Assignment 4: Aircraft Performance Analysis

Supervisor: Prof. Rajkumar Pant

Submitted by Team 4 -

Ambuj Nayan (24M0003)
Aparnathi Harshgiri Prakashgiri (21D170007)
Gulshan Kumar (22B0731)
Kharat Sahil Ashok (210110062)
M Kidron (24M0019)
Manasvi Kushwaha (22B2268)
Manish Sunil Lakhode (24M0040)
Omkar Lalsingh Bilawar (24M0050)
Prabhav Sanghavi (23B0346)
Rakesh Kumar Gaula (24M0060)
Ram Naresh Mahto (24M0065)
Shivangi Dhawan (24M0031)
Shivanshu Mishra (22B0764)
Shlok Zanwar (210100138)
Vishak V (24M0017)

Department of Aerospace Engineering
Indian Institute of Technology Bombay

November 2024

Table of Contents for Airbus A380 Report

Title **Page No.**

Table of Contents……...2

Abstract……..3

1. Introduction……3

2. Objective……...3

3. Methodology……..3

3.1 Input Parameters……...3

3.2 Calculations……..4

3.2.1 Aspect Ratio and Induced Drag Factor……4

3.2.2 Stalling Speeds……..4

3.2.3 Maximum Velocity…….....4

3.2.4 Take-off Distance……..…4

3.2.5 Balanced Field Length……..5

3.2.6 Climb Performance……...5

3.2.7 Ceilings……..……...5

3.2.8 Turning Performance…….……...5

3.2.9 Landing Distance……..……5

4. Output Results ……..……6

5. Code Explanation for Aircraft Performance Calculations…….……6

5.1 Importing the math module…….……6

5.2 Defining Aircraft and Environmental Parameters…….……..6

5.3 Calculating Aspect Ratio and Induced Drag Factor…….……....7

5.4 Stalling Speeds…….……....7

5.5 Maximum Velocity Calculations…….……....7

5.6 Take-off Distance Calculations…….…….….8

5.7 Balanced Field Length Calculations……..…….….8

5.8 Climb Performance Calculations……..……...9

5.9 Absolute and Service Ceilings……..……...9

5.10 Turning Performance……..…….……10

5.11 Landing Distance Calculation…….……10

5.12 Output Function……..…….……10

6. Output…….……11

7. Conclusion…….……11

8. Appendix…….……12

Abstract

This report presents a comprehensive performance analysis of the Airbus A380, utilizing Python-based data analysis techniques to assess key operational metrics including,stalling speed, maximum speed and turning capabilities. Additionally, the report analyses takeoff and landing distances. The analysis incorporates aerodynamic principles, aircraft specifications and atmospheric conditions at both sea level and cruising level.

1. Introduction

The Airbus A380, a wide-body, four-engine, long-haul aircraft, is one of the most advanced and ambitious commercial aircraft developed by Airbus. With its unprecedented passenger capacity and advanced engineering, the A380 was designed to cater to high-demand routes connecting major international hubs. However, as operational costs and fuel efficiency remain primary concerns for airlines, the need for a detailed performance analysis of the A380 has become increasingly important.

This report provides an analysis of the Airbus A380's performance characteristics, including takeoff distance and landing distance and operational performance. Python, with its extensive libraries for data analysis and visualization, is well-suited for performing detailed performance analysis.The code is structured to calculate several critical performance parameters for the Airbus A380, focusing on aspects such as stalling speeds, maximum speeds, takeoff distances, climb performance, ceilings, turning capabilities, and landing distance.

2. Objective

The primary objective of this report is to conduct a comprehensive performance analysis of the Airbus A380, with the aim of providing actionable insights that can help optimise its operational efficiency. Specifically, the analysis seeks to:

- Calculate Stalling Speed
- Access Maximum Velocity
- Evaluate Take off distance and Landing distance
- Analyse Climb Performance
- Analyse Turning Performance

3. Methodology

3.1 Input Parameters

The code begins by defining input parameters, including:

- **Atmospheric Conditions**: Standard sea-level pressure (p_sea), temperature (T_sea), and density (rho_sea) as well as density at cruising altitude (rho_cruise).
- **Aircraft Specifications**:
    - Wingspan (b), wing area (S), power output (Power), thrust (Thrust), and weights (W for gross weight, Wo for empty weight).
    - Drag and efficiency parameters, including parasite drag coefficient (Cdo), Oswald efficiency factor (e), maximum lift coefficient (Clmax), and friction coefficient (friction_coeff).
    - Performance-related values like maximum velocity (Vmax) and load factor (n).

3.2 Calculations

3.2.1 Aspect Ratio and Induced Drag Factor

- **Aspect Ratio (AR)**: The ratio of wingspan to wing area, calculated as AR= $\frac{b^2}{S}$.
- **Induced Drag Factor (k)**: Calculated using Oswald's efficiency factor, as k= $\frac{1}{\pi e AR}$

3.2.2 Stalling Speeds

- **Sea-level and Cruising Altitude Stalling Speeds**: Calculated using the formula Vstall=2WρSCl,maxV_{\text{stall}}=\sqrt{\frac{2W}{\rhoSC_{l,\text{max}}}}Vstall=ρSCl,max2W for sea level and cruising altitudes, where Clmax represents the maximum lift coefficient.

3.2.3 Maximum Velocity

- **Sea-level and Cruising Altitude Maximum Velocity**: Uses thrust availability and aerodynamic drag to determine V_max at both altitudes.

3.2.4 Take-off Distance

The takeoff distance is calculated in three parts:

- **Ground Roll (s_ground)**: Accounts for lift-off velocity, drag, rolling friction, and lift.
- **Transition Distance (s_transition)**: 10% of the ground roll.

- **Climb Distance (s_climb)**: Calculated using climb angle.
- **Total Takeoff Distance (takeoff_dist)**: Sum of the ground roll, transition distance, and climb distance.

3.2.5 Balanced Field Length

The balanced field length considers both **accelerate-go** and **accelerate-stop** distances to account for cases where an engine fails before takeoff.

3.2.6 Climb Performance

- **Maximum Climb Angle (theta_max)**: Determined using thrust-to-weight ratio and drag factors.
- **Rate of Climb (RoC)**: Calculated for the maximum climb angle and for maximum rate of climb, providing insight into the aircraft's altitude gain over time.

3.2.7 Ceilings

- **Absolute Ceiling**: The altitude at which the rate of climb reaches zero.
- **Service Ceiling**: The altitude at which the rate of climb is 0.508 m/s (100 feet/min), determined through iterative calculations using altitude-dependent air density.

3.2.8 Turning Performance

The turning performance calculates:

- **Radius and Rate of Turn** for a **level turn**, **pull-up maneuver**, and **pull-down maneuver**, based on speed, load factor, and gravity.

3.2.9 Landing Distance

The landing distance calculation includes touchdown velocity, lift, drag, and ground friction effects.

