# Indian Institute of Technology Bombay

AE 339 - High Speed Aerodynamics, 2024

---

## Project Report

## Compressible Flow Calculator

---

Shubhranil Chatterjee 210100144

# Contents

# List of Figures

## 1 Introduction

Compressible flow plays a crucial role in many engineering applications, from high-speed aerospace vehicles to gas dynamics in nozzles and pipelines. Compressible flow theory allows for the prediction of key flow properties, such as pressure, temperature, density, and velocity, especially in scenarios where the flow speed approaches or exceeds the speed of sound. However, the governing equations for compressible flow—such as isentropic flow relations and shock relations—are often complex and difficult to solve analytically due to their nonlinearity.

To address this difficulty, engineers have historically used gas tables, which provide precomputed values of flow properties for various conditions. While useful, these tables require manual interpolation and lookups, which can be time-consuming and prone to error. With the rise of modern computing, these manual methods can be efficiently replaced by automated tools that not only eliminate the need for gas tables but also provide faster and more accurate results.

In this project, a Python program that acts as a comprehensive calculator for key compressible flow scenarios has been developed. The program computes flow properties for:

- One Dimensional Isentropic flows
- Quasi One Dimensional Isentropic flows
- Normal shocks
- Oblique shocks

The Python program simplifies the process of determining flow parameters, eliminating the need for cumbersome gas tables. It can be run directly from the command line, providing an efficient, user-friendly solution for compressible flow analysis.
Key assumptions made in the program include:

- The flow is modeled as inviscid, i.e. neglecting any viscosity effects.
- The fluid is assumed to be an ideal gas with a constant specific heat ratio $\gamma$.
- In isentropic flows, the process is adiabatic and reversible (no heat transfer or entropy change).
- **N**ormal and oblique shocks are treated as adiabatic flow discontinuities.
- In quasi-one-dimensional flows, all flow properties are uniform across any given cross section of the flow and it is the area change that causes the flow properties to vary in the streamwise direction.

The remainder of this report is structured as follows: the governing equations for inviscid, adiabatic flow and the isentropic and shock relations are presented in Section 2. Section 3 explains the structure and flow of the Python code, describing how it computes the relevant flow parameters for different compressible flow scenarios. Handling of edge cases and error detection is also described in this section. A flowchart of the program is presented at the end of this section. The full program code can be found in Appendix A.

# 2 Theoretical Background

## 2.1 Governing Equations

The governing equations for steady-state, adiabatic, inviscid, compressible flow in the absence of body forces can be stated as:

$$\nabla \cdot (\rho \vec{u}) = 0 \tag{1}$$

$$\vec{u} \cdot \nabla \vec{u} = -\frac{1}{\rho} \nabla p \tag{2}$$

$$h_0 = e + \frac{p}{\rho} + \frac{|\vec{u}|^2}{2} = \text{const} \tag{3}$$

Eqns. 1 - 3 are closed by the state equation, which for an ideal gas reads:

$$p = \rho R T \tag{4}$$

The speed of sound in an ideal gas can be found as:

$$a = \left(\frac{\partial p}{\partial \rho}\right)_s = \sqrt{\frac{\gamma p}{\rho}} \tag{5}$$

where $\gamma = \frac{c_p}{c_v}$ is the ratio of specific heat capacities.

Using (4) and (5), the energy equation can be rewritten as:

$$T_0 = T\left(1 + \frac{\gamma - 1}{2} M^2\right) = \text{const} \tag{6}$$

where $M = \frac{|\vec{u}|}{a}$ is the Mach number.

## 2.2 Isentropic Flow Relations

For isentropic flows, we have:

$$ds = c_p \frac{dT}{T} + R \frac{dp}{p} = 0 \tag{7}$$

On integrating from state 1 to 2, we get:

$$\frac{T_2}{T_1} = \left(\frac{p_2}{p_1}\right)^{\frac{\gamma-1}{\gamma}} = \left(\frac{\rho_2}{\rho_1}\right)^{\gamma - 1} \tag{8}$$

Using (6), we recover the familiar form of the isentropic relations:

$$\frac{p_0}{p} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma-1}{\gamma}} \tag{9}$$

$$\frac{\rho_0}{\rho} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\gamma - 1} \tag{10}$$

