Faculty of Aerospace Engineering

AE4140 - Gas Dynamics

Task 2

Senne Hemelaar, 4573404

October 29, 2021

1 Introduction

In this report, the approach and results from AE4140 Gas Dynamics, task 2, will be presented. The written program is coded in Matlab. Task 2 is aimed at creating a program that can solve certain properties of a jet expansion (figure 1) using the 2-dimensional method of characteristics, where the number of characteristics (N) is a user input. This report consists of a section where the approach of solving the problem will be explained and afterwards, the results are shown and discussed. The entire code (including functions) for this assignment can be found in the appendix.

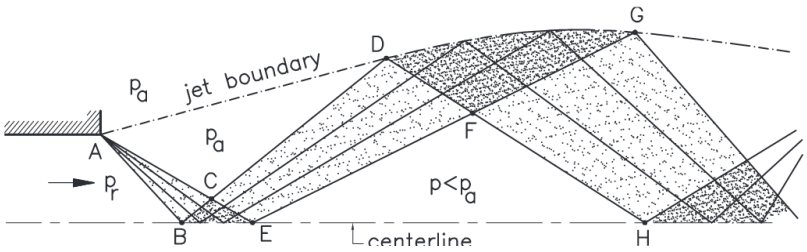


Figure 1: Problem Description Visualization

2 Approach

The taken approach to model each part of the assignment will be explained in this section step-by-step. The entire program can be found in appendix A. All the functions that are used can be found in appendix B

2.1 Discretization of the Expansion Fan

First, the expansion fan should be discretized. This is done by linearly interpolating between the initial deflection w.r.t. the horizontal (α_i) and the deflection w.r.t the horizontal after the expansion fan (α_B). The conditions for the end of the expansion fan (subscript B) are found using a Γ^+-charcateristic. The mach angle μ in computed using the function `Mu_From_Mach`. And the flow deflection at B is found from `Forward_Phi_Gamma_Plus`. From there, α_B can be computed easily using equality (1).

$$\alpha = \varphi - \mu \tag{1}$$

The code can be found in appendix A under the header "preliminary calculations". Figure 2 shows an example of the initial expansion fan for $N = 10$ characteristics.

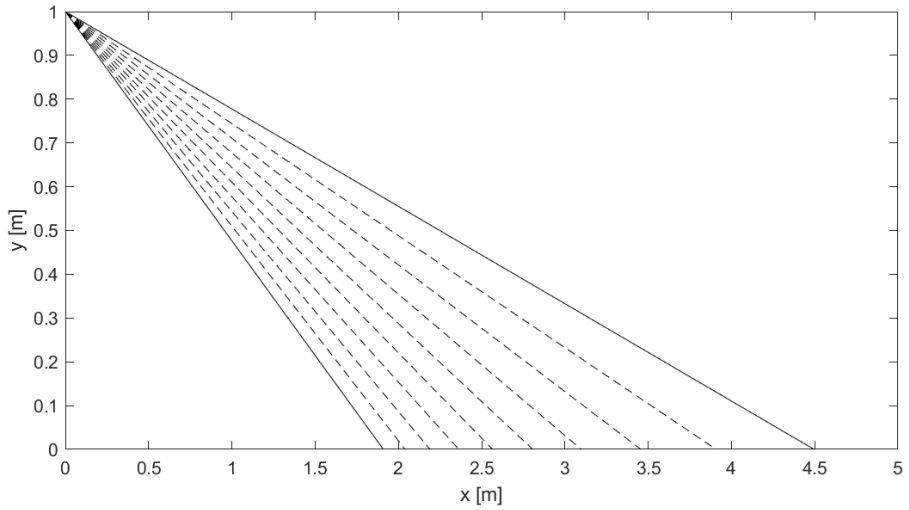


Figure 2: Expansion Fan Discretization, $N = 10$

2.2 Finding Variables for All Regions

Next, the code computes all the relevant variables in all regions of the underexpanded jet area and saves them in a struct. These variables are the following:

- Mach number M
- Prandtl-Meyer angle ν
- Flow deflection φ
- Deflection w.r.t. horizontal α
- Mach angle μ

In the code all regions are evaluated separately. The variables M, φ and ν in the expansion wave (simple wave region) are obtained using a bisection method. The function `Bisection_Method` can be found in appendix B.2. Now that the Prandtl-Meyer angle ν and flow deflection φ is known for all characteristics of the first simple wave, the method of characteristics can be applied. For the complex regions two for loops are being executed, one over the incoming characteristics and one over the reflected characteristics. The code determines the Prandtl-Meyer angle and flow deflection first. Then, from these values a bisection method is used again to determine the mach angle and mach number itself. The process can be found in the main script (appendix A).

2.3 Finding the Intersections

We have obtained all relevant variables for an input number of characteristics. However, what we haven't found yet is how the characteristics move through space. Equations (2) describe the slope of the characteristic. Again for loops over the incoming and reflected characteristics are taken to determine the intersection points of the characteristics and they are saved in a struct. Off course, boundary conditions are required to solve the problem. The first boundary condition is applied to the center line, where the flow deflection $\varphi = 0°$ For region DFG, the only noticeable difference is that φ varies and ν stays constant.

$$\begin{aligned} \Gamma^- &: \frac{\mathrm{d}y}{\mathrm{d}x} = \tan(\varphi - \mu) \\ \Gamma^+ &: \frac{\mathrm{d}y}{\mathrm{d}x} = \tan(\varphi + \mu) \end{aligned} \tag{2}$$

2.4 Creating Heat Map

The mach number is known for all regions as well as the intersection points. The function `Create_Heat_Map_Geometry` (appendix B.4) creates the patches to plot the mach number in the x,y field.

2.5 Streamlines

Personally, I found this the most challenging part. The first approach I took was to start with a straight line and find intersections using the matlab function `polyxpolyy`. If statements were used to find the intersection with the lowest x coordinate and then a linear function was applied to the initial straight streamline. However, this turned out to be a very time consuming method, while computing all the intersections with `polyxpolyy` took a lot of computing time. Alternatively a while loop was used to compute the steamline and the crossings were determined using a simpler method. The crossing index could be found by this simple line of code:

```
crossing_index = find(diff(sign(streamline(prev_idx:end) - gammas(i,prev_idx:end))));
```

The streamline function can be found in appendix B.5.