4. Output Results

The main() function prints all calculated values in an organized format, providing insights into the Airbus A380's performance characteristics at sea level and at altitude.

5. Code Explanation for Aircraft Performance Calculations

This code is a detailed computational tool for analyzing various performance metrics of the Airbus A380, including stalling speed, maximum speed, takeoff and landing distances, climb rate, turning performance, and ceiling limits. Each section performs a different calculation based on aerodynamic principles, making use of physics-based formulas to simulate real-world aircraft behavior under specified conditions.

5.1 Importing the Math Module

python
Copy code

```
import math
```

The math library provides essential mathematical functions (such as sqrt, sin, asin) necessary for aerodynamic calculations, including square roots and trigonometric functions.

5.2 Defining Aircraft and Environmental Parameters

These parameters define the basic properties of the Airbus A380 and environmental conditions:

python
Copy code

```
p_sea = 101325  # Standard sea-level pressure in Pascal
T_sea = 288  # Standard sea-level temperature in Kelvin
rho_sea = 1.225  # Standard sea-level density in kg/m^3
h_cruise = 10000  # Cruising altitude of A380 in meters
rho_cruise = 0.4135 # Cruising altitude density in kg/m^3
```

- **Atmospheric Conditions**: The standard atmospheric values are used to simulate conditions at sea level, while rho_cruise gives the density at cruising altitude (10,000 meters).
- **Aircraft Specifications**:
    - **Geometry and Power**: b is the wingspan, S is the wing area, and Power represents the engine power.

- **Thrust and Weight**: Thrust represents total thrust, and W represents the total weight, including passengers and cargo.
- **Drag and Efficiency**: Cdo is the parasite drag coefficient, representing drag from non-lift-related surfaces, while e is the Oswald efficiency factor for induced drag.
- **Other Factors**: Vmax (maximum speed), Clmax (maximum lift coefficient), and n (load factor) are values used in various performance calculations.

5.3 Calculating Aspect Ratio and Induced Drag Factor

python
Copy code

```
AR = b * b / S
k = 1 / (math.pi * e * AR)
```

- **Aspect Ratio (AR)**: The ratio of wingspan to wing area, which influences induced drag.
- **Induced Drag Factor (k)**: Calculated from the Oswald efficiency factor, this parameter is used to account for induced drag.

5.4 Stalling Speeds

python
Copy code

```
V_stall_sea = ((2 * W) / (rho_sea * S * Clmax)) ** 0.5
V_stall_cruise = ((2 * W) / (rho_cruise * S * Clmax)) ** 0.5
```

- **Stalling Speed at Sea Level and Cruise Altitude**: The code calculates the minimum speeds required for flight (stall speeds) at sea level and cruising altitude. These are essential for determining safe operating speeds.

5.5 Maximum Velocity Calculations

python
Copy code

```
T_avail_max = (0.5 * rho_sea * Vmax**2 * S * Cdo) + ((2 * k * W**2) / (rho_sea * Vmax**2 * S))
V_max_sea = (((T_avail_max / S) + ((W / S) * ((((T_avail_max / W) ** 2) - 4 * Cdo * k)**0.5))) / (rho_sea * Cdo)) ** 0.5
```

```
V_max_cruise = (((T_avail_max / S) + ((W / S) * ((((T_avail_max / W) ** 2) - 4 * Cdo * k)**0.5))) / (rho_cruise * Cdo)) ** 0.5
```

- **Maximum Velocity at Sea Level and Cruise Altitude**: Determines the maximum velocity achievable based on thrust and drag, both at sea level and at cruise altitude.

5.6 Take-off Distance Calculations

The code divides takeoff distance into three components:

- **Ground Roll (s_ground)**: Distance required to reach lift-off speed, considering lift, drag, and rolling friction.
- **Transition Distance (s_transition)**: Assumes 10% of the ground roll distance to account for the phase when the aircraft begins its climb.
- **Climb Distance (s_climb)**: Determined by the climb angle, calculated from thrust and drag.

python
Copy code

```
V_Lo = 1.2 * V_stall_sea
L1 = 0.5 * rho_sea * V_Lo**2 * S * Clmax
D1 = 0.5 * rho_sea * V_Lo**2 * S * Cdo
s_ground = (1.44 * W**2) / (9.81 * rho_sea * Clmax * S * (Thrust - D1 - (friction_coeff * (W - L1))))
s_transition = 0.1 * s_ground
climb_angle = math.asin((Thrust - D1) / W)
s_climb = 15 / math.tan(climb_angle)
takeoff_dist = s_ground + s_transition + s_climb
```

5.7 Balanced Field Length Calculation

Balanced field length takes into account **accelerate-go** and **accelerate-stop** distances for a scenario where one engine fails before reaching takeoff speed:

python
Copy code

```
deceleration = 3.0
V1 = 0.95 * V_Lo
D_go = 0.5 * rho_sea * V1**2 * S * Cdo
L_go = 0.5 * rho_sea * V1**2 * S * Clmax
a_go = ((Thrust / 2) - D_go - friction_coeff * (W - L_go)) / (W / 9.81)
```

```
s_go = V1**2 / (2 * a_go)
s_rotation = 0.1 * s_go
gamma = math.asin((Thrust / 2 - D_go) / W)
s_transition = 15 / math.tan(gamma)
s_total_go = s_go + s_rotation + s_transition
s_stop = V1**2 / (2 * deceleration)
balanced_field_length = max(s_total_go, s_stop)
```

5.8 Climb Performance Calculations

Calculates:

- **Maximum Climb Angle (theta_max)**: The maximum angle the aircraft can achieve.
- **Rate of Climb (RoC)**: Maximum rate of altitude gain per second.

python
Copy code

```
theta_max = math.asin((Thrust / W) - (4 * k * Cdo) ** 0.5)
V_theta_max = (2 * W * math.cos(theta_max) * ((k / Cdo) ** 0.5) / (rho_sea * S)) ** 0.5
RoC = V_theta_max * math.sin(theta_max)
```

5.9 Absolute and Service Ceilings

Calculates:

- **Absolute Ceiling**: The altitude at which RoC is zero.
- **Service Ceiling**: The altitude at which RoC reaches a minimum threshold (0.508 m/s).

python
Copy code

```
while altitude < 20000:
    ...
    if service_ceiling is None and RoC <= service_ceiling_roc:
        service_ceiling = altitude
    if RoC <= absolute_ceiling_roc:
        absolute_ceiling = altitude
        break
    altitude += 10
```

5.10 Turning Performance

Calculates turning radius and rate of turn for:

- **Level Turn**
- **Pull-up and Pull-down Maneuvers**

python
Copy code
```
turning_radius = (V_stall_cruise ** 2) / (9.81 * (((n * n) - 1) ** 0.5))
turning_rate = V_stall_cruise / turning_radius
```

5.11 Landing Distance Calculation

Determines the distance required for a safe landing, accounting for touchdown speed, drag, and ground friction.

python
Copy code
```
V_TD = 1.15 * V_stall_sea
s_landing = (1.69 * W**2) / (9.81 * rho_sea * S * Cl * (D2 + friction_coeff * W))
```

5.12 Output Function

The main() function organizes and prints the results for each performance metric, providing a clear summary of the Airbus A380's capabilities.