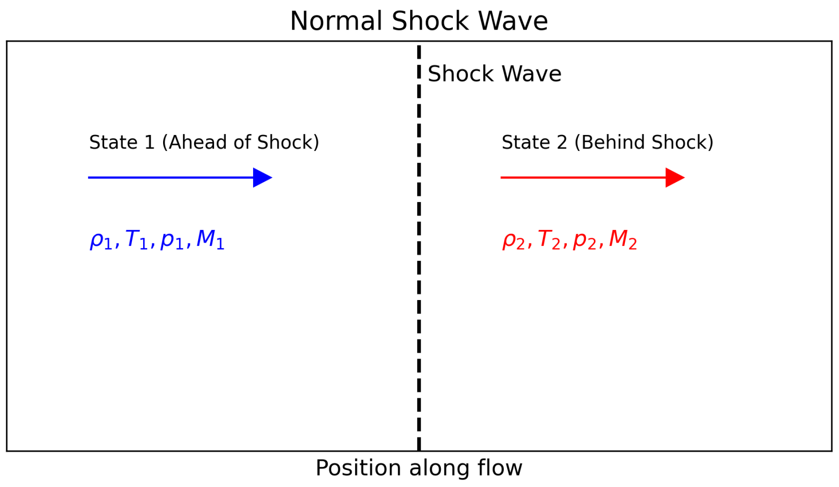

Figure 1: Schematic of a normal shock wave

## 2.3 Normal Shock Relations

A normal shock (Fig. 1) is a very thin region of discontinuity perpendicular to the flow where the flow properties change drastically. Since there is no heat transfer from the flow to the surroundings, the flow across the shock wave is adiabatic. Therefore, the basic shock equations are obtained from (1), (2) and (3).

$$\rho_1 u_1 = \rho_2 u_2 \tag{11}$$

$$p_1 + \rho_1 u_1^2 = p_2 + \rho_2 u_2^2 \tag{12}$$

$$h_1 + \frac{u_1^2}{2} = h_2 + \frac{u_2^2}{2} \tag{13}$$

An alternate form of these equations can be written as:

$$\frac{\rho_2}{\rho_1} = \frac{a_1}{a_2}\frac{M_1}{M_2} \tag{14}$$

$$\frac{\rho_2}{\rho_1} = \frac{a_1^2}{a_2^2}\frac{(1+\gamma M_1^2)}{(1+\gamma M_2^2)} \tag{15}$$

$$\frac{a_2^2}{a_1^2} = \frac{2+(\gamma-1)M_1^2}{2+(\gamma-1)M_2^2} \tag{16}$$

Eliminating $\frac{\rho_2}{\rho_1}$ and $\frac{a_2}{a_1}$ from (14) - (16), we get the mach number relation for normal shock waves:

$$M_2^2 = \frac{1+\frac{(\gamma-1)}{2}M_1^2}{\gamma M_1^2 - \frac{(\gamma-1)}{2}} \tag{17}$$

Substituting back (17) in (12) - (14), we get the remaining normal shock relations:

$$\frac{\rho_2}{\rho_1} = \frac{(\gamma+1)M_1^2}{2+(\gamma-1)M_1^2} \tag{18}$$

$$\frac{p_2}{p_1} = 1 + \frac{2}{\gamma+1}(M_1^2 - 1) \tag{19}$$

$$\frac{T_2}{T_1} = \left[1 + \frac{2}{\gamma+1}(M_1^2 - 1)\right]\left[\frac{2 + (\gamma-1)M_1^2}{(\gamma+1)M_1^2}\right] \tag{20}$$

