In [ ]:
```
from avipy import qty
import math

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

Individual assignment on Takeoff Weight Limitations

NOTE

If you want to run this yourself, you need to install: pandas, numpy and avipy. Numpy and Pandas are popular Python libraries that make working with data easier. AviPy is a library that I built myself purely for the purposes of FOE. This library contains a 'qty' module that takes care of unit conversions for commonly used units in aviation. The library uses SI units internally and converts it automatically to base unit, and to other units on demand. For example, If a value in knots is given, I can convert this to a `Velocity` type, by declaring `speed = qty.Velocity.Kts(120)`. The library will then automatically convert this unit to meters per second. When I use the speed variable later in the code, it is a value in meters per second.

To install these modules:

```
$ pip install pandas numpy avipy
```

Assignment

Make a TL (take-off weight limitation) table for a B777-300ER commercial aircraft for Hong Kong's Chek Lap Kok International Airport runway 25C with the environmental condition of 17° Celsius and no wind. The B777-300ER has two certified take-off flap settings of 5 and 15, 15 degrees will be used for the calculations. The aircraft is equipped with GE90-115B engines that deliver a thrust of 104,500 lbs per engine at sea level. The engine is flat rated up to $ISA + 15°$. Above $ISA + 15°$ the thrust decreases by 0.4% per degree. Explain and make a conclusion of your results!

Runway

- ASDA and TODA are 2900 m
- Elevation 0 ft
- No runway slope
- Obstacle 1600 m after end of the runway at 103 ft
- Coefficient of friction, $\mu = 0.027$

Aircraft

- MTOW, $m = 351534kg$
- wingspan, $b = 64.8m$
- wing area, $S = 436.8m^2$
- oswald factor, $e = 0.77$
- drag coefficient at flap 15, $C_{D_0} = 0.068$
- Stall speed at MTOW with flaps 15, $v_{stall} = 154.5kts$
- $v_2 = 1.18 \cdot v_{stall}$
- $v_{lof} = 1.15v_{stall}$
- Coefficient of friction with full brakes, $\mu = 0.56$

Assumptions

- During climb to 1500 ft, air pressure remains constant to ISA at sea level.
- Engine thrust is constant during takeoff.
- N-1 situation arises at lift-off.
- For ASDR, no need to consider pilot's reaction time => $v_1 == v_{lof}$.
- At v_{lof}, the ground run instantly changes into the take-off arc.

In [ ]:
```
toda = qty.Distance(2900)
asda = qty.Distance(2900)
height_b = qty.Distance.Ft(35)
mu = 0.027
mu_brake = 0.56
temp = qty.Temperature.Celsius(17)
```

```
pressure = qty.Pressure.Hpa(1013.25)
obstacle_distance = qty.Distance(1600)
obstacle_height = qty.Distance.Ft(103)

mtom = qty.Mass(351534)
mtow = qty.Force.Kg(mtom)
wingspan = qty.Distance(64.8)
surface = qty.Area(436.8)
oswald = 0.77
cd_0 = 0.068
v_stall_mtow = qty.Velocity.Kts(154)