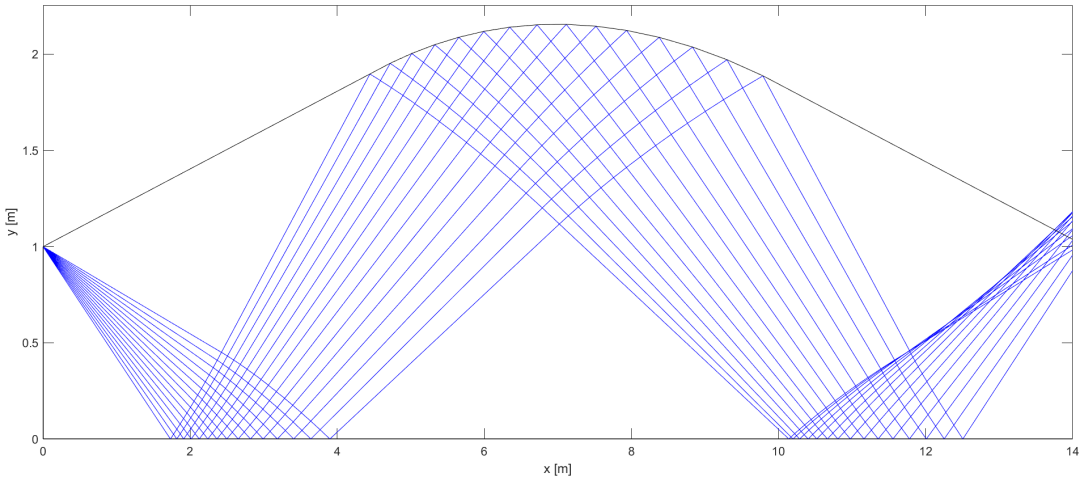
3 Results & Discussion

In this section all the obtained results from the script are shown. Additionally, comments and discussion points will be addressed.

3.1 Characteristics Pattern

Figure 3 shows the characteristic pattern for $N = 15$ and $N = 25$ characteristics. What can be observed is that the maximum height of the jet boundary decreases with an increasing number of characteristics. A higher number of characteristics also yields a higher deflection of the flow leaving region DFG, making the shock formation after region HIJ happen fasten.

(a) $N = 15$



(b) $N = 25$

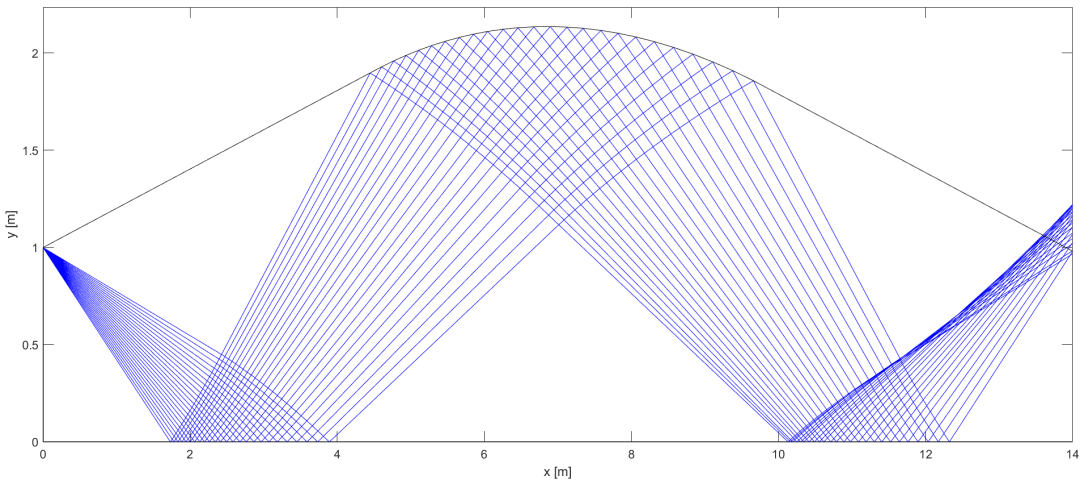
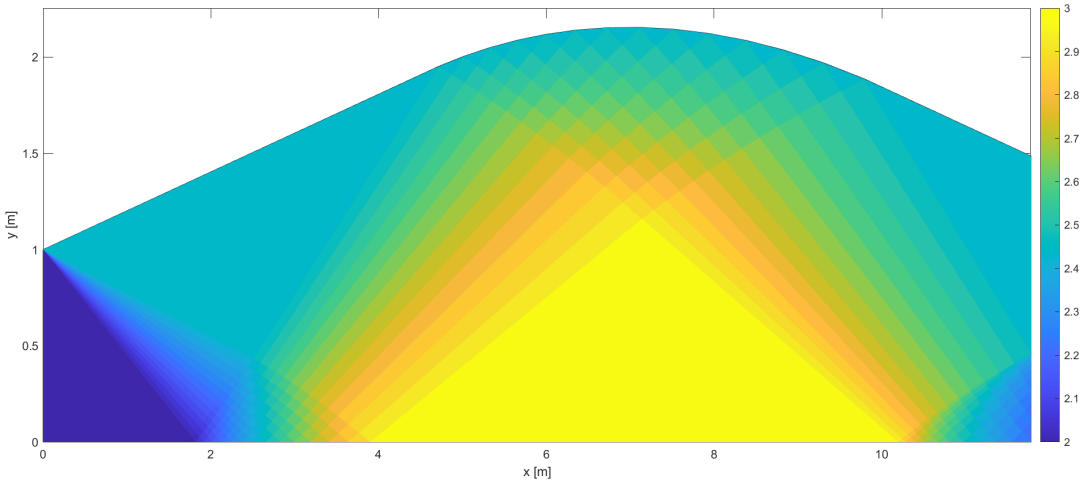


Figure 3: Characteristic Pattern

3.2 Mach Distribution

Figure 4 shows the mach distribution of the flow for $N = 15$ and $N = 25$ characteristics. the Mach number increases gradually over the expansion fan and has a constant value over the jet boundary. Due to even lower pressure (figure 6) in the core of the jet area, the mach number is the highest.

(a) $N = 15$



(b) $N = 25$

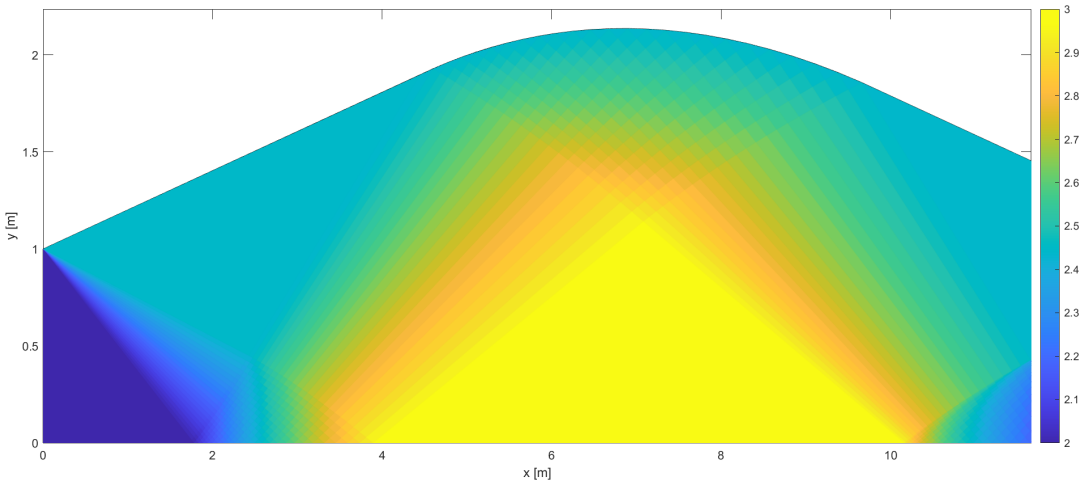


Figure 4: Mach Distribution

3.3 Streamlines and Pressure

Two streamlines are plotted in figure 5. What can be noticed is that the upper streamline straggling pattern when it goes though the complex region DFG. This is due to the approach that was taken to plot the streamlines. This effect logically decreases for a higher amount of characteristics. What also might be a better approach to increase the smoothness, is to interpolate the flow deflection φ over the entire domain and average. Then use that data set to construct the streamlines.