## 2.4 Oblique Shock Relations

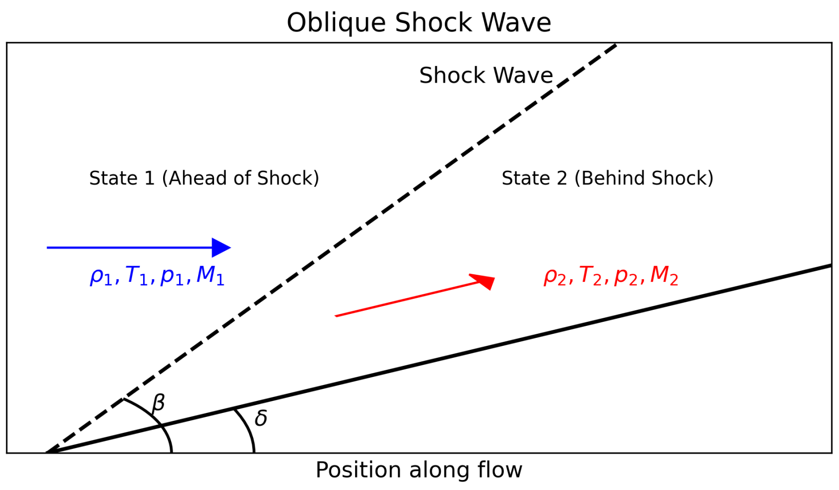

Figure 2: Schematic of an oblique shock wave

Oblique shock waves occur when supersonic flow is turned into itself. Fig. 2 shows a schematic of this phenomenon. The incoming supersonic flow is turned into itself by an angle $\delta$ called as the turning angle. This change in flow direction takes place across a shock wave that is at an angle $\beta$ to the upstream flow. This is called the wave angle.

To obtain the oblique shock relations, one can apply the normal shock relations to the components of velocities normal to the shock.

$$M_{n_1} = M_1 sin\beta \tag{21}$$

$$M_{n_2} = M_2 sin(\beta - \delta) \tag{22}$$

(21) and (22) can be substituted into (17)-(20) to get the oblique shock relations. However, these cannot be obtained unless the flow turning angle $\delta$ is found for a given shock angle $\beta$.

To obtain $\delta$, one can use the fact that the tangential velocity component remains unchanged across an oblique shock wave.

$$M_{n_1} tan\beta = M_{n_2} tan(\beta - \delta) \tag{23}$$

(23) can be used along with (17), (21) and (22) to get the $\delta - \beta - M$ relation:

$$tan\delta = 2cot\beta\left[\frac{M_1^2 sin^2\beta - 1}{M_1^2(\gamma + cos2\beta) + 2}\right] \tag{24}$$

A few important observations are to be made from (24).

For any value of $\delta$, two values of $\beta$ are predicted by (24). The higher value of $\beta$ leads to stronger changes in flow properties across the shock and hence is called as the strong shock solution, and the smaller $\beta$ is the weak shock solution.

There is a maximum value of $\delta$ for a given mach number, beyond which an oblique shock wave cannot exist. In such cases, a detached bow shock appears.

## 2.5 Quasi One Dimensional Flow

Quasi-one dimensional flow allows for variations in streamtube area along the flow direction while maintaining uniform flow properties across each cross-section. This approach is widely used to approximate flows for engineering applications, such as in wind tunnels and rocket engines.

Consider a flow through a duct having varying area. At the throat, i.e. the area of minimum cross-section the flow is sonic (M = 1). Denote the flow properties at this area $A^*$ as $p^*$, $\rho^*$, $a^*$. The continuity equation for quasi-one dimensional flow takes the form:

$$\rho_1 A_1 u_1 = \rho_2 A_2 u_2 \tag{25}$$

Using isentropic relations from section 2.2 in (25), we get the area-mach number relation:

$$\left(\frac{A}{A^*}\right)^2 = \frac{1}{M^2}\left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{\frac{\gamma+1}{\gamma-1}} \tag{26}$$

For a given area ratio, (26) gives two values of M, a subsonic value and a supersonic value. Depending on the flow regime, the flow Mach number can be found out from the given area ratio. The remaining property ratios can be found using isentropic flow relations for M = 1 at the throat.

# 3 Program Structure

## 3.1 Function Definitions

The program is structured around modular functions that handle different compressible flow relations, with limited core input parameters. The functions for isentropic flow relations and normal shock relations accept Mach number as the main input, since all property ratios are functions only of the mach number. For oblique shock relations, two input parameters, namely the upstream mach number and the shock angle are required.

## 3.2 Solution Methodology

### 3.2.1 (Quasi)-Isentropic Flow Relations

Results for isentropic and quasi-isentropic flow relations are combined together. The user can either input the flow mach number or one of the following property ratios: 1) $T/T_0$, 2) $P/P_0$, 3) $\rho/\rho_0$, 4) $A/A^*$ (subsonic), 5) $A/A^*$ (supersonic), 6) Mach angle, 7) Prandtl-Meyer Angle. Provided any of 1-7, the flow mach number is first calculated using Newton's method, and the remaining properties are subsequently calculated and displayed to the user.

### 3.2.2 Normal Shock Relations

The user can either input the upstream mach number ($M_1$) or one of the following properties: 1) $M_2$, 2) $P_2/P_1$, 3) $\rho_2/\rho_1$, 4) $T_2/T_1$, 5) $P0_2/P0_1$, 6) $P_1/P0_2$. Provided any of 1-6, the upstream mach number is first calculated using Newton's method, and the remaining properties are subsequently calculated and displayed to the user.