v_stall_to_lof = 1.15
v_stall_to_v2 = 1.18

stall_speeds: list[tuple[qty.Mass, qty.Velocity]] = [
    (mtom, v_stall_mtow),
    (qty.Mass.M_ton(350), qty.Velocity.Kts(153.5)),
    (qty.Mass.M_ton(345), qty.Velocity.Kts(151.5)),
    (qty.Mass.M_ton(340), qty.Velocity.Kts(149)),
    (qty.Mass.M_ton(335), qty.Velocity.Kts(147)),
    (qty.Mass.M_ton(330), qty.Velocity.Kts(145)),
    (qty.Mass.M_ton(325), qty.Velocity.Kts(142.5)),
    (qty.Mass.M_ton(320), qty.Velocity.Kts(140)),
    (qty.Mass.M_ton(315), qty.Velocity.Kts(137.5)),
    (qty.Mass.M_ton(310), qty.Velocity.Kts(135)),
    (qty.Mass.M_ton(305), qty.Velocity.Kts(132.5)),
    (qty.Mass.M_ton(300), qty.Velocity.Kts(130)),
    (qty.Mass.M_ton(295), qty.Velocity.Kts(127)),
    (qty.Mass.M_ton(290), qty.Velocity.Kts(124.5)),
    (qty.Mass.M_ton(285), qty.Velocity.Kts(121.5)),
    (qty.Mass.M_ton(280), qty.Velocity.Kts(118.5)),
    (qty.Mass.M_ton(275), qty.Velocity.Kts(115.5)),
    (qty.Mass.M_ton(270), qty.Velocity.Kts(112.5)),
    (qty.Mass.M_ton(265), qty.Velocity.Kts(109.5)),
    (qty.Mass.M_ton(260), qty.Velocity.Kts(106)),
]
```

Constants

Before any distance is calculated, some constant values for this scenario will be calculated.

1. Density with the given temperature and constant ISA pressure

$$\rho = \frac{P}{R \cdot T}$$

2. Thrust for one engine and with both

The engines are flat-rated up to $ISA + 15°C$, since the temperature in this scenario is $17°C$, the thrust does not decrease. The thrust values in LBS are converted to Newton using the qty module.

3. Aspect ratio and $C_{L_{max}}$

$$AR = \frac{b^2}{S}$$

The $C_{L_{max}}$ can be obtained using the lift equation with the MTOW and stall speed at MTOW:

$$C_{L_{max}} = \frac{2 \cdot W}{\rho \cdot v_{stall}^2 \cdot S}$$

In [ ]:
```
# 1. Density
density = pressure / (287 * temp)

# 2. Thrust
thrust_per_engine = qty.Force.Kg(qty.Mass.Lbs(104500))
thrust = qty.Force(thrust_per_engine * 2)

# 3. Aspect ratio and CL_max
aspect_ratio = wingspan**2 / surface
cl_max = (2 * mtow) / (density * surface * v_stall_mtow**2)

cl_max
```

Out[ ]:
```
2.0675363973645564
```

Takeoff Ground Run

To calculate the take-off run distance, assuming the acceleration is not constant, three steps are taken:

1. Calculate the optimal values of C_L and C_D for the minimum run distance

$C_L = \frac{1}{2} \cdot \mu \cdot \pi \cdot AR \cdot e$

$C_D = C_{D_0} + \frac{C_L^2}{\pi \cdot AR \cdot e}$

2. Calculate the aerodynamic factor and d factor needed in the ground run equation

$e = \frac{\rho \cdot S}{2 \cdot W} \cdot (C_D - \mu \cdot C_L)$

$d = \frac{T}{W} - \mu$

3. Calculate the ground run distance provided the acceleration is not constant

$S_A = \frac{1}{2 \cdot g \cdot e} \cdot \ln \frac{d}{d - e \cdot v_{LOF}^2}$

In [ ]:
```python
def get_gnd_run(weight: qty.Force, v_stall: qty.Velocity) -> qty.Distance:
    # 1. Get the optimal C_L and C_D values
    cl_run = 0.5 * mu * math.pi * aspect_ratio * oswald
    cd_run = cd_0 + cl_run**2 / (math.pi * aspect_ratio * oswald)

    # 2. Calculate aero-factor and d-factor
    aero_factor = (density * surface) / (2 * weight) * (cd_run - mu * cl_run)
    d_factor = thrust / weight - mu

    # 3. Get the ground run distance
    v_lof = v_stall_to_lof * v_stall

    ground_run = 1 / (2 * 9.81 * aero_factor) * math.log(d_factor / (d_factor - aero_factor * v_lof**2))