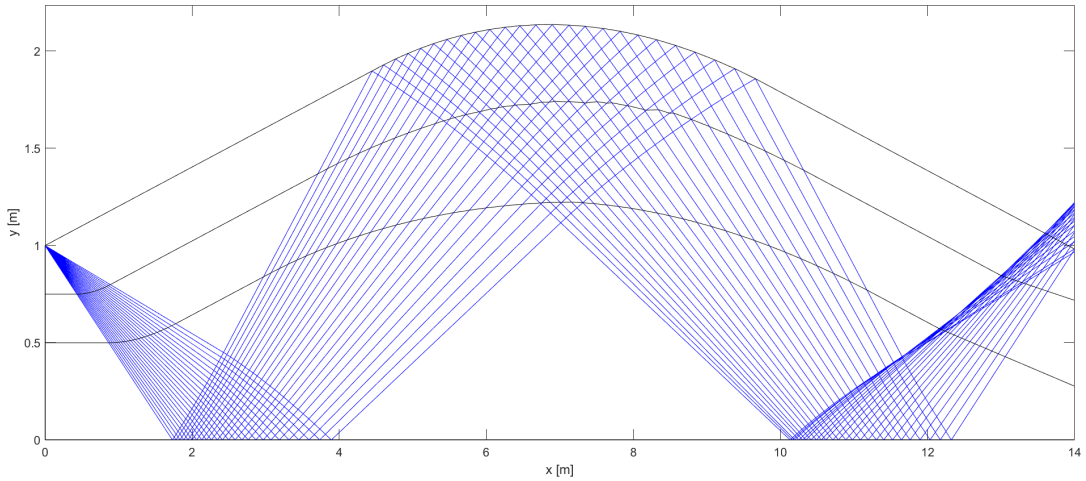


Figure 5: Streamlines starting at $H/2$ and $3H/4$, $N = 25$

Figure 6 shows the pressure along the streamlines from figure 5. It can be observed that the pressure drops rapidly over the expansion waves and it stays constant (equal to the ambient pressure) until the reflected wave and complex wave regions where it decreases and than increases again toward the ambient pressure. The total pressure P_{tot} is taken as the normal value $P_{tot} = 1$. The plot shows inconsistencies when it reaches the complex region DFG. I wasn't able to identify what caused this exactly. But my most logical theory would be that something goes wrong with using the right indices for obtaining the mach number (from which the pressure is calculated).

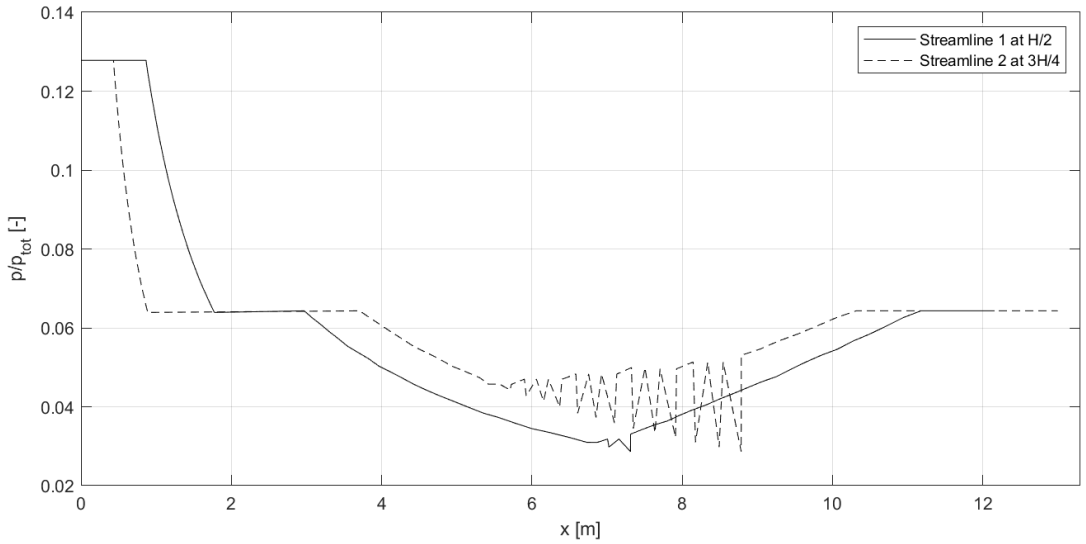
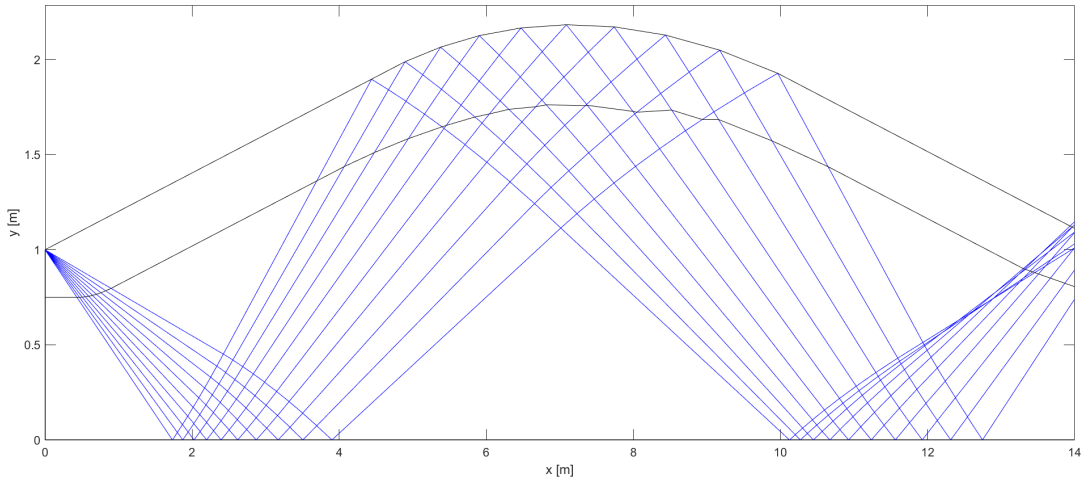


Figure 6: Pressure Along Streamlines

3.4 Accuracy

A higher number of characteristics definitely influences the results. As mentioned before in section 3.1 the jet boundary shape changes as well as the location of the first shock formation. Using the current method to determine the streamline, the accuracy is a lot higher for a higher amount on characteristics. This can be observed in figure 7. In terms of the mach distribution, increasing the number of characteristics mostly increases the resolution of the image (aside from the geometry effects mentioned before).