### 3.2.3 Oblique Shock Relation

The user needs to compulsorily input the upstream mach number. In addition, the user can input any of the following properties: 1) Turn Angle - $\delta$ (Weak Shock), 2) Turn Angle - $\delta$ (Strong Shock), 3) Shock Angle - $\beta$, 4) Normal Upstream Mach Number - $M_{1n}$. Provided any of 1-4, the shock angle is first computed using Newton's method and the remaining properties are subsequently calculated and displayed to the user.

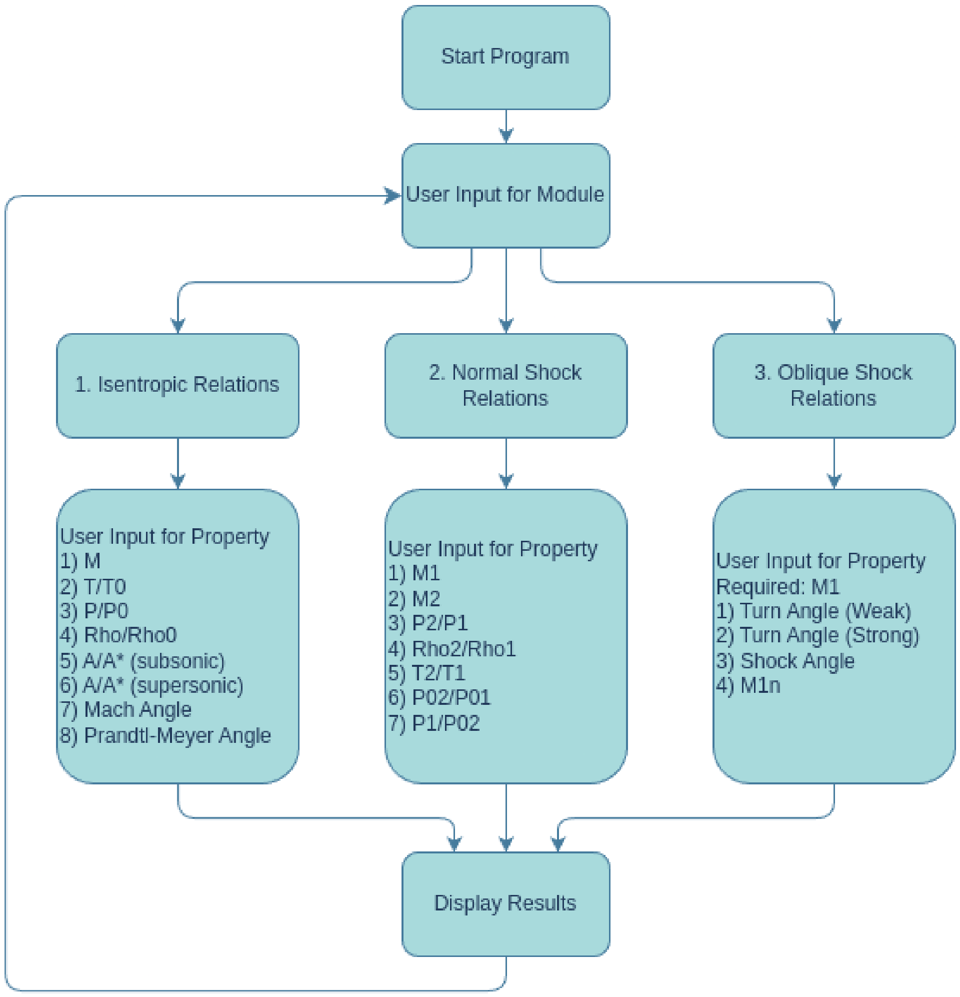

Figure 3: Program Workflow

## 3.3 Error Handling

The program incorporates error handling mechanisms to ensure that invalid inputs or unphysical conditions do not disrupt the flow of calculations. For each section, the program checks if the provided Mach number or property ratios fall within physically realistic ranges. If not, the program execution is interrupted and descriptive error message is displayed, prompting the user to correct their input.