    # Return the ground run as a distance for the given weight and stall speed
    return qty.Distance(ground_run)
```

Takeoff Air Run

At lift-off speed, an N - 1 situation arises. Which means the available thrust will be that of one engine. This value will be used in the following calculations.

The takeoff arc distance is calculated in multiple steps:

1. Calculate the C_L and C_D at point C

At point C, the end of the arc, the aircraft will be flying at v_2 speed. The lift coefficient in this configuration can be obtained by

$$C_{L_C} = \frac{2 \cdot W}{\rho \cdot S \cdot v_2^2}$$

The C_D in this configuration is obtained by

$$C_{D_C} = C_{D_0} + \frac{C_{L_C}^2}{\pi \cdot AR \cdot e}$$

2. Calculate the drag at point C

The drag is obtained by

$$D_C = W \cdot \frac{C_{D_C}}{C_{L_C}}$$

3. Calculate the climb gradient at point C

While in the given configuration, the climb angle is obtained by:

$$\sin \gamma_C = \frac{T - D_C}{W}$$

Then the gradient can be obtained using this angle. This gradient will not be used in this calculation, but will be used later.

$$g_C = \tan \gamma_C$$

4. Calculate the lift and drag coefficients at liftoff

By equalling the lift at stall speed and the lift at take-off speed, it is found that:

$$C_{L_{LOF}} = \frac{C_{L_{max}}}{ratio^2}$$

where $ratio$ is equal to the ratio between the liftoff speed and the stall speed, 1.15 in this scenario. Then the drag coefficient is obtained using

$$C_{D_{LOF}} = C_{D_0} + \frac{{CL_{LOF}}^2}{\pi \cdot AR \cdot e}$$

5. Calculate the average drag during the takeoff arc

The drag at liftoff is obtained using the drag equation

$$D_{LOF} = \frac{1}{2} \cdot \rho \cdot v_{LOF}^2 \cdot S \cdot C_{D_{LOF}}$$

Then the average drag is obtained by

$$D_{avg} = \frac{D_{LOF} + D_C}{2}$$

6. Calculate the air run distance

All known values can now be substituted into the energy equation to solve for the air run distance:

$$\frac{W}{2 \cdot g} \cdot (v_C^2 - v_{LOF}^2) = (T - D_{avg}) \cdot S_B - W \cdot h_B$$

This becomes

$$S_B = \frac{\frac{W}{2 \cdot g} \cdot (v_C^2 - v_{LOF}^2) + W \cdot h_B}{T - D_{avg}}$$

In [ ]:
```python
def get_air_run(weight: qty.Force, v_stall: qty.Velocity) -> tuple[qty.Distance, float]:
    # Define speeds for given v_stall
    v_lof = v_stall * v_stall_to_lof
    v_2 = v_stall * v_stall_to_v2

    # 1. Get C_L and C_D at point C
    cl_c = (2 * weight) / (density * surface * v_2**2)
    cd_c = cd_0 + cl_c**2 / (math.pi * aspect_ratio * oswald)
```

```
    # 2. Calculate the drag at point C
    drag_c = weight * (cd_c / cl_c)

    # 3. Calculate the climb gradient at point C
    angle_c = math.asin((thrust_per_engine - drag_c) / weight)
    gradient_c = math.tan(angle_c)

    # 4. Calculate the lift coefficent at liftoff
    cl_lof = cl_max / v_stall_to_lof**2
    cd_lof = cd_0 + cl_lof**2 / (math.pi * aspect_ratio * oswald)

    # 5. Calculate the average drag in the takeoff arc
    drag_lof = 0.5 * density * v_lof**2 * surface * cd_lof
    drag_avg = (drag_lof + drag_c) / 2

    # 6. Calculate the air run distance
    v_c = v_stall * v_stall_to_v2
    air_run = (weight / (2 * 9.81) * (v_c**2 - v_lof**2) + weight * height_b
        ) / (thrust_per_engine - drag_avg)

    # Return the air run distance for the given weight and stall speed
    return qty.Distance(air_run), gradient_c
```

Obstacle Clearance

The clearance between the net flight path of the aircraft and the obstacle needs to be 35 ft. To determine whether the aircraft actually meets this clearance with a certain weight, multiple steps are taken:

1. Calculate the climb gradient, g_C, from a given climb angle γ_C and subtract 0.8% to get the net flight path

$g_{net} = \tan \gamma_C - 0.008$

2. Calculate the climb angle of the climb gradient of the net takeoff flight path

$\gamma_{net} = \arctan g_{net}$

3. Calculate the vertical distance travelled during the climb

$\tan \gamma_C = \frac{hor}{ver} \rightarrow ver = hor \cdot \tan \gamma_C$

4. Calculate the clearance at the obstacle

The clearance at the obstacle is obtained by adding the vertical distance travelled during the climb and the height after the takeoff arc, h_b. This is the height of the net flight path at the obstacle. The height of the obstacle is then subtracted from the height of the net flight path to obtain the clearance of the net flight path to the obstacle. This obtained clearance must be greater than 35 ft.

In [ ]:
```python
def get_obstacle_clearance(residual_rwy_dist: qty.Distance, gradient_c: float) -> qty.Distance:
    # 1. Calculate the gradient of the net flight path
    net_gradient_c = gradient_c - 0.008

