AE 616 Gas Dynamics HW-2 Group 10

```
# Question 1
import numpy as np

def find_shock_location(γ, inlet_mach, duct_length, duct_diameter,Friction_factor,outlet_mach):

        if inlet_mach <= 1:
           return None  # No shock for subsonic or sonic inlet conditions

        # Calculate Mach number downstream of the shock
        for mach_at_shock in np.arange(1.01,inlet_mach,0.00001):
            mach_after_shock = float(np.sqrt(((2/(γ-1))+(mach_at_shock**2))/((2*γ*(mach_at_shock**2)/(γ-1))-1)))
        # Calculate duct lengths upstream and downstream of the shock
            upstream_length = (duct_diameter /(4*Friction_factor)) *(((1/γ)*((1 /(inlet_mach**2))-(1 /(mach_at_shock**2))))+(((γ+1)/(2*γ
            downstream_length = (duct_diameter /(4*Friction_factor)) *(((1/γ)*((1 /(mach_after_shock**2))-(1 /(outlet_mach**2))))+(((γ+1
            #upstream_length = (duct_diameter /(4*Friction_factor)) *(((1/γ)*((1 /(inlet_mach**2))-(1 /(mach_at_shock**2))))+((np.log(((
            #downstream_length = (duct_diameter /(4*Friction_factor)) *(((1/γ)*((1 /(mach_after_shock**2))-(1 /(outlet_mach**2))))+((np.

        # Update shock location based on error
            error = upstream_length + downstream_length - duct_length
        # Check for convergence
            if (abs(upstream_length + downstream_length - duct_length)<1e-03) and (upstream_length < duct_length):
               return upstream_length,downstream_length,mach_at_shock

γ = float(input("γ = "))
inlet_mach = float(input("inlet_mach = "))
duct_length = float(input("duct_length (m)= "))
duct_diameter = float(input("duct_diameter (m)= "))
Friction_factor = float(input("Friction_factor = "))
outlet_mach = float(input("outlet_mach = "))

shock_location = find_shock_location(γ, inlet_mach, duct_length, duct_diameter,Friction_factor,outlet_mach)

if shock_location is not None:
   print("Shock location:", shock_location[0])
   print("mach_at_shock:", shock_location[2])
else:
   print("No solution found.")

def No_shock_duct_length(γ, inlet_mach, duct_diameter,Friction_factor,outlet_mach):
    duct_length = (duct_diameter /(4*Friction_factor)) *(((1/γ)*((1 /(inlet_mach**2))-(1 /(outlet_mach**2))))+(((γ+1)/(2*γ))*(np.log(((1
    return duct_length

print(" ")

duct_length = No_shock_duct_length(γ, inlet_mach, duct_diameter,Friction_factor,outlet_mach)
print("duct_length when no normal shock occurs in the flow :",duct_length)
```

```
γ = 1.3
inlet_mach = 2
duct_length (m)= 10.8
duct_diameter (m)= 0.3
Friction_factor = 0.003
outlet_mach = 1
Shock location: 5.351122465628642
mach_at_shock: 1.4690500000030073

duct_length when no normal shock occurs in the flow : 8.931934142052128
```

```
# Cell 2, for installing pygasflow
# Need to be executed before executing Question 2 and Question 3
!pip install pygasflow
```

Show hidden output

```
# Question 2
# Note: Every time a kernel restarts in Google Colab, external libraries (pygasflow) should reinstalled.
# For installing cell 2 need to be excuted
import numpy as np
# pygasflow python package, Reference: https://pypi.org/project/pygasflow/
from pygasflow import shockwave_solver

γ = float(input("Enter gamma γ:",))
P1 = float(input("Enter P1(kPa):",))

# Create normal(M) function
# which returns β(weak solution), Upstream Normal component of M1 i.e. Mn1, Downstream Normal component of M i.e. Mn2 and Downstream Mach
# Pressure ratio across shock
```

```
# d = theta
def normal(M, d):