## Appendix A: Code Listing

```
import numpy as np
from scipy.optimize import fsolve, minimize
from tabulate import tabulate

'''
    This script calculates compressible aerodynamic relations for 1-D and quasi
    ↪  1-D
    isentropic flow, normal shock waves, and oblique shock waves using
    ↪  equations
    from compressible flow theory.

    Solution Strategy -
        (quasi)1-D Isentropic Flow Relations - Provided any of the flow
        ↪  property ratios
        or angles, the local Mach number is first calculated through iterative
        ↪  methods
        and the remaining property ratios are then computed directly using
        ↪  formulae.

        Normal Shock Relations - Provided and of the flow property ratios, the
        ↪  upstream
        Mach number is first calculated through iterative methods
        and the remaining property ratios are then computed directly using
        ↪  formulae.

        Oblique Shock Relations - Provided the upstream Mach number and turn
        ↪  angle,
        shock angle or incident normal Mach number, the shock angle is first
        ↪  calculated
        through iterative methods   and the remaining property ratios are then
        computed directly using formulae.

    Features:
            1) User-friendly command line interface for user input.
            2) Supports several property ratios as inputs.
            3) Ensures inputs are withing acceptable ranges.

    Dependencies : Numpy, Scipy, Tabulate
        To install the dependencies, use : pip install numpy scipy tabulate
'''
```

```
# Isentropic Flow Relations

def ImachAngle (M):
    if M>=1:
        return np.arcsin(1/M)*180/np.pi
    else:
        return None

def IprandtlMeyerAngle (M,gamma):
    if M>=1:
        return
         ↪  (np.sqrt((gamma+1)/(gamma-1))*np.arctan(np.sqrt((gamma-1)/(gamma+1)*(M**2-1)))
         ↪  - np.arctan(np.sqrt(M**2-1)))*180/np.pi
    else:
        return None

def ITbyT0 (M,gamma):
    return 1/(1 + (gamma-1)/2*M**2)

def IPbyP0 (M,gamma):
    return np.power(1 + (gamma-1)/2*M**2,-gamma/(gamma-1))

def IRhobyRho0 (M,gamma):
    return np.power(1 + (gamma-1)/2*M**2,-gamma/(gamma-1))*(1 + (gamma-1)/2*M**2)

def IAbyAcrit (M,gamma):
    return 1/M*np.power(2/(gamma+1)*(1+(gamma-1)/2*M**2),(gamma+1)/2/(gamma-1))

def IPbyPcrit (M,gamma):
    return IPbyP0(M,gamma)/IPbyP0(1,gamma)

def IRhobyRhocrit (M,gamma):
    return IRhobyRho0(M,gamma)/IRhobyRho0(1,gamma)

def ITbyTcrit (M,gamma):
    return ITbyT0(M,gamma)/ITbyT0(1,gamma)

# Normal Shock Relations

def NSM2 (M1,gamma):
    return np.sqrt((1+(gamma-1)/2*M1**2)/(gamma*M1**2-(gamma-1)/2))

def NSRho2byRho1 (M1,gamma):
    return (gamma+1)*M1**2/(2+(gamma-1)*M1**2)

def NSP2byP1 (M1,gamma):
    return 1+2*gamma/(gamma+1)*(M1**2-1)
```

```
def NST2byT1(M1, gamma):
    return NSP2byP1(M1,gamma)/NSRho2byRho1(M1,gamma)

def NSP02byP01 (M1,gamma):
    return
    ↪  np.power(((gamma+1)/2*M1**2/(1+(gamma-1)/2*M1**2)),gamma/(gamma-1))/np.power(2*gamm

def NSP1byP02 (M1,gamma):
    return IPbyP0(M1,gamma)/NSP02byP01(M1,gamma)

# Oblique shock relations

def OSdelta(M1,gamma,beta):
    beta = np.pi*beta/180
    return 180/np.pi*np.arctan((2/np.tan(beta)*(M1**2 * (np.sin(beta))**2 -
    ↪  1))/(2+M1**2*(gamma+np.cos(2*beta))))

def OSminbeta(M1,gamma):
    return fsolve(lambda beta: OSdelta(M1, gamma, beta), 20)

def OSmaxdelta(M1,gamma):
    result = minimize(lambda beta: -OSdelta(M1,gamma,beta), 40)
    maxdelta_beta = result.x[0]
    maxdelta = -result.fun
    return maxdelta, maxdelta_beta

def OSbetaWeak(M1,gamma,delta):
