Task II

Method of Characteristics for an Exhaust Jet Flow

Lorenz Veithen

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF AEROSPACE ENGINEERING

AE4140

GAS DYNAMICS

Task II

Method of Characteristics for an Exhaust Jet Flow

Lecturer: Ferdinand Schrijer

Name	**Student Number**
Lorenz Veithen	5075211

In the following discussion, Figure 0.1 will be used as reference.

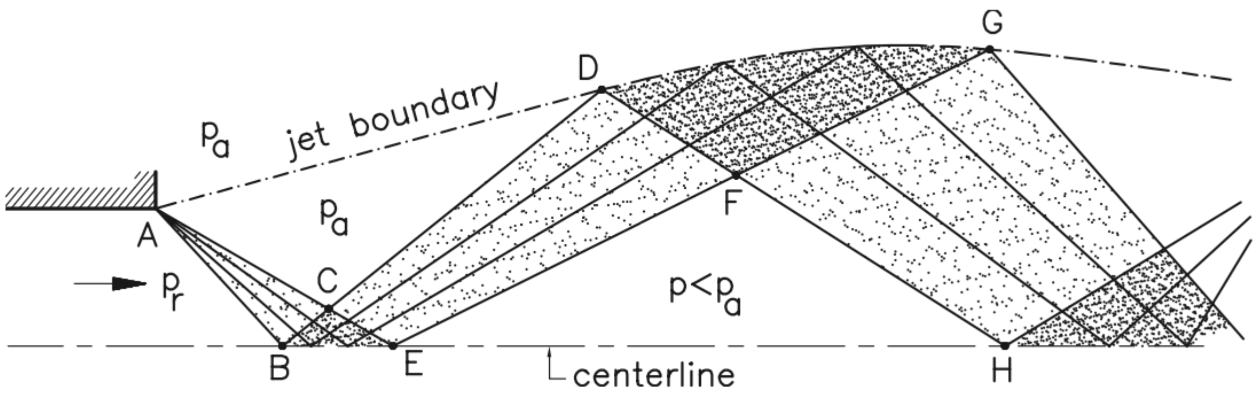


Figure 0.1: Jet exhaust reference image with three characteristics, as given in the assignment.

1) Problem Definition

First, the boundary conditions of the jet exhaust are determined. All along the centreline, the flow needs to be parallel to the horizontal, giving $\phi_c = 0°$ along the centreline. Following, a contact discontinuity is present along the jet boundary, meaning that the same pressure and velocity direction are present on both sides of the boundary. The boundary condition all along the jet boundary condition is then $p_{jet} = p_a$, which translates to the velocity profile through,

$$M_a = \sqrt{\left(\left(1 + (\gamma - 1)\frac{Me^2}{2}\right)\left(\frac{p_e}{p_a}\right)^{\frac{\gamma-1}{\gamma}} - 1\right)\frac{2}{\gamma - 1}} \tag{1}$$

Note that the subscript 'e' refers to the outlet conditions. This equation results from the isentropic expansion from the outlet to region ACD in Figure 0.1, as the total pressure stays constant and using equation (8.42) from Anderson[1]. To work with the characteristics, it is more convenient to express this boundary condition through its related Prandtl-Meyer angle ν_a, which is given in general from Equation 2[1]

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

Note that the upper jet boundary is not defined prior to the solving and will be delineated by the flow interactions, as explained later below. The deflection angle (and hence the slope of the first jet boundary), is given by $\phi_a = \nu_a - \nu_e + \phi_e$. The initial conditions to the problem are given by the outlet conditions at A: $M_e = 2$ and $phi_e = 0°$, and the pressure ratio $\frac{p_e}{p_a} = 2$. The boundary conditions are considered in lines 144 to 149 of the code shown qt the end of this report.

2) Initial Characteristics

The first expansion wave is entirely defined by the number of characteristics selected by the user, which defines a $\Delta\phi = \frac{\phi_a - \phi_e}{N-1}$ from one characteristic to another. This value is constant along the characteristic until the first interaction, as this is a simple wave region (all Γ^+ characteristics arise from a uniform region). The deflection angle of characteristic i (from 0 to N-1), among N in total, is then given by $\phi_i = i \cdot \Delta\phi + \phi_e$. The associated Prandtl-Meyer angle is then given by $\nu_i = \nu_e + \phi_i - \phi_e$ (Γ^+ characteristics going from the region before the expansion to region ACD). This is shown in the lines 151 to 159 of the code shown at the end of this report

The method to find the slope of the characteristics in that region is then given by $\alpha_i = -(\mu_i - \phi_i)$ (with $\sin\mu = \frac{1}{M}$, the Mach angle), as they are Γ^- characteristics. The characteristics being straight, those slopes are exact in the simple region. For other simple regions in the flow field, such as CEDF Figure 0.1, the characteristics slopes are given by $\alpha_i = (\mu_i + \phi_i)$ for Γ^+ characteristics.

[1]Anderson, J. (2011). EBOOK: Fundamentals of Aerodynamics (SI units). McGraw Hill.

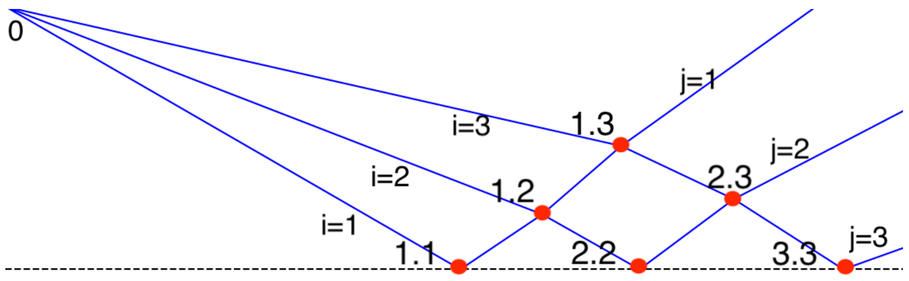


Figure 0.2: Interaction between the centre line and incoming characteristics. Note: this is just a sketch and is not supposed to represent a solution in any shape or form.

3) Data Handling and Non-Simple Regions Computations

A central part to the inner workings of the solver comes from the way the data is handled throughout the code. At each interaction region, a set of four matrices are generated: [X], [Y], [M], [ϕ], which store the values of the respective variables (x position, y position, Mach number, and deflection angle, respectively) at the characteristic intersections. As an example, the [X] matrix of x position of the intersections relative to Figure 0.2 is given by Equation 3.

$$[X] = \begin{bmatrix} x_{0.1} & x_{1.1} & 0 & 0 \\ x_{0.2} & x_{1.2} & x_{2.2} & 0 \\ x_{0.3} & x_{1.3} & x_{2.3} & x_{3.3} \end{bmatrix} \tag{3}$$

Where the first column gives the values relative to the incoming characteristic point of origin (such as point A in Figure 0.1 for the first expansion, or any point between C and E for the second), prior to any interaction, while the other three columns are relative to the intersection points between Γ^+ and Γ^- characteristics within the non-simple interaction region. The [Y], [M], and [ϕ] matrices take the same form. The powerfulness of this data storage method becomes clear when the interactions are computed. For such interactions, the general method consists in first computing the solution at the point P (resulting from the interaction), and then computing its position, and then storing the value. Different cases can be distinguished for both of those aspects.

