METUAEROSPACE

AEE445 HYPERSONIC FLOW

PROJECT II REPORT

Authors:

Selimhan Dağtaş – 2248052

Ömer Burkaş – 2268910

Zihni Berkay Günenç - 2311041

Büşra Eren - 2238301

Instructor: Prof. Dr. Sinan Eyi

Assistant: Melikşah Koca

Contents

1. INTRODUCTION

In this project we are asked to write a computer code and calculate the following flow variables around a circular-arc-airfoil by using the irrotational method of characteristics.

- Pressure contours
- Temperature contours
- Density contours
- Surface pressure distribution
- Surface temperature distribution
- Surface density distribution
- Compare the pressure coefficients calculated from the method of characteristics and the modified Newtonian theory.
- Compare the pressure coefficients calculated from the method of characteristics and the Newtonian theory.
- Compare the pressure coefficients calculated from the method of characteristics and the modified Newtonian theory.
- Compare the pressure coefficients calculated from the method of characteristics and the modified Newtonian theory with centrifugal force correction.

The thickness of the airfoil, free stream Mach number and the specific heat ratio should also be calculated according to following equations:

$$t = \frac{1}{N \times 10}\sum_{n=1}^{N} b_n$$

$$M_\infty = 6 + \frac{1}{N}\sum_{n=1}^{N} b_n$$

$$\gamma = 1 + \frac{1}{N \times 10}\sum_{n=1}^{N} b_n$$

where b_n is the group member's last digit student id number and N is the number of members in group. Then b_n taken as 1 and N taken as 4. The formulas become:

$$t = \frac{1}{4 \times 10}(2 + 0 + 1 + 1) = 0.1$$

$$M_\infty = 6 + \frac{1}{4}(2 + 0 + 1 + 1) = 7$$

$$\gamma = 1 + \frac{1}{4 \times 10}(2 + 0 + 1 + 1) = 1.1$$

2. METHOD

The Method of Characteristics is an iterative method that calculates a series of points under a shock wave and above a specified surface. A “known” point is a point in which the x coordinate, y coordinate, Mach number, and Theta (direction of flow velocity) is known. These points are calculated using three main calculation variations: Interior Point, Wall Point, Shock Point. Note that, these point calculations require some preliminary functions to be defined beforehand.

The first step is to use shock wave relations at the leading edge of the airfoil to calculate some initial points. This first set of points will be used to calculate the next set and so on. As the distance between the surface of the airfoil and the shock wave increases, it is expected that the distance between the points will increase resulting in a decrease in precision. However, if the points are adequately numerous, the result will have sufficient accuracy. The Mach and Theta values at these points will be used to calculate other flow properties, most importantly the pressure coefficient. Finally, the results will be compared with more analytical methods, Newtonian Methods.

2.1. Preliminary Calculations

Circular Arch

The airfoil is defined as a 10% thick circular arch airfoil. Meaning that the curvature will be circular and the thickness to chord ratio will be 0.1. This can be visualized in Figure 1.

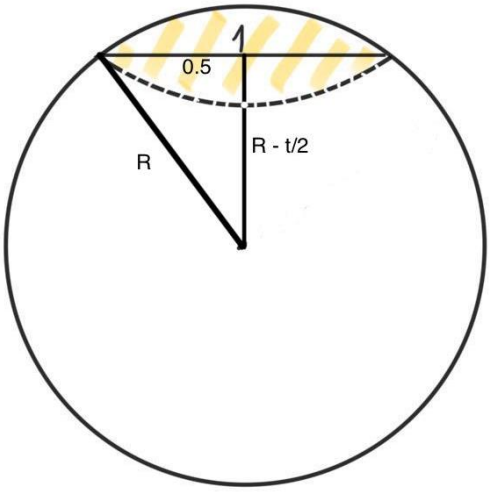


Figure 1: Circular Arch Airfoil

The first step is to determine the radius of the circle. Using Pythagorean's Theorem:

$$0.5^2 + (R - t/2)^2 = R^2$$

$$Rt = 0.5^2 + \frac{t^2}{4}$$

$$Rt = \frac{1}{4}(1 + t^2)$$

$$R = \frac{1 + t^2}{4t}$$

For t = 0.1:

$$R = 2.525$$

Now that we have obtained the value for the radius, we need to define the circular arch airfoil as a function y in terms of x. This can be derived by modifying the circle equation.

$$(y - y_0)^2 + (x - x_0)^2 = R^2$$

Taking Figure 1 as a reference, it is evident that if we take the origin of our coordinate system at the leading edge of the airfoil, we need to move the center of the circle to the right by a factor of 0.5 and down by a factor of (R – t/2). Applying these adjustments and isolating y results in the desired function:

$$y(x) = \sqrt{R^2 - (x - 0.5)^2} - (R - {}^{t}/_{2})$$

The slope of the tangent at a specified point will also be needed in the following calculations. This is a function with a simple analytical derivative; therefore, it is computationally efficient to derive it and place it in the code without computational methods.

$$\frac{dy}{dx} = \frac{0.5 - x}{\sqrt{R^2 - (x - 0.5)^2}}$$

Prandtl Meyer

$$\nu(M, \gamma) = \sqrt{\frac{\gamma + 1}{\gamma - 1}} arctan \sqrt{\frac{\gamma - 1}{\gamma + 1}(M^2 - 1)} - arctan\sqrt{M^2 - 1}$$

The Prandtl Meyer function is important for any aerodynamic calculations and it is even more critical in the Method of Characteristics. This is because to precede with the following calculations we need to be able to obtain the PM angle using Mach and obtain

Mach using the PM angle for a constant and known gamma. Calculating the angle from Mach is straight forward but the reverse requires numerical computation. For this computation the bisection method is preferred due to its efficiency and the limits of the Prandtl Meyer angle can easily be assumed.

Theta Beta Mach Relation

$$tan\theta = 2cot\beta \frac{M_1^2 \sin^2\beta - 1}{M_1^2(\gamma + cos2\beta) + 2}$$

The Theta-Beta-Mach relation is used to calculate the angle of a shock wave if the freestream Mach and the angle of the surface is known. It is also used for calculating the direction of a fluid flow directly behind an oblique shock wave. Therefore, similar to the Prandtl Meyer function. The revers of this relation must also be calculated. But first we need to calculate Theta using Beta. Mach is known.

$$\theta = tan^{-1}\left(2cot\beta \frac{M_1^2 \sin^2\beta - 1}{M_1^2(\gamma + cos2\beta) + 2}\right)$$

This same function is used alongside the bisection method to calculate Beta using Theta. The Bisection Method is useful in this situation since the upper limit of Beta is known to be 70 degrees and the lower limit is 0 degrees. Theta could also be used as the lower limit for increased computational efficiency.

Intersection Calculations

Calculating the intersection of various lines and functions is also vital for The Method of Characteristics. Firstly, calculating the location of an interior point requires the calculation of a point at the intersection of two lines. This could be done numerically, however, since there exists a relatively simple analytical solution exists, computational methods were avoided in this case.

The two linear lines can be written in the form of:

$$y - y_1 = m_1(x - x_1)$$
$$y - y_2 = m_2(x - x_2)$$

Define the intersection point of the two lines as (y_3, x_3) and solve for x_3:

$$(y_3)_1 = (y_3)_2 \;\rightarrow\; m_1(x_3 - x_1) + y_1 = m_2(x_3 - x_2) + y_2$$

Further algebraic simplifications result in:

$$x_3 = \frac{y_2 - y_1 - m_2 x_2 + m_1 x_1}{m_1 - m_2}$$

With the x location of the point is calculated, we can easily evaluate the y location by using either of the initial line equations above.

2.2. Interior Point

To calculate an interior point, we have two known points 1 and 2. The characteristic lines are not linear lines, they are curved. However, we can assume that they are straight if we assume the angle of that straight line to be the average of the slope of the characteristic lines at both the known points and the target point.