    return fsolve(lambda beta: delta - OSdelta(M1,gamma,beta),20)

def OSbetaStrong(M1,gamma,delta):
    return fsolve(lambda beta: delta - OSdelta(M1,gamma,beta),90)

def OSM1n(M1,gamma,beta):
    return M1*np.sin(beta*np.pi/180)

def OSM2n(M1,gamma,beta):
    return NSM2(OSM1n(M1,gamma,beta),gamma)

def OSM2(M1,gamma,beta):
    return OSM2n(M1,gamma,beta)/np.sin(np.pi/180*(beta-OSdelta(M1,gamma,beta)))

def OSP2byP1(M1,gamma,beta):
    return NSP2byP1(OSM1n(M1,gamma,beta),gamma)

def OSRho2byRho1(M1,gamma,beta):
    return NSRho2byRho1(OSM1n(M1,gamma,beta),gamma)

def OST2byT1(M1,gamma,beta):
    return NST2byT1(OSM1n(M1,gamma,beta),gamma)

def OSP02byP01(M1,gamma,beta):
    M2 = OSM2(M1,gamma,beta)
    return 1/IPbyP0(M2,gamma) * OSP2byP1(M1,gamma,beta) * IPbyP0(M1,gamma)

# Main

print("Welcome to Shubhranil's Compressible Aerodynamics Calculator.")
UserInput = 0

while (UserInput != 4):
    print("\nEnter '1' for Isentropic Flow Relations, '2' for Normal Shock
    ↪  Relations, '3' for Oblique Shock Relations, '4' to Exit the program. ")

    UserInput = int(input())

    if UserInput == 1:
        print("\nEnter heat capacity ratio (gamma).")

        gamma = float(input())

        if gamma <= 1:
            print("\nInvalid input. Gamma must be greater than 1.")
            continue

        print("\nSelect input : 1-Mach Number, 2-T/T0, 3-P/P0, 4-Rho/Rho0,
        ↪  5-A/A*(subsonic), 6-A/A*(supersonic), 7-Mach Angle (deg.),
        ↪  8-Prandtl-Meyer Angle (deg.). ")

        inputType = int(input())

        if inputType == 1:

            print("\nEnter Mach number.")

            M = float(input())

            if M<=0:
                print("\nInvalid input. Mach number must be greater than 0.")

        elif inputType == 2:

            print('\nEnter T/T0.')

            TbyT0 = float(input())

            if (TbyT0 >= 1) or (TbyT0 <= 0):
```

```
            print("\nInvalid input. T/T0 must be between 0 and 1.")
            continue

        M = fsolve(lambda M: TbyT0 - ITbyT0(M,gamma), 1)

    elif inputType == 3:

        print('\nEnter P/P0.')

        PbyP0 = float(input())

        if (PbyP0 >= 1) or (PbyP0 <= 0):
            print("\nInvalid input. P/P0 must be between 0 and 1.")
            continue

        M = fsolve(lambda M: PbyP0 - IPbyP0(M,gamma), 1)

    elif inputType == 4:

        print('\nEnter Rho/Rho0.')

        RhobyRho0 = float(input())

        if (RhobyRho0 >= 1) or (RhobyRho0 <= 0):
            print("\nInvalid input. Rho/Rho0 must be between 0 and 1.")
            continue

        M = fsolve(lambda M: RhobyRho0 - IRhobyRho0(M,gamma), 1)

    elif inputType == 5:

        print('\nEnter A/A* (subsonic).')

        AbyAcrit = float(input())

        if AbyAcrit < 1:
            print("\nInvalid input. A/A* must be greater than 1.")
            continue

        M = fsolve(lambda M: AbyAcrit - IAbyAcrit(M,gamma), 0.01)

    elif inputType == 6:

        print('\nEnter A/A* (supersonic).')

        AbyAcrit = float(input())

        if AbyAcrit < 1:
            print("\nInvalid input. A/A* must be greater than 1.")
            continue

        M = fsolve(lambda M: AbyAcrit - IAbyAcrit(M,gamma), 2)

    elif inputType == 7:

        print('\nEnter Mach Angle (deg.).')

        machAngle = float(input())

        if (machAngle > 90) or (machAngle < 0):
            print("\nInvalid input. Mach Angle must be between 0 and 90
             ↪  degrees.")
            continue

        M = fsolve(lambda M: machAngle - ImachAngle(M), 1)

    elif inputType == 8:

        print('\nEnter Prandtl-Meyer Angle (deg.).')