The following cases can be distinguished with respect to the solution a point P:

1. **Interaction with the centreline**: the solution can be computed from the boundary condition $\phi = \phi_c = 0$ and the constant Riemann invariant along Γ^- characteristics (only ones that reach the centreline). This gives, $\nu_p = \nu + \phi - \phi_c$ and $\phi_p = \phi_c$, where $J^- = \nu + \phi$ is the Riemann invariant along the incoming Γ^- characteristic.

2. **Interaction with the jet boundary**: the solution can be computed from the boundary condition $\nu = \nu_a$ along the jet contact discontinuity and the Riemann invariant along Γ^+ characteristics (only ones reaching the jet boundary). This gives, $\phi_{jet} = \nu_a - \nu + \phi$, where $J^+ = \nu - \phi$ is the Riemann invariant along the incoming Γ^+ characteristic. Furthermore, this ϕ_{jet} is then also the new slope of the jet boundary after the interaction, which will be used for the interaction with the next incoming characteristic.

3. **Interaction between two characteristics in the non-simple region**: the solution can be obtained from the Riemann invariants of both the Γ^- and Γ^+ characteristics, following the formulas below. A and B refer to the points in Figure 0.3. Using the data handling method shown above, for a non-simple region on the centreline, the point j,i is formed by a Γ^- characteristic from point j-1, i and a Γ^+ characteristic from point j, i-1 (for $x_{j,i}$ in the matrix [X] shown above). This is reversed for a non-simple region at the jet boundary (a Γ^+ characteristic comes from point j-1, i; and a Γ^- from j, i-1).

$$\nu_P = \frac{1}{2}(\nu_B + \nu_A) + \frac{1}{2}(\phi_B - \phi_A) \tag{4}$$

$$\phi_P = \frac{1}{2}(\phi_B + \phi_A) + \frac{1}{2}(\nu_B - \nu_A) \tag{5}$$

It should be kept in mind that the values on positions 0, i's in the matrices of the type of [X] are known from the previous interaction, or the initial expansion fan, and that values j, i=j are subject to the boundary conditions mentioned above.

Considering the position of the point P, the general method relies on the intersection of the Γ^+ and Γ^- characteristics, however, two main cases can be distinguished when considering the characteristics from A and B from Figure 0.3:

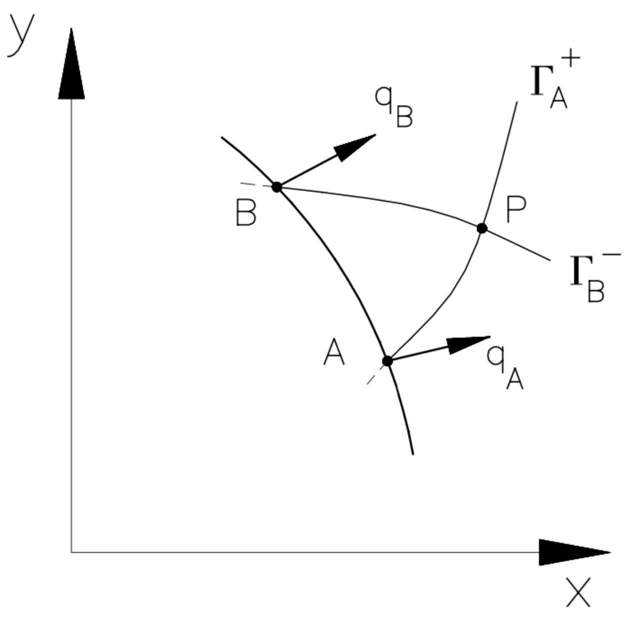


Figure 0.3: Position of point P from point A and B.

1. The previous point (A or B) arises from a **simple wave region**. The slope of the line from that point is then known exactly, as the characteristics are straight lines in that region. In this case, the exact straight line is computed like shown in the Initial Characteristics section above. This straight line is propagated until an intersection with a characteristic from the other type.

2. The previous point (A or B) arises from a **non-simple wave region**, in which the true characteristic is not a straight line, and is only approximated as one. The approximation as a straight line is done by computing the slope from the average solutions of the point P and the starting point (A or B, say A). The slope of the straight line is then given by $\alpha = -(\bar{\mu} - \bar{\phi})$ and $\alpha = (\bar{\mu} + \bar{\phi})$ for a Γ^- and Γ^+, respectively. Where $\bar{\mu} = \frac{1}{2}(\mu_A + \mu_B)$ and $\bar{\phi} = \frac{1}{2}(\phi_A + \phi_B)$. The line is then propagated until a characteristic of the other type intersects it.

Note that the intersection between two characteristics is simply computed from solving the system of two equations for two unknowns: $y = A_1 x + B_1$ and $y = A_2 x + B_2$, where the coefficients are fully defined by the point of origin and the slope angle the characteristics, as discussed above. In case a characteristic encounters a boundary, the same methods are used, simply that one of the points (A or B) is defined by the boundary condition: centreline ($\alpha_c = 0$ and point of origin from (1,0)) or the jet boundary (the point and slope used are from the last interaction with the jet boundary). This complete method is coded in function “interaction” (lines 79-139) in the code at the end of this report, where all options are differentiated, such that this function can be iterated upon for as many reflections as required by the user.

4) Plotting and streamline

A total of three plots were produced for the analysis presented below:

1. The important **characteristics in the flow field** are plotted based on the interaction positions found in the previous step.

2. The **pressure heatmap** was obtained by finding all polygons in the domain, defined by the interaction points found in the method shown earlier. The flow variables (Mach number and deflection angle) in the quadrilateral is assumed to be the average of the three or four points (triangle or quadrilateral). The only exception being in the first expansion fan, where the point A in Figure 0.1 is not considered in the average as it is multiply defined (does not have a proper value). In that case, only the two other points of the triangle are considered in the average.

3. **Streamline in the flow field** and the pressure along its path is found by propagating a particle from $x = 0$ for a specified y-position within the quadrilaterals mentioned above. In each quadrilateral, the deflection

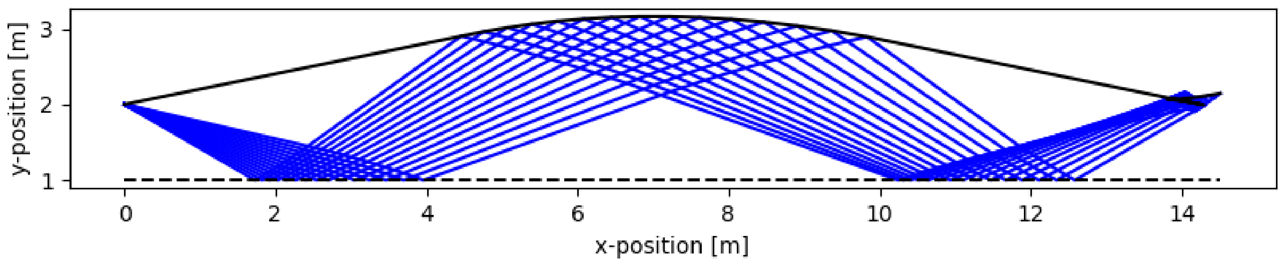


Figure 0.4: Characteristics in the flow field for 15 characteristics.

