s. Running the code taking altitude as 8000 feet, (i.e. setting the input density value to 0.95 kg/m^3), value of specific range at an airspeed of 63 m/s is obtained as 0.11 kg/km.

3. Both of the above points show a discrepancy between the experimental value and the code. This deviation can be attributed to the fact that the code does not take actual propulsive efficiencies into account. Further information is needed to determine the actual power available after accounting for the aerodynamic efficiency of the propulsion system.

4. From the results of the code, maximum L/D ratio is around 10.7, and from a different analysis [6] maximum L/D ratio is calculated as 10.9, which shows good agreement.

References

[1] John D. Anderson, Jr., *Fundamentals of Aerodynamics*, sixth ed., McGraw-Hill Education, New York, 2017.

[2] Arnold M. Kuethe and Chuen-Yen Chow, *Foundations of Aerodynamics Bases of Aerodynamic Design*, fifth ed., John Wiley & Sons, Inc., New York, 1998.

[3] Warren F. Phillips, *Mechanics of Flight*, John Wiley & Sons, Inc., New Jersey, 2004.

[4] Ira H. Abbott, Albert E. von Doenhoff, and Louis S. Stivers, Jr., *Report No. 824, Summary of Airfoil Data*, NACA.

[5] *Pilot's Operating Handbook, Cessna Skyhawk 172N*, Cessna Aircraft Company, Kansas, 1977.

[6] John McIver B.Eng., *Cessna Skyhawk Performance Assessment*, Temporal images. `http//temporal.com.au/c172.pdf`