This code efficiently calculates a comprehensive set of performance values based on the Airbus A380's design and operational parameters, making it a powerful tool for aircraft performance analysis.

6. Output -

Stalling speed-
At sea-level: 84.120 m/s
At cruise altitude: 144.788 m/s
Maximum speed-
At sea-level: 250.000 m/s
At cruise altitude: 430.299 m/s
Take-off distances-
Lift-off distance: 1076.505 m
Transition distance: 124.484 m
Climb distance: 55.151 m
Total take-off distance required: 1239.306 m
Balanced Field Length: 2038.111 m
Climb Performance-
Maximum climb angle: 12.41 degrees
Rate of Climb for maximum climb angle: -0.015 m/s
Climb angle for maximum Rate of Climb: 10.08 degrees
Maximum Rate of Climb: 33.239 m/s
Ceilings-
Absolute Ceiling: 11390.000 m
Service Ceiling: 11140.000 m

Turning performance-
Radius of turn for level turn: 551.758 m
Rate of turn for level turn: 0.262 rad/s
Radius of turn for pull-up manuever: 712.317 m
Rate of turn for pull-up maneuver: 0.203 rad/s
Radius of turn for pull-down manuever: 427.390 m
Rate of turn for pull-down maneuver: 0.339 rad/s
Landing distances-
Total landing distance required: 1386.198 m

7. Conclusion

The code provides a comprehensive set of performance metrics for the Airbus A380, useful for engineering analysis, flight simulation, or aerodynamic optimization. Each calculation aligns with standard aerodynamic principles, making this a practical tool for understanding large aircraft performance parameters.

