Delft University of Technology

Faculty of Aerospace Engineering
AE4140 Gas Dynamics

Task 2: Simulation of the Supersonic Underexpanded Flow After a Nozzle Exit Using the Method of Characteristics

Responsible teacher: Dr. Ferry Schrijer
October 23, 2022

Written by Andrei Pârvulescu
Student number: 5118344

Delft University of Technology

Contents

Nomenclature

Abbreviations

MoC Method of Characteristics

Greek Symbols

α Angle of Γ^+ characteristic [rad]

β Angle of Γ^- characteristic [rad]

μ Mach angle [rad]

ν Prandtl-Meyer angle [rad]

ϕ Flow angle [rad]

Latin Symbols

M Mach number [-]

p (Static) absolute pressure [atm]

p_t Total absolute pressure [atm]

x Horizontal spatial coordinate

y Vertical spatial coordinate

Theory 1

1.1 Set-up of the problem

The problem set-up is as follows: after a convergent-divergent nozzle, the supersonic gas flow enters the atmosphere. The scope is to find the development of the jet flow downstream of the nozzle. The flow is assumed underexpanded, such that the exit pressure p_e from the nozzle is higher than the atmospheric pressure p_a. The development is expected to look similar to Figure 1.1.

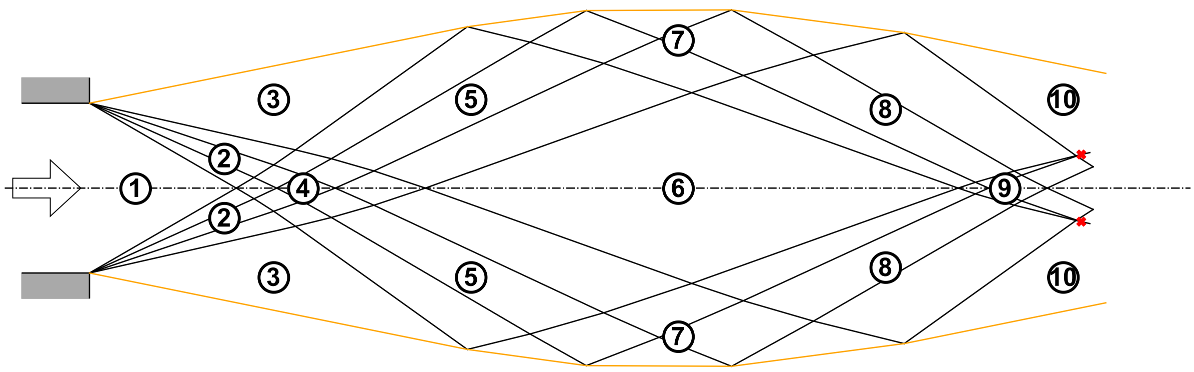

Figure 1.1: Diagram of the development after the exit of the nozzle of an underexpanded supersonic flow. The coordinate system is x horizontally positive to the right, and y vertically positive upwards.

In this figure, the solid orange lines represent the jet boundary, solid black lines the characteristics of the expansion / compression fans, the middle dash-dot line is the line of symmetry, and the red crosses represent the beginning of the shock formations.

As the flow exits the nozzle from the left into the uniform region 1, it remains uniform and at the same Mach number as the exit, M_e. It encounters the corner expansion fan, in the simple-wave region 2. The two expansion fans intersect and interact in the non-simple region 4, curving the characteristics. After passing region 2, the flow near the margin enters the uniform region 3, where the pressure has reached the atmospheric pressure p_a. Afterwards, the effect of the expansion fan from the opposite corner is felt by passing through the simple-wave region 5, once again accelerating. It then passes into the uniform region 6, where the maximum Mach number is reached. In the non-simple wave region 7, the incoming expansion fan is reflected as a compression fan, and the two fans interact. Characteristics originating from this region are convergent. In the simple-wave region 8, the flow experiences the deceleration of the compression waves. The two compression waves intersect and interact in the non-simple region 9. The method can no longer be applied when shocks begin to form, which happen when the converging characteristics intersect.

1.2 Assumptions and simplifications

The following assumptions and simplifications are made:

- The atmospheric pressure is positive and finite. A value of 0 would lead to a division by 0 when calculating the Mach number at the jet boundary. For the numerical exercise, the atmospheric pressure will be taken as $p_a = 1\,\text{atm}$.
- The nozzle exit flow needs to be supersonic and underexpanded, i.e. $M_e > 1$ and $p_e > p_a$. The method of characteristics only applies to a supersonic flow, which is expected to come from a convergent-divergent nozzle. Furthermore, if the flow were overexpanded, oblique shock waves

would start developing at the exit corners, rendering the method of characteristics inapplicable. In the numerical exercise, $M_e = 2$ and $p_e = 2 \cdot p_a$

- The flow is not only isentropic, but homentropic. This means that the entropy is constant and equal throughout the whole flow. This is a prerequisite for the MoC.

- The gas is assumed to be calorically perfect. This means that it respects the ideal gas law, and that the specific heats are constant regardless of temperature. This is applicable for not too extreme temperature values. This is a prerequisite for the MoC.

- The flow is assumed to be steady and all transient effects are ignored. This comes from assuming the variances with time in the Navier-Stokes equations to be zero. It is a prerequisite for the MoC, as without it characteristics cannot be found.

- The flow is considered two-dimensional. This is a prerequisite for the MoC.

- All viscous effects, heat conduction and external forces are assumed to b negligible or zero. This is a prerequisite for the MoC.

- The x-direction of the coordinate system is time-like. This means that any characteristics / Mach lines that would occur do not travel backwards towards a smaller x-value, as then initial value conditions would not be possible to be imposed. This is a prerequisite of the MoC.

1.3 Basics of the Method of Characteristics

The method of characteristics is based on the existence of two types of characteristics – Γ^- and Γ^- – at each point in the flow, along which a certain value is conserved and thus does not change. Along a Γ^- characteristic, this value is $\nu + \phi$, and along a Γ^+ it is $\nu - \phi$, where ϕ is the local flow angle with respect to the coordinate system, and ν is the local Prandtl-Meyer angle, defined in Equation 1.1.

$$\nu = \sqrt{\frac{\gamma+1}{\gamma-1}} \arctan\left(\sqrt{\frac{\gamma-1}{\gamma+1} \cdot (M^2-1)}\right) - \arctan\left(\sqrt{M^2-1}\right) \tag{1.1}$$

These characteristics also have an instantaneous direction with respect to the coordinate system. The Γ^- characteristic makes an angle of $\phi - \mu$ with respect to the x-axis, and the Γ^+ characteristic makes an angle of $\phi + \mu$, where μ is the local Mach angle, defined in Equation 1.2.

$$\sin \mu = \frac{1}{M} \tag{1.2}$$

To apply the method of characteristics, the domain starts with some initial values along the left-most boundary, and then the flow properties are calculated as the simulation proceeds towards the right. The basis of this propagation are explained in the next section.

1.3.1 Propagation of the flow properties

Assume the setup for propagation is similar to the one shown in Figure 1.2, and that the flow properties (ϕ and M, and thus also ν and μ) and their coordinates are known in points A and B. The properties in point P need to be calculated, as well as its position.

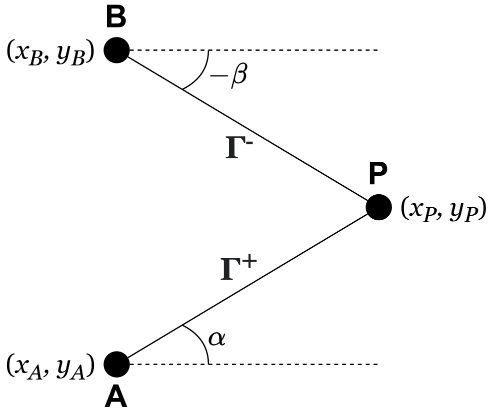

Figure 1.2: Propagation from points A and B to point P, along Γ^+ and Γ^- characteristics.

To calculate the properties at point P, the constant properties of the characteristics are used, i.e. $\nu_A - \phi_A = \nu_P - \phi_P$ and $\nu_B + \phi_B = \nu_P + \phi_P$. Rearranging, the relations in Equation 1.3 can be found for ν_P and ϕ_P. M_P and μ_P can be calculated from ν_P using Equation 1.1 and Equation 1.2.

$$\begin{aligned} \nu_P &= \frac{1}{2}(\nu_B + \nu_A) + \frac{1}{2}(\phi_B - \phi_A) \\ \phi_P &= \frac{1}{2}(\phi_B + \phi_A) + \frac{1}{2}(\nu_B - \nu_A) \end{aligned} \tag{1.3}$$

To calculate the position of point P, the angles the characteristics make wiht the horizontal are needed. As the distance between consecutive points is assumed to be small, the characteristics can be approximated with straight lines, and their angles (α for the Γ^+ and β for the Γ^-, as shown in Figure 1.2) can be taken as the average of the local characteristic angles in A or B and P. Their values are given in Equation 1.4.

$$\begin{aligned} \alpha &= \frac{1}{2}(\phi_A + \mu_A + \phi_P + \mu_P) \\ \beta &= \frac{1}{2}(\phi_B - \mu_B + \phi_P - \mu_P) \end{aligned} \tag{1.4}$$

The position of point P can be calculated from the geometry of Figure 1.2 and the angles α and β, with the values of x_P and y_P given in Equation 1.5, as functions of the positions of A and B.

$$\begin{aligned} x_P &= \frac{y_B - y_A + x_A \tan\alpha - x_B \tan\beta}{\tan\alpha - \tan\beta} \\ y_P &= y_A + x_P \tan\alpha - x_A \tan\alpha \end{aligned} \tag{1.5}$$

1.3.2 Centred expansion flows

At the corner of the walls at the nozzle exit, special care needs to be taken, as all characteristics of the expansion flow are centred on the corner. The value that is conserved along a characteristic is although different for all of them, and thus a singularity appears. How to tackle it is explained here for the top corner expansion flow. An analogous analysis can be made for the bottom corner expansion flow.

In order to avoid the singularity condition, the angles β of the Γ^- characteristics emanating from the top corner are set separately. A certain number of characteristics is chosen beforehand, and then they are distributed equally-spaced angle-wise. For this, the first and last angle are needed. As in general $\beta = \phi - \mu$, it is readily found that the angle of the first characteristic is $\beta_{first} = \phi_e - \mu_e$, where $\phi_e = 0$ is the flow angle at the nozzle exit, and μ_e is the Mach number at the nozzle exit. To find β_{last}, it is known that the flow properties in region 3 (defined earlier) are uniform, such that $\beta_{last} = \phi_{jet} - \mu_{jet}$, where ϕ_{jet} and μ_{jet} are the flow properties in region 3 at the jet boundary. To find μ_{jet} (therefore M_{jet}), the homentropic property of the flow is used, such that the relation given in Equation 1.6 is obeyed at any location, and that the total pressure is constant throughout the flow. Here, p_t is the total pressure and p is the static pressure.

$$\frac{p_t}{p} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma-1}} \tag{1.6}$$

Applying Equation 1.6 at the nozzle exit (i.e. using $M = M_e$, $p = p_e$), the total pressure p_t can be found. Afterwards, after applying the same equation at the jet boundary ($p = p_a$ because at the jet boundary there is no mass flow perpendicular to it, and thus the pressure forces are in equilibrium) and rearranging, it is found that:

$$M_{jet} = \sqrt{\frac{2}{\gamma - 1}\left[\left(\frac{p_t}{p_a}\right)^{\frac{\gamma-1}{\gamma}} - 1\right]} \tag{1.7}$$

As it can be seen, the fact that the jet boundary is in region 3 has not been used yet, showing that at the jet boundary the Mach number is constantly M_{jet} regardless of region. To find ϕ_{jet}, a Γ^+ characteristic can be taken from region 1 to region 3 for which $\nu_e - \phi_e = \nu_{jet} - \phi_{jet}$, and thus $\phi_e = \nu_e - \nu_{jet} + \phi_e$. With this information, β_{last} can be calculated.

Now, a number n_{char} of characteristics are created, with angles equally spaced from β_{first} to β_{last}. Let us concentrate now on one of these characteristics, for which the angle is β_i, as shown in Figure 1.3.