    # Using pygasflow python package to obtain β(weak solution) with Mach number(m1) and theta(flow deflection d) as input.
    result = shockwave_solver("m1", M, 'theta', d)
    M1, MN1, M2, MN2, beta, theta, pr, dr, tr, tpr = map(lambda x: round(x,4),result)
    beta_weak = beta

    Mn_1 = M*np.sin(np.radians(beta_weak))
    P_r = 1 + (2*γ / (γ+1)) * (Mn_1**2 - 1)
    Mn_2 = np.sqrt((2 + (γ-1)*Mn_1**2) / ((2*γ*Mn_1**2) - (γ-1)))
    M_2 = Mn_2/np.sin(np.radians(beta_weak-d))
    return beta_weak, d, Mn_1, P_r, Mn_2, M_2

# Iterate over a range of deflections
def slip_angle():
  for i in np.arange(-d_31, d_21, 0.1):
    # where d_31 = lower deflection  θ and d_21 = upper deflection  θ
    # "i" should be in a range(), such 0 <= d_42,d_43 <= 90. i.e. (0 <= theta <= 90)
    d_42 = d_21 - i  # Flow deflection for M2
    d_43 = d_31 + i  # Flow deflection for M3

    # Calling normal(M, theta) function to obtain flow parameter between 4 and 2
    b_42, d42, Mn_2, P_r_42, Mn_4, M_4 = map(lambda x:round(x,3), normal(M_2, d_42))

    # Calling normal(M, d) function to obtain flow parameter between 4 and 3
    b_43, d43, Mn_3, P_r_43, Mn_4, M_4 = map(lambda x:round(x,3), normal(M_3, d_43))

    # Check the condition for matching pressure ratios
    # return solution when difference between P42/P1 and P43/P1 = 0.002
    # P42/P1 = P4/P2 * P2/P1* P1 = P_r_42 * P_r_21 * P1
    # P43/P1 = P4/P3 * P3/P1* P1 = P_r_43 * P_r_31 * P1
    if np.isclose(P_r_42 * P_r_21 * P1, P_r_43 * P_r_31 * P1, 0.001):
        return f"Flow deflection after shock interaction \u0394: {i:.1f} | P42: {P_r_42 * P_r_21 * P1:.2f} kPa | P43: {P_r_43 * P_r_31 *
        break

# Flow parameters between 2 and 1
M_1 = float(input("Enter upstream, M1:",))
d_21 = float(input("Upper Flow deflection angle, θ:",)) # theta(θ)
b, d, Mn_1, P_r_21, Mn_2, M_2 = map(lambda x:round(x,3), normal(M_1, d_21))
print("β:", b, "θ:", d, "Mn1:", Mn_1, "P2/P1:", P_r_21, "Mn2:", Mn_2, "M2:", M_2)

# Flow parameters between 3 and 1
d_31= float(input("Lower Flow deflection angle, θ:",)) # theta(θ)
b, d, Mn_1, P_r_31, Mn_3, M_3 = map(lambda x:round(x,3), normal(M_1, d_31))
print("β:", b, "θ:", d, "Mn1:", Mn_1, "P3/P1:", P_r_31, "Mn3:", Mn_3, "M3:",M_3)

# Call Slipe angle function
print(slip_angle())
```

```
Enter gamma γ:1.4
Enter P1(kPa):30
Enter upstream, M1:3
Upper Flow deflection angle, θ:4
β: 22.354 θ: 4.0 Mn1: 1.141 P2/P1: 1.352 Mn2: 0.881 M2: 2.799
Lower Flow deflection angle, θ:3
β: 21.599 θ: 3.0 Mn1: 1.104 P3/P1: 1.256 Mn3: 0.908 M3: 2.848
Flow deflection after shock interaction Δ: 1.0 | P42: 50.29 kPa | P43: 50.30 kPa
```

```
# Question 3, Anderson Example 4.15
# Note: Every time a kernel restarts in Google Colab, external libraries(pygasflow) should reinstalled.
# For installing cell 2 need to be excuted
import numpy as np
# pygasflow python package, Reference: https://pypi.org/project/pygasflow/
from pygasflow import shockwave_solver

gamma = 1.4
a = float(input("Enter angle of attack: "))

# Stagnation ratio
def stag_Ratio(M):
    P0_r = ((1 + ((gamma - 1) / 2) * M**2)) ** (gamma / (gamma - 1))
    return P0_r

# Prandtl-Meyer Function
def pm_function(M):
    v1 = (np.sqrt((gamma + 1) / (gamma - 1))) * np.arctan(np.sqrt(((gamma - 1) / (gamma + 1)) * (M**2 - 1))) - np.arctan(np.sqrt(M**2 - 1
    return v1  # radians

# Ratios T2/T1, P2/P1, P01/P1 and P02/P1 for isentropic flow
# Used to obtain ratios across exapansion fan
def Ratio(M_1, M_2):
```

```
    T_r = (1 + ((gamma - 1) / 2) * M_1**2) / (1 + ((gamma - 1) / 2) * M_2**2)
    P_r = ((1 + ((gamma - 1) / 2) * M_1**2) / (1 + ((gamma - 1) / 2) * M_2**2)) ** (gamma / (gamma - 1))
    P01_r = stag_Ratio(M_1)
    P02_r = stag_Ratio(M_2)
    return T_r, P_r, P01_r, P02_r

# Mach Number behind expansion wave
def Mach(v2):
    for i in np.arange(1, 50, 0.01):
        v_2 = np.degrees(np.round(pm_function(i), 3))
        if np.isclose(v_2, v2, 0.01):
            return np.round(i, 2)

# Create normal(M) function
# which returns Upstream Normal component of M1 i.e. Mn1, Downstream Normal component of M i.e. Mn2 and Downstream Mach Number M2
# Pressure rise across oblique shock
def Normal(M_1, a):