(a) $N = 10$



(b) $N = 30$

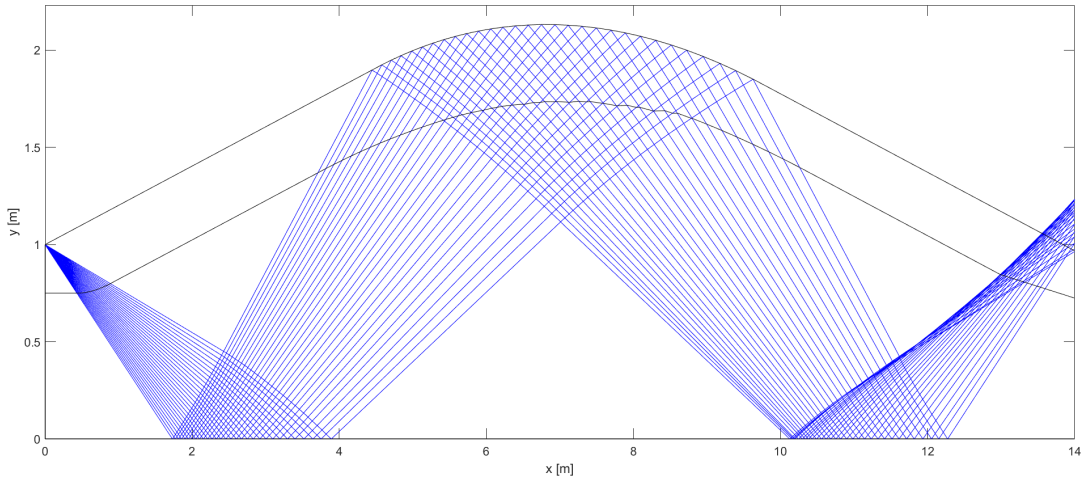


Figure 7: Streamline for Varying Number of Characteristics

4 Shock Location in Terms of Initial Mach

The location of the shock formation for varying initial mach number is also investigated. Additional code is written to find the first intersection happening between two characteristics. An example of such a shock location is shown in figure 8.

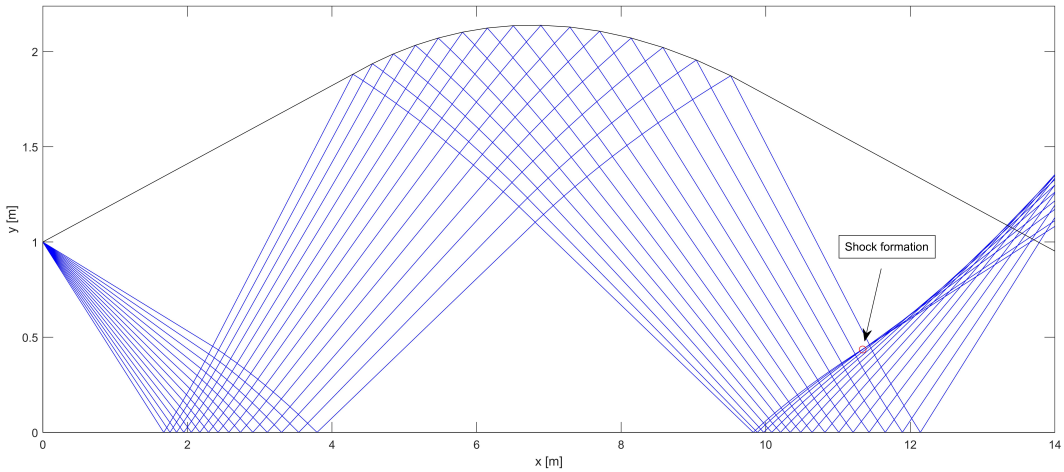


Figure 8: Shock Location

The x location of the first shock formation is plotted against initial mach number in figure 9. It can be clearly observed that the initial mach number has a linear relation with the location when shock formation starts to occur.

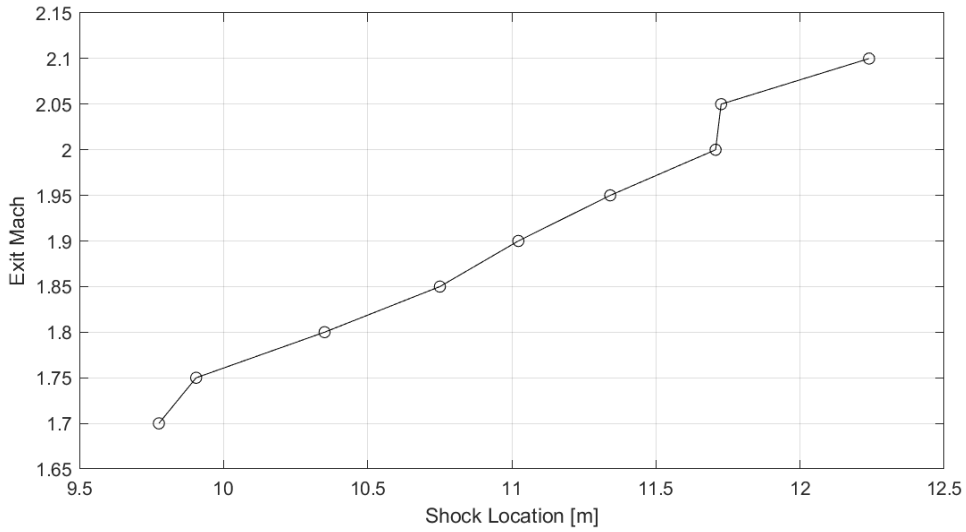


Figure 9: First shock location in terms of exit Mach

A Main Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                          Gas Dynamics: Task 2                              %%%
%%%                         Main processing script                             %%%
%%%                         Author: Senne Hemelaar                             %%%
%%%                              October 2021                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; close all;

%%%                               CONSTANTS                                    %%%
%%%============================================================================%%%
gamma = 7/5;   % Must also change in function 'Prandtl_Meyer_From_Mach.m'

%%%                             INITIAL VALUES                                 %%%
%%%============================================================================%%%
mach_initial = 2;
phi_initial = 0;
pressure_ambient = 1225;
pressure_initial = 2*pressure_ambient;
pressure_total = 1;

%%%                           VARIABLE PARAMETERS                              %%%
%%%============================================================================%%%
N = 10;
h = 1;
x_range = 14;
x_steps = 10000;
x_mesh  = linspace(0, x_range, x_steps);
error_tolerance = 0.00001;

%%%                         PRILIMINARY CALCULATIONS                           %%%
%%%============================================================================%%%

mach_B        = sqrt(2 / (gamma-1) * (2^((gamma - 1) / gamma) *...
                (1 + (gamma-1) / 2 * mach_initial^2) -1));
phi_B         = Forward_Phi_Gamma_Plus(Prandtl_Meyer_From_Mach(mach_initial),...
                Prandtl_Meyer_From_Mach(mach_B), phi_initial);
alpha_initial = Alpha(Mu_From_Mach(mach_initial),phi_initial);
alpha_B       = Alpha(Mu_From_Mach(mach_B),phi_B);
alpha_list    = linspace(alpha_initial,alpha_B,N);
alpha_steps   = linspace(1,length(alpha_list),length(alpha_list));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%         Setting up zero arrays and matrices for all regions            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Wave.ABC_mach  = zeros(N,1);
Wave.ABC_nu    = zeros(N,1);
Wave.ABC_phi   = zeros(N,1);