8.Appendix
(Code)

```
import math

# Specifications of the aircraft Airbus A380-
p_sea = 101325  # Standard sea-level pressure in Pascal
T_sea = 288  # Standard sea-level temperature in Kelvin
rho_sea = 1.225  # Standard sea-level density in kg/m^3
h_cruise = 10000  # Cruising altitute of A380 in metres
rho_cruise = 0.4135 # Crising altitude density in kg/m^3
b = 79.8  # Wingspan (b) in metres
S = 845  # Wing area (S) in sq. metres
Power = 300000000  # Power in watt
Thrust = 400000 * 4  # Thrust (T) in Newton
Cdo = 0.03  # Parasite Drag Coefficient (Cdo)
e = 0.87  # Oswald Efficiency factor (e)
W = 5493600 # Gross weight (W) in Newton
Wo = 2718870 # Empty weight (Wo) in Newton
Vmax = 250  # Maximum velocity (Vmax) in metre per second
Clmax = 1.5  # Clmax for typical aircrafts with triple-slotted flaps and eeading edge slats
friction_coeff = 0.5  # Rolling friction coefficient
n = 4  # Load factor


# Calculating aspect ratio and induced drag factor
AR = b * b / S
k = 1/(math.pi * e * AR)


# Calculating stalling speed for sea-level and cruising altitude
V_stall_sea = ((2 * W)/(rho_sea * S * Clmax)) ** 0.5
V_stall_cruise = ((2 * W)/(rho_cruise * S * Clmax)) ** 0.5


# Calculating maximum velocity for sea-level and cruising altitude
T_avail_max = (0.5 * rho_sea * Vmax * Vmax * S * Cdo) + ((2 * k * W * W)/(rho_sea * Vmax * Vmax *
S))
V_max_sea = (((T_avail_max / S) + ((W / S) * ((((T_avail_max / W) ** 2) - 4 * Cdo * k)**
0.5)))/(rho_sea * Cdo)) ** 0.5
```

```
V_max_cruise = (((T_avail_max / S) + ((W / S) * ((((T_avail_max / W) ** 2) - 4 * Cdo * k)**
0.5)))/(rho_cruise * Cdo)) ** 0.5


# Calculating the take-off distance required
V_Lo = 1.2 * V_stall_sea  # Lift-off velocity
L1 = 0.5 * rho_sea * V_Lo * V_Lo * S * Clmax  # Lift during take-off
D1 = 0.5 * rho_sea * V_Lo * V_Lo * S * Cdo  # Drag during take-off
s_ground = (1.44 * W * W)/(9.81 * rho_sea * Clmax * S * (Thrust - D1 - (friction_coeff * (W - L1))))
s_transition = 0.1 * s_ground
climb_angle = math.asin((Thrust - D1) / W)
s_climb = 15 / math.tan(climb_angle)
takeoff_dist = s_ground + s_transition + s_climb


# Calculating the balanced field length
deceleration = 3.0
V1 = 0.95 * V_Lo  # Decision speed V1
D_go = 0.5 * rho_sea * V1**2 * S * Cdo  # Drag with one engine
L_go = 0.5 * rho_sea * V1**2 * S * Clmax  # Lift with one engine
a_go = ((Thrust/2) - D_go - friction_coeff * (W - L_go)) / (W / 9.81)
s_go = V1**2 / (2 * a_go)  # Ground roll with one engine out

# Rotation and Transition Distances (estimated as in takeoff distance)
s_rotation = 0.1 * s_go
gamma = math.asin((Thrust/2 - D_go) / W)  # Climb angle with one engine
s_transition = 15 / math.tan(gamma)
s_total_go = s_go + s_rotation + s_transition  # Total Accelerate-Go Distance
s_stop = V1**2 / (2 * deceleration)  # Accelerate-Stop Distance
balanced_field_length = max(s_total_go, s_stop)  # Balanced Field Length (BFL)


# Calculating the parameters for climb performance
theta_max = math.asin((Thrust / W) - (4 * k * Cdo) ** 0.5)  # Value of maximum climb angle for rate of
climb
V_theta_max = (2 * W * math.cos(theta_max) * ((k / Cdo) ** 0.5) / (rho_sea * S)) ** 0.5  # Velocity at
maximum climb angle
RoC = V_theta_max * math.sin(theta_max)  # Value of Rate of climb for maximum climb angle

L_by_D_max = (1 / (4 * k * Cdo)) ** 0.5
z = 1 + ((1 + ((3 * W * W) / ((L_by_D_max * Thrust) ** 2))) ** 0.5)
V_RoC_max = (Thrust * z / (3 * S * rho_sea * Cdo)) ** 0.5  # Velocity for maximum rate of climb
```

```
RoC_max = ((W * z / (3 * S * rho_sea * Cdo)) ** 0.5) * ((Thrust / W) ** 1.5) * (1- (z/6) - (3 * W * W / (2 * (Thrust ** 2) * (L_by_D_max ** 2) * z)))  # Value of maximum rate of climb
theta_RoC_max = math.asin(RoC_max / V_RoC_max)  # Value of climb angle for maximum rate of climb


# Calculating the absolute ceiling and the service ceiling
# For absolute ceiling the maximum rate of climb is 0 m/s and for service ceiling the maximum rate of climb is 0.508 m/s
service_ceiling_roc = 0.508
absolute_ceiling_roc = 0.0
altitude = 0
service_ceiling = None
absolute_ceiling = None

# Finding absolute ceiling and service ceiling through iterations
while altitude < 20000:
    rho_cruise = rho_sea * math.exp(-altitude / 8500)
    V_cruise = ((2 * W) / (rho_cruise * S * (Cdo / k) ** 0.5)) ** 0.5
    Cl_c = (2 * W) / (rho_cruise * V_cruise**2 * S)
    D_c = 0.5 * rho_cruise * V_cruise**2 * S * (Cdo + k * Cl_c**2)
    Thrust_c = Thrust * (rho_cruise / rho_sea)
    RoC = V_cruise * (Thrust_c - D_c) / W

    # For service ceiling
    if service_ceiling is None and RoC <= service_ceiling_roc:
        service_ceiling = altitude

    # For absolute ceiling
    if RoC <= absolute_ceiling_roc:
        absolute_ceiling = altitude
        break
    altitude += 10


# Calculating the parameters for turning performance
# Turn radius and turning rate of level turns
turning_radius = (V_stall_cruise ** 2) / (9.81 * (((n * n) - 1) ** 0.5))
turning_rate = V_stall_cruise / turning_radius

# Turn radius and turning rate for pull up manneuver
pull_up_radius = (V_stall_cruise ** 2) / (9.81 * (n - 1))
pull_up_rate = V_stall_cruise / pull_up_radius
```

```
# Turn radius and turning rate for pull down maneuver
pull_down_radius = (V_stall_cruise ** 2) / (9.81 * (n + 1))
pull_down_rate = V_stall_cruise / pull_down_radius



# Calculating the landing distance
V_TD = 1.15 * V_stall_sea  # Touch down velocity
Cl = (2 * W)/(rho_sea * V_TD * V_TD * S)
L2 = 0.5 * rho_sea * V_TD * V_TD * S * Cl
D2 = 0.5 * rho_sea * V_TD * V_TD * S * (Cdo + (k * Cl * Cl))
s_landing = (1.69 * W * W)/(9.81 * rho_sea * S * Cl * (D2 + friction_coeff * (W)))



def main(V_stall_sea, V_stall_cruise, V_max_sea, V_max_cruise, s_ground, s_transition, s_climb,
takeoff_dist, theta_max, RoC, theta_RoC_max, RoC_max, absolute_ceiling, service_ceiling,
turning_radius, turning_rate, pull_up_radius, pull_up_rate, pull_down_radius, pull_down_rate, s_landing):
    print("Stalling speed-")
    print("At sea-level: {:.3f} m/s".format(V_stall_sea))
    print("At cruise altitude: {:.3f} m/s".format(V_stall_cruise))
    print("\nMaximum speed-")
    print("At sea-level: {:.3f} m/s".format(V_max_sea))
    print("At cruise altitude: {:.3f} m/s".format(V_max_cruise))
    print("\nTake-off distances-")
    print("Lift-off distance: {:.3f} m".format(s_ground))
    print("Transition distance: {:.3f} m".format(s_transition))
    print("Climb distance: {:.3f} m".format(s_climb))
    print("Total take-off distance required: {:.3f} m".format(takeoff_dist))
    print("Balanced Field Length: {:.3f} m".format(balanced_field_length))
    print("\nClimb Performance-")
    print("Maximum climb angle: {:.2f} degrees".format(math.degrees(theta_max)))
    print("Rate of Climb for maximum climb angle: {:.3f} m/s".format(RoC))
    print("Climb angle for maximum Rate of Climb: {:.2f}
degrees".format(math.degrees(theta_RoC_max)))
    print("Maximum Rate of Climb: {:.3f} m/s".format(RoC_max))
    print("\nCeilings-")
    print("Absolute Ceiling: {:.3f} m".format(absolute_ceiling))
    print("Service Ceiling: {:.3f} m".format(service_ceiling))
    print("\nTurning performance-")
    print("Radius of turn for level turn: {:.3f} m".format(turning_radius))
    print("Rate of turn for level turn: {:.3f} rad/s".format(turning_rate))
```

```
    print("Radius of turn for pull-up manuever: {:.3f} m".format(pull_up_radius))
    print("Rate of turn for pull-up maneuver: {:.3f} rad/s".format(pull_up_rate))
    print("Radius of turn for pull-down manuever: {:.3f} m".format(pull_down_radius))
    print("Rate of turn for pull-down maneuver: {:.3f} rad/s".format(pull_down_rate))
    print("\nLanding distances-")
    print("Total landing distance required: {:.3f} m".format(s_landing))


a = main(V_stall_sea, V_stall_cruise, V_max_sea, V_max_cruise, s_ground, s_transition, s_climb,
takeoff_dist, theta_max, RoC, theta_RoC_max, RoC_max, absolute_ceiling, service_ceiling,
turning_radius, turning_rate, pull_up_radius, pull_up_rate, pull_down_radius, pull_down_rate, s_landing)
a
```

Aircraft performance analysis for Lockheed Martin C-130 Hercules

Abstract **3**
1. Introduction **3**
2. Objective **3**
3. Methodology **4**
3.1 Input Parameters **4**
3.2 Calculations **4**
3.2.1 Aspect Ratio and Induced Drag Factor **4**
3.2.2 Stalling Speeds **5**
3.2.3 Maximum Velocity **5**
3.2.4 Take-off Distance **5**
3.2.5 Balanced Field Length **6**
3.2.6 Climb Performance **6**
3.2.7 Ceilings **7**
3.2.8 Turning Performance **7**
3.2.9 Landing Distance **7**
4. Parameters **8**
5. Output Results **8**
6. Conclusion **9**
1. Takeoff and Landing Performance 9
2. Maximum Speed 10
3. Climb Performance 10
4. Service Ceiling and Absolute Ceiling 10
5. Turn Performance 10
Implications and Practical Applications 11
7. Reference **12**
8. Appendix **13**

Abstract

This report presents a comprehensive performance analysis of the C-130 Hercules, utilizing Python-based data analysis techniques to assess key operational metrics including, stalling speed, maximum speed and turning capabilities. Additionally, the report analyzes takeoff and landing distances. The analysis incorporates aerodynamic principles, aircraft specifications and atmospheric conditions at both sea level and cruising level.