    # Using pygasflow python package to obtain β(weak solution) with Mach number(m1) and theta(flow deflection d) as input.
    # shockwave_solver gives flow parameters across a oblqiue shock wave
    result = shockwave_solver('m1', M_1, 'theta', a)
    M1, MN1, M2, MN2, beta, theta, pr, dr, tr, tpr = map(lambda x: round(x,4), result)
    b_w = beta
    Mn_1 = M_1 * np.sin(np.radians(b_w))
    Mn_3 = np.sqrt((2 + (gamma - 1) * Mn_1**2) / (2 * gamma * Mn_1**2 - (gamma - 1)))
    M_3 = Mn_3 / np.sin(np.radians(b_w - a))
    PN_r = 1 + (2 * gamma / (gamma + 1)) * (Mn_1**2 - 1)
    return Mn_1, Mn_3, M_3, PN_r

def slip_angle(M_2, M_3):
  # s in guess slip line angle
  # for loop continous until difference between Oblique_S and Expansion_S is 0.01
  # for postive angle of attack 's' in +ve, and 's' won't be higher than a = angle of attack
    for s in np.arange(0, a, 0.1):
        a2 = a + s
        # Calling Normal(M,a) function to find parameters across trailing edge oblique shock
        Mn_2, Mn_4, M_4, PN_r42 = np.round(Normal(M_2, a2), 3)
        # P4/P1 = P4/P2 * P2/P1
        Oblique_S = PN_r42 * P_r

        # Calling pm_function(M_2) function to find parameters across trailing edge expansion fan
        v3 = np.round(np.degrees(pm_function(M_3)), 2)
        v5 = v3 + a2
        M_5 = Mach(v5)
        T_r53, P_r53, P03_r, P05_r = map(lambda x: round(x, 3), Ratio(M_3, M_5))
        # P5/P1 = P5/P3 * P3/P1
        Expansion_S = P_r53 * PN_r

        print(f"Angle: {a2}, L: {Oblique_S}, R: {Expansion_S}, s: {s}")

        # Checking whether the difference b/w Oblique_S and Expansion_S is 0.01
        if np.isclose(Oblique_S, Expansion_S, 0.01):
            return s

    print("No suitable slip angle found.")
    return None  # Or return a specific value indicating failure


# Calling pm_function(M) to find prandtl meyer function  v(M) upstream of expansion fan
M_1 = float(input("Enter Mach Number, M_1: "))
v1 = np.round(np.degrees(pm_function(M_1)), 2)
print("Upstream v1:", v1)  # degrees

# prandtl meyer function v(M) for a downstream of expansion fan
v2 = v1 + a
print("Downstream v2:", v2)

# Mach Number behind expansion fan
M_2 = Mach(v2)
print("Mach behind L.E expansion fan M_2:", M_2)

# Parameters across Leading edge expansion fan
T_r, P_r, P01_r, P02_r = map(lambda x: round(x, 3), Ratio(M_1, M_2))
print("P2/P1:", P_r, "P01/P1:", P01_r, "P02/P2:", P02_r)
print("Pressure Ratio across expansion fan, P2/P1:", P01_r / P02_r)

# Parameters across Leading edge oblique shock
Mn_1, Mn_3, M_3, PN_r = np.round(Normal(M_1, a), 3)
print("Normal component of Mn1:", Mn_1)
print("Normal component of Mn3:", Mn_3)
print("Mach behind L.E oblique shock M_3:", M_3)
print("Pressure Ratio across normal shock, P3/P1:", PN_r)
```

```
# Lift and Drag per unit span
# L' = (P3 - P2) * cos(a)
# D' = (P3 - P2) * sin(a)
# Cl = L'/(qc) = (2 / gamma*M**2) * (P3/P1 - P2/P1) * cos(a)
# Cd = D'/(qc) = (2 / gamma*M**2) * (P3/P1 - P2/P1) * sin(a)
# where P3/P1 = PN_r is Pressure ration across oblique shock and P2/P1 = P_r is Pressure ration across expansion fan
Cl = np.round((2 / (gamma * M_1**2)) * (PN_r - P_r) * np.cos(np.radians(a)), 4)
Cd = np.round((2 / (gamma * M_1**2)) * (PN_r - P_r) * np.sin(np.radians(a)), 4)
print("Coeff lift, Cl:", Cl, "Coeff Drag, Cd:", Cd)

# Calling slip_angle function
slip_result = slip_angle(M_2, M_3)
print("Calculated slip angle, Φ:", slip_result)
```

```
Enter angle of attack: 5
Enter Mach Number, M_1: 2.6
Upstream v1: 41.41
Downstream v2: 46.41
Mach behind L.E expansion fan M_2: 2.81
P2/P1: 0.724 P01/P1: 19.954 P02/P2: 27.556