        prandtlMeyerAngle = float(input())

        maxPMAngle = IprandtlMeyerAngle(1e10,gamma)

        if (prandtlMeyerAngle < 0) or (prandtlMeyerAngle > maxPMAngle):
            print("Invalid input. Prandtl-Meyer Angle must be between 0 and
             ↪  {:.2f} degrees.".format(maxPMAngle))
            continue

        M = fsolve(lambda M: prandtlMeyerAngle - IprandtlMeyerAngle(M,gamma),
         ↪  1)

    else :
        print('\nInvalid input.')
        continue

    machAngle = ImachAngle(M)

    prandtlMeyerAngle = IprandtlMeyerAngle(M,gamma)

    PbyP0 = IPbyP0(M,gamma)

    RhobyRho0 = IRhobyRho0(M,gamma)

    TbyT0 = ITbyT0(M,gamma)
    AbyAcrit = IAbyAcrit(M,gamma)

    PbyPcrit = IPbyPcrit(M,gamma)

    RhobyRhocrit = IRhobyRhocrit(M,gamma)

    TbyTcrit = ITbyTcrit(M,gamma)

    headers_1 = ["Mach Number", "Mach Angle", "Prandtl-Meyer Angle", "P/P0",
    ↪  "Rho/Rho0"]

    headers_2 = ["T/T0", "P/P*", "Rho/Rho*", "T/T*", "A/A*"]

    result_1 = [[M, machAngle, prandtlMeyerAngle, PbyP0, RhobyRho0]]

    result_2 = [[TbyT0, PbyPcrit, RhobyRhocrit, TbyTcrit, AbyAcrit]]

    print("\n\t\t\t Isentropic Flow Relations \n")
    print(tabulate(result_1, headers = headers_1))
    print("")
    print(tabulate(result_2, headers = headers_2))

elif UserInput == 2:
    print("\nEnter heat capacity ratio (gamma).")

    gamma = float(input())

    if gamma <= 1:
        print("\nInvalid input. Gamma must be greater than 1.")
        continue

    print("\nSelect input : 1-M1, 2-M2, 3-P2/P1, 4-Rho2/Rho1, 5-T2/T1,
    ↪  6-P02/P01, 7-P1/P02. ")

    inputType = int(input())

    if inputType == 1:
        print("\nEnter M1.")

        M1 = float(input())

        if M1 < 1:
            print("\nInvalid input. M1 must be greater than 1.")
            continue

    elif inputType == 2:
        print("\nEnter M2.")
        M2 = float(input())

        minM2 = NSM2(1e10,gamma)

        if (M2 > 1) or (M2 < minM2):
            print('\nInvalid input. M2 must be between {:.2f} and
             ↪  1.'.format(minM2))
            continue

        M1 = fsolve(lambda M1: M2 - NSM2(M1,gamma), 2)

    elif inputType == 3:
        print("\nEnter P2/P1.")

        P2byP1 = float(input())

        if P2byP1 < 1:
            print("\nInvalid input. P2/P1 must be greater than 1.")
            continue

        M1 = fsolve(lambda M1: P2byP1 - NSP2byP1(M1,gamma),2)

    elif inputType == 4:
        print("\nEnter Rho2/Rho1.")

        Rho2byRho1 = float(input())

        maxRho2byRho1 = NSRho2byRho1(1e10,gamma)

        if (Rho2byRho1 < 1) or (Rho2byRho1 > maxRho2byRho1):
            print("\nInvalid input. Rho2/Rho1 must be between 1 and
             ↪  {:.2f}.".format(maxRho2byRho1))
            continue

        M1 = fsolve(lambda M1: Rho2byRho1 - NSRho2byRho1(M1,gamma),2)

    elif inputType == 5:
        print("\nEnter T2/T1.")

        T2byT1 = float(input())

        if T2byT1 < 1:
            print("\nInvalid input. T2/T1 must be greater than 1.")
            continue

        M1 = fsolve(lambda M1: T2byT1 - NST2byT1(M1,gamma),2)

    elif inputType == 6:
```

```
    print("\nEnter P02/P01.")

    P02byP01 = float(input())

    if (P02byP01 >= 1) or (P02byP01 <= 0):
        print("\nInvalid input. P02/P01 must be between 0 and 1.")
        continue

    M1 = fsolve(lambda M1: P02byP01 - NSP02byP01(M1,gamma),2)

elif inputType == 7:
    print("\nEnter P1/P02.")

    P1byP02 = float(input())

    maxP1byP02 = NSP1byP02(1,gamma)

    if (P1byP02 >= maxP1byP02) or (P1byP02 <= 0):
        print("\nInvalid input. P1byP02 must be between 0 and
         ↪  {:.3f}.".format(maxP1byP02))