1. Introduction

The **C-130 Hercules** is a four-engine, tactical transport aircraft developed by **Lockheed Martin**. Known for its versatility, rugged design, and ability to operate from short and unprepared airstrips, the C-130 has been widely used for military transport, medical evacuation, and humanitarian aid missions.

This report provides an analysis of the **C-130 Hercules's** performance characteristics, including takeoff and landing distances, stalling speeds, maximum speeds, climb performance, ceilings, turning capabilities, and landing performance. Python, with its extensive libraries for data analysis and visualization, is well-suited for performing detailed performance analysis. The code is structured to calculate several critical performance parameters specific to the **C-130 Hercules**, addressing operational aspects essential for military and logistical applications.

2. Objective

The primary objective of this report is to conduct a comprehensive performance analysis of the C-130 Hercules. Specifically, the analysis seeks to:

- Calculate Stalling Speed
- Access Maximum Velocity
- Evaluate Take off distance and Landing distance
- Analyze Climb Performance

- Analyze Turning Performance

3. Methodology

3.1 Input Parameters

The code begins by defining input parameters, including:

- **Atmospheric Conditions**: Standard sea-level pressure ($p_{sea-level}$), temperature ($T_{sea-level}$), and density ($\rho_{sea\ level}$) as well as density at cruising altitude (ρ_{cruise}).
- **Aircraft Specifications**:
  - Wingspan (b), wing area (S), power output (Power), thrust (Thrust), and weights (W for gross weight, Wo for empty weight).
  - Drag and efficiency parameters, including parasite drag coefficient (C_{D_o}), Oswald efficiency factor (e), maximum lift coefficient ($C_{L_{max}}$), and friction coefficient (μ_r).
  - Performance-related values like maximum velocity (V_{max}) and load factor (n).

3.2 Calculations

3.2.1 Aspect Ratio and Induced Drag Factor

- **Aspect Ratio (AR)**: The ratio of wingspan to wing area, calculated as $AR=\frac{(b)^2}{S}$.
- **Induced Drag Factor (k)**: Calculated using Oswald's efficiency factor, as $k=\frac{1}{\pi e AR}$.

3.2.2 Stalling Speeds

- **Sea-level and Cruising Altitude Stalling Speeds**: Calculated using the formula $V_{stall,sea\ level} = \sqrt{\frac{2W}{\rho_{sea\ level} S C_{L_{max}}}}$ $V_{stall,cruise} = \sqrt{\frac{2W}{\rho_{cruise} S C_{L_{max}}}}$ for sea level and cruising altitudes, where $C_{L_{max}}$ represents the maximum lift coefficient.

3.2.3 Maximum Velocity

- **Sea-level and Cruising Altitude Maximum Velocity**: Uses power availability and power required to determine V_{max} at both altitudes.

$$P_{available} = P_{sealevel} \times \left(\frac{\rho}{\rho_{sealevel}}\right)^{0.7}$$

$$P_{available} = P_{required}$$

$$P_{required} = \frac{1}{2} \times \rho \times v^3 \times S \times C_D$$

$$C_D = C_{D_o} + kC_L^2$$

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

3.2.4 Take-off Distance

The takeoff distance is calculated in three parts:

- **Ground Roll (S_{Ground})**: Accounts for lift-off velocity, drag, rolling friction, and lift.

$$a_{ground} = \frac{\left(\left(T - D_{ground} - \mu_r \times \left(W - L_{ground}\right)\right)\right)}{\left(\frac{W}{9.81}\right)}$$

$$S_{ground} = \frac{(V_{takeoff})^2}{2 \times a_{ground}}$$

- **Rotation Distance($S_{Rotation}$)**: 10% of the ground roll.

$$S_{rotation} = 0.1 \times S_{ground}$$

- **Transition Distance ($S_{Transition}$)**: Distance to clear 50 ft obstacle

$$S_{transition} = \frac{50}{\tan \gamma}$$

- **Total Takeoff Distance (S_{Total})**: Sum of the ground roll, transition distance, and climb distance.

$$S_{total\_takeoff} = S_{ground} + S_{rotation} + S_{transition}$$

3.2.5 Balanced Field Length

The balanced field length considers both **accelerate-go** and **accelerate-stop** distances to account for cases where an engine fails before takeoff.

$$S_{Go} = S_{ground\ (with\ one\ engine\ failure)}$$

$$S_{Total\ Go} = S_{Go} + S_{Transition} + S_{rotation}$$

$$S_{stop} = \frac{V^2}{2*a_{Retardation}}$$

$a_{Retardation}$ is taken as 3 generally

$$Balanced\ Field\ Length = Max(S_{Total\ Go}, S_{Stop})$$

3.2.6 Climb Performance

- **Rate of Climb (RoC)**: Calculated for the maximum climb angle and for maximum rate of climb, providing insight into the aircraft's altitude gain over time.

$$ROC = \frac{P_{available} - P_{required}}{W}$$

$$P_{required} = \frac{1}{2} \times \rho \times V^3_{(R/C)_{max}} \times S \times C_D$$

$$V_{R/C_{max}} = \left(\frac{2}{\rho_\infty} \sqrt{\frac{k}{3C_{D_O}}} \frac{W}{S} \right)^{\frac{1}{2}}$$

3.2.7 Ceilings

- **Absolute Ceiling**: The altitude at which the rate of climb reaches zero.
- **Service Ceiling**: The altitude at which the rate of climb is 0.508 m/s (100 feet/min), determined through iterative calculations using altitude-dependent air density.

$$(R/C)_{max} = \frac{\eta \frac{P}{pr}}{W} - \frac{2}{\rho_\infty} \sqrt{\frac{K}{3C_{D,O}}} \left(\frac{W}{S} \right)^{\frac{1}{2}} \times \frac{1.155}{\left(\frac{L}{D} \right)_{max}}$$

3.2.8 Turning Performance

The turning performance calculates:

- **Radius and Rate of Turn** for a **level turn** based on speed, load factor, and gravity.

$$R_{turning} = \frac{V^2}{g \times \sqrt{n^2 - 1}}$$

$$\omega = \frac{V}{R_{turning}}$$

3.2.9 Landing Distance

The landing distance calculation includes touchdown velocity, lift, drag, and ground friction effects.

$$S_{Landing} = \left(\frac{1.69 \times W^2}{g \times \rho \times S \times Cl_{max} \times F_{effective}} \right)$$

Where $F_{effective}$ is the total effective retardation force acting on the aircraft.

$$F_{effective} = D + \mu \times (W - L) + T_{rev}$$

4. Parameters

Density at sea level = 1.225 kg/m^3
Gross mass = 70000 kg
Area = 201 m^2
$C_{L_{max}}$ = 1.6 (Maximum lift coefficient (without flaps))
$C_{L_{max,takeoff}}$ = 2.23
$C_{L_{max,landing}}$ = 3.03
C_{D_0} = 0.025 (Zero-lift drag)
Oswald Efficiency factor = 0.85 m
Propeller efficiency(η) = 0.8
span = 40.4 m
Power = 11065 KW (At Sea Level)
Thrust available = 144.36 KN (At Sea Level)
Rolling friction coefficient(μ_r)=0.4

5. Output Results

A jupyter notebook was made with all the input parameters defined. Then functions to calculate the required parameters were defined. Then all the outputs were printed on the terminal.