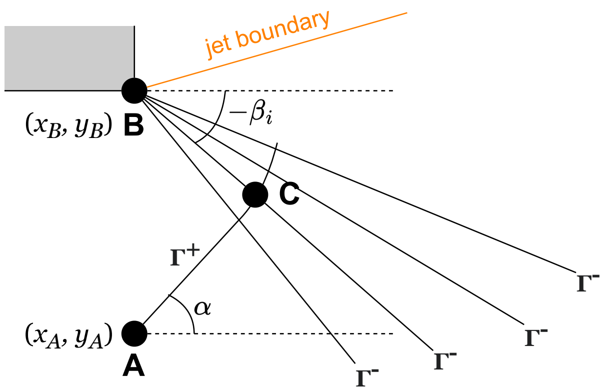

Figure 1.3: The set-up for the calculation of flow values in the origin of the corner expansion flow. Point B is the corner.

The target is to calculate the conditions and position of point C, which represents the first datapoint of characteristic i starting from the corner. It will be used to propagate the calculation further. Point B represents the top corner of the nozzle exit, and point A represents the datapoint immediately below it in the initial condition boundary of the flow domain. Its position is determined by the number of initial points (n_{init}) on this boundary that are chosen to be generated.

Point C being on the i Γ^- characteristic, it can be found that $\beta_i = \phi_C - \mu_C$. Therefore, $\phi_C = \beta_i + \mu_C$. Along the Γ^+ characteristic from A: $\nu_A - \phi_A = \nu_C - \phi_C$. Using the previously derived equation for ϕ_C and rearranging, we get that:

$$\nu_C - \mu_C - \beta_i - \nu_A + \phi_A = 0 \tag{1.8}$$

As both ν_C and μ_C are functions of M_C and the other values in the formula are known, this equation can be solved numerically for M_C, using for example Newton's method or a method already implemented in the `Scipy` library for Python. After that, ϕ_C can also be calculated, and thus all flow properties of point C are known. To calculate it position, the angle α of the Γ^+ characteristic is needed, which can be found as:

$$\alpha = \frac{1}{2}\left(\phi_A + \mu_A + \phi_P + \mu_P\right) \tag{1.9}$$

Afterwards, to find x_P and y_P, Equation 1.5 can be used, with the following changes: β_i instead of β and x_C and y_C instead of x_P and y_P.

1.3.3 Jet boundary

When propagating along the jet boundary, only one incoming characteristic is available for calculation (Γ^+ for the top jet boundary, Γ^- for the bottom). Therefore, a special case for the propagation algorithm is needed at these boundaries. In this subsection, the algorithm for the top jet boundary is described, with an analogous analysis possible for the bottom jet boundary.

The set-up of the analysis is shown in Figure 1.4. Point B represents the last known datapoint along the jet boundary. Point A is the first known datapoint along the Γ^- characteristic originating from B, immediately adjacent to it. We need to calculate the properties and position of point D, the intersection between the jet boundary (from B) and the Γ^+ characteristic originating in A. As D is on the jet boundary, as shown before, $M_C = M_{jet}$. From the Γ^+ characteristic: $\nu_A - \phi_A = \nu_D - \phi_D$. Rearranging, it is found that $\phi_D = \nu_D - \nu_A + \phi_A$.

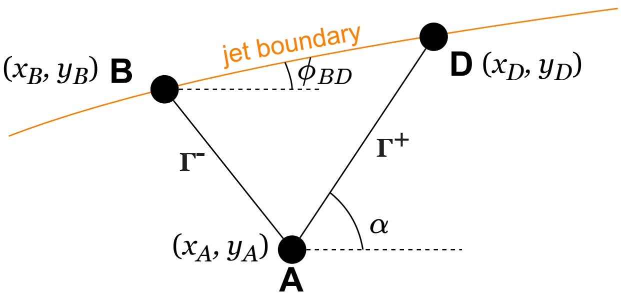

Figure 1.4: The set-up for the propagation of the flow along the jet boundary.

As now all the properties of point D are known, we have to calculate its position. The streamline angle ϕ_{BD} between points B and D is taken as the average of the local angle in the two points, i.e. $\phi_{BD} = \frac{1}{2}\left(\phi_B + \phi_D\right)$. Angle α is calculated as follows:

$$\alpha = \frac{1}{2}\left(\phi_A + \mu_A + \phi_D + \mu_D\right) \tag{1.10}$$

The coordinates x_D and y_D can be found using Equation 1.5, with the following changes: ϕ_{BD} instead of β, and x_D and y_D instead of x_P and y_P.

1.4 Shock formation and streamline calculation

When convergent characteristics intersect, as explained in Section 1.1, shocks start to form, and the MoC breaks down. Therefore, the program needs to be stopped when these intersections are encountered. In the first subsections of this section, an algorithm for checking if two line segments intersect and where the intersection happens will be explained. This will also be used when calculating the streamline path, as the intersection between a characteristic and the streamline up to that point is needed.

1.4.1 Checking if two line segments intersect[1]

Assume we have two line segments, $[AB]$ and $[CD]$. For them to intersect, A and B need to be separated by $[CD]$, and C and D need to be separated by $[AB]$. This is shown in Figure 1.5.

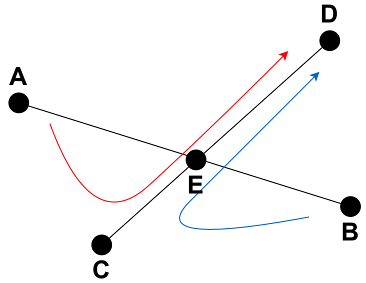

Figure 1.5: Two line segments $[AB]$ and $[CD]$ intersecting.

For A and B to be separated by $[CD]$, the point sequences $\{A, C, D\}$ (shown in red) and $\{B, C, D\}$ (shown in blue) need to have different orientations. Similarly, for C and D to be separated by $[AB]$, the point sequences $\{A, B, C\}$ and $\{A, B, D\}$ need to have different orientations.

In order to see the orientation of each set of points $\{A, B, C\}$, we can check if the set is counter-clockwise (ccw), and if it is not, then it is clock-wise (cw). It is assumed that the points cannot be collinear. To check if the set of points $\{A, B, C\}$ is ccw, we have to check that the slope of the line (AC) is larger than the slope of line (AB). If we know the coordinates of the points A, B and C, this is equivalent to checking that:

$$(y_C - y_A) \cdot (x_B - x_A) > (y_B - y_A) \cdot (x_C - x_A) \tag{1.11}$$

Therefore, if we know the coordinates of points A, B, C and D, it can be checked whether they intersect.

1.4.2 Calculating intersection position

Once two line segments $[AB]$ and $[CD]$ have been checked to intersect, the intersection position is required. This can be done by deriving line equations from the points positions, of the type $y = m \cdot x + n$, where the constants m and n define the line. Let the line (AB) be defined as $y = m_1 \cdot x + n_1$, and the line (CD) as $y = m_2 \cdot x + n_2$. Then, using the geometry of the figure Figure 1.5 and the positions of the points, it is found that for line (AB):

$$m_1 = \frac{y_B - y_A}{x_B - x_A} \qquad n_1 = y_A - m_1 \cdot x_A \tag{1.12}$$

and similar relations for line (CD). Then, using the fact that point E lies on both line (AB) and line (CD), it is found that its position is given by:

$$x_E = \frac{n_2 - n_1}{m_1 - m_2} \qquad y_E = m_1 \cdot x_E + n_1 \tag{1.13}$$

1.4.3 Checking the shock formation and propagating the streamline

A shock begins forming when two converging characteristics of the same type intersect. Exactly which points to use to check for intersections is explained in the next chapter. Also in the next chapter, the intersection between a characteristic and the streamline will be used to propagate its path and value further.

[1]Method adapted from `https://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/`

Numerical implementation 2

In this chapter, the numerical implementation of the program will be discussed, along its data structure and the order of operations that the program follows. The program is written in Python, and uses the libraries `NumPy`, `SciPy` and `Matplotlib`. It contains 19 functions used for solving the problem, and 6 functions used for plotting.

2.1 Data structure

The main data structure consists of 6 globally defined matrices (2D NumPy arrays) for each of the flow properties and node positions: `x` (x-position), `y` (y-position), `phi` (flow angle ϕ), `M` (Mach number M), `nu` (Prandtl-Meyer angle ν) and `mu` (Mach angle μ). The first index (`i`) represents the number of the Γ^- characteristic that passes through that point/node, and the second index (`j`) represents the number of the Γ^+ characteristic that passes through that point/node. This is shown in Figure 2.1, showing the relationship between a node at coordinates `[i, j]` and its relation to other node positions.

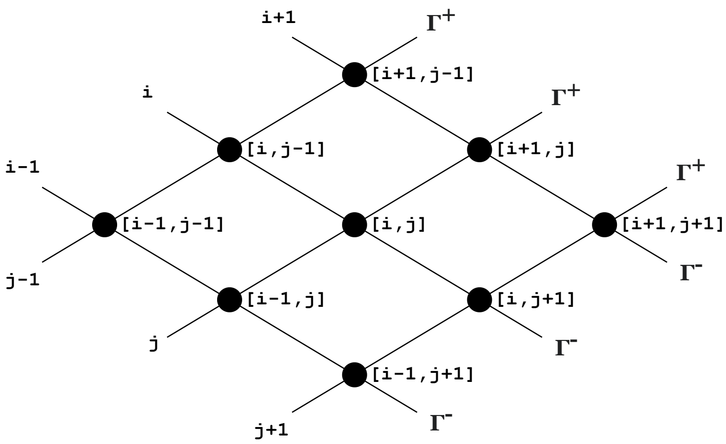

Figure 2.1: Index notation in the matrix data structure. Highlighting the relation between neighbouring nodes. The flow direction is from left to right.

When initialising the problem, firstly points are put along the initial boundary at the nozzle exit, defining new Γ^+ and Γ^- characteristics, but also near the corners in the expansion fans. Furthermore, new nodes are created at the intersection between characteristics and the jet boundary. In order to make the indexing convention clear for these special cases, the diagram in Figure 2.2 is provided.

As it can be seen, a number of `no_init` initial nodes are created along the initial nozzle exit boundary. Along with these, the same number of Γ^- and of Γ^+ characteristics are created, numbered from `0` to `no_init-1`. As further down the stream, new Γ^- characteristics will be created starting from the top jet boundary (as seen with the addition of point D in Figure 1.4), the numbering of the Γ^- characteristics (index `i`) starts from the bottom up, and then to the right, as shown in Figure 2.2. For an analogous reason, the Γ^+ characteristics (index `j`) start from the top, and then to the right. As such, starting from the top, the indices of the initial boundary nodes are `[no_init-1,0]`, `[no_init-2,1]`, ... `[0, no_init-1]`.

The presence of the sharp top corner will generate a number of `no_char` Γ^- characteristics, emanating from the corner, including `i=no_init-1`, which comes from the initial boundary. Therefore, they will vary from `i=no_init-1` to `i=no_init+no_char-2`, in order to be `no_char` in total. In order to have a consistent numbering, the corner will have to be copied a total of `no_char` times (including the original point created in the initial boundary). This is because all the `no_char` Γ^- characteristics created by the centred expansion fan intersect there. Therefore, the corner will have the following set of indices: `[no_init-1,0]`, `[no_init,0]` ... `[no_init+no_char-2,0]`, proving its singularity properties.

Lastly, when a Γ^+ characteristic intersects a jet boundary in a place where there does not exist already a node, it will create one, and thus also a new Γ^- characteristic. Thus, its index `i` will be the one of the previous streamline node, plus 1. In Figure 2.2, the previous node is the corner, for which its last possible index is used in this case, i.e. `i = no_init+no_char-2`. Therefore, the indices of the new node on the streamline is `[no_init+no_char-1,1]`, as it is also on the intersection with the `j=1` Γ^+ characteristic.