        continue

    M1 = fsolve(lambda M1: P1byP02 - NSP1byP02(M1,gamma),2)

else :
    print("\nInvalid input.")
    continue

M2 = NSM2(M1,gamma)

P2byP1 = NSP2byP1(M1,gamma)

Rho2byRho1 = NSRho2byRho1(M1,gamma)

T2byT1 = NST2byT1(M1,gamma)

P02byP01 = NSP02byP01(M1,gamma)

P1byP02 = NSP1byP02(M1,gamma)

headers_1 = ["M1", "M2", "P2/P1", "Rho2/Rho1","T2/T1", "P02/P01",
 ↪  "P1/P02"]

result_1 = [[M1, M2, P2byP1, Rho2byRho1,T2byT1, P02byP01, P1byP02]]

print("\n\t\t\t Normal Shock Relations \n")
print(tabulate(result_1, headers = headers_1))
```

```
elif UserInput == 3:
    print("\nEnter heat capacity ratio (gamma).")

    gamma = float(input())

    if gamma <= 1:
        print("\nInvalid input. Gamma must be greater than 1.")
        continue

    print("\nEnter M1.")

    M1 = float(input())

    if M1 <= 1:
        print("\nInvalid input. M1 must be greater than 1.")
        continue

    deltaMax = OSmaxdelta(M1,gamma)[0]
    deltaMax_beta = OSmaxdelta(M1,gamma)[1]

    betaMin = OSminbeta(M1,gamma)[0]

    print("\nSelect input : 1-Turn Angle (weak shock), 2-Turn Angle (strong
     ↪  shock), 3-Shock Angle, 4-M1n.")

    inputType = int(input())

    if inputType == 1:
        print("\nEnter turn angle (weak shock).")

        delta = float(input())

        if (delta < 0):
            print("\nInvalid input. Turn angle must be greater than 0.")
            continue

        if (delta > deltaMax):
            print("\nTurn angle greater than {:.2f}. Detached bow shock
             ↪  appears.".format(deltaMax))
            continue

        beta = OSbetaWeak(M1,gamma,delta)

    elif inputType == 2:
        print("\nEnter turn angle (strong shock).")

        delta = float(input())
        if (delta < 0):
            print("\nInvalid input. Turn angle must be greater than 0.")
            continue

        if (delta > deltaMax):
            print("\nTurn angle greater than {:.2f}. Detached bow shock
             ↪  appears.".format(deltaMax))
            continue

        beta = OSbetaStrong(M1,gamma,delta)

    elif inputType == 3:
        print("\nEnter shock angle.")

        beta = float(input())

        if beta < betaMin:
            print("\nInvalid input. Shock angle must be greater than Mach
             ↪  angle ({:.2f}).".format(betaMin))
            continue

    elif inputType == 4:
        print("\nEnter M1n.")

        M1n = float(input())

        beta = fsolve(lambda beta: M1n - OSM1n(M1,gamma,beta),40)

    else :
        print("\nInvalid input.")
        continue

    M2 = OSM2(M1,gamma,beta)

    delta = OSdelta(M1,gamma,beta)

    P2byP1 = OSP2byP1(M1,gamma,beta)

    Rho2byRho1 = OSRho2byRho1(M1,gamma,beta)

    T2byT1 = OST2byT1(M1,gamma,beta)

    P02byP01 = OSP02byP01(M1,gamma,beta)

    M1n = OSM1n(M1,gamma,beta)

    M2n = OSM2n(M1,gamma,beta)
```

```
        headers_1 = ["M1", "M2", "Turn Angle", "Shock Angle", "P2/P1"]

        headers_2 = ["Rho2/Rho1", "T2/T1", "P02/P01", "M1n", "M2n"]

        result_1 = [[M1, M2, delta, beta, P2byP1]]

        result_2 = [[Rho2byRho1, T2byT1, P02byP01, M1n, M2n]]

        print("\n\t\t\t Oblique Shock Relations \n")
        print(tabulate(result_1, headers = headers_1))
        print("")
        print(tabulate(result_2, headers = headers_2))

    elif UserInput !=4 :
        print("\nInvalid input.")
```

## References

[1] J. D. Anderson, "Modern Compressible Flow with Historical Perspective," Third Edition, McGraw Hill Professional, New York, 2003.

[2] Compressible Aerodynamics Calculator, William Devenport https://devenport.aoe.vt.edu/aoe3114/calch.html