Pressure Ratio across expansion fan, P2/P1: 0.7241254173319785
Normal component of Mn1: 1.157
Normal component of Mn3: 0.87
Mach behind L.E oblique shock M_3: 2.384
Pressure Ratio across normal shock, P3/P1: 1.394
Coeff lift, Cl: 0.1411 Coeff Drag, Cd: 0.0123
Angle: 5.0, L: 1.0317, R: 1.027378, s: 0.0
Calculated slip angle, Φ: 0.0
```

```
# Question 3, Anderson Example 4.16, Calculated Slip angle Φ
# Note: Every time a kernel restarts in Google Colab, external libraries(pygasflow) should reinstalled.
# For installing cell 2 need to be excuted
import numpy as np
# pygasflow python package, Reference: https://pypi.org/project/pygasflow/
from pygasflow import shockwave_solver

gamma = 1.4
a = float(input("Enter angle of attack: "))

# Stagnation ratio
def stag_Ratio(M):
    P0_r = ((1 + ((gamma - 1) / 2) * M**2)) ** (gamma / (gamma - 1))
    return P0_r

# Prandtl-Meyer Function
def pm_function(M):
    v1 = (np.sqrt((gamma + 1) / (gamma - 1))) * np.arctan(np.sqrt(((gamma - 1) / (gamma + 1)) * (M**2 - 1))) - np.arctan(np.sqrt(M**2 -
    return v1  # radians

# Ratios T2/T1, P2/P1, P01/P1 and P02/P1 for isentropic flow
# Used to obtain ratios across exapansion fan
def Ratio(M_1, M_2):
    T_r = (1 + ((gamma - 1) / 2) * M_1**2) / (1 + ((gamma - 1) / 2) * M_2**2)
    P_r = ((1 + ((gamma - 1) / 2) * M_1**2) / (1 + ((gamma - 1) / 2) * M_2**2)) ** (gamma / (gamma - 1))
    P01_r = stag_Ratio(M_1)
    P02_r = stag_Ratio(M_2)
    return T_r, P_r, P01_r, P02_r

# Mach Number behind expansion wave
def Mach(v2):
    for i in np.arange(1, 50, 0.01):
        v_2 = np.degrees(np.round(pm_function(i), 3))
        if np.isclose(v_2, v2, 0.01):
            return np.round(i, 2)

# Create normal(M) function
# which returns Upstream Normal component of M1 i.e. Mn1, Downstream Normal component of M i.e. Mn2 and Downstream Mach Number M2
# Pressure rise across oblique shock
def Normal(M_1, a):

    # Using pygasflow python package to obtain β(weak solution) with Mach number(m1) and theta(flow deflection d) as input.
    # shockwave_solver gives flow parameters across a oblqiue shock wave
    result = shockwave_solver('m1', M_1, 'theta', a)
    M1, MN1, M2, MN2, beta, theta, pr, dr, tr, tpr = map(lambda x: round(x,4), result)
    b_w = beta
    Mn_1 = M_1 * np.sin(np.radians(b_w))
    Mn_3 = np.sqrt((2 + (gamma - 1) * Mn_1**2) / (2 * gamma * Mn_1**2 - (gamma - 1)))
    M_3 = Mn_3 / np.sin(np.radians(b_w - a))
    PN_r = 1 + (2 * gamma / (gamma + 1)) * (Mn_1**2 - 1)
    return Mn_1, Mn_3, M_3, PN_r
```

```
def slip_angle(M_2, M_3):
  # s in guess slip line angle
  # for loop continous until difference between Oblique_S and Expansion_S is 0.01
  # for postive angle of attack 's' in +ve, and 's' won't be higher than a = angle of attack
    for s in np.arange(0, a, 0.1):
        a2 = a + s
        # Calling Normal(M,a) function to find parameters across trailing edge oblique shock
        Mn_2, Mn_4, M_4, PN_r42 = np.round(Normal(M_2, a2), 3)
        # P4/P1 = P4/P2 * P2/P1
        Oblique_S = PN_r42 * P_r

        # Calling pm_function(M_2) function to find parameters across trailing edge expansion fan
        v3 = np.round(np.degrees(pm_function(M_3)), 2)
        v5 = v3 + a2
        M_5 = Mach(v5)
        T_r53, P_r53, P03_r, P05_r = map(lambda x: round(x, 3), Ratio(M_3, M_5))
        # P5/P1 = P5/P3 * P3/P1
        Expansion_S = P_r53 * PN_r

        print(f"Angle: {a2}, L: {Oblique_S}, R: {Expansion_S}, s: {s}")

        # Checking whether the difference b/w Oblique_S and Expansion_S is 0.01
        if np.isclose(Oblique_S, Expansion_S, 0.01):
            return s

    print("No suitable slip angle found.")