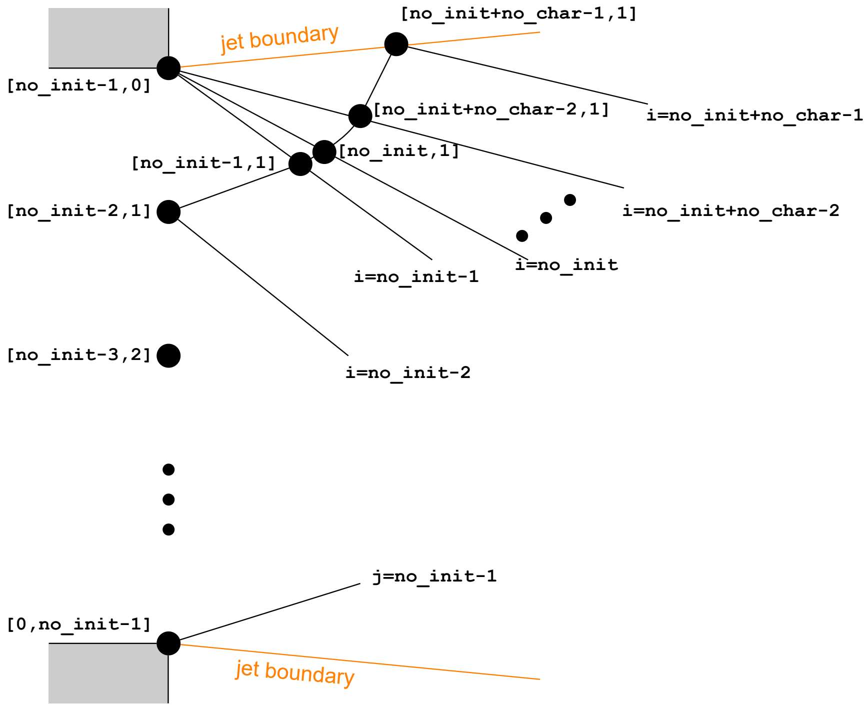

Figure 2.2: Index notation of the nodes that are calculated in the beginning. Flow is from the left to the right.

There are two more data structures used through the code: one for the shock, and one for the streamline. The one for the shock consists of two 1D NumPy array, namely `x_shock` and `y_shock`, which stores the locations at which shocks start to develop. The one for the streamline consists of six 1D NumPy arrays, namely `xs`, `ys`, `phis`, `Ms`, `nus` and `mus` (with the same meaning as for the global node structure), each for a parameter of the flow along the streamline.

All the above-mentioned arrays are initialised filled with not-a-number values (`np.nan`), and these values are replaced with real numbers as they are calculated through propagation. This is to make it easier to check whether a point was already calculated at a particular location, and also to make plotting easier.

2.1.1 Input variables

The following variables have to be provided as input for the program:

- `Me`: the nozzle exit Mach number M_e. Problem value is `2.0`.
- `p_a`: the atmospheric pressure p_a. Problem value is `1` atm.

- `p_e`: the nozzle exit (static) pressure p_e. Problem value is `2 * p_a`.
- `phi_e`: the nozzle exit flow angle phi_e. Problem value is `0.0`.
- `gamma`: the specific heat ratio of the gas γ. Problem value is `1.4` (air).
- `h`: height (diameter) of the nozzle exit. Problem value is `1.0`.
- `no_init`: number of nodes to be created on the initial boundary.
- `no_char`: number of characteristics to be created on each corner expansion fan.
- `dim`: dimension of (each axis of) each data structure matrix.
- `no_steps`: number of propagation steps after which the program terminates, if it has encountered no shock.
- `option`: variable for choosing between plotting node-based values of the Mach number field, or a continuous field calculated by interpolation. `0` for node-based, `1` for interpolation.
- `xs[0]`: the initial x-coordinate of the streamline. Problem value is `0.0`
- `ys[0]`: the initial y-coordinate of the streamline.
- `phis[0]`: the initial flow angle of the streamline. Problem value is `phi_e`
- `Ms[0]`: the initial Mach number of the streamline. Problem value is `Me`

2.2 Functions used in the program

In this section, the functionality and purpose of each of the functions is shortly explained, with short comments where necessary. For a more detailed explanation, the code is provided in Appendix A, which includes a multitude of comments at every step, explaining its functionality. An exception is made for checking for the shock formation and propagating the streamline, which are explained in separate subsections and the end of this section. The non-plotting functions used in the program are:

- `nu_from_M()`, `mu_from_M()`, `M_from_mu()` and `M_from_nu()`: functions that convert between the M, ν and μ values at a certain data point. To convert from ν to M, the roots of the function `nu_M_function()` need to be found numerically, which is done with `scipy.optimize()`
- `calc_Mjet()`: calculates the Mach number at the jet boundary M_{jet}.
- `calc_e_jet()`: calculates ν_e, μ_e, M_{jet}, ν_{jet}, μ_{jet}, ϕ_{jet} (in region 3 of Figure 1.1, p_t.
- `ccw()`: checks whether three points are in counter-clockwise order, as explained in Subsection 1.4.1.
- `check_intersect()`: checks whether two line segments intersect, as explained in Subsection 1.4.1
- `get_intersect()`: calculates intersection position of two line segments, as explained in Subsection 1.4.2
- `init_exit()`: initialises the values at the initial boundary of the nozzle exit, as explained in Section 2.1.
- `assign_top_corner()`: assigns the values of the first set of nodes at the expansion flow of near top corner, as explained in Section 2.1. It also creates the necessary copies for the top corner. It follows the procedure shown in Subsection 1.3.2 by first calculating β_{first} and β_{last}, and then the required values, which correspond to point(s) C in Figure 1.3. Point A is the point with coordinates `[no_init-2,1]`. To use this function, the roots of the function `beta_M_function()` need to be found numerically, which is done with `scipy.optimize()`.

- `assign_bottom_corner()`: equivalent version of `assign_top_corner()` for the bottom expansion fan. To use this function, the roots of the function `alpha_M_function()` need to be found numerically, which is done with `scipy.optimize()`.

- `prop_top_BC()`: propagate (calculate) the parameters along the top jet boundary, as explained in Subsection 1.3.3. For a point D with coordinates `[i,j]`, point A has coordinates `[i-1,j]` and point B has coordinates `[i-1,j-1]`.

- `prop_bottom_BC()`: equivalent version of `prop_top_BC()` for the bottom jet boundary.

- `prop_normal()`: propagate (calculate) the parameters within the flow, as explained in Subsection 1.3.1. For a point P with coordinates `[i,j]`, point A has coordinates `[i-1,j]` and point B has coordinates `[i,j-1]`.

- `propagation()`: the main loop of the propagation. The parsing is done along a diagonal that has the formula `i+j==no_init-1+k`, where `k` represents the step number, starting from `0`. When `k==0`, the points represent the initial nozzle exit boundary. For one step, `j` (the second index) is varied, while `i` is calculated to maintain the relation of step `k`. Afterwards, the program moves on to step `k+1`. For each point, it is checked if it already exists, and if not, one of the three propagation functions described earlier are applied, if possible. The function also checks for shock formation while propagating, and retains the values where this happens. The loop is stopped once the step containing the first shock is completed. This is done in order to find the symmetric shock. Also during propagation, the function checks if it can propagate the streamline. This and the shock formation are explained in more detail in the next subsection.

2.2.1 Checking for shock formation

A shock starts developing when two characteristics of the same type intersect. Therefore, there are two cases: two Γ^+ characteristics intersect, or two Γ^-. In this subsection, only the first case is treated, with the second one being analogous. Two Γ^+ characteristics can be first detected to intersect when the program reaches the black square in Figure 2.3.

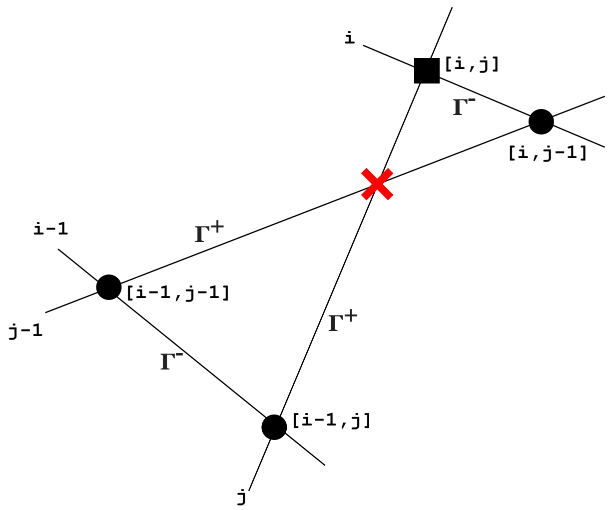

Figure 2.3: First detection of a shock from the intersection of two Γ^+ characteristics. The current calculated node is the black square, with indices `[i,j]`.

As it can be seen from the figure, it results that the code has to check for intersection between the segments [`[i,j]`, `[i-1,j]`] and [`[i,j-1]`, `[i-1,j-1]`]. It does so for every node that is calculated, using `check_intersect()`. Afterwards, it uses `get_intersect()` to get the position of the intersection (shock), and stores it in the arrays `x_shock` and `y_shock`.

2.2.2 Propagating the streamline

A streamline can be propagated whenever a new intersection is found between the last expected path of the streamline, and either a Γ^- or a Γ^+ characteristic. In this section, only the first case is treated, the second one being analogous. The set-up is shown in Figure 2.4.

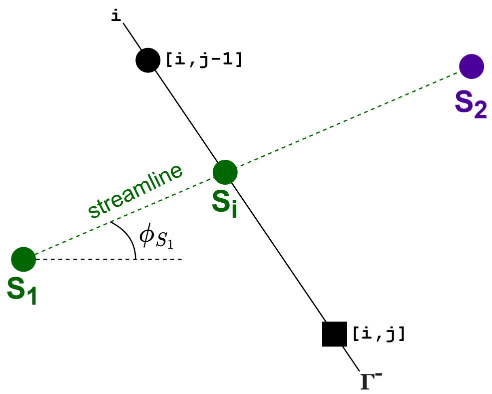

Figure 2.4: Propagation of the streamline from the intersection with a Γ^- characteristic. The current calculated node is the black square, with indices `[i,j]`.

The black square represents the current node in the global propagation. S_1 is the last known point on the streamline, and ϕ_{S_1} is its flow angle. S_1 has spatial coordinates (`xs[i_s]`, `xs[i_s]`), where `i_s` is the index of the last data entry of the streamline arrays. S_2 is a projection along the streamline with flow angle ϕ_{S_1} with a certain length, long enough to assure an intersection with a characteristic. If it taken that $x_{S_2} = x_{S_1} + 10 \cdot h$, where h is the nozzle exit height (diameter), then $y_{S_2} = y_{S_1} + 10 \cdot h \cdot \tan \phi_{S_1}$. It is clear now that we need to check the intersection between line segments $[S_1, S_2]$ and [`[i,j-1]`, `[i,j]`] using `check_intersect()`. If they intersect, the intersection position is calculated using `get_intersect()`, and these values are added to the end of the streamline position arrays. For the parameter values (M, ϕ etc.) the values are taken from the global node with indices `[i,j-1]`, as along a characteristic, until it reaches another node, the parameters are assumed constant. To calculate the (static) pressure along the streamline, a rearranged Equation 1.6 is used, where p_t is constant throughout the whole flow, and M is `Ms`.

2.3 Overall order of operations

Now that all the building components have been explained, the order in which these operations are performed is shortly presented. For more detailed information, please have a look at the code in Appendix A.

The program begins with initialising the initial nozzle exit boundary, using `init_exit()`. Then, the nodes in the expansion fans near the corners are initialised, using `assign_top_corner()` and `assign_bottom_corner()`. Finally, the propagation is performed using `propagation()`, which checks all three propagation cases, checks for shock and also propagates the streamline. The program stops after a shock has been encountered and the step where the shock has been found is finished, or when the maximum number of steps `no_steps` has been reached. Finally, the plotting routines are called.

Results and Discussions 3

In this chapter, the results will be presented and discussed. The terminology regarding the different regions is the same as explained in Section 2.1.

3.1 Characteristics and jet bundary

In Figure 3.1, the characteristics originating in the expansion fans and the jet boundaries are plotted.