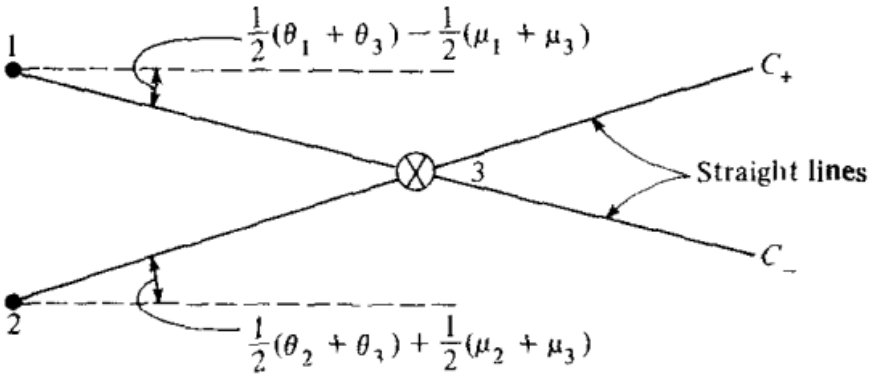


Figure 2: Interior Point Calculation

As seen above, we need to know the properties at point 3 to be able to identify the location of point 3. That can be done using the characteristic equations. First, we need to evaluate the Prandtl Meyer angle and the two known points.

$$\nu_1 = \nu(M_1, \gamma) \quad \nu_2 = \nu(M_2, \gamma)$$

Next, we can use the compatibility equations for the negative characteristic line at point 1 and the positive characteristic line at point 2.

$$\theta_1 + \nu_1 = \theta_3 + \nu_3$$

$$\theta_2 - \nu_2 = \theta_3 - \nu_3$$

Using these we can formulate the equations for the values at point 3

$$\theta_3 = \frac{(\theta_1 + \nu_1) + (\theta_2 - \nu_2)}{2}$$

$$\nu_3 = \frac{(\theta_1 + \nu_1) - (\theta_2 - \nu_2)}{2}$$

We found Theta and the Prandtl Meyer angle. Now we need to calculate the Mach number

$$\nu_3 = \nu(M_3, \gamma) \rightarrow M_3$$

Now that we know the Mach number at all three points, we can calculate their individual Mach angles using:

$$\mu = \sin^{-1}\left(\frac{1}{M}\right)$$

Using the Mach angles and the Theta angles at all three points, we can calculate the slope of the estimated characteristic lines.

$$m_1 = \tan^{-1}\left(\frac{(\theta_1 - \mu_1) + (\theta_3 - \mu_3)}{2}\right)$$

$$m_2 = \tan^{-1}\left(\frac{(\theta_2 + \mu_2) + (\theta_3 + \mu_3)}{2}\right)$$

In combination with coordinates of the initial known points, the slopes calculated above can be used to evaluate the equation for the two lines. The intersection of these two lines gives us the coordinates of the desired point.

2.3. Wall Point

The wall point calculations are more complex than the interior point calculations because they require some computational iterations to find the exact, or estimated solution with minimized error. There is one point known, point 4.

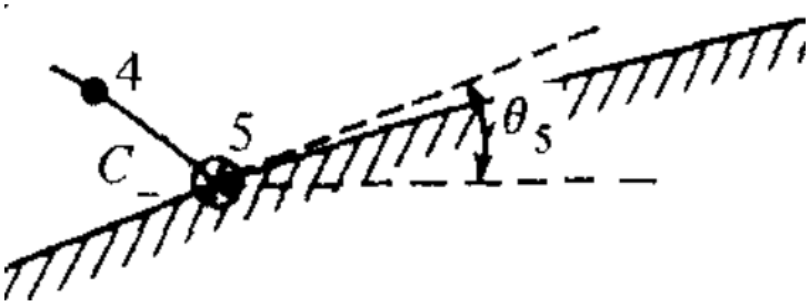


Figure 3: Wall Point

As demonstrated in Figure 3, we also use the known curvature of the surface and the fact that Theta is always tangent to the surface of wall points. Using this information, we first estimate the slope of the line connecting point 4 and the desired point 5 by the following equations:

$$\mu_4 = \sin^{-1}\left(\frac{1}{M_4}\right)$$

$$m_4 = \tan(\theta_4 - \mu_4)$$

$$y = m_4(x - x_4) + y_4$$

Using this line, we can calculate its intersection with the airfoil giving us an estimation for the position of point 5. The angle at that point, Theta 5, can be calculated using the derivative of the circular arch function.

$$\left.\frac{dy}{dx}\right|_{(x_5,y_5)} = \tan\theta_5$$

Using this angle and the compatibility equations we can calculate the Prandtl Meyer function at point 5.

$$\theta_4 + \nu_4 = \theta_5 + \nu_5$$

$$\nu_5 = \theta_4 + \nu_4 - \theta_5$$

The Prandtl Meyer angle allows us to calculate the Mach number at point 5:

$$\nu_5 = \nu(M_5, \gamma) \rightarrow M_5$$

With the Mach number, the Mach angle can be evaluated:

$$\mu_5 = \sin^{-1}\left(\frac{1}{M_5}\right)$$

Using this angle, we can make a more accurate estimation for the slope connecting the two lines.

$$m_4 = \tan\left(\frac{(\theta_4 - \mu_4) + (\theta_5 - \mu_5)}{2}\right)$$

This slope will be used again to estimate the position of point 5 and so on. This loop can be competed as many times as it takes to achieve an error value low enough to achieve sufficient accuracy.

2.4. Shock Point

Calculating a point on the shock wave is similar yet slightly more complex than calculating a point on the wall. Here we have a known point 6 that sits right below the shock wave. The desired point to be calculated is point 7. Furthermore, there is another point, point 8, which is a previous shock point or the leading edge. For point 6, all four critical variables are known. For point 8, the position and the shock wave angle at that point is known. The first step in calculating point 7 is to calculate the Prandtl Meyer function and the Mach angle at the known point 6.

$$\nu_6 = \nu(M_6, \gamma) \qquad \mu_6 = \sin^{-1}\left(\frac{1}{M_6}\right)$$

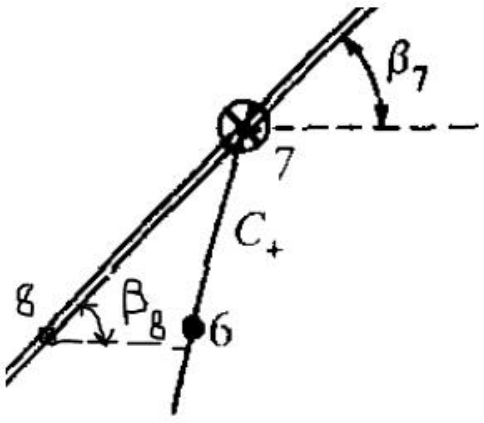


Figure 4: Shock Point

The next step involves an iterative process. The shock angle, Beta 7, and the Mach number, Mach 7, is unknown. There are two methods to calculate the Mach number using Beta 7: using the Characteristic Equations and using Shock Wave Relations. In this iteration, we will calculate Mach 7 with the two aforementioned methods, using an estimated value of Beta 7. Since the curvature of the shock wave is slightly downwards, it is expected that Beta 7 will be slightly less than Beta 5. Therefore, Beta 5 is an appropriate initial guess. The computational iteration is carried out using the bisection algorithm.

To use the Characteristic Equations method, we must first calculate Theta using Beta. This is done using the Theta-Beta-Mach Relations previously explained in the preliminary calculations section.

$$\upsilon_7 = \theta_7 - \theta_6 + \upsilon_6$$

Using the equation above, we obtain the Prandtl-Meyer function at point 7. With this, we can obtain the first estimation of Mach 7.

$$\nu_7 = \nu(M_7, \gamma) \rightarrow M_7$$

The first step when using Shockwave Relations to calculate Mach 7 is to calculate the Mach number of the freestream normal to the shock wave.

$$M_{n_\infty} = M_\infty * \sin\beta$$

Next, using Normal Shockwave Relations, we can calculate the Mach normal at point 7.

$$M_{n_7}^2 = \frac{(\gamma - 1)M_{n_\infty}^2 + 2}{2\gamma M_{n_\infty}^2 - (\gamma - 1)}$$

Using the Normal Mach at point 7 and the relevant angles, we can calculate Mach 7.

$$M_7 = \frac{M_{n_7}}{\sin(\beta_7 - \theta_7)}$$

Using the bisection-iteration method the difference between these two Mach numbers is minimized by finding the optimal value for Beta 7. Once Mach 7 is calculated the true position of point 7 can be calculated using an estimation of the slope that connects points 6 and 7.

$$\mu_7 = \sin^{-1}\left(\frac{1}{M_7}\right)$$

$$m_6 = \tan\left(\frac{(\theta_6 + \mu_6) + (\theta_7 + \mu_7)}{2}\right)$$

Using the intersection of the line formulated by the slope m_6 and the shockwave propagating from point 8.