    # 2. Calculate the net flight path angle
    net_angle_c = math.atan(net_gradient_c)

    # 3. Calculate the vertical distance travelled in the climb
    climb_height = (obstacle_distance + residual_rwy_dist) * math.tan(net_angle_c)

    # 4. Calculate the clearance of the aircaft to the obstacle
    clearance = climb_height + height_b - obstacle_height
    return qty.Distance(clearance)
```

Stop distance at V_1

To calculate the stop distance when the aircaft is at v_1 speed, multiple steps are taken.

1. Calculate the C_L and C_D values at v_1 speed

For this, the C_L and C_D $ values for the aircraft on ground are used. These are the same as the ones used for the ground run.

$C_{L_{run}} = \frac{1}{2} \cdot \mu \cdot pi \cdot AR \cdot e$

$C_{D_{run}} = C_{D_0} + \frac{C_{Lrun}^2}{\pi \cdot AR \cdot e}$

2. Calculate the aero factor and d-factor for the stop run

$$e = \frac{\rho \cdot S}{2 \cdot W} \cdot (C_{D_{run}} - \mu_{brake} \cdot CL_{run})$$

$$d = -\mu_{brake}$$

3. Calculate the stopping distance using the obtained values

$$S_u = \frac{1}{2 \cdot g \cdot e} \cdot \ln \frac{d - e \cdot TAS_{LOF}^2}{d}$$

In [ ]:
```python
def get_stop_dist(weight, v_stall):
    # 1. Calculate C_L and C_D
    cl_run = 0.5 * mu * math.pi * aspect_ratio * oswald
    cd_run = cd_0 + cl_run**2 / (math.pi * aspect_ratio * oswald)

    # 2. Calculate the aero-factor and the d-factor during the stop run
    aero_factor = (density * surface) / (2 * weight)  * (cd_run - mu_brake * cl_run)
    d_factor = - mu_brake

    # 3. Calculate the stopping distance
    v_lof = v_stall * v_stall_to_lof
    stop_dist = 1 / (2 * 9.81 * aero_factor) * math.log((d_factor - aero_factor * v_lof**2) / d_factor)

    return qty.Distance(stop_dist)
```

Calculate values for every mass and stall speed

Now that functions have been defined to calculate the ground run, air run, stopping distance, obstacle clearance and the climb gradient, these values can be calculated for every mass from 260 tons to the structural MTOM.

This process is done in the following steps:

1. Define an empty dictionary to store the values in

2. Iterate over all defined mass and stall speed value pairs

3. Calculate the values for ground run, air run, takeoff distance, stopping distance, acceleration stop distance using the earlier defined functions

4. Calculate any residual runway distance after 35ft has been reached, with that calculate the clearance over the obstacle

5. Store all calculated values for the mass, stall speed pair in the dictionary

6. Convert the dictionary to a pandas dataframe for further analysis

In [ ]:
```python
# 1. Define empty dictionary for values
distances = {
    "mass": [],
    "tod": [],
    "asd": [],
    "clearance": [],
    "gradient": [],
}

# 2. Iterate over mass, stall speed value pairs
for mass, v_stall in stall_speeds:
    weight = qty.Force.Kg(mass)

    # 3. Calculate the distances
    ground_run = get_gnd_run(weight, v_stall)
    air_run, gradient_c = get_air_run(weight, v_stall)
    takeoff_dist = ground_run + air_run
    stop_dist = get_stop_dist(weight, v_stall)
    accel_stop_dist = ground_run + stop_dist

    # 4. Calculate the obstacle clearance
    residual_rwy_dist = toda - takeoff_dist
    obstacle_clearance = get_obstacle_clearance(residual_rwy_dist, gradient_c)

    # 5. Store the values in the dictionary
    distances["mass"].append(mass.base)
    distances["tod"].append(takeoff_dist)
    distances["asd"].append(accel_stop_dist)
    distances["clearance"].append(obstacle_clearance.ft)
    distances["gradient"].append(gradient_c)