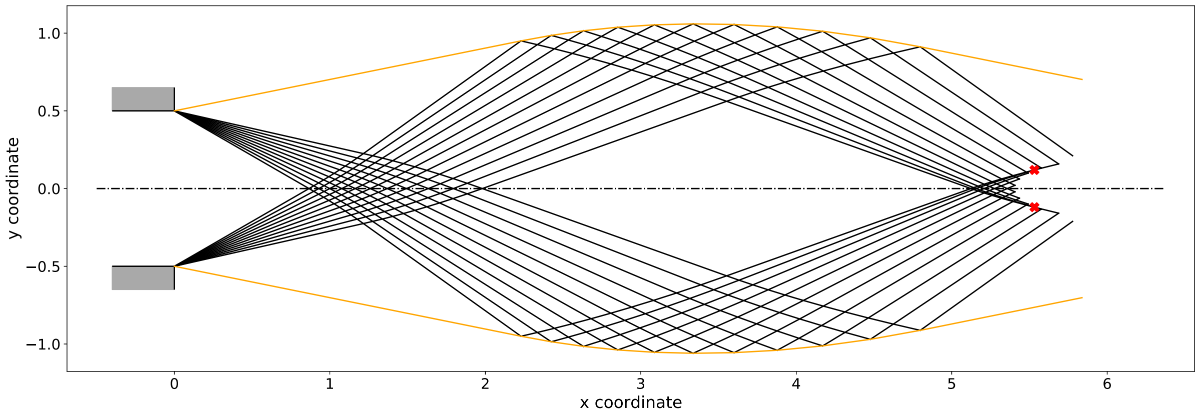

Figure 3.1: Characteristics of the centred expansion fans. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=31`, `no_char=11`. The characteristics are plotted in black, and the jet boundary in orange.

The characteristics start straight from the expansion fans, distributed equidistantly in region 2, as expected. Then, in region 4, the top and bottom expansion fans interact, curving each other. The characteristics originating from the bottom expansion fans are bent upwards by the top expansion fan. The characteristics originating from the top expansion fan are bent downwards by the bottom expansion fan. These curving directions are to be expected, as the streamlines are bent in the same directions by the respective expansion fans. After they interact, the characteristics continue straight (in region 5), until they encounter the jet boundary.

The jet boundary begins as a straight line from the corners of the nozzle exit walls. It maintains its straight line behaviour, until it intersects the expansion fan characteristics. At these regions (region 7), the interaction between the characteristics and the jet boundary bends the latter inwards, going from a divergent to a convergent boundary. Afterwards, the jet boundaries continue again as straight lines.

Coming back to the characteristics, these are also affected by the interaction with the jet boundary. At the intersection, the characteristics reflect and change nature (from Γ^+ to Γ^- and vice-versa). Furthermore, the direct and reflected characteristics intersect with each other (region 7), once again curving. All of them are bent inwards (i.e. towards the symmetry line), opposite to the effect of the interaction in region 4. Afterwards, the characteristics continue straight (region 8). However, (mainly) due to their reflection, and the interaction with the original characteristics, they are now convergent, and will eventually lead to the formation of shocks. They are now compression fans, as they no longer expand the flow, but compress it. In region 9, the two compression fans intersect and interact. However, around this region is also where shocks start forming, and thus the calculation is stopped.

3.2 Mach distribution

In this section, the results of the Mach number distribution and a discussion on it will be given. The "exact" results are presented in Figure 3.2, where the value of the Mach number is only plotted at the points where it is actually calculated. In Figure 3.3, these values have been interpolated to fill up the jet area, and to make the figure easier to understand. However, due to this reason, the figure has to be interpreted with care. Around and after the region where shocks form (plotted with red crosses), the interpolation does not have the necessary data to calculate in the centre after the shock (as can be seen from the lack of datapoints in Figure 3.2) and thus is not realistic. In spite of this, the future analysis will be done using the interpolated figures, for ease of understanding and interpretation.

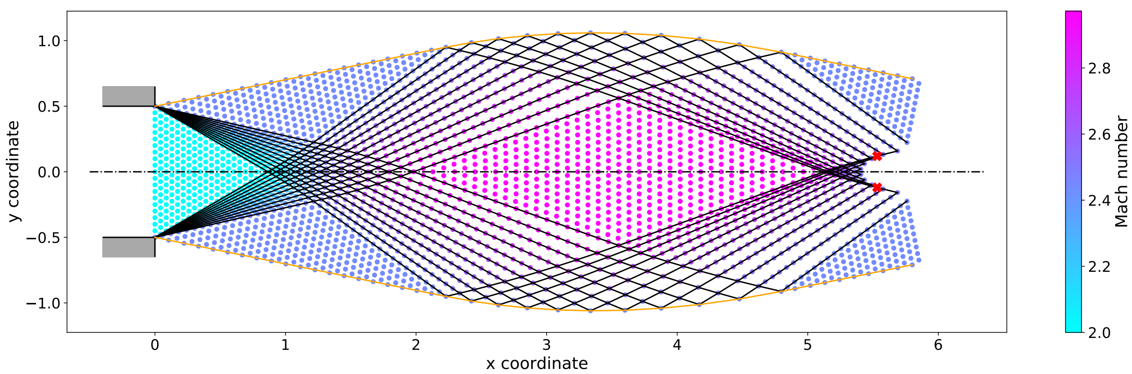

Figure 3.2: Mach number values at the calculated nodes. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`. The characteristics are plotted in black, and the jet boundary in orange. The Mach number legend is on the right.

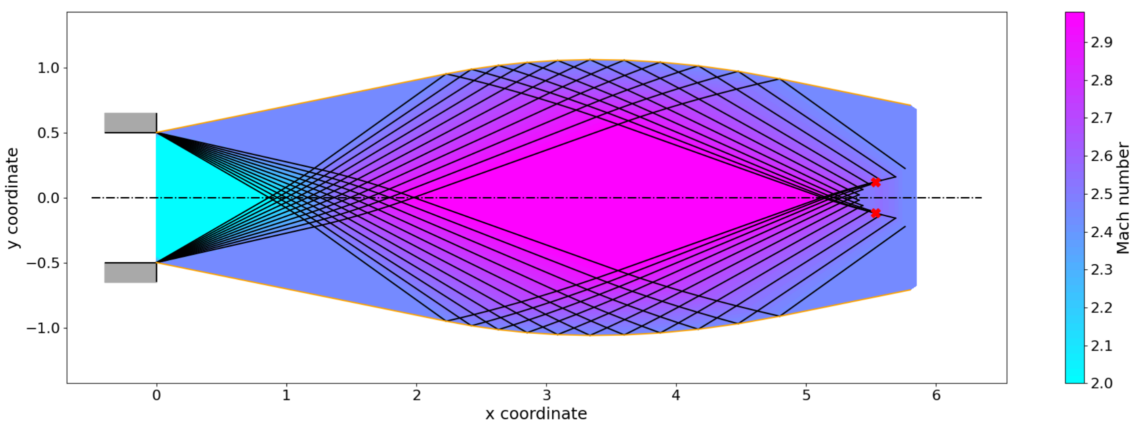

Figure 3.3: Mach number field calculated as an interpolation of the data available at the calculation nodes. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`. The characteristics are plotted in black, and the jet boundary in orange. The Mach number legend is on the right.

In region 1, immediately after the nozzle exit, the flow maintains its initial conditions of $M_e = 2.0$. As the flow passes through the expansion fans in regions 2, the Mach number increases gradually, until it reaches $M_{jet} \approx 2.44$. This makes sense, as expansion fans decrease the pressure, in term accelerating the flow. Near the corners, this jump is more sudden. In region 3, it maintains the uniform value of M_{jet}. In region 5, the Mach number increases further gradually from M_{jet} to $M_{max} \approx 2.97$. This is the maximum Mach number the flow will experience. In region 4, the Mach number increases from M_e to M_{jet} and then to M_{max}, smoothly connecting all the previously-mentioned regions. In region 6, the flow is uniformly at M_{max}. When it interacts with the compression fans in regions 8, the Mach number decreases from M_{max} back to M_{jet}. This again makes sense, as compression fans increase the pressure of the flow, which in turn decelerates it. In region 7, the flow has a gradual variation, from M_{jet} at the jet boundary, to M_{max} near region 6. Theoretically, in region 9 the flow would further decelerate. However, this is not perfectly clear in the figure.

An interesting and logical observation to be made is that at the jet boundary, the Mach number is always M_{jet}. The reason for it was explained earlier in the report, and is due to the equilibrium of pressure forces at the boundary with the atmospheric pressure. This property can be seen throughout regions 3 and at the external sides of regions 7.

3.3 Streamline properties

In this section, the path and pressure distribution of two streamlines will be plotted and discussed. These are the centreline streamline and the quarter-height streamline.

3.3.1 Centreline streamline

In Figure 3.4, the path of the centreline streamline is shown in golden yellow. In Figure 3.5, the variation of the (static) pressure along the streamline is shown.

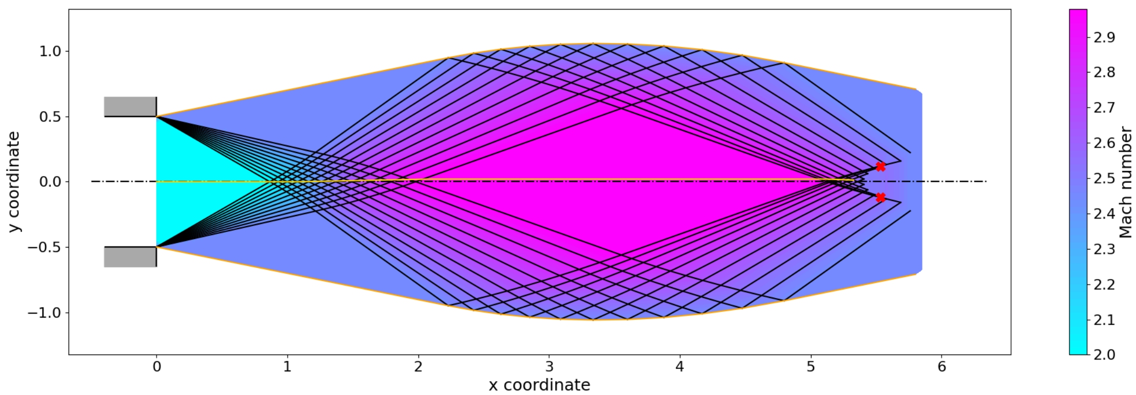

Figure 3.4: Path of the centreline streamline, superimposed on the Mach field and characteristics. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`. The characteristics are plotted in black, and the jet boundary in orange. The Mach number legend is on the right. The streamline path is plotted in golden yellow.

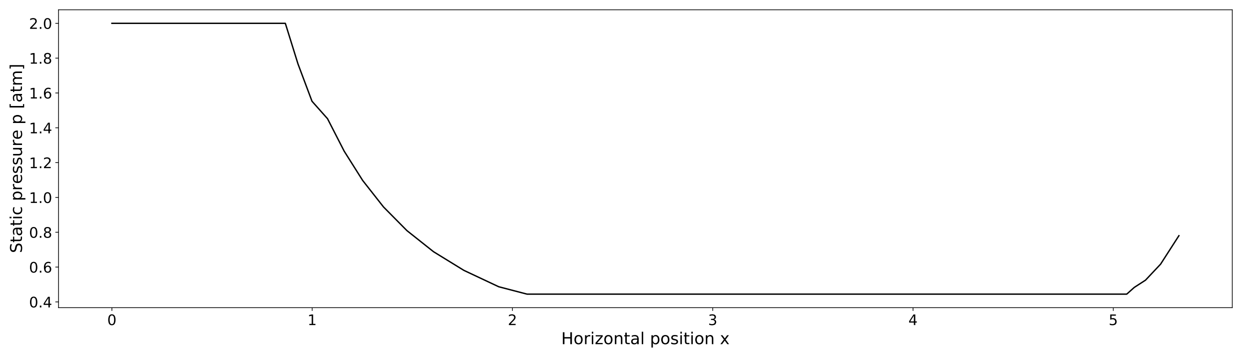

Figure 3.5: Plot of the static pressure (in atm) along the centreline streamline, as a function of horizontal distance x. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`.

As it can be seen, the orientation of the centreline streamline is practically unaffected, maintaining a zero-angle deflection along the whole domain. The small deviation seen in Figure 3.4 is most probably sue to calculation and discretisation errors. It starts from region 1, and then moves to region 4, where its Mach number increases, reaching a maximum in region 6. It then enters region 9, where the Mach number starts decreasing.

This variation can be seen also in the pressure plot, as when the Mach number increases, the pressure decreases, and vice-versa. Thus, while the streamline is in region 1, the pressure remains at $p_e = 2\,\text{atm}$. When it enters region 4, it has a sudden drop, which gradually shifts to the minimum pressure value in region 6. It remains at that pressure, as region 6 is uniform. It makes sense for the smallest pressure to be at the fastest region. When it enters region 9, the pressure starts increasing again, but the calculation stops, as a shock was encountered.