2.5. Method Of Characteristics

The Method of Characteristics starts by initializing a set of N number of points. These points are initialized near the leading edge, under the shock wave and above the surface. To do this, an r is defined to represent its distance from the origin, the leading edge. Increasing the number of points increases accuracy. In one iteration, one set of points is used to calculate the next. There are two types of point sets, one being all interior points, the other set including one wall and one shock point. Therefore, the total number of points in each consecutive set will increase or decrease by one depending on the type of set. The flow properties of these points are all assumed to be the same and are calculated using Oblique Shockwave Relations.

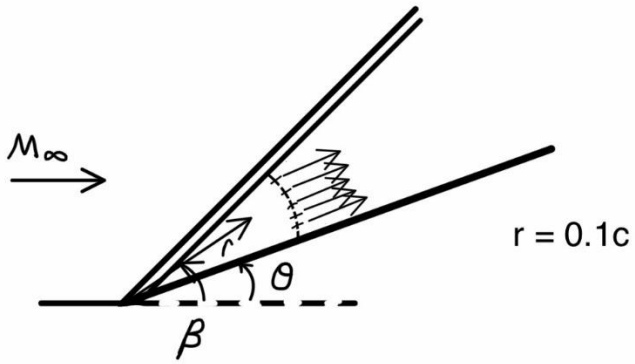


Figure 5: Initial points at leading edge

The initial points start very close to each other, however as the distance between the surface of the airfoil and the shockwave increases, the points inside the set gradually become spread apart, decreasing the accuracy. This can be counteracted by increasing the number of points to a sufficient degree. Increasing accuracy can also be achieved by decreasing the initial points' distance from the leading edge.

2.6. Pressure Temperature and Density

Pressure

Due to the irrotational flow assumption made previously, the flow after the shockwave can be accepted as isentropic. This assumption means that the total pressure of the flow will stay constant and will vary depending on the Mach number. The total pressure of the flow can be calculated using Oblique Shockwave Relations. The shock angle at the leading edge, Beta 0, can be calculated using Theta-Beta-Mach relations, allowing us to then calculate the total pressure.

$$M_{n_\infty} = M_\infty * \sin\beta$$

$$\frac{P_{t_0}}{P_{t_\infty}} = \left[\frac{(\gamma+1)M_{n_\infty}^2}{(\gamma-1)M_{n_\infty}^2+2}\right]^{\frac{\gamma}{\gamma-1}} \left[\frac{(\gamma+1)}{2\gamma M_{n_\infty}^2-(\gamma-1)}\right]^{\frac{1}{\gamma-1}}$$

Now that the total pressure is calculated, we can use the isentropic equation below to calculate the pressure at any point as long as the Mach number is known.

$$P = P_{t_0}\left(1+\frac{\gamma-1}{2}M^2\right)^{\frac{-\gamma}{\gamma-1}}$$

Temperature

Similar to the pressure calculations, calculating temperature involves using isentropic equations. The difference is that total temperature does not change across a shockwave unlike total pressure. Therefore, we can calculate the total temperature of the incoming free flow. Then we can use this total temperature to calculate the temperature of any point with a known Mach number.

$$T_{t_\infty} = T_\infty * \left(1+\frac{\gamma-1}{2}M_\infty^2\right)$$

$$T = \frac{T_{t_\infty}}{\left(1+\frac{\gamma-1}{2}M^2\right)}$$

Density

Since we now know pressure and temperature, we can use the Ideal Gas Law instead of the more involved method of using isentropic equations.

$$PV = nRT \quad \rightarrow \quad P = \rho RT$$

$$\rho = \frac{P}{RT} \qquad R = 8.314\ J/mol.K$$

2.7. Newtonian Methods

The Newtonian Methods use Newton's Laws of Motion to calculate an analytical solution for the pressure coefficient distribution on a surface under hypersonic flow. These methods result in equations mostly based off of the angle of the tangent at the point of interest on the surface. The Newtonian Method is as follows.

$$C_p = 2\sin^2\theta$$

Correcting this to a certain extent is The Modified Newtonian Method replacing the 2 with a calculated maximum Mach number that is expected to be achieved.

$$C_{p,max} = \frac{2}{\gamma M_\infty^2}\left\{\left[\frac{(\gamma+1)^2 M_\infty^2}{4\gamma M_\infty^2 - 2(\gamma-1)}\right]^{\frac{\gamma}{\gamma-1}}\left[\frac{1-\gamma+2\gamma M_\infty^2}{\gamma+1}\right] - 1\right\}$$

$$C_p = C_{p,max}\sin^2\theta$$

This can be further optimized by applying a centrifugal force correction that accounts for a fluid's tendency to stay in motion in the direction of its motion.

$$C_p = C_{p,max}\sin^2\theta_i + 2\left(\frac{d\theta}{dy}\right)_i sin\theta_i \int_0^{y_i} cos\theta dy$$

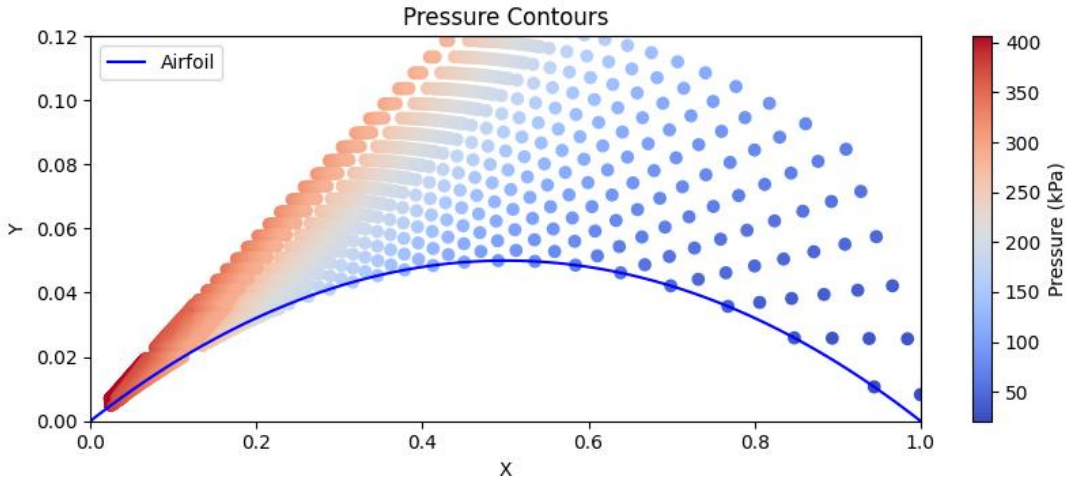
The integral portion can be analytically altered into the form:

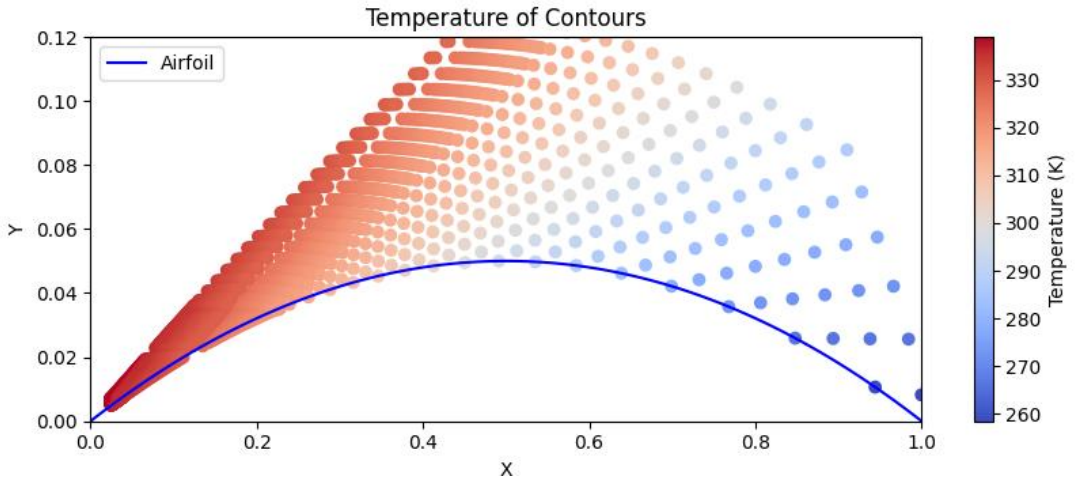
$$\int_0^{y_i} cos\theta dy = \int_0^{x_i} \cos\left(-\tan^{-1}\left(\frac{dy}{dx}\right)\right)\frac{dy}{dx}dx$$

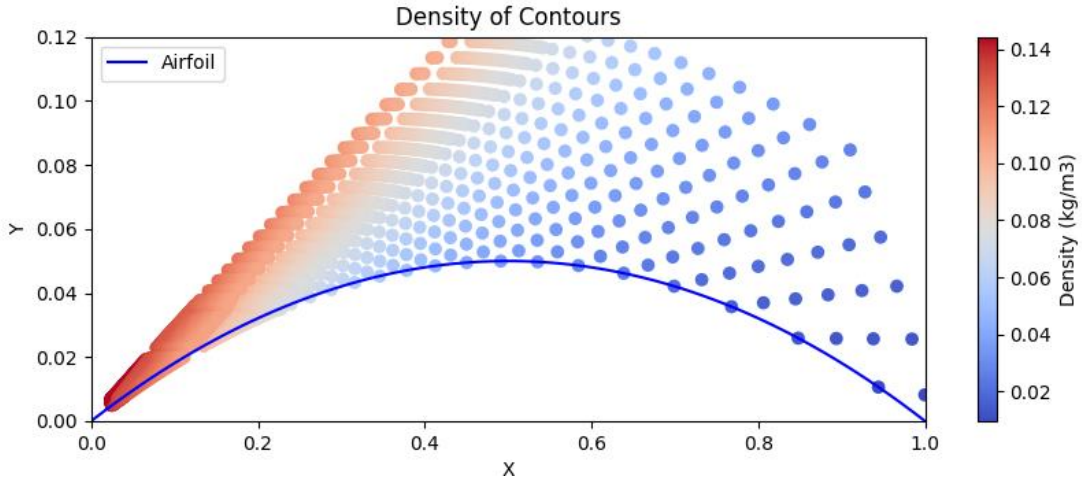
This improves the computational solvability of the Coefficient of pressure; dy/dx was already calculated in the preliminary calculations section, making this practically a function of x.