Max speed at sea level : 147.76 m/s

Max speed at cruising altitude : 139.23 m/s

Cruise speed: 125.42m/s

Stall speed at sea level and stall speed at cruise altitude: 59.04m/s and 85.07m/s

Total takeoff distance: 622.49m

Turn radius: 925.77

Rate of turn: 7.76°/s

Balanced field length: 798.5m

Max rate of climb at sea level: 10.93m/s

Max rate of climb at cruise altitude: 2.19m/s

Service ceiling: 8541m

Absolute ceiling: 9548m

Landing Distance: 417.59m

6. Conclusion

The analysis conducted on the C-130 Hercules provides a comprehensive understanding of the aircraft's performance characteristics, focusing on various critical aspects such as maximum speed, takeoff and landing distance, climb performance, service ceiling, and turn performance. By combining theoretical principles with computational methods, we were able to model and analyze the behavior of the C-130J Hercules in different flight regimes and at varying altitudes, including both sea level and cruise conditions.

Key Findings

1. Takeoff and Landing Performance:

The takeoff and landing distances were calculated by accounting for factors such as ground roll, rotation, transition, climb-out phase, and obstacle clearance. For the C-130 Hercules, an aircraft primarily designed for tactical airlift and short-field operations, it was critical to estimate these distances accurately. The calculations show that the

C-130 requires a moderate takeoff distance under International Standard Atmosphere (ISA) conditions. With the inclusion of a 50-foot obstacle clearance, the total takeoff distance aligns well with the aircraft's design purpose, enabling it to operate from shorter runways, which is advantageous for military and emergency operations.

2. Maximum Speed:

The maximum speed of the C-130 at both sea level and cruise altitude was determined using the power available and power required methodology. By applying the performance equation, we observed that the maximum speed at sea level is significantly higher due to the denser air, which increases both drag and power output. At cruising altitude, the thinner air reduces both power available and drag, resulting in a slightly lower maximum speed. The speed values derived through these calculations are in line with typical operational speeds for the C-130, reinforcing the robustness of the performance estimation methods used.

3. Climb Performance:

Climb performance is essential for an aircraft like the C-130, which may need to ascend quickly for terrain clearance or operational demands. By calculating the maximum rate of climb and climb angle, we demonstrated that the C-130 has a relatively steep climb angle and a strong rate of climb at sea level. These performance characteristics decrease with altitude, as the power available reduces with decreasing air density. However, even at higher altitudes, the C-130 maintains sufficient climb performance to meet typical mission requirements.

4. Service Ceiling and Absolute Ceiling:

The service ceiling and absolute ceiling of the C-130 were calculated based on the conditions where the rate of climb approaches zero. The service ceiling, where the aircraft can sustain a climb rate of around 100 feet per minute, highlights the operational altitude limit for routine missions, while the absolute ceiling represents the maximum attainable altitude under ideal conditions. The C-130s service ceiling is well-suited for both tactical and strategic operations, allowing it to fly above most weather systems and potential threats in low-altitude environments.

5. Turn Performance:

The turn performance analysis focused on the rate of turn and turn radius. These parameters are particularly relevant in tactical scenarios where maneuverability is essential. Our calculations indicated that the C-130s turning performance is moderate, as expected for a large turboprop aircraft. While not optimized for rapid or tight turns,

the aircraft maintains sufficient maneuverability to perform necessary in-flight maneuvers, especially during low-level flight or when evading potential threats.

Implications and Practical Applications

The performance analysis of the C-130J Hercules provides insights into the aircraft's operational capabilities, limitations, and suitability for various mission profiles. The findings can be valuable in planning and executing missions that require specific performance characteristics, such as short takeoff and landing distances, high rate of climb, and reasonable cruise efficiency. The aircraft's capabilities are well-suited for military applications that demand flexibility and adaptability, especially in austere and semi-prepared environments.

From a design perspective, the C-130s turboprop engines, wing configuration, and structural layout contribute to its balanced performance in both high-speed and low-speed regimes. The aircraft's ability to operate efficiently at a wide range of altitudesand perform necessary maneuvers under tactical conditions makes it a versatile assetin modern military operations.

7. Reference

1. Rolls-Royce, “Allison T56 Turboprop Engine Specifications,” Rolls-Royce Holdings, 2023. [Online]. Available: https://www.rolls-royce.com/products-and-services/defence/aerospace/t56

2. Lockheed Martin, “C-130 Hercules Specifications,” Lockheed Martin Corporation, 2023. [Online]. Available: https://www.lockheedmartin.com/en-us/products/c130

3. J. D. Anderson, Aircraft Performance and Design. New York: McGraw-Hill, 1999.

4. D. P. Coiro, F. Nicolosi, and F. Grasso, “Multi-Component Airfoil Design and Analysis for STOL Ultra-Light Aircraft,” Department of Aerospace Engineering, University of Naples “Federico II,” Italy. [Online]. Available: ResearchGate. Accessed: Nov. 4, 2024

8.Appendix

[12]:
```
import math
```

[13]:
```
# Constants
rho_sea_level = 1.225  # kg/m^3 (density at sea level)
rho_cruise = 0.59       # Approximate air density at cruise altitude (7000 m)
gross_wt = 70000 * 9.81 # Gross Weight
empty_wt = 39000 *9.81 # Empty Weight
S = 201                 # Wing area in m^2
CL_max = 1.6            # Maximum lift coefficient (without flaps)
C_D0 = 0.025            # Zero-lift drag
e = 0.85 # Oswald efficiency factor
b = 40.4 # span in m
AR = b**2/S # Aspect Ratio
k = 1/(math.pi *e * AR )  # Induced drag factor
```