Wave.BCE_mach         = zeros(N);
Wave.BCE_nu           = zeros(N);
Wave.BCE_phi          = zeros(N);
Wave.BCE_alpha        = zeros(N);
Wave.BCE_mu           = zeros(N);
intersections.BCE_x   = zeros(N);
intersections.BCE_y   = zeros(N);
```

```
Wave.DFG_mach         = zeros(N);
Wave.DFG_nu           = zeros(N);
Wave.DFG_phi          = zeros(N);
Wave.DFG_alpha        = zeros(N);
Wave.DFG_mu           = zeros(N);
intersections.DFG_x   = zeros(N);
intersections.DFG_y   = zeros(N);

Wave.HIJ_mach         = zeros(N);
Wave.HIJ_nu           = zeros(N);
Wave.HIJ_phi          = zeros(N);
Wave.HIJ_alpha        = zeros(N);
Wave.HIJ_mu           = zeros(N);
intersections.HIJ_x   = zeros(N);
intersections.HIJ_y   = zeros(N);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%            Finding relevant variables* for all regions                 %%%
%%%                   *: Mach, nu, phi, alpha, mu                          %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Wave.ABC_mach(1)  = mach_initial;
Wave.ABC_nu(1)    = Prandtl_Meyer_From_Mach(mach_initial);
Wave.ABC_phi(1)   = phi_initial;

%%%                          REGION ABC, BCE                               %%%
%%%=========================================================================%%%
for j = 2:N
    [Wave.ABC_mach(j), Wave.ABC_phi(j), Wave.ABC_nu(j)] =...
    Bisection_Method(...
    Wave.ABC_mach(j-1),...
    mach_B,...
    error_tolerance,...
    alpha_list(j),...
    Wave.ABC_phi(j-1),...
    Prandtl_Meyer_From_Mach(Wave.ABC_mach(j-1)));
    Wave.BCE_nu(1,1) = Wave.ABC_nu(1) + Wave.ABC_phi(1);
    for i = 1:j
        if i == j
            Wave.BCE_nu(i,j) = Wave.ABC_nu(j) + Wave.ABC_phi(j);
        elseif i == 1
            Wave.BCE_nu(i,j) =  0.5*(Wave.BCE_nu(i,j-1) + Wave.ABC_nu(j))+...
                                0.5*(Wave.ABC_phi(j) - Wave.BCE_phi(i,j-1));
            Wave.BCE_phi(i,j) = 0.5*(Wave.BCE_phi(i,j-1) + Wave.ABC_phi(j))+...
                                0.5*(Wave.ABC_nu(j) - Wave.BCE_nu(i,j-1));
        else
            Wave.BCE_nu(i,j) =  0.5*(Wave.BCE_nu(i,j-1) + Wave.BCE_nu(i-1,j))+...
                                0.5*(Wave.BCE_phi(i-1,j) - Wave.BCE_phi(i,j-1));
            Wave.BCE_phi(i,j) = 0.5*(Wave.BCE_phi(i,j-1) + Wave.BCE_phi(i-1,j))+...
                                0.5*(Wave.BCE_nu(i-1,j) - Wave.BCE_nu(i,j-1));
        end
        [Wave.BCE_mu(1,1), Wave.BCE_mach(1,1)] = Mu_From_Nu_Bisection(...
                                                 Wave.BCE_nu(1,1), mach_initial-0.007,...
                                                 3, error_tolerance);
        [Wave.BCE_mu(i,j), Wave.BCE_mach(i,j)] = Mu_From_Nu_Bisection(...
                                                 Wave.BCE_nu(i,j), mach_initial,...
                                                 20, error_tolerance);
        Wave.BCE_alpha(1,1) = Positive_Slope(Wave.BCE_mu(1,1), Wave.BCE_phi(1,1));
        Wave.BCE_alpha(i,j) = Positive_Slope(Wave.BCE_mu(i,j), Wave.BCE_phi(i,j));
```

```
    end
end

%%%                             REGION DFG                                  %%%
%%%=========================================================================%%%
for j = 2:N
    for i = 1:j
        Wave.DFG_nu(1,1)  = Prandtl_Meyer_From_Mach(mach_B);
        Wave.DFG_phi(1,1) = Wave.BCE_phi(1,end) - Wave.BCE_nu(1,end) + Wave.DFG_nu(1,1);
        if i == j
            Wave.DFG_nu(i,j)  = Prandtl_Meyer_From_Mach(mach_B);
            Wave.DFG_phi(i,j) = Wave.DFG_phi(i-1,j) - Wave.DFG_nu(i-1,j) + Wave.DFG_nu(i,j);
        elseif i == 1
            Wave.DFG_nu(i,j)  = 0.5*(Wave.BCE_nu(j,end)  + Wave.DFG_nu(i,j-1))+...
                                0.5*(Wave.DFG_phi(i,j-1) - Wave.BCE_phi(j,end));
            Wave.DFG_phi(i,j) = 0.5*(Wave.BCE_phi(j,end) + Wave.DFG_phi(i,j-1))+...
                                0.5*(Wave.DFG_nu(i,j-1)  - Wave.BCE_nu(j,end));
        else
            Wave.DFG_nu(i,j)  = 0.5*(Wave.DFG_nu(i,j-1)  + Wave.DFG_nu(i-1,j))+...
                                0.5*(Wave.DFG_phi(i,j-1) - Wave.DFG_phi(i-1,j));
            Wave.DFG_phi(i,j) = 0.5*(Wave.DFG_phi(i,j-1) + Wave.DFG_phi(i-1,j))+...
                                0.5*(Wave.DFG_nu(i,j-1)  - Wave.DFG_nu(i-1,j));
        end
        [Wave.DFG_mu(1,1), Wave.DFG_mach(1,1)] = Mu_From_Nu_Bisection(...
                                                 Wave.DFG_nu(1,1), mach_initial,...
                                                 20, error_tolerance);
        [Wave.DFG_mu(i,j), Wave.DFG_mach(i,j)] = Mu_From_Nu_Bisection(...
                                                 Wave.DFG_nu(i,j), mach_initial,...
                                                 20, error_tolerance);
        Wave.DFG_alpha(1,1) = -Negative_Slope(Wave.DFG_mu(1,1), Wave.DFG_phi(1,1));
        Wave.DFG_alpha(i,j) = -Negative_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
    end
end