3. RESULTS

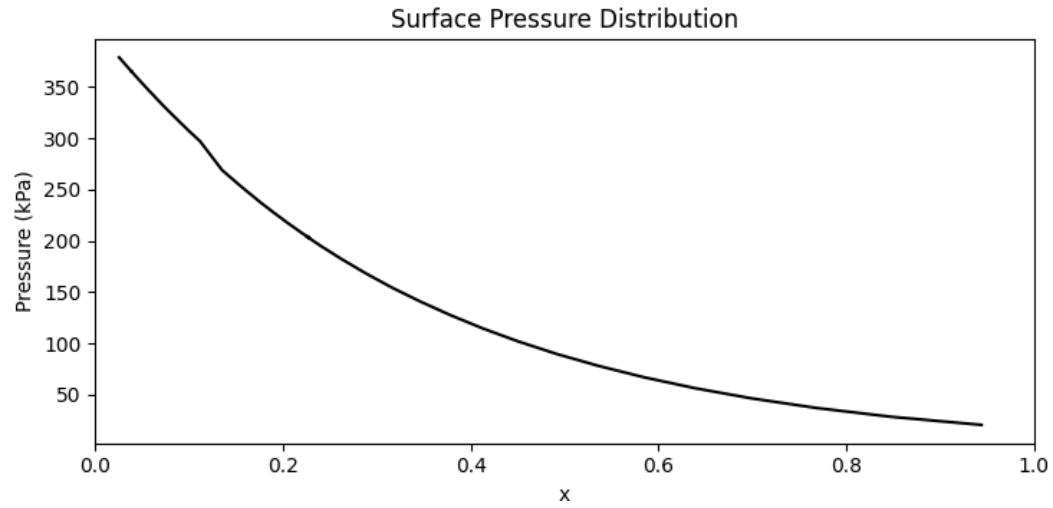
3.1. Contours

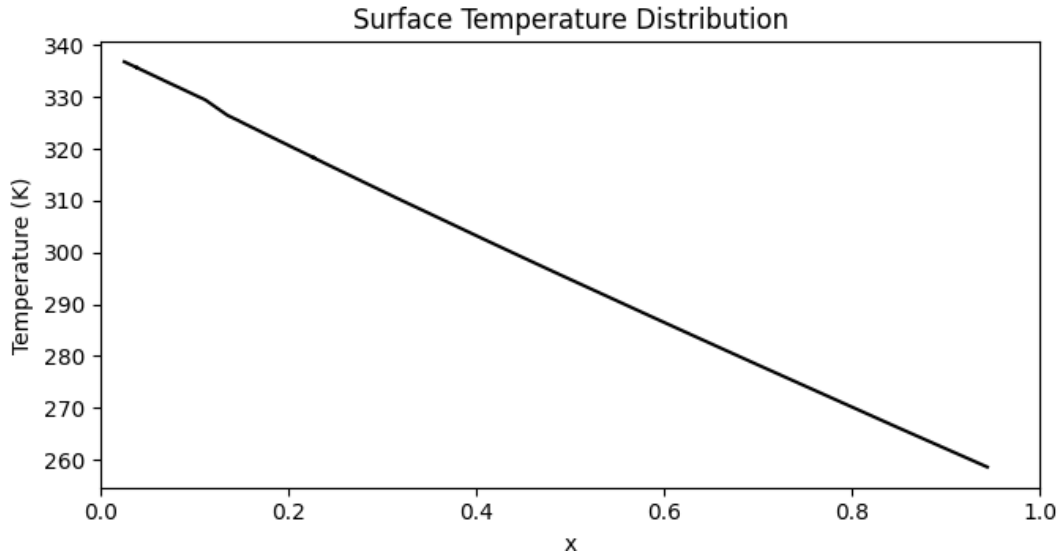


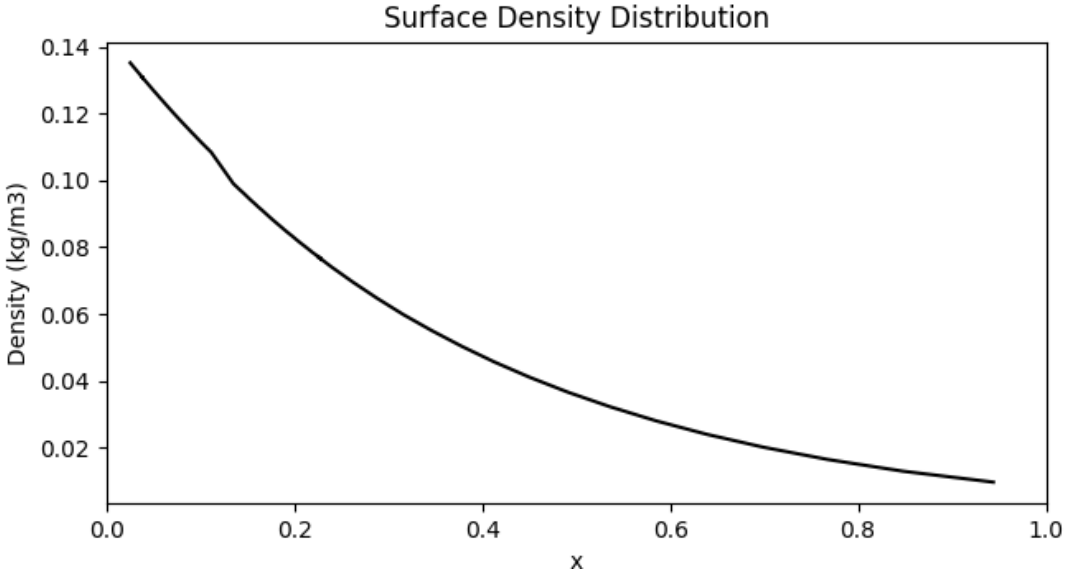




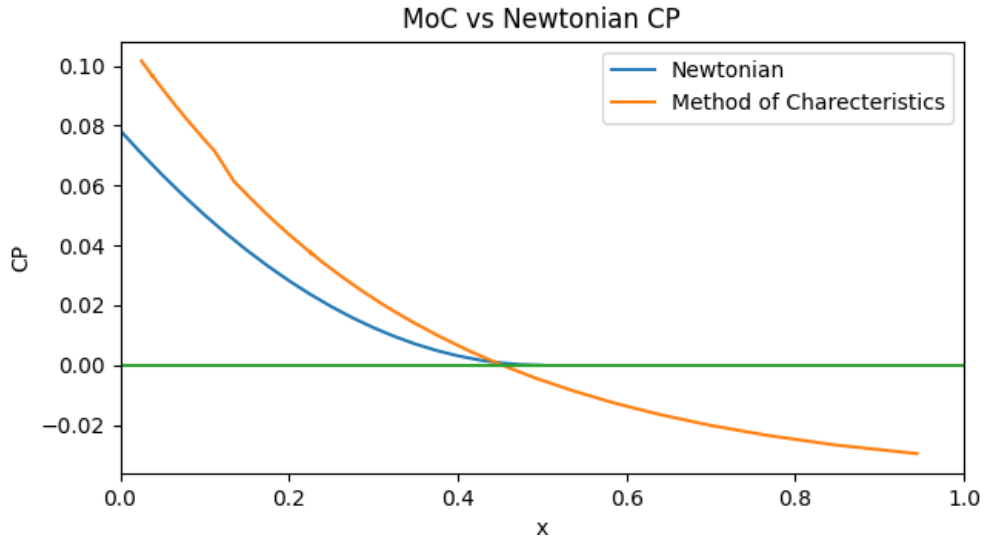
3.2. Surface Distributions

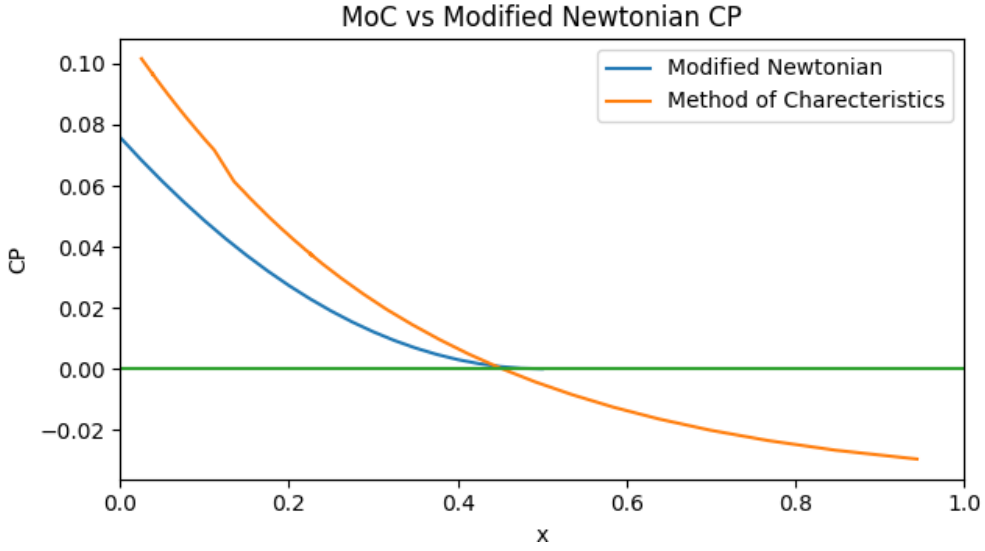


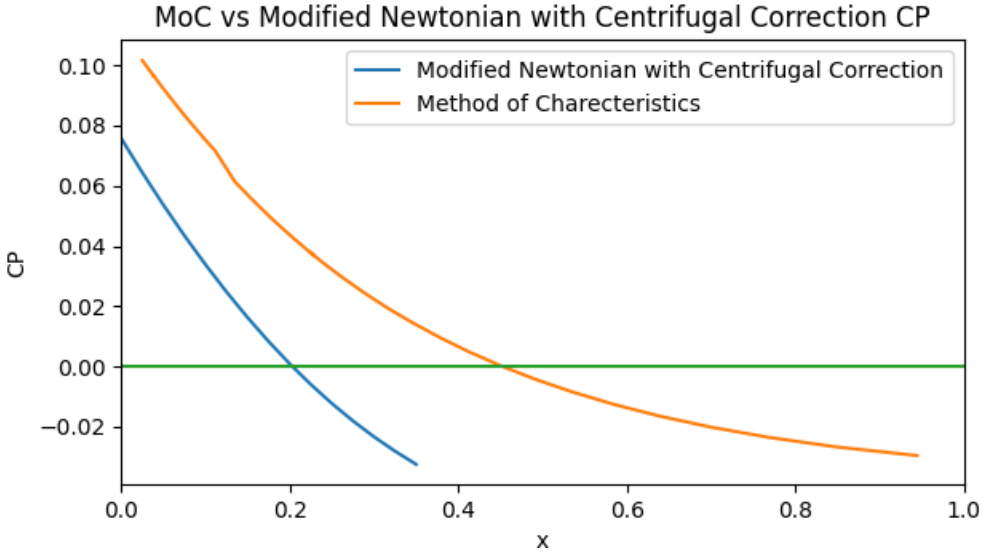




3.3. Pressure Coefficient Comparisons with Method of Characteristics







4. CONCLUSION

The Method of Characteristics is a useful and versatile tool that can be used to calculate various properties of a flow field. The calculations done in this project were done under the assumption that the flow is irrotational although the method of characteristics can also be used to calculate rotational flow. The results are not out of the ordinary. Most of the results are within the margins of expectation however there are some key findings that were significant enough to note in the conclusion.

The first unexpected nuance was the fact that the shock and wall points of each set are extended significantly more forward than the interior points. This makes it difficult to have a consistent spread of point within the control volume. The increased distance between the shock point and the next point below in, extend the interior point in the next set further than the rest of the points in that set. This phenomenon propagates down until it reaches the wall in which the created gap is even further evacuated by the extension of the wall point. This is reflected in the contour graphs where there are visible gaps between the points near the shock wave region. Number of points, edge clearance of outermost points, and initial points' origin distance were adjusted to minimize the effect of those gaps but loss of accuracy is inevitable.

The results of the surface distribution graphs were relatively expected. It was evident that pressure, temperature, and density would all decrease. It was slightly unexpected however that the graph of the decreasing temperature was relatively linear while the drop in pressure and density seemed parabolic. This is an interesting phenomenon.

The final unexpected detail was the fact that the pressure coefficient calculated from the Newtonian Method was much closed to the result of the Method of Characteristics than the Centrifugal Corrected Modified Newtonian Method, which is expected to be more accurate, hence closer to the Method of Characteristics values of the pressure coefficient.

To conclude, The Method of Characteristics is a versatile tool in aerodynamic calculations. Although it has its challenges such as the propagating gap, its accuracy is inevitably much higher than other methods such as the Newtonian Methods.