Maximum Speed at Sea Level and Cruising Altitude

[14]:
```
P_max = 4*4637*745.7*0.8    # Available power in watts (example: 1200 kW)
from scipy.optimize import fsolve

def power_balance(V, P_max, weight, rho, S, C_D0, k):
    # Calculate lift coefficient (CL) based on current speed
    C_L = (2 * weight) / (rho * V**2 * S)

    # Calculate drag coefficient (CD) based on CL
    C_D = C_D0 + k * C_L**2

    # Calculate power required (P_req) at the current speed
    P_req = 0.5 * rho * V**3 * C_D * S

    # Return the difference between available power and required power
    return P_max*(rho/rho_sea_level)**0.7 - P_req

def calculate_vmax_fsolve(P_max, weight, rho, S, C_D0, k, initial_guess=100):
    vmax_solution = fsolve(power_balance, initial_guess, args=(P_max, weight,
 ↪rho, S, C_D0, k))
    return vmax_solution[0]
```

```
weight = gross_wt

v_max_sl = calculate_vmax_fsolve(P_max, weight, rho_sea_level, S, C_D0, k)
print("Maximum Speed (Vmax):", v_max_sl, "m/s", v_max_sl*3.6, "km/h")

v_max_cruise = calculate_vmax_fsolve(P_max, weight, rho_cruise, S, C_D0, k)
print("Maximum Speed (Vmax):", v_max_cruise, "m/s", v_max_cruise*3.6, "km/h")
```

```
Maximum Speed (Vmax): 147.4510656306297 m/s 530.823836270267 km/h
Maximum Speed (Vmax): 139.23135901043258 m/s 501.2328924375573 km/h
```

Calculate Cruise Speed

[15]:
```
W = gross_wt

V_cruise = math.sqrt((2 * W) / (rho_cruise * S * math.sqrt(C_D0 / k)))

V_cruise_knots = V_cruise * 1.94384  # 1 m/s  1.94384 knots

# Output Results
print(f"Cruise Speed (V_cruise): {V_cruise:.2f} m/s ({V_cruise_knots:.2f}␣
↪knots)")
```

```
Cruise Speed (V_cruise): 125.42 m/s (243.79 knots)
```

Stall Speed at Sea level and Cruise

[16]:
```
def stall_speed(W, rho, S, CL_max):
    return math.sqrt((2 * W) / (rho * S * CL_max))

V_stall_sea_level = stall_speed(gross_wt, rho_sea_level, S, CL_max)
V_stall_cruise = stall_speed(gross_wt, rho_cruise, S, CL_max)

print("Stall Speed at Sea Level:", V_stall_sea_level, "m/s, ",␣
↪V_stall_sea_level*18/5 , "km/h")
print("Stall Speed at Cruise Altitude:", V_stall_cruise, "m/s, ",␣
↪V_stall_cruise*18/5 , "km/h")
```

```
Stall Speed at Sea Level: 59.04354939319531 m/s,  212.55677781550312 km/h
Stall Speed at Cruise Altitude: 85.07745611549655 m/s,  306.2788420157875 km/h
```

Takeoff

[17]:
```
# Constants (for C-130J Hercules)
W = gross_wt                  # Weight in Newtons
rho = rho_sea_level           # Air density at sea level in kg/m^3
C_L_max = 2.23                # Maximum lift coefficient
T = 144360                    # Total thrust available at takeoff in Newtons
mu = 0.4                      # Rolling friction coefficient
```

```
# Takeoff speed (1.2 times the stalling speed)
V_stall = math.sqrt((2 * W) / (rho * S * C_L_max))
V_TO = 1.2 * V_stall

# Ground Roll Distance
D_ground = 0.5 * rho * V_TO**2 * S * C_D0  # Drag during ground roll
L_ground = 0.5 * rho * V_TO**2 * S * C_L_max  # Lift during ground roll
a_ground = (T - D_ground - mu * (W - L_ground)) / (W / 9.81)  # Net acceleration
S_ground = V_TO**2 / (2 * a_ground)  # Ground roll distance

# Rotation Distance (approx. 10% of ground roll)
S_rotation = 0.1 * S_ground

# Transition Distance
gamma = math.asin((T - D_ground) / W)  # Climb angle
S_transition = 15.24/ math.tan(gamma)  # Distance to clear 50 ft obstacle

# Total Takeoff Distance
S_total_takeoff = S_ground + S_rotation + S_transition

print(f"Ground Roll Distance: {S_ground:.2f} m")
print(f"Rotation Distance: {S_rotation:.2f} m")
print(f"Transition Distance: {S_transition:.2f} m")
print(f"Total Takeoff Distance: {S_total_takeoff:.2f} m")
```

```
Ground Roll Distance: 496.05 m
Rotation Distance: 49.61 m
Transition Distance: 77.03  m
Total Takeoff Distance: 622.69 m
```

Turning Radius

[18]:
```
V = 125.42  # Velocity in m/s
g = 9.81    # Gravity in m/s^2

def turning_radius(V, n):
    return V**2 / (g * math.sqrt(n**2 - 1))

# Assuming a load factor (n)
n = 2
radius = turning_radius(V, n)

print("Turning Radius:", radius, "meters")

print("Turn rate:", (V/radius)*180/math.pi, "degree/s")
```

```
Turning Radius: 925.7718225219469 meters
Turn rate: 7.76221147772126 degree/s
```

Accelerate Go Distance, Accelerate Stop distance, Balanced Field Length

[19]:
```
# Constants (for C-130J Hercules)
W = gross_wt
rho = rho_sea_level
C_L_max = 2.23              # Maximum lift coefficient
T_total = 144360            # Total thrust available at takeoff in Newtons
T_one_engine = T_total * 0.75  # Thrust with one engine inoperative
mu = 0.4                    # Rolling friction coefficient
braking_deceleration = 3.0  # Typical braking deceleration (m/s^2)

# Takeoff and Decision Speed (approximation)
V_stall = math.sqrt((2 * W) / (rho * S * C_L_max))
V_takeoff = 1.2 * V_stall   # Takeoff speed
V1 = 0.95 * V_takeoff       # Approximate decision speed V1

# Accelerate-Go Distance (with one engine out)
D_go = 0.5 * rho * V1**2 * S * C_D0  # Drag with one engine
L_go = 0.5 * rho * V1**2 * S * C_L_max  # Lift with one engine
a_go = (T_one_engine - D_go - mu * (W - L_go)) / (W / 9.81)  # Net acceleration
S_go = V1**2 / (2 * a_go)  # Ground roll with one engine out

# Rotation and Transition Distances (estimated as in takeoff distance)
S_rotation = 0.1 * S_go
gamma = math.asin((T_one_engine - D_go) / W)  # Climb angle with one engine
S_transition = 15.24/ math.tan(gamma)  # Distance to clear 50 ft obstacle

# Total Accelerate-Go Distance
S_total_go = S_go + S_rotation + S_transition

# Accelerate-Stop Distance
S_stop = V1**2 / (2 * braking_deceleration)

# Balanced Field Length (BFL)
BFL = max(S_total_go, S_stop)

print(f"Accelerate-Go Distance (S_go): {S_total_go:.2f} m")
print(f"Accelerate-Stop Distance (S_stop): {S_stop:.2f} m")
print(f"Balanced Field Length (BFL): {BFL:.2f} m")
```

```
Accelerate-Go Distance (S_go): 798.53 m
Accelerate-Stop Distance (S_stop): 541.77 m
Balanced Field Length (BFL): 798.53 m
```