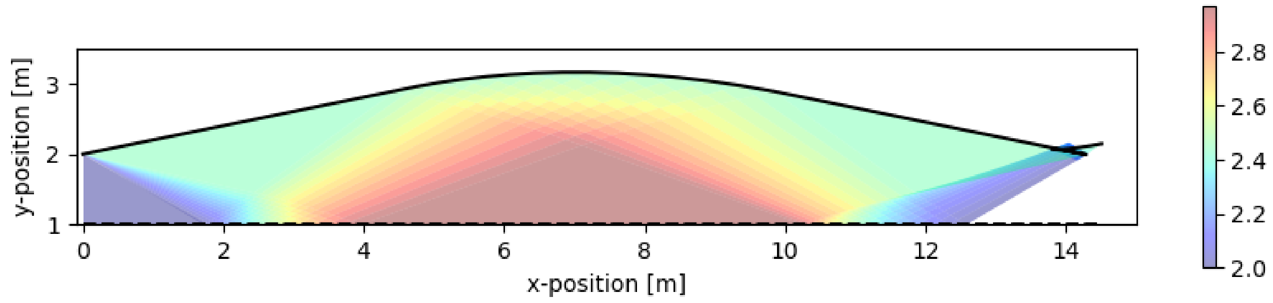


Figure 0.5: Mach distribution in the jet area for 15 characteristics.

angle of the particle in the region of the flow is known from the average mentioned above. The particle is then propagated in a time-like fashion: $x_{i+1} = x_i + \Delta x$ and $y_{i+1} = y_i + \tan\phi\Delta x$, where Δx is defined by the user, and needs to be smaller than the smallest x-position difference between two quadrilateral or triangle region boundaries, that will be encountered by the particle (user needs to check that manually as a verification step). The pressure along the streamline is obtained from the isentropic relation (8.42) from Anderson[1] and is given as the ratio of the pressure and the atmospheric pressure (taken as reference throughout the flow field).

5) Results

The results for the different graphs described in the previous section are given below. Note that the height of the outlet is taken as $H = 2\ m$ for all the simulations shown in this report. Meaning that the centreline is at $y = 1\ m$. Additionally, $\gamma = \frac{7}{5}$

5.1) Characteristics

The characteristics in the flow field are given in Figure 0.4.

Going from $x = 0$ to $x \approx 14$, the flow is first expanded in the centred expansion wave, where the flow deflection is in increased and the flow accelerates to a Mach number of around 2.44. The centreline acts like a wall in this model (in reality, the reflected characteristics emanate from the second half of the jet exhaust. The characteristics diverge in the centred expansion fan and after their first reflection. In the second non-simple wave region, the exhaust jet boundary is bent down through the interactions of the characteristics with the boundary. This results in a reflection of converging characteristics, which will eventually intersect with each other to form a shock, during the third reflection (second interaction with the symmetry line). The Method of Characteristics (MOC), then predicts the formation of a shock towards $x \approx 11.3\ m$, after which the solution through this method is not reliable anymore. The forming process of the shock clearly arises from the intersection of characteristics of the same family/type (here Γ^+. A total of two completely simple regions (straight lines), two complete non-simple regions (curved lines) and one shockwave can be distinguished in the plots (the shockwave arises in the middle of the third non-simple region).

The shape of the jet boundary is typical for under-expanded jet exhausts, showing that the flow physics have been modelled in a seemingly correct manner. The jet exhaust is curved as a result of the non-simple region arising from the reflection of the flow on its boundary.

5.2) Mach Distribution

Following, the Mach distribution in the jet area can be seen in Figure 0.5.

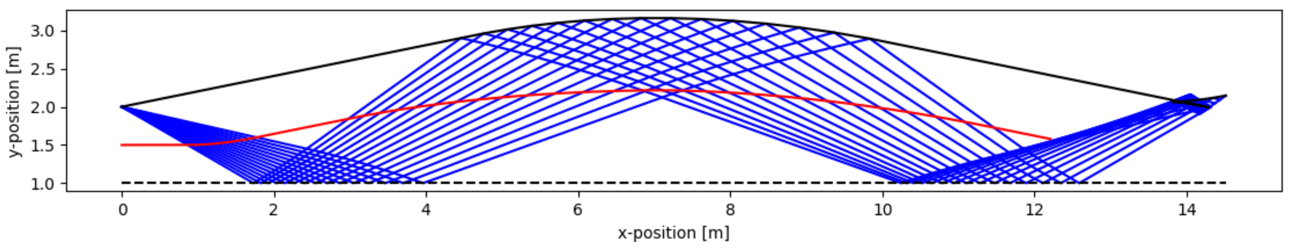


Figure 0.6: Streamline starting at y = 3H/4, x=0

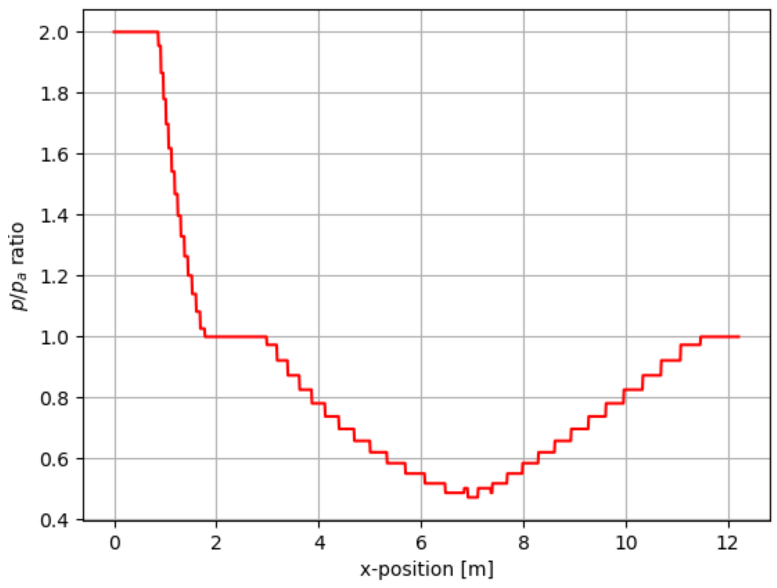


Figure 0.7: Static pressure on streamline starting at y = 3H/4, x = 0

First, the flow is acceleration through the centred expansion fan, reaching a value of about Mach 2.44 above the first non-simple region. The reflection of the characteristics still diverge, meaning that the flow is further expanded, nearly reaching a value of Mach 3 below the second non-simple wave region. However, the third simple wave region has converging characteristics, meaning that the flow is being decelerated to again. However, the shock wave forms and the solution after $x \approx 11.3$ shown in the heatmap is not reliable, it is known from shock theory that the flow becomes subsonic after the shockwave, but this is not taken into account by the solver.

5.3) Streamline

Note that all streamline simulations were stopped before the particle encounters the shockwave (if it was going to at some point).