APPENDIX

```
import numpy as np
import math
import matplotlib.pyplot as plt

# Given variables from Student IDs
t = 0.1  # thickness ratio
M_inf = 7  # Freestream Mach number
gamma = 1.1  # Specific Heat Ratio

R = (1 + t**2)/(4*t)  # Radius of the Circle that defines the Circular Arch
Airfoil

# Initial Points Adjustments
N = 24  # number of initialized points
r = 0.025  # distance from origin (leading edge)
iterations = 400  # number of max loops in method of charecteristics
edge_clearance = 0.01  # edge clearence for the shock and wall

# Free Flow Characteristics (assume sea level ISA standards)
P_inf = 101.325  # pressure (kPa)
T_inf = 288  # temperature (K)
rho_inf = 1.225  # density (kg/m3)


def circular_arch(x):

    # Circular Arch Airfoil function
    y = np.sqrt(R**2 - (x-0.5)**2) - (R - (t/2))

    return y


def circular_arch_slope(x):

    # Derivative of the Circular Arch Airfoil Function
    dy_dx = (0.5-x)/(np.sqrt((R**2) - ((x-0.5)**2)))

    return dy_dx


def prandtl_meyer(M, gamma):
    # Prandtl-Meyer function
    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))


def find_mach_number(target_prandtl_meyer, gamma, a=0.1, b=30, tolerance=1e-6,
max_iterations=100):
    # Find the input value (Mach number) that produces the desired target Prandtl
Meyer
    # Using the bisection method
    if prandtl_meyer(a, gamma) == target_prandtl_meyer:
        return a
    if prandtl_meyer(b, gamma) == target_prandtl_meyer:
        return b

    for _ in range(max_iterations):
```

```
        # Calculate the midpoint and Prandtl Meyer at midpoint
        c = (a + b) / 2
        PM = prandtl_meyer(c, gamma)

        # Check if c is the solution
        if np.isclose(PM, target_prandtl_meyer, atol=tolerance):
            return c

        # Update the interval based on the sign of the function value at c
        if PM < target_prandtl_meyer:
            a = c
        else:
            b = c

    # If no solution is found within the maximum iterations
    raise RuntimeError(
        "Mach not found within the maximum number of iterations.")


def theta_from_beta(beta):

    # Theta Beta Mach Relation
    theta = math.atan(
        ((2/math.tan(beta))*(((M_inf**2)*math.sin(beta)**2-1)) /
((M_inf**2)*(gamma+math.cos(2*beta))+2)))

    return theta


def beta_from_theta(theta, a=1 * np.pi/180, b=70 * np.pi/180, tolerance=1e-9,
max_iterations=100):

    # Reversed Theta Beta Mach Relation using Bisection
    if theta_from_beta(a) == theta:
        return a
    if theta_from_beta(b) == theta:
        return b

    for _ in range(max_iterations):
        # Calculate the midpoint
        c = (a + b) / 2

        # Check if solution is close enough
        if np.abs(theta_from_beta(c) - theta) < tolerance:
            return c

        # Update the interval based on the value at c
        if theta_from_beta(c) < theta:
            a = c
        else:
            b = c

    # If no solution is found within the iterations
    return 0


def find_linear_intersection(x1, y1, m1, x2, y2, m2):
    # Intersection of two lines in the form of: y - y0 = m0(x - x0)
```

```
    # Solve the equations for x-coordinate
    x3 = (y2 - y1 - m2*x2 + m1*x1) / (m1 - m2)

    # Calculate the y-coordinate of the intersection point using either line
equation
    y3 = m1*(x3 - x1) + y1

    return x3, y3


def find_arch_intersection(x0, y0, m0, tolerance=1e-6, max_iterations=100):
    # Find the intersection point of the neg char line and circular arch using the
bisection method

    # The negative characteristic line
    def line(x):
        return m0*(x - x0) + y0

    a = 0.0  # Initial lower bound for x
    b = 1.0  # Initial upper bound for x

    for _ in range(max_iterations):
        c = (a + b) / 2  # midpoint of bounds

        # Evaluate the functions at the midpoint and adjust the interval
        if line(c) - circular_arch(c) < 0:
            b = c
        else:
            a = c

        # Check for convergence
        if abs(line(c) - circular_arch(c)) < tolerance:
            return c, line(c)

    # If no solution is found within the iterations
    # This indicates thet the calculation point has passed the trailing edge
    return 0, 0


def interior_point(x1, y1, M1, Theta1, x2, y2, M2, Theta2):

    # Find Mach Anges
    Mangle1 = np.arcsin(1 / M1)
    Mangle2 = np.arcsin(1 / M2)

    # Find Prandtl Meyer angles
    PM1 = prandtl_meyer(M1, gamma)
    PM2 = prandtl_meyer(M2, gamma)

    # Find k konstant for characteristic lines from compatibility equations
    K1neg = Theta1 + PM1
    K2pos = Theta2 - PM2

    # Use these constants to find PM and Theta for point 3
    Theta3 = (K1neg + K2pos)/2
    PM3 = (K1neg - K2pos)/2

    # Use Prandtl-Meyer3 to find Mach Angle 3
    M3 = find_mach_number(PM3, gamma)
```

```
    Mangle3 = np.arcsin(1/M3)

    # Find slope of lines connecting the points
    m1 = np.arctan(((Theta1 - Mangle1) + (Theta3 - Mangle3)) / 2)
    m2 = np.arctan(((Theta2 +
                     Mangle2) + (Theta3 + Mangle3)) / 2)

    # Find the intersection: x3 and y3\
    x3, y3 = find_linear_intersection(x1, y1, m1, x2, y2, m2)

    return x3, y3, M3, Theta3


def wall_point(x4, y4, M4, Theta4, max_iterations=100):

    # Calculate Mach Angle and Prandtl-Meyer angle at point 4
    Mangle4 = np.arcsin(1/M4)
    PM4 = prandtl_meyer(M4, gamma)

    # Initial guess for point 5 values
    M5 = M4
    Theta5 = Theta4
    Mangle5 = np.arcsin(1 / M5)

    # Loop to find the location of point 5
    for _ in range(max_iterations):
        m4 = np.tan(((Theta4 - Mangle4) + (Theta5 - Mangle5)) / 2)

        x5, y5 = find_arch_intersection(x4, y4, m4)
        if x5 == 0:
            return 0, 0, 0, 0

        Theta5 = np.arctan(circular_arch_slope(x5))

        PM5 = Theta4 + PM4 - Theta5
        M5 = find_mach_number(PM5, gamma)
        if (abs(Mangle5 - np.arcsin(1 / M5)) < 1e-9):
            return x5, y5, M5, Theta5
        Mangle5 = np.arcsin(1 / M5)

    raise RuntimeError(
        "No solution found within the maximum number of iterations .")


def mach_difference_given_beta(Theta6, PM6, Beta7):
    # The Difference between the Mach number calculated with the two methods

    # Calculate M7 using the charecteristic Equations
    Theta7 = theta_from_beta(Beta7)
    PM7 = Theta7 - Theta6 + PM6
    M7_char = find_mach_number(PM7, gamma)

    # Calculate M7 using shock wave relations
    Mn_inf = M_inf*math.sin(Beta7)
    Mn_7 = np.sqrt(((gamma-1)*(Mn_inf**2) + 2) /
                   ((2*gamma*Mn_inf**2) - (gamma-1)))
    M7_shock = Mn_7 / math.sin(Beta7 - Theta7)

    # The Difference:
```

```
    return M7_char - M7_shock


def shock_point(x6, y6, M6, Theta6, x8, y8, Beta8, max_iterations=100,
tolerance=1e-6):

    # Calculate Prantdl Meyer angle and Mach angle
    PM6 = prandtl_meyer(M6, gamma)
    Mangle6 = np.arcsin(1 / M6)

    # Beta 7 will be calculated with Bisection Method
    # Beta 7 will be around Beta8
    a = Beta8 + (1 * math.pi/180)  # upper bound for Beta 7
    b = Beta8 - (4 * math.pi/180)  # lower bound for Beta 7

    # Check if Bisection method is viable
    if np.sign(mach_difference_given_beta(Theta6, PM6, a)) ==