%%%                             REGION HIJ                                  %%%
%%%=========================================================================%%%
for j = 2:N
    for i = 1:j
        Wave.HIJ_nu(1,1) = Wave.DFG_nu(1,end) + Wave.DFG_phi(1,end);
        if i == j
            Wave.HIJ_nu(i,j)  = Wave.DFG_nu(j,end) + Wave.DFG_phi(j,end);
        elseif i == 1
            Wave.HIJ_nu(i,j)  = 0.5*(Wave.HIJ_nu(i,j-1)  + Wave.DFG_nu(j,end))+...
                                0.5*(Wave.DFG_phi(j,end) - Wave.HIJ_phi(i,j-1));
            Wave.HIJ_phi(i,j) = 0.5*(Wave.HIJ_phi(i,j-1) + Wave.DFG_phi(j,end))+...
                                0.5*(Wave.DFG_nu(j,end)  - Wave.HIJ_nu(i,j-1));
        else
            Wave.HIJ_nu(i,j)  = 0.5*(Wave.HIJ_nu(i,j-1)  + Wave.HIJ_nu(i-1,j))+...
                                0.5*(Wave.HIJ_phi(i-1,j) - Wave.HIJ_phi(i,j-1));
            Wave.HIJ_phi(i,j) = 0.5*(Wave.HIJ_phi(i,j-1) + Wave.HIJ_phi(i-1,j))+...
                                0.5*(Wave.HIJ_nu(i-1,j)  - Wave.HIJ_nu(i,j-1));
        end
        [Wave.HIJ_mu(1,1), Wave.HIJ_mach(1,1)] = Mu_From_Nu_Bisection(...
                                                 Wave.HIJ_nu(1,1), mach_initial-0.007,...
                                                 20, error_tolerance);
        [Wave.HIJ_mu(i,j), Wave.HIJ_mach(i,j)] = Mu_From_Nu_Bisection(...
                                                 Wave.HIJ_nu(i,j), mach_initial-0.007,...
                                                 20, error_tolerance);
```

```
        Wave.HIJ_alpha(1,1) = Positive_Slope(Wave.HIJ_mu(1,1), Wave.HIJ_phi(1,1));
        Wave.HIJ_alpha(i,j) = Positive_Slope(Wave.HIJ_mu(i,j), Wave.HIJ_phi(i,j));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%         Finding intersection coordinates and create plotting        %%%
%%%                     arrays for regions BCE, DFG, HIJ                %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%                          REGION ABC, BCE                            %%%
%%%=====================================================================%%%
gammas = zeros(N,length(x_mesh));
for i = 1:N
    for j = i:N
        positive_slope = 0;
        negative_slope = 0;
        if i == 1
            negative_slope = -Negative_Slope(Mu_From_Mach(Wave.ABC_mach(j)),...
                             Wave.ABC_phi(j));
            incoming_characteristic = h + negative_slope * x_mesh;
            if j == 1
                reflected_characteristic = 0 * x_mesh;
            else
                positive_slope = Positive_Slope(Wave.BCE_mu(i,j), Wave.BCE_phi(i,j));
                diff_positive_slopes = intersections.BCE_y(i,j-1) - positive_slope*...
                    intersections.BCE_x(i,j-1);
                reflected_characteristic = positive_slope * x_mesh+...
                                           diff_positive_slopes;
            end
        else
            negative_slope = -Negative_Slope(Wave.BCE_mu(i,j), Wave.BCE_phi(i,j));
            if j == i
                positive_slope = 0;
            else
                positive_slope = Positive_Slope(Wave.BCE_mu(i,j), Wave.BCE_phi(i,j));
            end
            incoming_characteristic  = intersections.BCE_y(i-1,j) + negative_slope*...
                                       (x_mesh - intersections.BCE_x(i-1,j));
            reflected_characteristic = intersections.BCE_y(i,j-1) + positive_slope*...
                                       (x_mesh - intersections.BCE_x(i,j-1));
        end
        y_i = find(diff(sign(incoming_characteristic - reflected_characteristic)));
        intersections.BCE_y(i,j) = reflected_characteristic(y_i(end));
        intersections.BCE_x(i,j) = x_mesh(y_i(end));

        x_i = find((x_mesh >= intersections.BCE_x(i,j)));
        if i == 1
            gammas(j,1:y_i(1)) = h + negative_slope * x_mesh(y_i(1));
            if j == 1
                prev_x_idx_reflected = find(x_mesh >= intersections.BCE_x(i,end));
            else
                prev_x_idx_reflected = find(x_mesh >= intersections.BCE_x(i,j-1));
            end
        elseif i == j
            prev_x_idx = find((x_mesh >= intersections.BCE_x(i-1,j)));
            gammas(j,prev_x_idx(1,1):end) = intersections.BCE_y(i-1,j) +...
                                            negative_slope *...
```

```
                                        (x_mesh(prev_x_idx(1,1:end)) -...
                                        x_mesh(prev_x_idx(1,1)));
        else
            prev_x_idx = find(x_mesh >= intersections.BCE_x(i-1,j));
            prev_x_idx_reflected = find(x_mesh >= intersections.BCE_x(i,j-1));
            gammas(j,prev_x_idx(1,1):end) = intersections.BCE_y(i-1,j) + ...
            negative_slope * (x_mesh(prev_x_idx(1,1):end) - x_mesh(prev_x_idx(1,1)));
            gammas(i,prev_x_idx_reflected(1,1):end) = ...
            intersections.BCE_y(i,j-1) + positive_slope *...
            (x_mesh(prev_x_idx_reflected(1,1):end) - x_mesh(prev_x_idx_reflected(1,1)));
        end
    end
end
for i = 1:N
    x_i = find(x_mesh <= intersections.BCE_x(1,end-(i-1)));
    negative_slope = (intersections.BCE_y(1,end-(i-1))-1)/...
                     (intersections.BCE_x(1,end-(i-1)))*x_range/x_steps;
    gammas(end-(i-1),1:length(x_i)) = Linear_Function(negative_slope,1,x_i,0);
    if i == N
        break
    else
        x_i = find(x_mesh >= intersections.BCE_x(1,i) &...
                   x_mesh <= intersections.BCE_x(1,i+1));
        positive_slope = ((intersections.BCE_y(1,i+1)-intersections.BCE_y(1,i))...
        / (intersections.BCE_x(1,i+1)-intersections.BCE_x(1,i))) * x_range/x_steps;
        gammas(1, x_i(1):x_i(end)) = Linear_Function(positive_slope,...
                                     intersections.BCE_y(1,i), x_i - x_i(1), 0);
    end
end

%%%                            REGION DFG                                  %%%
%%%========================================================================%%%
jet_boundary = h + tan(phi_B) * x_mesh;
for i = 1:N
    for j = i:N
        diff_slopes_positive = 0;
        diff_slopes_negative = 0;
        if i == 1
            if j == 1
            diff_slopes = intersections.DFG_y(i,end) - Wave.BCE_alpha(j,end)...
                          *intersections.DFG_x(i,end);
            else
            diff_slopes = intersections.DFG_y(i,j-1) - Wave.BCE_alpha(j,end)...
                          *intersections.DFG_x(i,j-1);
            end
            incoming_characteristic = Linear_Function(Wave.BCE_alpha(j,end),...
            intersections.BCE_y(j,end), x_mesh, intersections.BCE_x(j,end));
            positive_slope = Positive_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
            negative_slope = -Negative_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
            if j == i
                reflected_characteristic = jet_boundary;
            else
                negative_slope = -Negative_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
                diff_slopes_negative = intersections.DFG_y(i,j-1) -...
                                       negative_slope * intersections.DFG_x(i,j-1);
                reflected_characteristic = Linear_Function(negative_slope,...
                                           diff_slopes_negative, x_mesh, 0);
            end
```