Finally, two streamlines are considered in the characteristics' domain. First, a particle starting at $x = 0$, $y = \frac{3H}{4}$ (note that H is taken as 2 m in all the simulations provided in this report, and the centreline at y = 1 m). Figure 0.6 shows the streamline within the domain of the characteristics. The pressure along the streamline is shown by Figure 0.7. Following the path of the particle, it is first deflected upwards as it goes through the simple region, before reaching a maximum where the jet boundary achieves its maximum. After that point, the particle deflects back down to finally reach it's original y-position right before the shockwave. The pressure varies more than for the streamline considered below, as it is predominantly going through regions where characteristics are 'active' (non-uniform regions). The pressure along the streamline can be seen to be first expanded, before reaching a constant value for a short time and going further down in the second non-simple region. The inverse process then occurs until the shockwave itself. The simulation is stopped right before the shockwave is reached, after which the solution would be unreliable. It is however noted that the pressure would be expected to rise tremendously right after the shock, as the flow becomes subsonic.

Similarly, the second streamline starting at $x = 0$, $y = \frac{2H}{4}$ (centreline) can be considered in Figures 0.8 and 0.9. In this case, the flow path is going straight in the non-simple region, and stays completely horizontal as the quadrilaterals it goes through have a zero average deflection angle (or should have a zero average the-

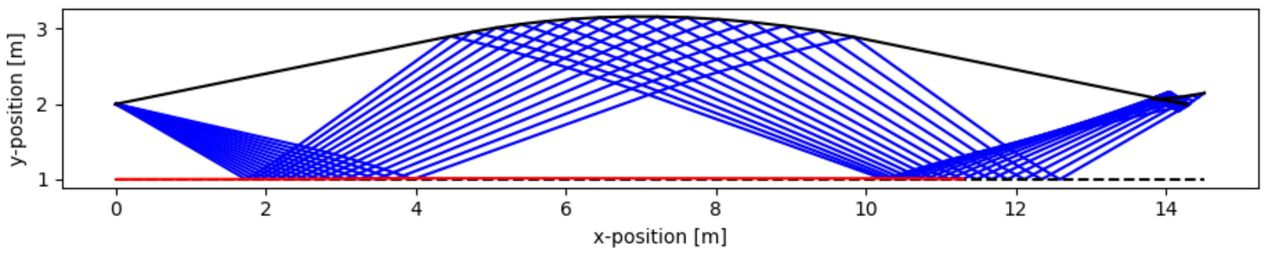


Figure 0.8: Streamline starting at y = 2H/4, x=0

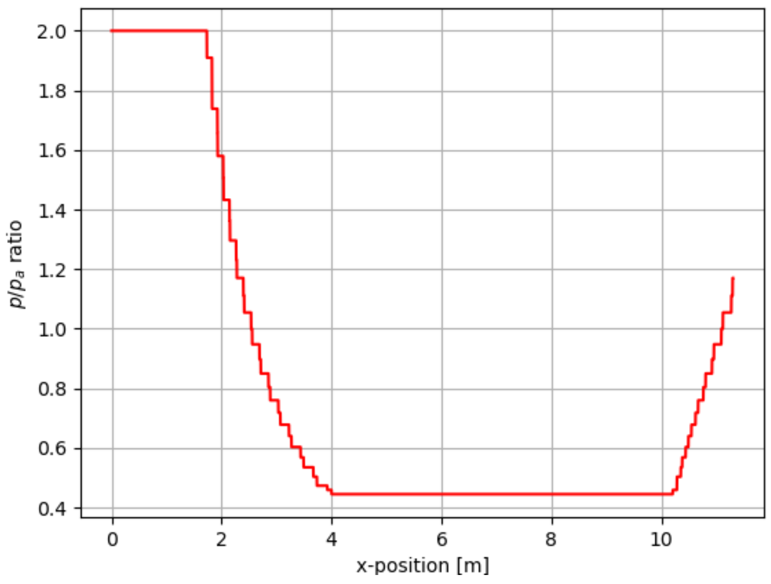


Figure 0.9: Static pressure on streamline starting at y = 2H/4, x = 0

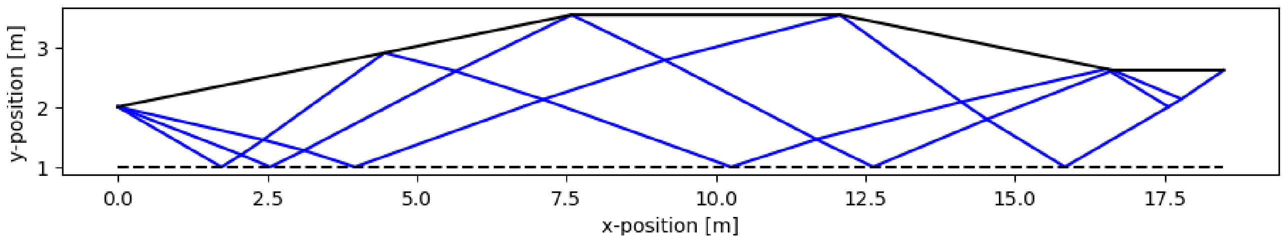
oretically due to the symmetry of the jet). Therefore, overall, the particle stays on the centreline of the flow. Furthermore, the pressure plot shows the expansion resulting in a pressure ratio of about 0.45 with respect to the atmospheric pressure at minimum: as the particle is travelling in the non-simple region, the pressure drops before staying constant in the uniform region below the second non-simple region. Following, the pressure builds up isentropically again in the third non-simple region. The streamline does not go through the shock, however, the flow likely would undergo some process to ensure that the particles going through the shock and the ones going under it can 'cohabitate' after the non-simple region. The simulation is stopped at $x \approx 11.3$ as well, as it is expected that some non-modelled physics are at play there.

6) Accuracy of the Computations

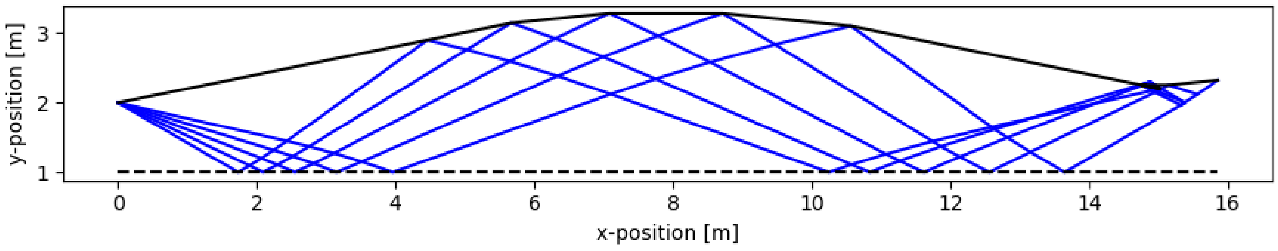
The accuracy of the method clearly relies on the number of characteristics used to obtain details of the flow field. However, when it comes to determining the flow properties within the uniform regions, those values for the Mach number and the deflection angle can also be completely known from considering only two characteristics (the head and tail characteristics of the centred expansion). Hence, increasing the number of characteristics serves to obtain a deeper insight in the flow field simple and non-simple regions, but the properties present in the uniform regions can be found to the same level of accuracy using only two characteristics. This can be compared using the characteristics in the flow profile at several N's (number of characteristics), as shown in Figure 0.10.