# 6. Convert the dictionary to a pandas dataframe
df = pd.DataFrame(distances)
```

```
df
```

Out[ ]:

	mass	tod	asd	clearance	gradient
0	351534	3248.453926	2712.702386	2.619021	0.025198
1	350000	3183.432370	2684.527561	8.675637	0.025751
2	345000	2963.498103	2580.528234	30.257330	0.027492
3	340000	2740.557559	2461.676622	53.965190	0.029129
4	335000	2566.627475	2364.249497	77.507059	0.030939
5	330000	2408.314776	2269.734504	102.144657	0.032793
6	325000	2243.123078	2161.726231	128.288400	0.034510
7	320000	2092.521544	2057.524025	155.068218	0.036242
8	315000	1954.319995	1957.026086	182.475653	0.037990
9	310000	1826.817665	1860.133597	210.497136	0.039755
10	305000	1708.669325	1766.750591	239.115370	0.041536
11	300000	1598.793236	1676.783821	268.310199	0.043333
12	295000	1484.045366	1576.965696	296.338312	0.044821
13	290000	1389.026411	1494.031381	326.202850	0.046622
14	285000	1289.364116	1402.048045	353.824926	0.048046
15	280000	1196.592627	1314.070073	380.775246	0.049408
16	275000	1110.101764	1229.978619	406.884460	0.050699
17	270000	1029.372937	1149.658374	431.961421	0.051908
18	265000	953.960372	1072.997380	455.789862	0.053022
19	260000	876.150316	990.135951	471.392842	0.053368

The defined dataframe can then be used to obtain the limiting weight in each category. For this, multiple steps are again taken.

A function is defined to interpolate the mass where the target value for the category is reached. This function takes the following steps:

1. Get the rows in the dataframe that are closest to the target value

This is done by either selecting the biggest value that's bigger than the target value, and the smallest value that's lower than the target value. Or vice-versa, depending on whether the value needs to be bigger or smaller than the target value (TOD should be smaller than TODA, while gradient should be bigger than the required climb gradient). If there is no upper value, than the maximum value is the structural MTOM, this is thus returned in this case.

2. Calculate the coefficients of a linear function between the two rows in the dataframe

3. Interpolate the mass using the linear equation.

Using the function the mass is interpolated where:

- The TOD is lower than or equal to the TODA
- The ASD is lower than or equal to the ASDA
- The clearance height is higher than or equal to equal to 35 ft
- The Climb gradient is higher than or equal to 2.4%

In [ ]:
```
def get_mass_target(df: pd.DataFrame, target: str, target_value: float, goal: str):
    """
    Interpolates the mass between two points of the dataframe where the required target value is in between

    Parameters
    ----------
    df: DataFrame
        The dataframe of required values
    target: str
        The category that has to be calculated: tod, asd, clearance, gradient
    target_value: str
        The value of the target category
    goal:
        Where the value has to be higher or lower than the target value

    Returns
    -------
```

```
    The interpolated mass where the mass results in the target value
    """
    # 1. Get closest rows in dataframe
    if goal == 'lower':
        upper_index = df.index[df[target] > target_value].max()
        lower_index = df.index[df[target] < target_value].min()
    elif goal == 'higher':
        upper_index = df.index[df[target] < target_value].max()
        lower_index = df.index[df[target] > target_value].min()

    if np.isnan(upper_index):
        return mtom.base

    upper = df.iloc[upper_index]
    lower = df.iloc[lower_index]

    # 2. Calculate linear coefficients
    mass1, target1 = lower['mass'], lower[target]
    mass2, target2 = upper['mass'], upper[target]

    line_slope = (target2 - target1) / (mass2 - mass1)
    line_y_int = target2 - line_slope * mass2

    # 3. Interpolate the mass value where the target value is reached
    mass_target = (target_value - line_y_int) / line_slope

    return mass_target

tod_mass = get_mass_target(df, 'tod', 2900, 'lower')
asd_mass = get_mass_target(df, 'asd', 2900, 'lower')
clearance_mass = get_mass_target(df, 'clearance', 35, 'higher')
gradient_mass = get_mass_target(df, 'gradient', 0.024, 'higher')

tod_mass, asd_mass, clearance_mass, gradient_mass
```

Out[ ]: (343575.8960195389, 351534, 343999.7684655137, 351534)