3.3.2 Quarter-height streamline

In Figure 3.6, the path of the quarter-height streamline is shown in golden yellow. In Figure 3.7, the variation of the (static) pressure along the streamline is shown.

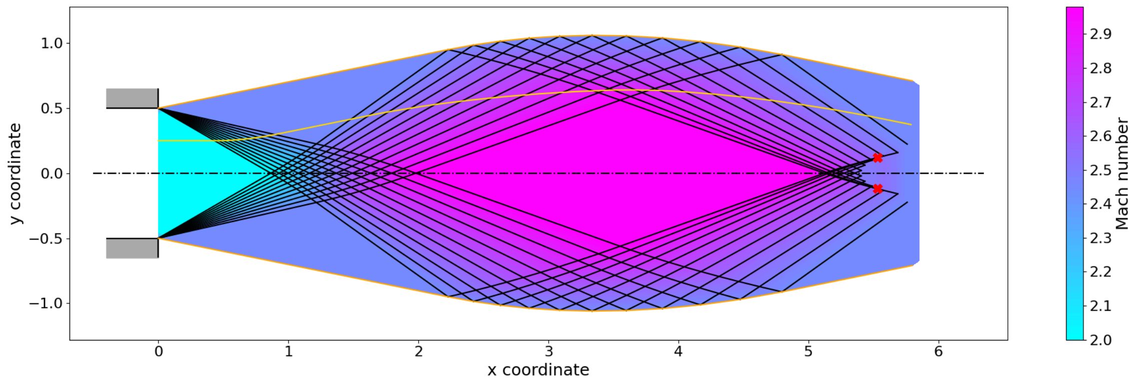

Figure 3.6: Path of the quarter-height streamline, superimposed on the Mach field and characteristics. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`. The characteristics are plotted in black, and the jet boundary in orange. The Mach number legend is on the right. The streamline path is plotted in golden yellow.

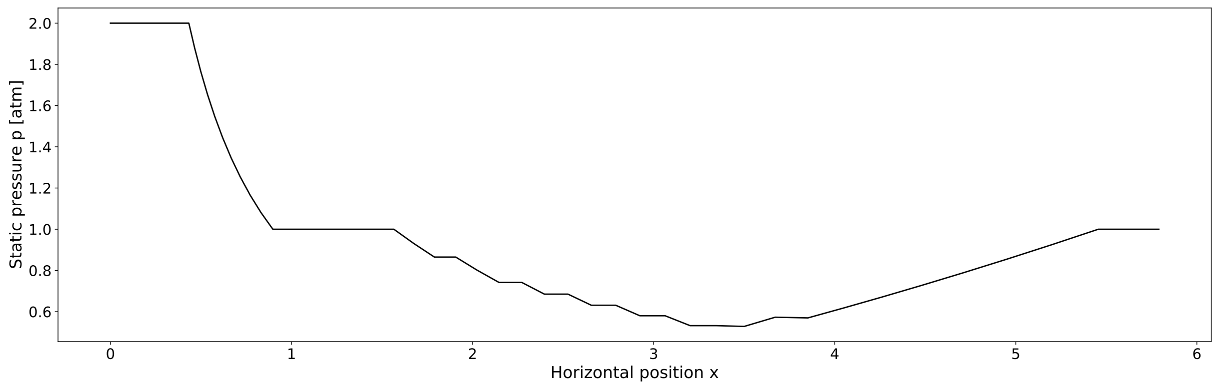

Figure 3.7: Plot of the static pressure (in atm) along the quarter-height streamline, as a function of horizontal distance x. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=11`.

As it can be seen, the orientation of the streamline is affected by the expansion and compression flows. It starts straight during region 1, at Mach number M_e, then is bent upward by the top expansion fan, and its Mach number increases to M_{jet}. In region 3 it travels straight, unaltered, with a constant Mach number. When it enters region 5, it is bent downwards by the bottom expansion fan, while its Mach number increases. it shortly enters region 7, still being bent downwards. Here it reaches its maximum velocity. In then enters region 8, where it is bent further down by the action of the converging compression fan. Its Mach number now decreases. It then enters region 10, where it travels unaltered. Its Mach number remains at $Mjet$.

This variation can be seen also in the pressure plot, as when the Mach number increases, the pressure decreases, and vice-versa. While the streamline is in region 1, it has pressure p_e. In region 2 the pressure decreases due to the expansion fan, and reaches the atmospheric pressure $p_a = 1\,\mathrm{atm}$ in region 3. This makes sense, as the region is uniform and is in equilibrium with the atmosphere through the jet boundary. The pressure then continues to decrease through region 5, until it reaches a minimum in region 7 (when the Mach number reaches a maximum). It then starts increasing in region 8 due to the compression fan, and reaches back p_a in region 10. It is interesting to note that the two pressures of the streamline in region 3 and 10 are both the same and constant, as they both border the jet boundary and are uniform regions.

Another interesting note appears when comparing this streamline with the centreline one. Here, there are two negative pressure jumps, as the flow passes two distinct expansion flows, compared to the combined interaction as for the centreline streamline. Furthermore, the quarter-height streamline does not reach such a low minimum pressure as the centreline one, as it never enters region 8, and thus never achieves the maximum Mach number of the domain.

3.4 Accuracy of computation

The accuracy of the computation, as many simplifications and assumptions have been made, that are given and explained in Section 1.2. Besides these, there are also some computation-specific simplifications, that affect the computation. The most important would be that the expansion fan consists of discrete characteristics, whereas in reality it would be made out of an infinity of them, for a very smooth and gradual transition. Another simplification is that during propagation, the position of the new nodes are estimated based on the averages of the angles of the characteristics in the old and new nodes, where in reality the characteristics are not necessarily straight between the points.

The effects of these simplifications can be seen in multiple locations, from the step-wise "curvature" of the jet boundary in Figure 1.1, step-wise curving of the characteristics when interacting, step-wise curving of the streamline when encountering characteristics, the step-wise decreasing in pressure in Figure 3.7 etc. Most of these issues can be diminished by using a larger number of characteristics in the expansion fans (`no_char`), which would decrease the distances between consecutive nodes and characteristics. To show this, the pressure plot for the quarter-height streamline is shown in Figure 3.8 for `no_char=101`.

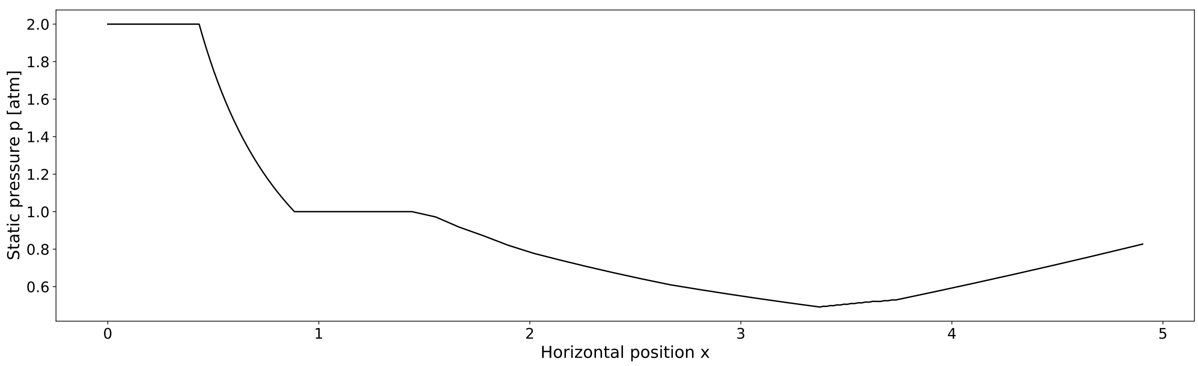

Figure 3.8: Plot of the static pressure (in atm) along the quarter-height streamline, as a function of horizontal distance x. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=21`, `no_char=101`.

As it can be seen, the pressure function is now much smoother, showing that the pressure does not reach a horizontal plateau of minimum value, but actually has a more "pointy" nature, of decreasing and then sharply increasing. It also shows that a (slightly) smaller pressure than initially estimated is achieved. Another way to increase the resolution of these graphs and plots would be to increase `no_init`. However, this number would mostly affect the values in the uniform regions, and thus not have a significant effect.

3.5 Location of the shock formation

The location of the shock formation is given in Table 3.1 for a varying exit Mach number, and in Table 3.2 for a varying pressure ratio. Furthermore, in Figure 3.9 to Figure 3.12 there are given the shock locations (red crosses) in context of the whole jet flow, for varying pressure ratio.

Table 3.1: Position of the shock formation as a function of exit Mach number and pressure ration, with a varying exit Mach number.

M_e	p_e/p_a	x_{shock}	y_{shock}
1.5	2	3.933853	0.156
2	2	5.447934	0.097671
3	2	8.687244	0.105703
4	2	11.91832	0.119611

Table 3.2: Position of the shock formation as a function of exit Mach number and pressure ration, with a varying p_e/p_a pressure ratio.

M_e	p_e/p_a	x_{shock}	y_{shock}
2	1.5	4.517683	0.259944
2	2	5.447934	0.097671
2	3	6.932256	0.360697
2	4	8.010467	0.961293

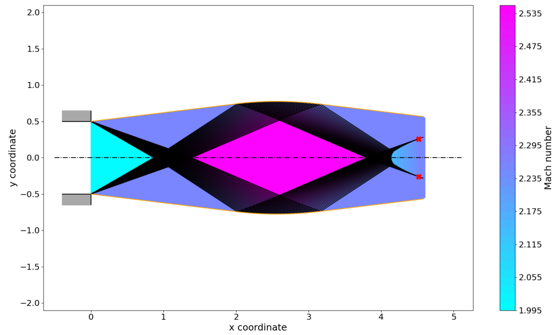

Figure 3.9: Mach number field. $M_e = 2.0$, $p_e/p_a = 1.5$, `no_init=31`, `no_char=31`.

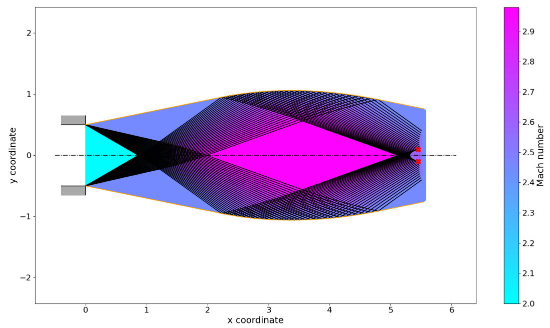

Figure 3.10: Mach number field. $M_e = 2.0$, $p_e/p_a = 2$, `no_init=31`, `no_char=31`.

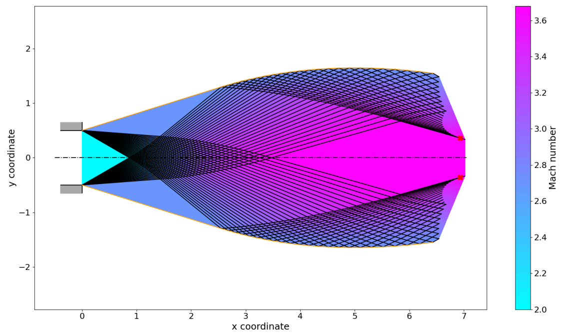

Figure 3.11: Mach number field. $M_e = 2.0$, $p_e/p_a = 3$, `no_init=31`, `no_char=31`.

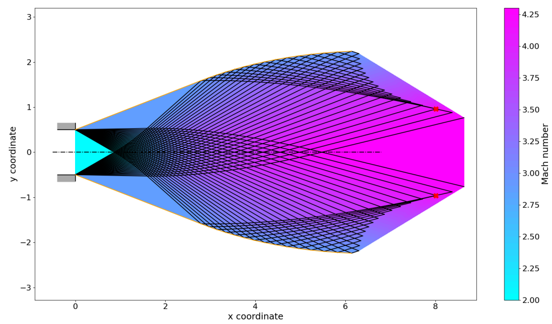

Figure 3.12: Mach number field. $M_e = 2.0$, $p_e/p_a = 4$, `no_init=31`, `no_char=31`.