A first observation which can be made from Figure 0.10 is that more characteristics result in a shorter jet exhaust, but this shrinkage seems to converge to a certain value as the jet exhaust lengths are similar for N=7 and N=10. A similar picture can be used for the Mach distribution, as shown in Figure 0.11. As explained earlier, the resolution of the flow becomes higher, but the values in the uniform regions are always estimated to the same level of accuracy.

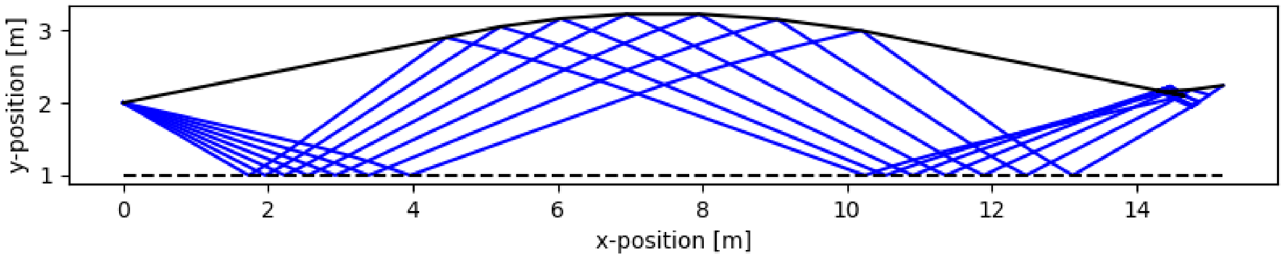
Furthermore, the number of characteristics as a direct influence on both the pressure along a streamline and the path of the streamline. The former is seen from the jumps in the static pressure plots, which would be continuous (rather than discrete jumps) if an infinite amount of characteristics was considered. The streamline



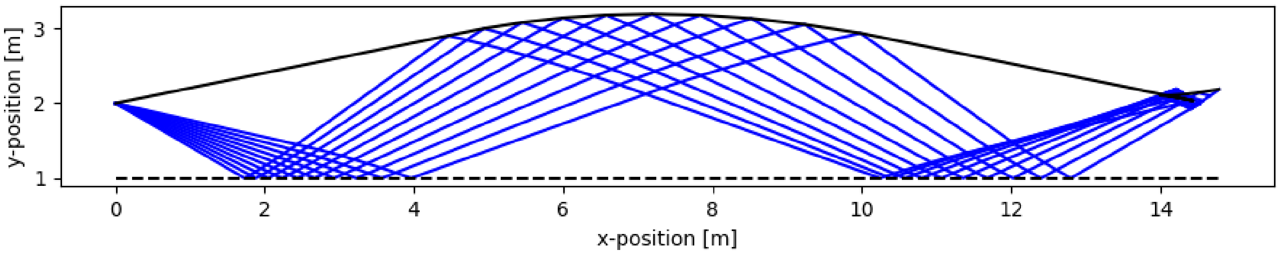
(a) N=3



(b) N=5



(c) N=7



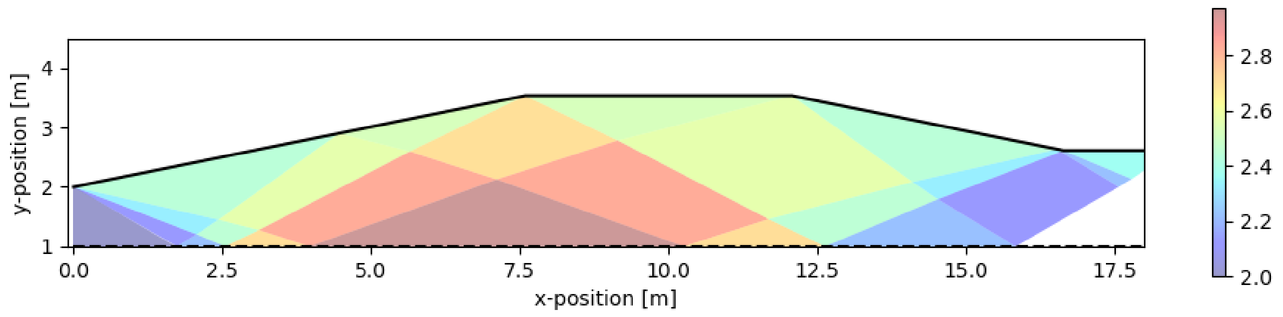
(d) N=10

Figure 0.10: Characteristics in the flow path for different number of characteristics N.

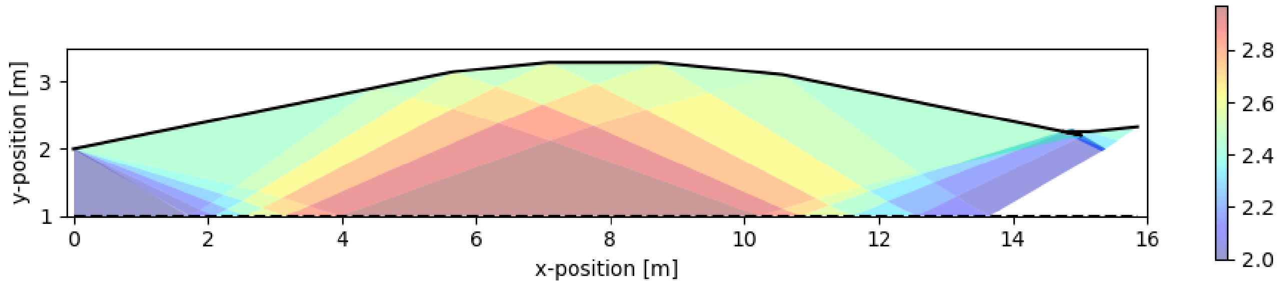
trajectory is more accurate when a larger number of characteristics is considered, as the flow direction angle would be more accurate within each polygon considered for the propagation. Additionally, note that the streamline trajectory accuracy also depends on the step in x inputted by the user, which needs to be sufficiently small. For the context of this assignment, it is no issue as a quite limited amount of characteristics are considered, but a more efficient manner to propagate the streamline should be considered for industry level implementations (eg. by computing the intersections of streamline line segments with the characteristics instead). Furthermore, the centreline streamline was approximated by taken $y = \frac{H}{2} + 0.0000000000001$ and not $y = \frac{H}{2}$, as the implementation would consider the latter as outside the domain of the solution. However, this inaccuracy is as small as the machine accuracy and can therefore be safely neglected.

In general, the code shows limitations when the shockwave appears, resulting in an unreliable solution of the flow field when for regions 'below' the shock. This is simply the case due to the model implemented, which assumes a homentropic flow (this assumption is valid up and until the shockwave formation). Additionally, it can be seen that the solving method implementation also somewhat breaks down after the shock as the characteristics seem to fold over the jet boundary, again emphasising that the solution after the shock is not reliable.