```
else
    incoming_characteristic = 0;
    reflected_characteristic = 0;
    negative_slope = -Negative_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
    positive_slope = Positive_Slope(Wave.DFG_mu(i,j), Wave.DFG_phi(i,j));
    if j ~= i
        diff_slopes_positive = intersections.DFG_y(i-1,j)...
        - positive_slope * intersections.DFG_x(i-1,j);
        diff_slopes_negative = intersections.DFG_y(i,j-1)...
        - negative_slope * intersections.DFG_x(i,j-1);
        incoming_characteristic = Linear_Function(positive_slope,...
                                  diff_slopes_positive, x_mesh, 0);
        reflected_characteristic = Linear_Function(negative_slope,...
                                    diff_slopes_negative, x_mesh, 0);
    else
        diff_slopes_positive = intersections.DFG_y(i-1,j)...
        - positive_slope * intersections.DFG_x(i-1,j);
        incoming_characteristic = Linear_Function(positive_slope,...
                                  diff_slopes_positive, x_mesh, 0);
        reflected_characteristic = jet_boundary;
    end
end
y_i = find(diff(sign(incoming_characteristic - reflected_characteristic)));
intersections.DFG_y(i,j) = incoming_characteristic(y_i(end));
intersections.DFG_x(i,j) = x_mesh(y_i(end));

x_idx = find(x_mesh >= intersections.DFG_x(i,j));
if i == 1
    prev_Wave.x_idx = find(x_mesh >= intersections.BCE_x(j,end));
    if j == 1
        prev_x_idx_reflected = find(x_mesh >= intersections.DFG_x(i,end));
    else
        prev_x_idx_reflected = find(x_mesh >= intersections.DFG_x(i,j-1));
    end
    gammas(j,prev_Wave.x_idx(1,1):x_idx(1,1)) = intersections.BCE_y(j,end)...
    + positive_slope * (x_mesh(prev_Wave.x_idx(1,1):x_idx(1,1))...
    - x_mesh(prev_Wave.x_idx(1,1)));
    if j > 1
        gammas(j,prev_Wave.x_idx(1,1):x_idx(1,1)) = intersections.BCE_y(j,end)...
        + Wave.BCE_alpha(j,end) * (x_mesh(prev_Wave.x_idx(1,1):x_idx(1,1))...
        - x_mesh(prev_Wave.x_idx(1,1)));
        gammas(i,prev_x_idx_reflected(1,1):x_idx(1,1) + 1) =...
       intersections.DFG_y(i,j-1) + negative_slope * ...
       (x_mesh(prev_x_idx_reflected(1,1):x_idx(1,1)+1) -...
       x_mesh(prev_x_idx_reflected(1,1)));
    else
        diff_slopes = jet_boundary(x_idx(1,1)) - x_mesh(x_idx(1,1))...
        * tan(Wave.DFG_phi(i,j));
        jet_boundary(x_idx(1,1):end) = tan(Wave.DFG_phi(i,j))...
        * x_mesh(x_idx(1,1):end) + diff_slopes;
    end
elseif j == i
    prev_x_idx = find(x_mesh >= intersections.DFG_x(i-1,j));
    gammas(j,prev_x_idx(1,1):x_idx(1,1) +1) = intersections.DFG_y(i-1,j)...
    + positive_slope * (x_mesh(prev_x_idx(1,1):x_idx(1,1) +1)...
    - x_mesh(prev_x_idx(1,1)));
    diff_slopes = jet_boundary(x_idx(1,1)) - x_mesh(x_idx(1,1))...
    * tan(Wave.DFG_phi(i,j));
```

```
            jet_boundary(x_idx(1,1):end) = tan(Wave.DFG_phi(i,j))...
            * x_mesh(x_idx(1,1):end) + diff_slopes;
        else
            prev_x_idx = find(x_mesh >= intersections.DFG_x(i-1,j));
            idx = find(x_mesh >= intersections.DFG_x(i,j));
            prev_x_idx_reflected = find(x_mesh >= intersections.DFG_x(i,j-1));
            gammas(i,prev_x_idx_reflected(1,1):end) = negative_slope *...
            (x_mesh(prev_x_idx_reflected(1,1):end)) + diff_slopes_negative;
            gammas(j,prev_x_idx(1,1):end) = positive_slope *...
            (x_mesh(prev_x_idx(1,1):end)) + diff_slopes_positive;
        end
        if j == N - 1
            gammas(i,x_idx(1,1):end) = intersections.DFG_y(i,j)...
            + Wave.DFG_alpha(i,j) * (x_mesh(x_idx(1,1):end) - x_mesh(x_idx(1,1)));
        end
    end
end

%%%                            REGION HIJ                                  %%%
%%%========================================================================%%%
for i = 1:N
    for j = i:N
        diff_slopes_positive = 0;
        diff_slopes_negative = 0;
        positive_slope = 0;
        negative_slope = 0;
        if i == 1
            negative_slope = Wave.DFG_alpha(j,end);
            diff_slopes_negative = intersections.DFG_y(j,end) ...
            - negative_slope * intersections.DFG_x(j,end);
            incoming_characteristic = diff_slopes_negative + negative_slope...
            * x_mesh;
            if j == 1
                reflected_characteristic = 0 * x_mesh;
            else
                positive_slope = Positive_Slope(Wave.HIJ_mu(i,j-1),...
                                 Wave.HIJ_phi(i,j-1));
                diff_slopes_positive = intersections.HIJ_y(i,j-1)...
                - positive_slope * intersections.HIJ_x(i,j-1);
                reflected_characteristic = positive_slope...
                * x_mesh + diff_slopes_positive;
            end
        else
            negative_slope = -Negative_Slope(Wave.HIJ_mu(i-1,j), Wave.HIJ_phi(i-1,j));
            if j == i
                reflected_characteristic = 0 * x_mesh;
            else
                positive_slope = Positive_Slope(Wave.HIJ_mu(i,j), Wave.HIJ_phi(i,j));
                diff_slopes_positive = intersections.HIJ_y(i,j-1)...
                - positive_slope * intersections.HIJ_x(i,j-1);
                reflected_characteristic = intersections.HIJ_y(i,j-1)...
                + positive_slope * (x_mesh - intersections.HIJ_x(i,j-1));
            end
            incoming_characteristic = intersections.HIJ_y(i-1,j) +...
            negative_slope * (x_mesh - intersections.HIJ_x(i-1,j));
        end
        y_i = find(diff(sign(incoming_characteristic - reflected_characteristic)));
        intersections.HIJ_y(i,j) = reflected_characteristic(y_i);
```