For a constant pressure ratio and varying Mach number, from Table 3.1 it can be seen that the higher the Mach number, the later the shock develops. This is because the jet flow gets longer, and the shock still happens somewhere near region 9. The variation in vertical position is small. No significant effects are seen in the plots, and as such they are not included.
For a constant Mach number and varying pressure ratios, it can be seen from the figures that the position of the shock varies significantly. Looking at Table 3.2, we can deduct that the higher the ratio, at a larger x-coordinate the shock forms. However, the figures show that for increasing pressure ratios, the shock appears earlier from the point of view of the regions defined at the beginning of the report. For $p_e/p_a = 1.5$, the shock forms past region 9, after the intersection between the two compression fans. For $p_e/p_a = 2$, the shock forms around region 9, near the intersection of the two compression fans. For $p_e/p_a = 3$, the shock forms in region 8, and the calculation does not even reach region 9. For $p_e/p_a = 4$, the shock forms in region 7, no calculation being preformed for regions 8 and further. Also important to note is that with increasing p_e/p_a, the jet flow becomes significantly wider.

Code A

In this appendix, the code of the program is given, including all its functions and plotting routines. Comments have been written throughout, to ease the understanding of the reader/user. The code can also be found online on GitHub at the following address: `https://github.com/aparvulescu/moc-nozzle-exit`.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize


# Commands for making font size in matplotlib bigger
SMALL_SIZE = 16
MEDIUM_SIZE = 18
BIGGER_SIZE = 22

plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title


# -----------------------------------------------------------
# Solver functions
# -----------------------------------------------------------

def nu_from_M(M):
    """
    Convert the Mach number to its respective Prandtl-Meyer angle in radians.

    :param M: Mach number
    :return: Prandtl-Meyer angle [radians]
    """
    return np.sqrt((gamma + 1) / (gamma - 1)) * np.arctan(np.sqrt((gamma - 1) / (gamma + 1)
    * (M * M - 1))) \
           - np.arctan(np.sqrt(M * M - 1))


def mu_from_M(M):
    """
    Convert the Mach number to its respective Mach angle in radians.

    :param M: Mach number
    :return: Mach angle [radians]
    """
    return np.arcsin(1 / M)


def M_from_mu(mu):
    """
    Convert the Mach angle to the Mach number.

    :param mu: Mach angle [radians]
    :return: Mach number
    """
    return 1 / np.sin(mu)


def M_from_nu(nu):
    """
    Convert the Prandtl-Meyer angle to the Mach number.

    :param nu: Prandtl-Meyer angle [radians]
    :return: Mach number
    """
    global nu_global
    nu_global = nu
```

```python
    sol = optimize.root(nu_M_function, np.array([Me]), tol=1e-8)
    return np.squeeze(sol.x)


def nu_M_function(M):
    """
    Function required for the root-finding method for finding the Mach number from the Prandtl-Meyer angle.
    Equivalent with the expression nu(M) - nu_global == 0.

    :param M: Mach number to be solved for
    :return: nu(M) - nu_global
    """
    return np.sqrt((gamma + 1) / (gamma - 1)) * np.arctan(np.sqrt((gamma - 1) / (gamma + 1) * (M * M - 1))) \
           - np.arctan(np.sqrt(M * M - 1)) - nu_global


def calc_Mjet(Me, pe, pa):
    """
    Calculate the jet boundary Mach number (Mjet) from the exit and atmospheric conditions.

    :param Me: Exit Mach number
    :param pe: Exit static pressure [atm]
    :param pa: Atmospheric static pressure [atm]
    :return: Jet boundary Mach number
    """
    pt = pe * (1 + (gamma - 1) / 2 * Me * Me) ** (gamma / (gamma - 1))
    return np.sqrt(2 / (gamma - 1) * ((pt / pa) ** ((gamma - 1) / gamma) - 1))


def beta_M_function(M):
    """
    Function required for the root-finding method of the nodes immediately adjacent to the top corner. Equivalent with
    the expression nu(M) - mu(M) - beta_global - nu_a_global + phi_a_global == 0.

    :param M: Mach number to be solved for
    :return: nu(M) - mu(M) - beta_global - nu_a_global + phi_a_global
    """
    return np.sqrt((gamma + 1) / (gamma - 1)) * np.arctan(np.sqrt((gamma - 1) / (gamma + 1) * (M * M - 1))) \
           - np.arctan(np.sqrt(M * M - 1)) - np.arcsin(1 / M) - beta_global - nu_a_global + phi_a_global


def alpha_M_function(M):
    """
    Function required for the root-finding method of the nodes immediately adjacent to the bottom corner. Equivalent
    with the expression nu(M) - mu(M) + alpha_global - nu_b_global + phi_b_global == 0.

    :param M: Mach number to be solved for
    :return: nu(M) - mu(M) + alpha_global - nu_b_global + phi_b_global
    """
    return np.sqrt((gamma + 1) / (gamma - 1)) * np.arctan(np.sqrt((gamma - 1) / (gamma + 1) * (M * M - 1))) \
           - np.arctan(np.sqrt(M * M - 1)) - np.arcsin(1 / M) + alpha_global - nu_b_global - phi_b_global


def calc_e_jet(Me, pe, pa, phi_e):
    """
    Calculate the remaining exit conditions and jet (boundary) conditions. The total pressure is constant throughout
    the whole flow, as it is assumed to be isentropic.

    :param Me: Exit Mach number
    :param pe: Exit static pressure [atm]
    :param pa: Atmospheric static pressure [atm]
    :param phi_e: Exit flow angle [radians]
    :return: In this order: Exit Prandtl-Meyer angle [rad], exit Mach angle [rad], jet Mach number, jet Prandtl-Meyer
    angle [rad], jet Mach angle [rad], jet flow angle [rad], total pressure of the flow [atm]
```

```python
    """
    Mjet = calc_Mjet(Me, pe, pa)
    nu_e = nu_from_M(Me)
    mu_e = mu_from_M(Me)
    nu_jet = nu_from_M(Mjet)
    mu_jet = mu_from_M(Mjet)
    phi_jet = nu_jet - nu_e + phi_e
    pt = pe * (1 + (gamma - 1) / 2 * Me * Me) ** (gamma / (gamma - 1))
    return nu_e, mu_e, Mjet, nu_jet, mu_jet, phi_jet, pt


def ccw(xa, ya, xb, yb, xc, yc):
    """
    Adapted from https://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/

    Check whether points A, B and C (in this order) are found in a 2D-plane in a counter-clockwise orientation. This is
    done by checking that the slope of line (A, C) is larger than that of line (A, B). Will be used to check if two line
    segments intersect.

    :param xa: x-position of point A
    :param ya: y-position of point A
    :param xb: x-position of point B
    :param yb: y-position of point B
    :param xc: x-position of point C
    :param yc: y-position of point C
    :return: True if the points are in a counter-clockwise orientation, False otherwise
    """
    return (yc - ya) * (xb - xa) > (yb - ya) * (xc - xa)


def check_intersect(xa, ya, xb, yb, xc, yc, xd, yd):
    """
    Adapted from https://bryceboe.com/2006/10/23/line-segment-intersection-algorithm/

    Check whether segments [A, B] and [C, D] intersect, by using the ccw() function. For this to happen, A and B need to
    be separated by [C, D], and C and D by [A, B]. For A and B to be separated by [C, D], then the point sequences
    A, C, D and B, C, D should have different orientations. Similarly, for C and D to be separated by [A, B], then the
    point sequences A, B, C and A, B, D should have different orientations.

    :param xa: x-position of point A
    :param ya: y-position of point A
    :param xb: x-position of point B
    :param yb: y-position of point B
    :param xc: x-position of point C
    :param yc: y-position of point C
    :param xd: x-position of point D
    :param yd: y-position of point D
    :return:
    """
    return ccw(xa, ya, xc, yc, xd, yd) != ccw(xb, yb, xc, yc, xd, yd) and ccw(xa, ya, xb, yb, xc, yc) != \
           ccw(xa, ya, xb, yb, xd, yd)


def get_intersect(xa, ya, xb, yb, xc, yc, xd, yd):
    """
    Return the x- and y-coordinates of the intersection point between segments [A, B] and [C, D].

    :param xa: x-position of point A
    :param ya: y-position of point A
    :param xb: x-position of point B
    :param yb: y-position of point B
    :param xc: x-position of point C
    :param yc: y-position of point C
    :param xd: x-position of point D
    :param yd: y-position of point D
    :return: The x- and y-coordinates of the intersection point, in this order.
    """
    m1 = (yb - ya) / (xb - xa)
```

```
    n1 = ya - m1 * xa
    m2 = (yd - yc) / (xd - xc)
    n2 = yc - m2 * xc

    xe = (n2 - n1) / (m1 - m2)
    ye = m1 * xe + n1

    return xe, ye


def init_exit(no_init, h):
    """
    Assign the initial condition values of the nozzle exit to the global data structure.

    :param no_init: Number of nodes created on the exit nozzle initial boundary
    :param h: The height (diameter) of the nozzle exit
    :return: Returns nothing
    """
    # Cycle through all the initial boundary nodes and assign their properties and positions
    for i in range(no_init):
        x[no_init - i - 1, i] = 0
        y[no_init - i - 1, i] = h / 2 - i * h / (no_init - 1)
        phi[no_init - i - 1, i] = 0
        M[no_init - i - 1, i] = Me
        nu[no_init - i - 1, i] = nu_e
        mu[no_init - i - 1, i] = mu_e


def assign_top_corner():
    """
    Assign the values for the first set of nodes at the intersection between the first Gamma + characteristic with the
    expansion fan of the top corner to the global data structure.

    :return: Returns nothing
    """
    # Calculate the slope angles of the last and first Gamma- characteristics of the top corner expansion fam
    beta_last = phi_jet - mu_jet
    beta_first = phi_e - mu_e
    # Create no_char (also counting the first and last) intermediate characteristics
    beta = np.linspace(beta_first, beta_last, no_char)

    # Assign needed values to nodes A and B, where A is the first node under the top corner in the initial boundary, and
    # B is the top corner
    nu_a = nu[no_init - 2, 1]
    phi_a = phi[no_init - 2, 1]
    mu_a = mu[no_init - 2, 1]
    x_a = x[no_init - 2, 1]
    y_a = y[no_init - 2, 1]
    x_b = x[no_init - 1, 0]
    y_b = y[no_init - 1, 0]

    # Cycle through all created expansion fan Gamma- characteristics
    for i, beta_i in enumerate(beta):
        # Assign jet boundary value to corner with each new characteristic, for consistency. Will be used when
        # calculating the top jet boundary condition.
        phi[no_init - 1 + i, 0] = phi_jet
        M[no_init - 1 + i, 0] = Mjet
        nu[no_init - 1 + i, 0] = nu_jet
        mu[no_init - 1 + i, 0] = mu_jet
        x[no_init - 1 + i, 0] = x[no_init - 1, 0]
        y[no_init - 1 + i, 0] = y[no_init - 1, 0]

        # Initialise global values needed for performing a root-finding method
        global beta_global, nu_a_global, phi_a_global
        # Assign values to them
        beta_global = beta_i
        nu_a_global = nu_a
        phi_a_global = phi_a

        # Calculate Mach number and other properties of C, where C is the intersection of the Gamma+ characteristic
```

```
        # from A and the current Gamma- characteristic of the expansion flow
        sol = optimize.root(beta_M_function, np.array([Me]), tol=1e-8)
        Mc = np.squeeze(sol.x)
        nu_c = nu_from_M(Mc)
        mu_c = mu_from_M(Mc)
        phi_c = beta_i + mu_c

        # Calculate position of C
        alpha_i = 0.5 * (phi_a + mu_a + phi_c + mu_c)
        x_c = (y_b - y_a + x_a * np.tan(alpha_i) - x_b * np.tan(beta_i)) / (np.tan(alpha_i) - np.tan(beta_i))
        y_c = y_a + x_c * np.tan(alpha_i) - x_a * np.tan(alpha_i)

        # Assign values of C to the global data structure matrices
        phi[no_init - 1 + i, 1] = phi_c
        M[no_init - 1 + i, 1] = Mc
        nu[no_init - 1 + i, 1] = nu_c
        mu[no_init - 1 + i, 1] = mu_c
        x[no_init - 1 + i, 1] = x_c
        y[no_init - 1 + i, 1] = y_c


def assign_bottom_corner():
    """
    Assign the values for the first set of nodes at the intersection between the first Gamma- characteristic with the
    expansion fan of the bottom corner.