Rate of Climb

[20]:
```
P_sea_level = 4 * 4637 * 745.7 *0.8  # Total power available at sea level (in␣
↪watts)
```

```python
# Environmental conditions
rho_sea_level = 1.225     # Air density at sea level (kg/m^3)
rho_cruise = 0.59         # Air density at cruise altitude (20,000 ft, approx.)
n = 0.7                   # Power reduction exponent for turboprop engines

# Calculate power available at cruise altitude
P_cruise = P_sea_level * (rho_cruise / rho_sea_level) ** n

# Function to calculate power required at a given velocity and altitude
def calculate_power_required(V, W, S, C_D0, k, rho):
    # Calculate Lift Coefficient (C_L) to maintain level flight
    C_L = W / (0.5 * rho * V**2 * S)
    # Calculate Drag
    D = 0.5 * rho * V**2 * S * (C_D0 + k * C_L**2)
    # Calculate Power Required
    P_required = D * V
    return P_required

# Function to calculate the maximum rate of climb
def calculate_max_rate_of_climb(P_available, W, S, C_D0, k, rho):
    # Estimate the velocity for maximum rate of climb (V_best_climb)
    V_best_climb = math.sqrt((2 * W) / (rho * S) * math.sqrt(k /(3* C_D0)))
    # Calculate Power Required at V_best_climb
    P_required = calculate_power_required(V_best_climb, W, S, C_D0, k, rho)
    # Calculate Rate of Climb
    max_rate_of_climb = (P_available - P_required) / W
    return max_rate_of_climb, V_best_climb

# Calculate maximum rate of climb at sea level
max_rc_sea_level, V_best_climb_sea_level = calculate_max_rate_of_climb(P_sea_level, W, S, C_D0, k, rho_sea_level)

# Calculate maximum rate of climb at cruise altitude
max_rc_cruise, V_best_climb_cruise = calculate_max_rate_of_climb(P_cruise, W, S, C_D0, k, rho_cruise)

# Convert rate of climb from m/s to ft/min (1 m/s = 196.85 ft/min)
max_rc_sea_level_ft_min = max_rc_sea_level * 196.85
max_rc_cruise_ft_min = max_rc_cruise * 196.85

# Output results
print(f"Maximum Rate of Climb at Sea Level: {max_rc_sea_level:.2f} m/s ({max_rc_sea_level_ft_min:.2f} ft/min)")
print(f"Best Climb Speed at Sea Level: {V_best_climb_sea_level:.2f} m/s")
```

```
print(f"Maximum Rate of Climb at Cruise Altitude: {max_rc_cruise:.2f} m/s
↪({max_rc_cruise_ft_min:.2f} ft/min)")
print(f"Best Climb Speed at Cruise Altitude: {V_best_climb_cruise:.2f} m/s")
```

Maximum Rate of Climb at Sea Level: 10.93 m/s (2151.03 ft/min)
Best Climb Speed at Sea Level: 66.14 m/s
Maximum Rate of Climb at Cruise Altitude: 2.19 m/s (431.05 ft/min)
Best Climb Speed at Cruise Altitude: 95.30 m/s

Ceiling

[21]:
```
P_ser=(0.508+2/(rho_sea_level)*math.sqrt(k/(3*C_D0))*(W/S)**0.5*1.155/14.72)*W/
↪0.8
r_ser=(P_ser/P_sea_level)**(1/0.7)

import math

def calculate_altitude_from_density(density, sea_level_density=1.225,
↪scale_height=8500):
    if density >= sea_level_density:
        return 0  # At or below sea level

    # Calculate altitude based on density
    altitude = -scale_height * math.log(density / sea_level_density)
    return altitude

# Example usage
density_ser = r_ser*1.225  # Air density in kg/m^3 (example value)
altitude_ser = calculate_altitude_from_density(density_ser)
print(f"Service ceiling: {altitude_ser:.2f} meters")

P_abs=(2/(rho_sea_level)*math.sqrt(k/(3*C_D0))*(W/S)**0.5*1.155/14.72)*W/0.8
r_abs=(P_abs/P_sea_level)**(1/0.7)
density_abs = r_abs*1.225  # Air density in kg/m^3 (example value)
altitude_abs = calculate_altitude_from_density(density_abs)
print(f"Absolute ceiling: {altitude_abs:.2f} meters")
```

Service ceiling: 8541.26 meters
Absolute ceiling: 9548.86 meters

Landing Distance

[22]:
```
def calculate_landing_distance(weight, air_density, wing_area,
↪max_lift_coefficient, zero_lift_drag_coefficient, wingspan,
↪oswald_efficiency, friction_coefficient, thrust_reversal=0):

    # Constants
    g = 9.81  # Gravity acceleration in m/s^2
```

```
    # Calculate stall speed (V_stall)
    stall_speed = math.sqrt((2 * weight) / (air_density * wing_area *␣
↪max_lift_coefficient))

    # Calculate touchdown velocity (V_T = 1.3 * V_stall)
    touchdown_velocity = 1.3 * stall_speed

    # Calculate lift coefficient (C_L) at touchdown velocity
    lift_coefficient = (2 * weight) / (air_density * touchdown_velocity ** 2 *␣
↪wing_area)

    # Calculate the aspect ratio (AR) of the wing
    aspect_ratio = wingspan ** 2 / wing_area

    # Calculate the drag coefficient (C_D) including zero-lift drag and␣
↪lift-induced drag
    drag_coefficient = zero_lift_drag_coefficient + (lift_coefficient ** 2) /␣
↪(math.pi * oswald_efficiency * aspect_ratio)

    # Calculate drag force (D) at touchdown velocity using the drag coefficient
    drag = 0.5 * air_density * touchdown_velocity ** 2 * drag_coefficient *␣
↪wing_area

    # Calculate lift force (L) at touchdown velocity
    lift = 0.5 * air_density * touchdown_velocity ** 2 * wing_area * lift_coeff

    # Effective force (F_eff) during landing including drag, friction, and␣
↪thrust reversal if applicable
    effective_force = (drag + friction_coefficient * (weight-lift) +␣
↪thrust_reversal)

    # Calculate landing distance (s_L) using the given formula
    landing_distance = (1.69 * weight ** 2) / (g * air_density * wing_area *␣
↪max_lift_coefficient * effective_force)

    return landing_distance

# Example usage
lift_coeff=0.9 #Average lift coefficient
max_lift_coefficient = 3.03 # Typical maximum lift coefficient for C-130J
friction_coefficient = 0.4  # Typical coefficient of dynamic friction for a dry␣
↪runway
thrust_reversal = 57600 # Example thrust reversal force in newtons (negative as␣
↪it acts against motion)
```

```
landing_distance = calculate_landing_distance(gross_wt, rho_sea_level, S,␣
↪max_lift_coefficient, C_D0, b, e, friction_coefficient, thrust_reversal)
print(f"Estimated landing distance: {landing_distance:.2f} meters")
```

```
Estimated landing distance: 417.59 meters
```