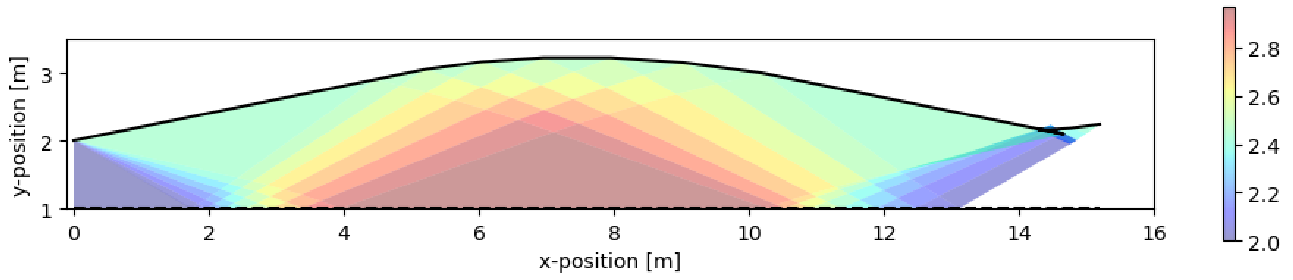
In practice, a verification of the code would include a sensitivity analysis of the number of characteristics to ensure that the solution with N=15 characteristics (considered in the discussions) is indeed close enough to one if a very large number of characteristics. This could not be performed in the context of this assignment due



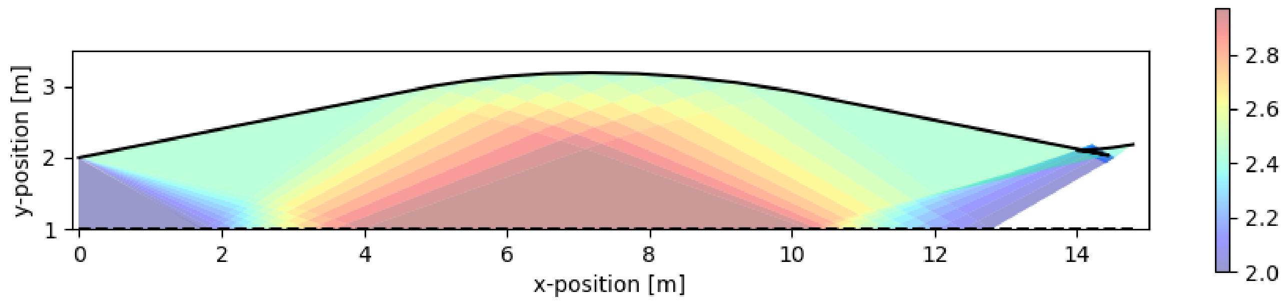
(a) N=3



(b) N=5



(c) N=7



(d) N=10

Figure 0.11: Mach number distribution for different number of characteristics N.

to time constraints, but should be done if this code is used for any further design or analysis.