    :return: Returns nothing
    """
    # Calculate the slope angles of the last and first Gamma+ characteristics of the bottom corner expansion fam
    alpha_last = - phi_jet + mu_jet
    alpha_first = - phi_e + mu_e
    # Create no_char (also counting the first and last) intermediate characteristics
    alpha = np.linspace(alpha_first, alpha_last, no_char)

    # Assign needed values to nodes A and B, where B is the first node above the bottom corner in the initial boundary,
    # and A is the bottom corner
    nu_b = nu[1, no_init - 2]
    phi_b = phi[1, no_init - 2]
    mu_b = mu[1, no_init - 2]
    x_b = x[1, no_init - 2]
    y_b = y[1, no_init - 2]
    x_a = x[0, no_init - 1]
    y_a = y[0, no_init - 1]

    # Cycle through all created expansion fan Gamma+ characteristics
    for i, alpha_i in enumerate(alpha):
        # Assign jet boundary value to corner with each new characteristic, for consistency. Will be used when
        # calculating the bottom jet boundary condition.
        phi[0, no_init - 1 + i] = - phi_jet
        M[0, no_init - 1 + i] = Mjet
        nu[0, no_init - 1 + i] = nu_jet
        mu[0, no_init - 1 + i] = mu_jet
        x[0, no_init - 1 + i] = x[0, no_init - 1]
        y[0, no_init - 1 + i] = y[0, no_init - 1]

        # Initialise global values needed for performing a root-finding method
        global alpha_global, nu_b_global, phi_b_global
        # Assign values to them
        alpha_global = alpha_i
        nu_b_global = nu_b
        phi_b_global = phi_b

        # Calculate Mach number and other properties of C, where C is the intersection of the Gamma- characteristic
        # from B and the current Gamma+ characteristic of the expansion flow
        sol = optimize.root(alpha_M_function, np.array([Me]), tol=1e-8)
        Mc = np.squeeze(sol.x)
        nu_c = nu_from_M(Mc)
        mu_c = mu_from_M(Mc)
        phi_c = alpha_i - mu_c
```

```python

        # Calculate position of C
        beta_i = 0.5 * (phi_b - mu_b + phi_c - mu_c)
        x_c = (y_b - y_a + x_a * np.tan(alpha_i) - x_b * np.tan(beta_i)) / (np.tan(alpha_i) - np.tan(beta_i))
        y_c = y_a + x_c * np.tan(alpha_i) - x_a * np.tan(alpha_i)

        # Assign values of C to the global data structure matrices
        phi[1, no_init - 1 + i] = phi_c
        M[1, no_init - 1 + i] = Mc
        nu[1, no_init - 1 + i] = nu_c
        mu[1, no_init - 1 + i] = mu_c
        x[1, no_init - 1 + i] = x_c
        y[1, no_init - 1 + i] = y_c


def prop_top_BC(ia, ja, ib, jb):
    """
    Propagate to (calculate) the next node D in the top jet boundary from the nodes A (index [ia, ja]) and
    B (index [ib, jb]). Point B is the previous node in the top jet boundary, and A is the node which is on the same
    Gamma+ characteristic as node D, and on the same Gamma- characteristic as node B. Practically, a new Gamma-
    characteristic will be created with the addition of point D. After calculation, assign values to the global
    data structure.

    :param ia: Gamma- coordinate of node A
    :param ja: Gamma+ coordinate of node A
    :param ib: Gamma- coordinate of node B
    :param jb: Gamma+ coordinate of node B
    :return: Returns nothing
    """
    # At the jet boundary, the mach number is constant == Mjet. Assign these values to node D
    Md = Mjet
    nu_d = nu_jet
    mu_d = mu_jet

    # Assign needed values for calculation to nodes A and B
    nu_a = nu[ia, ja]
    phi_a = phi[ia, ja]
    mu_a = mu[ia, ja]
    x_a = x[ia, ja]
    y_a = y[ia, ja]
    phi_b = phi[ib, jb]
    x_b = x[ib, jb]
    y_b = y[ib, jb]

    # Calculate values of node D
    phi_d = nu_d - nu_a + phi_a
    alpha = 0.5 * (phi_a + mu_a + phi_d + mu_d)
    beta = 0.5 * (phi_b + phi_d)
    x_d = (y_b - y_a + x_a * np.tan(alpha) - x_b * np.tan(beta)) / (np.tan(alpha) - np.tan(beta))
    y_d = y_a + x_d * np.tan(alpha) - x_a * np.tan(alpha)

    # Add values of node D to the global data structure
    phi[ia + 1, ja] = phi_d
    M[ia + 1, ja] = Md
    nu[ia + 1, ja] = nu_d
    mu[ia + 1, ja] = mu_d
    x[ia + 1, ja] = x_d
    y[ia + 1, ja] = y_d


def prop_bottom_BC(ia, ja, ib, jb):
    """
    Propagate to (calculate) the next node D in the bottom jet boundary from the nodes A (index [ia, ja]) and
    B (index [ib, jb]). Point A is the previous node in the top jet boundary, and B is the node which is on the same
    Gamma- characteristic as node D, and on the same Gamma+ characteristic as node B. Practically, a new Gamma+
```

```
    characteristic will be created with the addition of point D. After calculation, assign values to the global
    data structure.

    :param ia: Gamma- coordinate of node A
    :param ja: Gamma+ coordinate of node A
    :param ib: Gamma- coordinate of node B
    :param jb: Gamma+ coordinate of node B
    :return: Returns nothing
    """
    # At the jet boundary, the mach number is constant == Mjet. Assign these values to node D
    Md = Mjet
    nu_d = nu_jet
    mu_d = mu_jet

    # Assign needed values for calculation to nodes A and B
    nu_b = nu[ib, jb]
    phi_b = phi[ib, jb]
    mu_b = mu[ib, jb]
    x_b = x[ib, jb]
    y_b = y[ib, jb]
    phi_a = phi[ia, ja]
    x_a = x[ia, ja]
    y_a = y[ia, ja]

    # Calculate values of node D
    phi_d = nu_b + phi_b - nu_d
    alpha = 0.5 * (phi_a + phi_d)
    beta = 0.5 * (phi_b - mu_b + phi_d - mu_d)
    x_d = (y_b - y_a + x_a * np.tan(alpha) - x_b * np.tan(beta)) / (np.tan(alpha) - np.tan(beta))
    y_d = y_a + x_d * np.tan(alpha) - x_a * np.tan(alpha)

    # Add values of node D to the global data structure
    phi[ib, jb + 1] = phi_d
    M[ib, jb + 1] = Md
    nu[ib, jb + 1] = nu_d
    mu[ib, jb + 1] = mu_d
    x[ib, jb + 1] = x_d
    y[ib, jb + 1] = y_d


def prop_normal(ia, ja, ib, jb):
    """
    Propagate to (calculate) the next node P using the classic method of characteristics, from the nodes
    A (index [ia, ja]) and B (index [ib, jb]). Node A is the node before C that is on the same Gamma+ characteristic,
    and node B is the node before C that is on the same Gamma- characteristic.

    :param ia: Gamma- coordinate of node A
    :param ja: Gamma+ coordinate of node A
    :param ib: Gamma- coordinate of node B
    :param jb: Gamma+ coordinate of node B
    :return: Returns nothing
    """
    # Assign needed values for calculation to nodes A and B
    nu_a = nu[ia, ja]
    mu_a = mu[ia, ja]
    phi_a = phi[ia, ja]
    x_a = x[ia, ja]
    y_a = y[ia, ja]
    nu_b = nu[ib, jb]
    mu_b = mu[ib, jb]
    phi_b = phi[ib, jb]
    x_b = x[ib, jb]
    y_b = y[ib, jb]

    # Calculate values of node P
    nu_p = 0.5 * (nu_b + nu_a) + 0.5 * (phi_b - phi_a)
    phi_p = 0.5 * (phi_b + phi_a) + 0.5 * (nu_b - nu_a)
    Mp = M_from_nu(nu_p)
    mu_p = mu_from_M(Mp)

```

```
    alpha = 0.5 * (phi_a + mu_a + phi_p + mu_p)
    beta = 0.5 * (phi_b - mu_b + phi_p - mu_p)
    x_p = (y_b - y_a + x_a * np.tan(alpha) - x_b * np.tan(beta)) / (np.tan(alpha) - np.tan(beta))
    y_p = y_a + x_p * np.tan(alpha) - x_a * np.tan(alpha)

    # Add values of node D to the global data structure
    phi[ib, ja] = phi_p
    M[ib, ja] = Mp
    nu[ib, ja] = nu_p
    mu[ib, ja] = mu_p
    x[ib, ja] = x_p
    y[ib, ja] = y_p


def propagation(steps):
    """
    Function that propagates (calculates) the next nodes' values and positions, using the initial nozzle exit points,
    and the ones created for the expansion flows. It also checks for shock formation, stops the propagation if one is
    found, after that step is finished (to check for symmetric shocks), and return their coordinates. Furthermore, it
    calculates the values and positions along the path of predefined streamline.

    :param steps: Maximum number of steps performed in the propagation
    :return: Returns nothing
    """
    # Initialise the last index of the streamline data structure arrays (i_s), the last index of the shock data
    # structure arrays (index_shock), and the step at which the shock forms (k_shock)
    global i_s, index_shock
    k_shock = np.nan

    # Cycle through all the steps. A step means calculating the values and positions of nodes for indices that respect
    # the relation i + j = no_init - 1 + k (a diagonal in the data structure matrices)
    for k in range(steps + 1):
        # Cycle through all the possible node indices for a given step
        for j in range(no_init + k):
            i = no_init - 1 + k - j

            # Propagate values according to the m.o.c.
            if not np.isnan(M[i, j]):
                # Point already exists, passing...
                pass
            elif (not np.isnan(M[i - 1, j])) and (not np.isnan(M[i, j - 1])):
                # Two characteristics exist for this point's intersection, doing normal propagation
                prop_normal(i - 1, j, i, j - 1)
            elif (not np.isnan(M[i - 1, j])) and (not np.isnan(M[i - 1, j - 1])):
                # Point qualifies for top jet BC
                prop_top_BC(i - 1, j, i - 1, j - 1)
            elif (not np.isnan(M[i, j - 1])) and (not np.isnan(M[i - 1, j - 1])):
                # Point qualifies for bottom jet BC
                prop_bottom_BC(i - 1, j - 1, i, j - 1)

            # Check if shock develops by intersecting neighbouring characteristics
            if check_intersect(x[i, j], y[i, j], x[i - 1, j], y[i - 1, j], x[i, j - 1], y[i, j - 1], x[i - 1, j - 1],
                               y[i - 1, j - 1]):
                # Store the shock locations
                index_shock += 1
                x_shock[index_shock] = get_intersect(x[i, j], y[i, j], x[i - 1, j], y[i - 1, j], x[i, j - 1],
                                                     y[i, j - 1], x[i - 1, j - 1], y[i - 1, j - 1])[0]
                y_shock[index_shock] = get_intersect(x[i, j], y[i, j], x[i - 1, j], y[i - 1, j], x[i, j - 1],
                                                     y[i, j - 1], x[i - 1, j - 1], y[i - 1, j - 1])[1]
                k_shock = k
            elif check_intersect(x[i, j], y[i, j], x[i, j - 1], y[i, j - 1], x[i - 1, j], y[i - 1, j],
                                 x[i - 1, j - 1], y[i - 1, j - 1]):
```

```
                # Store the shock locations
                index_shock += 1
                x_shock[index_shock] = get_intersect(x[i, j], y[i, j], x[i, j - 1], y[i, j - 1], x[i - 1, j],
                                                     y[i - 1, j], x[i - 1, j - 1], y[i - 1, j - 1])[0]
                y_shock[index_shock] = get_intersect(x[i, j], y[i, j], x[i, j - 1], y[i, j - 1], x[i - 1, j],
                                                     y[i - 1, j], x[i - 1, j - 1], y[i - 1, j - 1])[1]
                k_shock = k
            # If a shock has been found and if the current step number is greater than that of the shock, it exits the
            # loop
            if not np.isnan(k_shock) and k > k_shock:
                break