    return None  # Or return a specific value indicating failure


# Calling pm_function(M) to find prandtl meyer function  v(M) upstream of expansion fan
M_1 = float(input("Enter Mach Number, M_1: "))
v1 = np.round(np.degrees(pm_function(M_1)), 2)
print("Upstream v1:", v1)  # degrees

# prandtl meyer function v(M) for a downstream of expansion fan
v2 = v1 + a
print("Downstream v2:", v2)

# Mach Number behind expansion fan
M_2 = Mach(v2)
print("Mach behind L.E expansion fan M_2:", M_2)

# Parameters across Leading edge expansion fan
T_r, P_r, P01_r, P02_r = map(lambda x: round(x, 3), Ratio(M_1, M_2))
print("P2/P1:", P_r, "P01/P1:", P01_r, "P02/P2:", P02_r)
print("Pressure Ratio across expansion fan, P2/P1:", P01_r / P02_r)

# Parameters across Leading edge oblique shock
Mn_1, Mn_3, M_3, PN_r = np.round(Normal(M_1, a), 3)
print("Normal component of Mn1:", Mn_1)
print("Normal component of Mn3:", Mn_3)
print("Mach behind L.E oblique shock M_3:", M_3)
print("Pressure Ratio across normal shock, P3/P1:", PN_r)

# Lift and Drag per unit span
# L' = (P3 - P2) * cos(a)
# D' = (P3 - P2) * sin(a)
# Cl = L'/(qc) = (2 / gamma*M**2) * (P3/P1 - P2/P1) * cos(a)
# Cd = D'/(qc) = (2 / gamma*M**2) * (P3/P1 - P2/P1) * sin(a)
# where P3/P1 = PN_r is Pressure ration across oblique shock and P2/P1 = P_r is Pressure ration across expansion fan
Cl = np.round((2 / (gamma * M_1**2)) * (PN_r - P_r) * np.cos(np.radians(a)), 4)
Cd = np.round((2 / (gamma * M_1**2)) * (PN_r - P_r) * np.sin(np.radians(a)), 4)
print("Coeff lift, Cl:", Cl, "Coeff Drag, Cd:", Cd)

# Calling slip_angle function
slip_result = slip_angle(M_2, M_3)
print("Calculated slip angle, Φ:", slip_result)
```

```
Enter angle of attack: 20
Enter Mach Number, M_1: 3
Upstream v1: 49.76
Downstream v2: 69.75999999999999
Mach behind L.E expansion fan M_2: 4.27
P2/P1: 0.17 P01/P1: 36.733 P02/P2: 216.255
Pressure Ratio across expansion fan, P2/P1: 0.16985965642412892
Normal component of Mn1: 1.837
Normal component of Mn3: 0.608
Mach behind L.E oblique shock M_3: 1.994
Pressure Ratio across normal shock, P3/P1: 3.771
Coeff lift, Cl: 0.5371 Coeff Drag, Cd: 0.1955
Angle: 20.0, L: 0.9633900000000001, R: 1.0596510000000001, s: 0.0
Angle: 20.1, L: 0.9698500000000001, R: 1.0596510000000001, s: 0.1
Angle: 20.2, L: 0.9763100000000001, R: 1.0596510000000001, s: 0.2
```

```
Angle: 20.3, L: 0.98277, R: 1.044567, s: 0.30000000000000004
Angle: 20.4, L: 0.98923, R: 1.044567, s: 0.4
Angle: 20.5, L: 0.9956900000000001, R: 1.0294830000000001, s: 0.5
Angle: 20.6, L: 1.00232, R: 1.0294830000000001, s: 0.6000000000000001
Angle: 20.7, L: 1.00878, R: 1.014399, s: 0.7000000000000001
Calculated slip angle, Φ: 0.7000000000000001
```