np.sign(mach_difference_given_beta(Theta6, PM6, b)):
        raise ValueError(
            "Error values at a and b must have opposite signs.")

    # Check the validity of the boundaries first
    if mach_difference_given_beta(Theta6, PM6, a) == 0:
        Beta7 = a
    if mach_difference_given_beta(Theta6, PM6, b) == 0:
        Beta7 = b
    else:
        # Bisection Loop:
        for _ in range(max_iterations):
            c = (a + b) / 2
            Beta7 = c

            if np.isclose(mach_difference_given_beta(Theta6, PM6, c), 0,
atol=tolerance):
                break

            if np.sign(mach_difference_given_beta(Theta6, PM6, c)) ==
np.sign(mach_difference_given_beta(Theta6, PM6, a)):
                a = c
            else:
                b = c

    # Calculate Theta for determined Beta angle and freeflow Mach
    Theta7 = theta_from_beta(Beta7)

    # Calculate Mach 7 using charecteristic equations
    PM7 = Theta7 - Theta6 + PM6
    M7 = find_mach_number(PM7, gamma)

    # Mach angle at point 7
    Mangle7 = np.arcsin(1/M7)

    # angle of linearized char line connecting point 6 and 7
    m6 = ((Theta6 + Mangle6) + (Theta7 + Mangle7))/2

    # Intersection of Shock Wave and char line
    x7, y7 = find_linear_intersection(x6, y6, m6, x8, y8, np.tan(Beta8))

    return x7, y7, M7, Theta7, Beta7
```

```
def Method_of_Characteristics():
    # The first two sets of points will be initialized before the main loop
    x = np.zeros([2, N])
    y = np.zeros([2, N])
    M = np.zeros([2, N])
    Theta = np.zeros([2, N])

    # The shock angles will be kept track of in a seperate array
    Beta = np.zeros(1)

    # Define Theta and Beta at the leading edge
    Theta0 = np.arctan(circular_arch_slope(0))
    Beta0 = beta_from_theta(Theta0)

    # Calculate Mach using oblique shock wave relations
    Mn_inf = M_inf*math.sin(Beta0)
    Mn_0 = math.sqrt(((gamma-1)*(Mn_inf**2) + 2) /
                     ((2*gamma*Mn_inf**2) - (gamma-1)))
    M0 = Mn_0 / math.sin(Beta0 - Theta0)

    # # # Calculate the minimum and maximum angle for the initial points # # #
    # Theta min is at the point on the airfoil with r distance from origin

    def difference_between_arch_y(x):
        y_r = np.sqrt(r**2 - x**2)
        y_airfoil = circular_arch(x)
        return y_r - y_airfoil

    # Calculate X min using bisection method
    a = 0
    b = r
    for _ in range(50):
        # Calculate the midpoint
        X_min = (a + b) / 2

        # Check if c is the solution
        if np.abs(difference_between_arch_y(X_min)) < 1e-9:
            break

        # Update the interval based on the sign of the function value at c
        if difference_between_arch_y(X_min) > 0:
            a = X_min
        else:
            b = X_min

    # Define the min and max angles for initial points
    Theta_min = np.arccos(X_min/r)
    Theta_max = Beta0

    # Initialize the first set of points all interior points
    for i in range(N-1):

        # evenly distributed points in a circular arch with defined edge clearance
        x[0][i] = r * np.cos(Theta0 + (((Theta_max - Theta_min) *
                                        (i + edge_clearance)) / ((N-1) + (2 *
edge_clearance))))
        y[0][i] = r * np.sin(Theta0 + (((Theta_max - Theta_min) *
```

```
                                        (i + edge_clearance)) / ((N-1) + (2 *
edge_clearance))))

        # The initial points assumed to have the Mach and Theta of the leading
edge
        M[0][i] = M0
        Theta[0][i] = Theta0

    # Second set of points will include one wall and one shock point
    for i in range(N):
        if i == 0:  # Lower-most point is a wall point
            x[1][i], y[1][i], M[1][i], Theta[1][i] = wall_point(
                x[0][i], y[0][i], M[0][i], Theta[0][i])
        elif i == N-1:  # Upper-most point is a shock point
            x[1][i], y[1][i], M[1][i], Theta[1][i], Beta[0] = shock_point(
                x[0][i-1], y[0][i-1], M[0][i-1], Theta[0][i-1], 0, 0, Beta0)
        else:  # Others are all interior points
            x[1][i], y[1][i], M[1][i], Theta[1][i] = interior_point(
                x[0][i], y[0][i], M[0][i], Theta[0][i], x[0][i-1], y[0][i-1],
M[0][i-1], Theta[0][i-1])

    # Has the latest wall point surpassed the trailing edge?
    Passed_Trailing_Edge = False

    # j represents the set of points used to calculate the next set
    for j in range(1, iterations):
        x_next = np.zeros(N)
        y_next = np.zeros(N)
        M_next = np.zeros(N)
        Theta_next = np.zeros(N)

        Beta_next = 0

        # Point sets with an even j have one wall and one shock point
        if j % 2 == 0:
            for i in range(N):
                if i == 0:
                    x_next[i], y_next[i], M_next[i], Theta_next[i] = wall_point(
                        x[j][i], y[j][i], M[j][i], Theta[j][i])
                    if x_next[i] == 0:
                        # Check if the edge is surpassed
                        Passed_Trailing_Edge = True
                        break
                elif i == N-1:
                    x_next[i], y_next[i], M_next[i], Theta_next[i], Beta_next =
shock_point(
                        x[j][i-1], y[j][i-1], M[j][i-1], Theta[j][i-1], x[j-1][i],
y[j-1][i], Beta[(int(j/2)-1)])
                    Beta = np.append(Beta, Beta_next)
                else:
                    x_next[i], y_next[i], M_next[i], Theta_next[i] =
interior_point(
                        x[j][i], y[j][i], M[j][i], Theta[j][i], x[j][i-1], y[j][i-
1], M[j][i-1], Theta[j][i-1])
            if Passed_Trailing_Edge:
                # Break out of the loop if the airfoil has been completely
calculated
                break
        # Point sets with an odd j have only interior points
```

```
        else:
            for i in range(N - 1):
                x_next[i], y_next[i], M_next[i], Theta_next[i] = interior_point(
                    x[j][i+1], y[j][i+1], M[j][i+1], Theta[j][i+1], x[j][i],
y[j][i], M[j][i], Theta[j][i])

        # Add the next set of points into the matrices
        x = np.vstack((x, x_next))
        y = np.vstack((y, y_next))
        M = np.vstack((M, M_next))
        Theta = np.vstack((Theta, Theta_next))

    if Passed_Trailing_Edge:
        print("Loop DID reach the trailing edge")
    else:
        print("Loop DID NOT reach the trailing edge")

    return x, y, M, Theta


def calculate_pres_temp_dens(x, y, M, Theta):
    # Assume flow is isentropic

    # Freestream total pres and temp (isentropic relations)
    Pt_inf = P_inf * np.power((1 + ((gamma - 1) / 2)
                               * M_inf**2), (gamma / (gamma - 1)))
    Tt_inf = T_inf * (1 + ((gamma - 1) / 2) * M_inf**2)

    # Calculate Mach Normal to Leading Edge Shock Wave
    Theta0 = np.arctan(circular_arch_slope(0))
    Beta0 = beta_from_theta(Theta0)
    Mn_inf = M_inf*math.sin(Beta0)

    # Total pres and temp after shockwave
    Pt_0 = Pt_inf
    Pt_0 *= np.power((((gamma + 1) * Mn_inf**2) /
                     (((gamma - 1) * Mn_inf**2) + 2)), (gamma / (gamma - 1)))
    Pt_0 *= np.power(((gamma + 1) / ((2 * gamma * Mn_inf**2) -
                     (gamma - 1))), (1 / (gamma - 1)))