            # Propagate streamline if an intersection is found with either of the immediate Gamma+ or Gamma- segment
            # before the current node
            if not np.isnan(M[i, j]) and not np.isnan(M[i, j - 1]):
                # Check for intersection with Gamma- segment
                # Create virtual possible segment of the streamline propagation, with x-length 10 * h
                x_s1 = xs[i_s]
                y_s1 = ys[i_s]
                phi_s = phis[i_s]
                x_s2 = x_s1 + 10 * h
                y_s2 = y_s1 + 10 * h * np.tan(phi_s)

                # Check actual intersection
                if check_intersect(x_s1, y_s1, x_s2, y_s2, x[i, j - 1], y[i, j - 1], x[i, j], y[i, j]):
                    xe, ye = get_intersect(x_s1, y_s1, x_s2, y_s2, x[i, j - 1], y[i, j - 1], x[i, j], y[i, j])

                    # Move to the next index for the streamline arrays, and store the values in the data structure
                    i_s += 1
                    xs[i_s] = xe
                    ys[i_s] = ye
                    Ms[i_s] = M[i, j - 1]
                    phis[i_s] = phi[i, j - 1]
                    nus[i_s] = nu[i, j - 1]
                    mus[i_s] = mu[i, j - 1]

            elif not np.isnan(M[i, j]) and not np.isnan(M[i - 1, j]):
                # Check for intersection with Gamma+ segment
                # Create virtual possible segment of the streamline propagation, with x-length 10 * h
                x_s1 = xs[i_s]
                y_s1 = ys[i_s]
                phi_s = phis[i_s]
                x_s2 = x_s1 + 10 * h
                y_s2 = y_s1 + 10 * h * np.tan(phi_s)

                # Check actual intersection
                if check_intersect(x_s1, y_s1, x_s2, y_s2, x[i - 1, j], y[i - 1, j], x[i, j], y[i, j]):
                    xe, ye = get_intersect(x_s1, y_s1, x_s2, y_s2, x[i - 1, j], y[i - 1, j], x[i, j], y[i, j])

                    # Move to the next index for the streamline arrays, and store the values in the data structure
                    i_s += 1
                    xs[i_s] = xe
                    ys[i_s] = ye
                    Ms[i_s] = M[i - 1, j]
                    phis[i_s] = phi[i - 1, j]
                    nus[i_s] = nu[i - 1, j]
                    mus[i_s] = mu[i - 1, j]


# ----------------------------------------------------------
# Plotting functions
```

```python
# ----------------------------------------------------------

def plot_all_M(option):
    """
    Plot the Mach number field, either node-based (only at the calculated location), or by (smooth) interpolation over
    the whole domain.

    :param option: Option variable. 0 to plot node-based Mach number field, 1 to plot (smooth) the interpolation
    :return: Returns nothing
    """
    # Remove np.nan values from data structure
    xg = x[np.logical_not(np.isnan(x))]
    yg = y[np.logical_not(np.isnan(y))]
    Mg = M[np.logical_not(np.isnan(M))]

    # Plot the corresponding field according to option
    if option == 0:
        p1 = ax1.scatter(xg, yg, s=20, c=Mg, cmap="cool")
    elif option == 1:
        p1 = ax1.tricontourf(xg, yg, Mg, levels=50, cmap="cool")
    else:
        raise Exception("This value for the plotting option is not supported! Choose option equal to 0 or 1")
    fig1.colorbar(p1, label="Mach number")


def plot_char():
    """
    Plot the characteristic lines originating from the expansion fans.

    :return: Returns nothing
    """
    for k in range(no_char):
        # Plot the direct Gamma- characteristics arising from the top corner
        xt = x[no_init - 1 + k][np.logical_not(np.isnan(x[no_init - 1 + k]))]
        yt = y[no_init - 1 + k][np.logical_not(np.isnan(y[no_init - 1 + k]))]
        ax1.plot(xt, yt, color="black")

        # Plot the direct Gamma+ characteristics arising from the bottom corner
        xb = x[:, 2 * no_init - 1 + k + no_char - 2][np.logical_not(np.isnan(x[:, 2 * no_init - 1 + k + no_char - 2]))]
        yb = y[:, 2 * no_init - 1 + k + no_char - 2][np.logical_not(np.isnan(y[:, 2 * no_init - 1 + k + no_char - 2]))]
        ax1.plot(xb, yb, color="black")

        # Plot the indirect Gamma+ characteristics arising from the reflections of the Gamma- from the top corner
        xb = x[:, no_init - 1 + k][np.logical_not(np.isnan(x[:, no_init - 1 + k]))]
        yb = y[:, no_init - 1 + k][np.logical_not(np.isnan(y[:, no_init - 1 + k]))]
        ax1.plot(xb, yb, color="black")

        # Plot the indirect Gamma- characteristics arising from the reflections of the Gamma+ from the bottom corner
        xb = x[2 * no_init - 1 + k + no_char - 2][np.logical_not(np.isnan(x[:, 2 * no_init - 1 + k + no_char - 2]))]
        yb = y[2 * no_init - 1 + k + no_char - 2][np.logical_not(np.isnan(y[:, 2 * no_init - 1 + k + no_char - 2]))]
        ax1.plot(xb, yb, color="black")


def plot_jet_BC():
    """
    Plot the jet boundaries that encompass the flow with an orange line.

    :return: Returns nothing
    """
    # Initialise data arrays for the top and bottom jet boundaries
    xt = np.array([])
    yt = np.array([])
    xb = np.array([])
    yb = np.array([])

```

```
    # Loop over the coordinates of the nodes that are on the top jet boundary and add their
    position to xt and yt
    i = 0
    while not np.isnan(x[no_init + no_char - 2 + i, i]):
        xt = np.append(xt, x[no_init + no_char - 2 + i, i])
        yt = np.append(yt, y[no_init + no_char - 2 + i, i])
        i += 1

    # Loop over the coordinates of the nodes that are on the bottom jet boundary and add
    their position to xb and yb
    i = 0
    while not np.isnan(x[i, no_init + no_char - 2 + i]):
        xb = np.append(xb, x[i, no_init + no_char - 2 + i])
        yb = np.append(yb, y[i, no_init + no_char - 2 + i])
        i += 1

    # Plot the jet boundaries
    ax1.plot(xt, yt, color="orange")
    ax1.plot(xb, yb, color="orange")


def plot_shock():
    """
    Plot the locations of the shocks with red x crosses.

    :return: Returns nothing
    """
    # Remove np.nan values from data structure
    xg = x_shock[np.logical_not(np.isnan(x_shock))]
    yg = y_shock[np.logical_not(np.isnan(y_shock))]

    # Plot the shock formation locations
    ax1.scatter(xg, yg, marker="X", color="red", s=100, zorder=500)


def plot_streamline():
    """
    Plot the path of the streamline for which the initial conditions are specified globally.
     Also plot the (static)
    pressure versus x-coordinate graph along the streamline.

    :return: Returns nothing
    """
    # Remove np.nan values from data structure
    xg = xs[np.logical_not(np.isnan(xs))]
    yg = ys[np.logical_not(np.isnan(ys))]
    Mg = Ms[np.logical_not(np.isnan(Ms))]

    # Calculate static pressure along streamline
    pg = pt / ((1 + (gamma - 1) / 2 * Mg * Mg) ** (gamma / (gamma - 1)))

    # Plot the streamline path on the main figure
    ax1.plot(xg, yg, color="gold")

    # Plot the pressure vs x  graph of the streamline
    fig2, ax2 = plt.subplots()
    ax2.plot(xg, pg, color="black", label=rf"$y_0$ = {ys[0]}")
    ax2.set_xlabel("Horizontal position x")
    ax2.set_ylabel("Static pressure p [atm]")


def plot_aux():
    """
    Plot auxiliary items, such as symmetry line and nozzle exit walls.

    :return: Returns nothing
    """
    # Remove np.nan values from data structure
    xg = x[np.logical_not(np.isnan(x))]

    # Plot symmetry line
    ax1.hlines(0, -0.5, xg[-1] + 0.5, linestyles="dashdot", color="black")

    # Plot the exit nozzle walls
    ax1.hlines(h / 2, -0.4, 0, color="black")
```

```
    ax1.hlines(-h / 2, -0.4, 0, color="black")
    ax1.vlines(0, h / 2, h / 2 * 1.3, color="black")
    ax1.vlines(0, -h / 2, -h / 2 * 1.3, color="black")

    xsh = np.linspace(-0.4, 0, 1001)
    y1 = h / 2 * 1.3 * np.ones(xsh.shape[0])
    y2 = h / 2 * np.ones(xsh.shape[0])
    y3 = - h / 2 * np.ones(xsh.shape[0])
    y4 = - h / 2 * 1.3 * np.ones(xsh.shape[0])
    ax1.fill_between(xsh, y1, y2, color="darkgrey")
    ax1.fill_between(xsh, y3, y4, color="darkgrey")

    fig1.tight_layout()


# ----------------------------------------------------------
# Global program
# ----------------------------------------------------------


# Values needed for root-finding. Do not work with them outside the root-finding functions!
nu_global = np.nan
beta_global = np.nan
nu_a_global = np.nan
phi_a_global = np.nan
alpha_global = np.nan
nu_b_global = np.nan
phi_b_global = np.nan

# Initial conditions & initialisations
Me = 2.0  # Mach number at the nozzle exit
pa = 1  # Static pressure of the ambient atmosphere [atm]
pe = 2 * pa  # Static pressure at the nozzle exit
phi_e = 0.0  # Flow angle at the nozzle exit [rad]
gamma = 1.4  # Specific heat ratio of air
h = 1.0  # Height (diameter) of the nozzle exit
no_init = 31  # Number of nodes to be created at the exit of the nozzle
no_char = 31  # Number of characteristics to be created in the expansion fans at the top and
     bottom corners
dim = 3010  # Dimension of (each axis of) each data structure matrix. Needs to be bigger
    than (no_init + no_steps + 1)!!
no_steps = 2500  # Maximum number of propagation steps
option = 1  # Variable for choosing between plotting node-based values of the Mach number
    field, or a continuous field
# calculated by interpolation. 0 for node-based, 1 for interpolation

# Initialisation of global data structure matrices
x = np.empty((dim, dim))
x[:] = np.nan
y = np.empty((dim, dim))
y[:] = np.nan
phi = np.empty((dim, dim))
phi[:] = np.nan
M = np.empty((dim, dim))
M[:] = np.nan
nu = np.empty((dim, dim))
nu[:] = np.nan
mu = np.empty((dim, dim))
mu[:] = np.nan

# Initialisation of possible shock data structure array
x_shock = np.empty((2 * dim + 10))
x_shock[:] = np.nan
y_shock = np.empty((2 * dim + 10))
y_shock[:] = np.nan
index_shock = -1

# Initialisation of the streamline data structure array
xs = np.empty((2 * dim + 10))
xs[:] = np.nan
ys = np.empty((2 * dim + 10))
ys[:] = np.nan
phis = np.empty((2 * dim + 10))
phis[:] = np.nan
Ms = np.empty((2 * dim + 10))
```

```python
Ms[:] = np.nan
nus = np.empty((2 * dim + 10))
nus[:] = np.nan
mus = np.empty((2 * dim + 10))
mus[:] = np.nan

# Calculation of global nozzle exit and jet boundary values needed for calculations
nu_e, mu_e, Mjet, nu_jet, mu_jet, phi_jet, pt = calc_e_jet(Me, pe, pa, phi_e)

# Values for streamline initial conditions
xs[0] = 0.0  # x-position
ys[0] = h / 4  # y-position
phis[0] = phi_e  # Flow ange [rad]
Ms[0] = Me  # Mach number
nus[0] = nu_e  # Prandtl-Meyer angle [rad]
mus[0] = mu_e  # Mach angle [rad]
i_s = 0  # Index of last entry in streamline arrays. Do not change!

# Initialise the nozzle exit boundary, the expansion fans, and perform the global propagation
init_exit(no_init, h)
assign_top_corner()
assign_bottom_corner()
propagation(no_steps)

# Plot Mach number distribution, characteristics, jet boundaries, chosen streamline and shock formation location
fig1, ax1 = plt.subplots()
plot_all_M(option)
plot_char()
plot_jet_BC()
plot_streamline()
if not np.isnan(x_shock[0]):
    plot_shock()
plot_aux()
ax1.axis("equal")
ax1.set_xlabel("x coordinate")
ax1.set_ylabel("y coordinate")
plt.show()
```