7) Solver Code

```python
import numpy as np
from scipy.optimize import fsolve
from PlottingRoutines import plotCharacteristics
import matplotlib.pyplot as plt
from matplotlib import path
from collections import Counter
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
import matplotlib

# User inputs
PLOT = True
PLOT_QUADS = False
dx = 0.01  # Flow path resolution
last_flow_x = 0.1  # Last flow path
N = 15 # Number of characteristics
Num_reflections = 4
Me = 2  # Exhaust mac number
Pe_Pa_ratio = 2  # Exhaust pressure is twice the ambient atmospheric pressure
gamma = 7/5  # Type of gas present in exhaust flow
phi_e = 0  # The flow deflection at the entry of the exhaust in degrees
H = 2  # m, the height of the total (!!!) exhaust. Only half of it is modelled
x0 = 0  # m, x position of the outlet
y0 = H  # m, y position of the outlet
yc = H/2

# Main functions
def v(gamma, M):
    '''Returns the Prandtl-Meyer angle related to a Mach number in degrees.
    Note: Formula taken from Anderson equation (9.42), as the reader has the wrong definition
    Verification: was verified manually and against calculators online'''
    return np.rad2deg(np.sqrt((gamma + 1)/(gamma-1)) * np.arctan(np.sqrt((gamma-1)*((M**2)-1)/(gamma+1)))
                      - np.arctan(np.sqrt((M**2 - 1))))

def M(gamma, v):
    '''Returns the Mach number related to a Prandtl-Meyer angle given in degrees.
    Verification: was verified manually and against calculators online'''
    def func(x): return -np.rad2deg(np.sqrt((gamma + 1)/(gamma-1)) *
                                   np.arctan(np.sqrt((gamma-1)*((x[0]**2)-1)/(gamma+1)))
                                    - np.arctan(np.sqrt((x[0]**2 - 1)))) + v
    root = fsolve(func, [2])  # 2 is the initial guess used
    return float(root[0])

def mu(M):
    '''Returns the mach angle in degrees
    Verification: was verified manually'''
    return np.rad2deg(np.arcsin(1/M))

def charac_direction(mu, phi_char, type=False):
    '''Returns the direction of the wave in degrees
    Note: the angle is defined with respect to the horizontal, positive upwards'''
    if type:  #    plus
        alpha_char = mu + phi_char
    else:  #    minus
        alpha_char = -(mu - phi_char)
    return alpha_char

def characteristics_equations(x0, y0, alpha):
    '''Returns the coefficients of the linear approximation of the characteristics equations in the field
    Note: alpha given in degrees'''
    A = np.tan(np.deg2rad(alpha))
    B = y0 - A * x0
    return A, B

def lines_intersection(A1, B1, A2, B2):
    '''Returns the intersection point of two lines defined from y = A1 x + B1 and y = A2
    x + B2 in the form (xp, yp)'''
    xp = np.divide((B2-B1), (A1-A2))
    yp = np.multiply(A1, xp) + B1
    return xp, yp

def G_int_sol(M1, phi1, M2, phi2, gamma):
    vp = 0.5 * (v(gamma, M1) + v(gamma, M2)) + 0.5 * (phi2 - phi1)
    phip = 0.5 * (phi1 + phi2) + 0.5 * (v(gamma, M2) - v(gamma, M1))
    return vp, phip

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

def interaction(inc_M, inc_phi, inc_x, inc_y, jetBond, type='center'):
    if type == 'center':
        mainChar = False
    elif type =='jetBoundary':
        mainChar = True
    else:
        print("wrong wave type inserted")
        return False
    # Interactions matrix for the first interaction
    X_int = np.zeros((N, N + 1))
    Y_int = np.zeros((N, N + 1))
    phi_int = np.zeros((N, N + 1))
    M_int = np.zeros((N, N + 1))

    X_int[:, 0] = inc_x[:, 0]
    Y_int[:, 0] = inc_y[:, 0]
    phi_int[:, 0] = inc_phi[:, 0]
    M_int[:, 0] = inc_M[:, 0]

    # First interaction with the center line - use the wall boundary condition phi_c = 0
    for j in range(1, N + 1):
        for i in range(j - 1, N):
            if i == j - 1:
                if type == 'jetBoundary':
                    A_bc, B_bc = characteristics_equations(jetBond[-1, 1], jetBond[-1,
    2], jetBond[-1, 0])
                    vp = va
                    phip = va - v(gamma, M_int[i, j - 1]) + phi_int[i, j - 1]
                    if j-1 == 0:
                        A_minus, B_minus = characteristics_equations(X_int[i, j - 1],
    Y_int[i, j - 1], charac_direction(mu(M_int[i, j - 1]), phi_int[i, j - 1], type=
    mainChar))
                    else:
                        A_minus, B_minus = characteristics_equations(X_int[i, j - 1],
    Y_int[i, j - 1], charac_direction((mu(M_int[i, j - 1]) + mu(M(gamma, vp)))*0.5, (
    phi_int[i, j - 1] + phip)*0.5, type=mainChar))
                    xp, yp = lines_intersection(A_minus, B_minus, A_bc, B_bc)
                    jetBond = np.vstack((jetBond, np.array([phip, xp, yp])))
                else:
                    A_bc, B_bc = 0, yc
                    phip = phi_c
                    vp = v(gamma, M_int[i, j - 1]) + phi_int[i, j - 1] - phip
                    if j - 1 == 0:
                        A_minus, B_minus = characteristics_equations(X_int[i, j - 1],
    Y_int[i, j - 1], charac_direction(mu(M_int[i, j - 1]), phi_int[i, j - 1], type=
    mainChar))
                    else:
                        A_minus, B_minus = characteristics_equations(X_int[i, j - 1],
```

```python
Y_int[i, j - 1], charac_direction((mu(M_int[i, j - 1]) + mu(M(gamma, vp))) * 0.5, (phi_int[i, j - 1] + phip) * 0.5, type=mainChar))
                    xp, yp = lines_intersection(A_minus, B_minus, A_bc, B_bc)
                Mp = M(gamma, vp)
            else:
                if type == 'jetBoundary':
                    vp, phip = G_int_sol(M_int[i, j-1], phi_int[i, j-1], M_int[i-1, j], phi_int[i-1, j], gamma)
                else:
                    vp, phip = G_int_sol(M_int[i - 1, j], phi_int[i - 1, j], M_int[i, j - 1], phi_int[i, j - 1], gamma)

                if j-1 ==0:
                    A_minus, B_minus = characteristics_equations(X_int[i, j - 1], Y_int[i, j - 1], charac_direction(mu(M_int[i, j - 1]), phi_int[i, j - 1], type=mainChar))
                else:
                    A_minus, B_minus = characteristics_equations(X_int[i, j - 1], Y_int[i, j - 1], charac_direction((mu(M_int[i, j - 1])+mu(M(gamma, vp)))*0.5, (phi_int[i, j - 1]+phip)*0.5,type=mainChar))
                A_plus, B_plus = characteristics_equations(X_int[i - 1, j], Y_int[i - 1, j], charac_direction((mu(M_int[i - 1, j]) + mu(M(gamma, vp)))*0.5, (phi_int[i - 1, j] + phip)*0.5, type= not mainChar))
                xp, yp = lines_intersection(A_minus, B_minus, A_plus, B_plus)
                Mp = M(gamma, vp)
            X_int[i, j] = xp
            Y_int[i, j] = yp
            phi_int[i, j] = phip
            M_int[i, j] = Mp
    return X_int, Y_int, phi_int, M_int, jetBond

# Initial conditions
ve = v(gamma, Me)

# BCs
phi_c = 0  # Center line phi = 0
Ma = np.sqrt(((1 + (gamma-1) * (Me**2)/2) * (Pe_Pa_ratio)**((gamma-1)/gamma) - 1)*(2/(gamma-1)))  # Mach number jet BC
va = v(gamma, Ma)  # Prandtl-Meyer angle after the first expansion wave
phi_a = va - ve  # Slope of first segment of the jet boundary
jetphi = np.array([[phi_e, x0, y0], [phi_a, x0, y0]])

# Initial expansion fan definition
phi_vector = np.zeros((N, 1))
M_vector = np.zeros((N, 1))
dPhi = (phi_a-phi_e)/(N-1)  # Incremental flow deflection over the Expansion wave.
for i in range(0, N):
    Phi_i = i * dPhi + phi_e
    vi = ve + Phi_i; Mi = M(gamma, vi)
    phi_vector[i] = Phi_i
    M_vector[i] = Mi

X_vector = (np.ones((N, 1)) * x0)
Y_vector = (np.ones((N, 1)) * y0)

X_int, Y_int = [], []
M_int, phi_int = [], []
w = 1

# Main calculations !
while w <= Num_reflections:
    if (w % 2) == 0:
        X_int0, Y_int0, phi_int0, M_int0, jetphi = interaction(M_vector, phi_vector, X_vector, Y_vector, jetphi, type='jetBoundary')
    else:
        X_int0, Y_int0, phi_int0, M_int0, jetphi = interaction(M_vector, phi_vector, X_vector, Y_vector, jetphi, type='center')
    # Retrieve the initial conditions for the next step
    X_vector = np.array(np.matrix(X_int0[-1, 1:]).T)
    Y_vector = np.array(np.matrix(Y_int0[-1, 1:]).T)
    phi_vector = np.array(np.matrix(phi_int0[-1, 1:]).T)
    M_vector = np.array(np.matrix(M_int0[-1, 1:]).T)
    X_int.append(X_int0[:, 1:]); Y_int.append(Y_int0[:, 1:]); M_int.append(M_int0[:, 1:])
    phi_int.append(phi_int0[:, 1:])
    w += 1

#print(phi_int[0])

# Obtain all quadrilaterals
# First combine all matrices in
X_total,Y_total,M_total,phi_total = np.ones((1, N))*x0, np.ones((1, N))*y0, np.ones((1, N)) * Me, np.ones((1, N))*phi_e

for i in range(0, len(X_int)):
    for k in range(0, N):
        for l in range(k+1, N):
            X_int[i][k, l] = X_int[i][l, k]
            Y_int[i][k, l] = Y_int[i][l, k]
            M_int[i][k, l] = M_int[i][l, k]
            phi_int[i][k, l] = phi_int[i][l, k]
    X_total = np.vstack((X_total, X_int[i]))
    Y_total = np.vstack((Y_total, Y_int[i]))
    M_total = np.vstack((M_total, M_int[i]))
    phi_total = np.vstack((phi_total, phi_int[i]))

# Find the values of the quadrilaterals
# quadri_matrix = np.zeros((1, 4+4+2)) # four x coords, 4 y coords, the average mach number and the average phi angle
quadri_matrix = np.array([x0, y0, x0, yc, X_total[1, 0], Y_total[1, 0], X_total[1, 0],
                          Y_total[1, 0], Me, phi_e])  # Origin zone

for i in range(0, np.shape(X_total)[0]-1):
    for j in range(0, np.shape(X_total)[1]-1):
        myList = [X_total[i, j], X_total[i, j+1], X_total[i+1, j], X_total[i+1, j+1]]
        counter = Counter(myList)
        duplicates = [i for i, j in counter.items() if j > 1]
        if len(duplicates) > 0:
            ind = np.where(X_total == duplicates[0])
            avg_M = ((M_total[i, j] + M_total[i, j + 1] + M_total[i + 1, j] + M_total[i + 1, j + 1]) / 4 - M_total[ind[0][1], ind[1][1]]/4)*4/3
            avg_phi = ((phi_total[i, j] + phi_total[i, j + 1] + phi_total[i + 1, j] + phi_total[i + 1, j + 1]) / 4 - phi_total[ind[0][1], ind[1][1]]/4)*4/3

            if len(np.where(np.array(myList) == x0)[0]) > 0:
                avg_M = ((M_total[i, j] + M_total[i, j + 1] + M_total[i + 1, j] + M_total[i + 1, j + 1]) / 4 - 2 *  M_total[0, 0] / 4)*2
                avg_phi = ((phi_total[i, j] + phi_total[i, j + 1] + phi_total[i + 1, j] + phi_total[i + 1, j + 1]) / 4 - 2*  phi_total[0, 0] / 4)*2
                # Checked
            new_quad = np.array([[X_total[i, j], Y_total[i, j], X_total[i + 1, j + 1], Y_total[i + 1, j + 1], X_total[i, j + 1], Y_total[i, j + 1], X_total[i + 1, j], Y_total[i + 1, j], avg_M, avg_phi]])
        else:
            avg_M = (M_total[i, j] + M_total[i, j+1] + M_total[i+1, j] + M_total[i+1 , j+1])/4
            avg_phi = (phi_total[i, j] + phi_total[i, j+1] + phi_total[i+1, j] + phi_total[i+1 , j+1])/4
            new_quad = np.array([[X_total[i, j], Y_total[i, j], X_total[i, j+1], Y_total[i, j+1], X_total[i+1, j], Y_total[i+1, j], X_total[i+1, j+1], Y_total[i+1, j+1], avg_M, avg_phi]])
        quadri_matrix = np.vstack((quadri_matrix, new_quad))

# Add special quads --> the very large ones outside the simple regions
```

```python
for i in range(0, Num_reflections-1):
    if i == 0:  # contains the origin, this is the first large upper region, should have Ma
        avg_M = (M_total[N*i+1, -1] + M_total[N*(i+1)+1, 0]) / 2
        avg_phi = (phi_total[N*i+1, -1] + phi_total[N*(i+1)+1, 0]) / 2
        # Checked
    else:
        avg_M = (M_total[N*i, -1] + M_total[N*i+1, -1] + M_total[N*(i+1)+1, 0]) / 3
        avg_phi = (phi_total[N*i, -1] + phi_total[N*i+1, -1] + phi_total[N*(i+1)+1, 0]) / 3
    new_quad = np.array([[X_total[N*i, -1], Y_total[N*i, -1], X_total[N*i+1, -1], Y_total[N*i+1, -1], X_total[N*(i+1)+1, 0], Y_total[N*(i+1)+1, 0], X_total[N*(i+1)+1, 0], Y_total[N*(i+1)+1, 0], avg_M, avg_phi]])
    quadri_matrix = np.vstack((quadri_matrix, new_quad))

# Some quadrilaterals might be present twice, clean up those rows
i = 0
while i < np.shape(quadri_matrix)[0]-1:
    r1 = quadri_matrix[i, :]
    sum_r1 = sum(r1)
    for j in range(0, np.shape(quadri_matrix)[0]-1):
        if j == i:
            pass # do nothing
        else:
            r2 = quadri_matrix[j, :]
            sum_r2 = sum(r2)
            if abs(sum_r1-sum_r2) < 1e-5:
                # rows 1 and 2 are the same
                quadri_matrix = np.delete(quadri_matrix, j, 0)
                i -= 1
        if j>=np.shape(quadri_matrix)[0]-1: break
    i += 1

# Flow path in the mess of characteristics
x, shock = 0, False
y = 2*H/4 + 0.0000000000001
list_M, list_x, list_y = [], [], []
while shock == False:
    # Determine in which quad the particle is
    in_quad, m = False, 0
    while in_quad == False:
        a_quad = quadri_matrix[m, :]
        p = path.Path([(a_quad[0], a_quad[1]), (a_quad[2], a_quad[3]), (a_quad[6], a_quad[7]), (a_quad[4], a_quad[5])])
        coords = [[a_quad[0], a_quad[1]], [a_quad[2], a_quad[3]], [a_quad[6], a_quad[7]], [a_quad[4], a_quad[5]]]
        coords.append(coords[0])
        xs, ys = zip(*coords)
        if PLOT_QUADS:
            plt.plot(xs, ys, 'g')
        in_quad = p.contains_points([(x, y)])
        m += 1
        if m >= np.shape(quadri_matrix)[0]:
            print('point not in domain !!!')
            print(x, y)
            break
    the_quad = quadri_matrix[m-1, :]
    Mx = the_quad[8]
    phi_x = the_quad[9]
    if x > last_flow_x:
        shock = True
    list_M.append(Mx)
    list_x.append(x)
    list_y.append(y)
    print(x, Mx, phi_x)   # Uncomment to monitor the pathline progress
    x += dx
    y += np.tan(np.deg2rad(phi_x))*dx

if PLOT:
    plt.figure(1)
    plt.axes().set_aspect('equal')
    for i in range(0, N):
        plt.plot(X_total[:, i], Y_total[:,i], 'b')
    plt.plot(jetphi[:, 1], jetphi[:, 2], 'black')
    plt.plot([min(jetphi[:, 1]), max(jetphi[:, 1])], [1,1], color='black', linestyle='
    dashed')
    plt.xlabel("x-position [m]")
    plt.ylabel("y-position [m]")
    #plt.plot(list_x, list_y, 'r')

    plt.figure(2)
    array_M = np.array(list_M)
    p_p0 = pressure_ratio(gamma, array_M)
    p_pa = Pe_Pa_ratio * p_p0**(-1) * pressure_ratio(gamma, Me)
    plt.plot(list_x, p_pa, 'r')
    plt.grid(True)
    plt.xlabel('x-position [m]')
    plt.ylabel(r'$p/p_{a}$ ratio')

in_quad, m = False, 0
M_regions = quadri_matrix[:, 8]

# Mach number heatmap
p = 0
patches = []
while p < np.shape(quadri_matrix)[0]:
    a_quad = quadri_matrix[p, :]
    # Make polygon patch
    patches.append(Polygon(np.array([[a_quad[0], a_quad[1]], [a_quad[2], a_quad[3]], [
    a_quad[6], a_quad[7]], [a_quad[4], a_quad[5]]]), True))
    p += 1

fig = plt.figure(3)
ax = fig.add_subplot(111)
ax.set_aspect('equal')
p = PatchCollection(patches, cmap=matplotlib.cm.jet, alpha=0.4)
colors = M_regions
p.set_array(colors)
ax.add_collection(p)
fig.colorbar(p)
plt.plot(jetphi[:, 1], jetphi[:, 2], 'black')
plt.plot([min(jetphi[:, 1]), max(jetphi[:, 1])], [1, 1], color='black', linestyle='dashed
    ')
plt.xlabel("x-position [m]")
plt.ylabel("y-position [m]")

ax.set_xlim(-0.1, 15)
ax.set_ylim(1, 3.5)
plt.show()
```

Listing 1: Method of Characteristics Solver for the Exhaust Jet Problem.