    Tt_0 = Tt_inf

    # Use Isentropic equations to calculate pres and temp at all points
    P = Pt_0 / np.power((1 + ((gamma - 1) / 2) *
                         np.power(M, 2)), (gamma / (gamma - 1)))
    T = Tt_0 / (1 + ((gamma - 1) / 2) * np.power(M, 2))

    # Ideal Gas equation to calculate density
    R = 8.314  # Ideal gas constant in J/(mol·K)

    rho = (P / (R * T))

    # Delete empty points
    P[np.where(M == 0)] = 0
    T[np.where(M == 0)] = 0
    rho[np.where(M == 0)] = 0

    return P, T, rho
```

```
def surface_distribution(A):
    # extract wall points of matrix A

    # initialize array
    a = np.zeros(0)

    for i in range(len(A)-1):
        if (A[i][N-1] == 0):
            a = np.append(a, A[i+1, 0])
    return a


def newtonian_methods(t, M, gamma):

    R = (1 + t**2)/(4*t)

    # Defining Theta in terms of Chord (0 -> 1)
    x = np.linspace(0, 1, 41)  # x = [0, 0.025, 0.05 ... 0.95, 0.975, 1]
    Theta = np.zeros(0)
    for i in range(len(x)):
        Theta = np.append(Theta, -math.asin((x[i]-0.5)/R))

    # Newtonian CP Calculation
    CP_newtonian = np.zeros(0)
    for i in range(len(Theta)):
        CP_newtonian = np.append(
            CP_newtonian, 2*math.pow(math.sin(Theta[i]), 2))

    # Find CP max for modification
    CP_max = (2/(gamma*M**2))*((math.pow((((gamma+1)**2)*(M**2)) /
                                         (4*gamma*M**2-2*(gamma-1)), gamma/(gamma-
1))*((1-gamma+2*gamma*M**2)/(gamma+1)))-1)

    # Modified Newtonian CP Calculation
    CP_modified = np.zeros(0)
    for i in range(len(Theta)):
        CP_modified = np.append(CP_modified, CP_max *
                                math.pow(math.sin(Theta[i]), 2))

    # Determine y coordinate of points using Theta
    gamma = np.zeros(0)
    for i in range(len(Theta)):
        gamma = np.append(
            gamma, (R*math.cos(Theta[i])) - (R-(t/2)))

    # Determine the derivative of Theta
    dTheta_dy = np.zeros(0)
    for i in range(len(Theta)):
        if i < len(Theta)/2:
            dTheta_dy = np.append(
                dTheta_dy, (Theta[i+1] - Theta[i])/(gamma[i+1] - gamma[i]))
        else:
            dTheta_dy = np.append(
                dTheta_dy, (Theta[i] - Theta[i-1])/(gamma[i] - gamma[i-1]))

    # Centrifugal Corrected CP Calculation
    CP_centrifugal = np.zeros(0)
    CP_mod_centrifugal = np.zeros(0)
```

```
    for i in range(len(Theta)):
        I = 0  # the integration term must be calculated with a loop
        for j in range(i):
            I += math.cos(Theta[i]) * (gamma[j+1]-gamma[j])
        correction = 2*dTheta_dy[i]*math.sin(Theta[i])*I
        CP_centrifugal = np.append(
            CP_centrifugal, CP_newtonian[i] + correction)
        CP_mod_centrifugal = np.append(
            CP_mod_centrifugal, CP_modified[i] + correction)

    return x[0:21], CP_newtonian[0:21], CP_modified[0:21],
CP_mod_centrifugal[0:21]


def CP_Method_of_Characteristics(gamma, P_inf, P, M):
    CP_MoC = (P-P_inf) / ((gamma * P_inf * M**2) / 2)
    return CP_MoC


def plot_contours(x, y, M, title, label):
    # Plot contour in the area below the shockwave

    # Calculate coordinates for the Airfoil
    x_circular_arch = np.linspace(0, 1, 100)
    y_circular_arch = circular_arch(x_circular_arch)

    # Create a figure and axis
    fig, ax = plt.subplots()

    # Plot The Airfoil
    ax.plot(x_circular_arch, y_circular_arch, color='blue', label='Airfoil')

    # turn matricies into arrays for easy plotting
    x = x.flatten()
    y = y.flatten()
    M = M.flatten()

    # Remove the zero points
    x = x[np.nonzero(x)]
    y = y[np.nonzero(y)]
    M = M[np.nonzero(M)]

    # Calculate min and max values
    vmin = np.min(M)
    vmax = np.max(M)

    # Define the colormap
    cmap = plt.cm.get_cmap('coolwarm')

    # Plot the points with color-coded temperatures
    plt.scatter(x, y, c=M, cmap=cmap, vmin=vmin, vmax=vmax)

    # Set colorbar
    cbar = plt.colorbar()
    cbar.set_label(label)

    # Set labels and title
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
```

```
    ax.set_title(title)

    # Add legend
    ax.legend(loc='upper left')

    # Show the plot
    plt.ylim(0, 0.12)
    plt.xlim(0, 1)
    plt.show()


def plot_surface_distribution(x, distribution, title, label):

    # Create a scatter plot with lines connecting the points
    plt.plot(x, distribution, 'k-')
    plt.scatter(x, distribution, marker='None')

    # Set the axis labels
    plt.xlabel('x')
    plt.ylabel(label)
    plt.title(title)

    plt.xlim([0, 1])

    # Show the plot
    plt.show()


def plot_two_surface_distribution(x1, distribution1, x2, distribution2, title,
axis_label, label1, label2):
    # Create a scatter plot with lines connecting the points
    plt.plot(x1, distribution1, label=label1)
    plt.plot(x2, distribution2, label=label2)
    plt.plot([0, 1], [0, 0])

    # Set the axis labels
    plt.xlabel('x')
    plt.ylabel(axis_label)
    plt.title(title)

    plt.xlim([0, 1])

    # Add legend
    plt.legend(loc="upper right")

    # Show the plot
    plt.show()


# Calculate points in the control volume and their properties
x, y, M, Theta = Method_of_Characteristics()
P, T, rho = calculate_pres_temp_dens(x, y, M, Theta)

# Plot the Contours of the Properties in the Control Volume
plot_contours(x, y, M, 'Mach Number Contours', 'Mach Number')
plot_contours(x, y, P, 'Pressure Contours', 'Pressure (kPa)')
plot_contours(x, y, T, 'Temperature of Contours', 'Temperature (K)')
plot_contours(x, y, rho, 'Density of Contours', 'Density (kg/m3)')
```

```
# Extract the fluid proporties on the surface of the airfoil
x_surface = surface_distribution(x)
M_surface = surface_distribution(M)
P_surface = surface_distribution(P)
T_surface = surface_distribution(T)
rho_surface = surface_distribution(rho)

# Plot these Proporties as Surface Distributions
plot_surface_distribution(x_surface, P_surface,
                          'Surface Pressure Distribution', 'Pressure (kPa)')
plot_surface_distribution(
    x_surface, T_surface, 'Surface Temperature Distribution', 'Temperature (K)')
plot_surface_distribution(x_surface, rho_surface,
                          'Surface Density Distribution', 'Density (kg/m3)')

# Calculate the Coefficient of Pressure with values from MoC
CP_MoC = CP_Method_of_Characteristics(gamma, P_inf, P_surface, M_inf)

# Calculate CP with the three newtonian methods
X_newtonian, CP_newtonian, CP_modified, CP_mod_centrifugal = newtonian_methods(
    t, M_inf, gamma)

# Plot these methods in comparison
plot_two_surface_distribution(X_newtonian, CP_newtonian, x_surface, CP_MoC,
                              "MoC vs Newtonian CP", "CP", 'Newtonian', 'Method of
Charecteristics')
plot_two_surface_distribution(X_newtonian, CP_modified, x_surface, CP_MoC,
                              "MoC vs Modified Newtonian CP", "CP", 'Modified
Newtonian', 'Method of Charecteristics')
plot_two_surface_distribution(X_newtonian[0:15], CP_mod_centrifugal[0:15],
x_surface, CP_MoC, "MoC vs Modified Newtonian with Centrifugal Correction CP",
                              "CP", 'Modified Newtonian with Centrifugal
Correction', 'Method of Charecteristics')
```