```
        intersections.HIJ_x(i,j) = x_mesh(y_i);

        x_idx = find(x_mesh >= intersections.HIJ_x(i,j));
        if i == 1
            prev_Wave.x_idx = find(x_mesh >= intersections.DFG_x(j,end));
            gammas(j,prev_Wave.x_idx(1,1):end) = diff_slopes_negative...
            + negative_slope * x_mesh(prev_Wave.x_idx(1,1):end);
            if j > 1
                prev_x_idx_reflected = find(x_mesh >= intersections.HIJ_x(i,j-1));
                gammas(i,prev_x_idx_reflected(1,1):end) =...
                intersections.HIJ_y(i,j-1) + positive_slope...
                *(x_mesh(prev_x_idx_reflected(1,1):end)-...
                x_mesh(prev_x_idx_reflected(1,1)));
            end
        elseif j == i
            prev_x_idx = find(x_mesh >= intersections.HIJ_x(i-1,j));
            gammas(j,prev_x_idx(1,1):end) = intersections.HIJ_y(i-1,j)...
            + negative_slope * (x_mesh(prev_x_idx(1,1):end)...
            - x_mesh(prev_x_idx(1,1)));
        else
            prev_x_idx = find(x_mesh >= intersections.HIJ_x(i-1,j));
            prev_x_idx_reflected = find(x_mesh >= intersections.HIJ_x(i,j-1));
            idx = find(x_mesh >= intersections.HIJ_x(i,j));
            gammas(j,prev_x_idx(1,1):end) = intersections.HIJ_y(i-1,j)...
            + negative_slope * (x_mesh(prev_x_idx(1,1):end)-...
            x_mesh(prev_x_idx(1,1)));
            gammas(i,prev_x_idx_reflected(1,1):end) = intersections.HIJ_y(i,j-1)...
            + positive_slope * (x_mesh(prev_x_idx_reflected(1,1):end)...
            - x_mesh(prev_x_idx_reflected(1,1)));
            if j == N-1
                gammas(i,x_idx(1,1):end) = intersections.HIJ_y(i,j)...
                + Wave.HIJ_alpha(i,j) * (x_mesh(x_idx(1,1):end)...
                - x_mesh(x_idx(1,1)));
            end
        end
    end
end

%%%                       MINOR IRRELEVANT FIX                              %%%
x_i = find(x_mesh >= intersections.HIJ_x(N,N));
gammas(N,x_i(1):end) = -gammas(N,x_i(1):end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                         Creating heat map                               %%%
%%%     function: Create_Heat_Map_Geometry constructs all patches           %%%
%%%              where local mach numbers will be plotted                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
colormap = Create_Heat_Map_Geometry(intersections, N, x_range, jet_boundary);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%           Creating Streamline for given initial height                  %%%
%%%         function: Create_Streamline constructs streamline               %%%
%%%          coordinates and pressure along those coordinates               %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
initial_height_streamline_1 = 0.5;
initial_height_streamline_2 = 0.75;
initial_height_streamline_3 = 0.25;
```

```
[streamline_1, press_dist_1, x_crossings_1] =...
Create_Streamline(initial_height_streamline_1, x_mesh, mach_initial,...
gamma, pressure_total, N, x_steps, intersections, gammas, Wave);
[streamline_2, press_dist_2, x_crossings_2] =...
Create_Streamline(initial_height_streamline_2, x_mesh, mach_initial,...
gamma, pressure_total, N, x_steps, intersections, gammas, Wave);
[streamline_3, press_dist_3, x_crossings_3] =...
Create_Streamline(initial_height_streamline_3, x_mesh, mach_initial,...
gamma, pressure_total, N, x_steps, intersections, gammas, Wave);
```

B Functions

All functions used in the main script can be found here.

B.1 Alpha

```
function [alpha] = Alpha(mu,phi)
%ALPHA Calculates alpha from mu and phi in radians
alpha = phi-mu;
end
```

B.2 Bisection Method

```
function [mach_i, phi_i, nu_i] = Bisection_Method(mach_left, mach_right, tolerance,...
alpha, backward_phi, backward_nu)
%BISECTION_METHOD Bisection iteration method to find variables (mach, phi
%and nu) in simple wave region ABC
mach_i = (mach_left+mach_right)/2;
while (abs(mach_left - mach_right) >= tolerance)
    mach_i = (mach_left+mach_right)/2;
    step = Bisection_Steps(mach_i, alpha, backward_phi, backward_nu);
    if (abs(step) < abs(tolerance))
        break
    elseif (step * Bisection_Steps(mach_left, alpha, backward_phi, backward_nu) < 0)
        mach_right = mach_i;
    else
        mach_left = mach_i;
    end
end
mu_i  = Mu_From_Mach(mach_i);
phi_i = mu_i + alpha;
nu_i  = Prandtl_Meyer_From_Mach(mach_i);
end
```

B.3 Center Pressure Distribution

```
function [press_dist, x_crossings] = Center_Pressure_Distribution(press_dist,...
x_crossings, Wave_BCE_mach, Wave_HIJ_mach, gamma, p_total)
%CENTER_PRESSURE_DISTRIBUTION
for i = 1:N
    x_pos       = x_intersections_BCE(i,i);
    press_dist  = [ [press_dist],  ...
    [Pressure_From_Total_Pressure(Wave_BCE_mach(i,i), gamma, p_total)] ];
    x_crossings = [ [x_crossings] , [x_pos] ];
    x_pos       = x_intersections_HIJ(i,i);
    press_dist  = [ [press_dist] ,...
    [Pressure_From_Total_Pressure(Wave_HIJ_mach(i,i), gamma, p_total)] ];
    x_crossings = [ [x_crossings] , [x_pos] ];
```

```
end

end
```

B.4 Create Heat Map Geometry

```
function [colormap] = Create_Heat_Map_Geometry(intersections, N, x_range, jet_boundary)
%CREATE_HEAT_MAP_GEOMETRY Creates triangles and squares where local mach
%values can be plotted

%%%                          REGION OUTLET                                  %%%
%%%=========================================================================%%%
colormap.AB0_x = [0; 0; intersections.BCE_x(1,1)];
colormap.AB0_y = [0; 1; intersections.BCE_y(1,1)];

%%%                           REGION ABC                                    %%%
%%%=========================================================================%%%
for i = 1:N-1
colormap.ABC_x{i} = [0; intersections.BCE_x(1,i); intersections.BCE_x(1,i+1)];
colormap.ABC_y{i} = [1; intersections.BCE_y(1,i); intersections.BCE_y(1,i+1)];
end

%%%                           REGION ACD                                    %%%
%%%=========================================================================%%%
colormap.ACD_x = [0; intersections.BCE_x(1,end); intersections.DFG_x(1,1)];
colormap.ACD_y = [1; intersections.BCE_y(1,end); intersections.DFG_y(1,1)];

%%%                           REGION BCE                                    %%%
%%%=========================================================================%%%
for i = 1:N-1
    for j = i:N-1
        if j == i
        colormap.BCE_x{i,j} = [intersections.BCE_x(i,j);...
        intersections.BCE_x(i,j+1); intersections.BCE_x(i+1,j+1)];
        colormap.BCE_y{i,j} = [intersections.BCE_y(i,j);...
        intersections.BCE_y(i,j+1); intersections.BCE_y(i+1,j+1)];
        else
        colormap.BCE_x{i,j} = [intersections.BCE_x(i,j);...
        intersections.BCE_x(i,j+1); intersections.BCE_x(i+1,j+1);...
        intersections.BCE_x(i+1,j)];
        colormap.BCE_y{i,j} = [intersections.BCE_y(i,j);...
        intersections.BCE_y(i,j+1); intersections.BCE_y(i+1,j+1);...
        intersections.BCE_y(i+1,j)];
        end
    end
end

%%%                           REGION CDEF                                   %%%
%%%=========================================================================%%%
for i = 1:N-1
    colormap.CDEF_x{i} = [intersections.BCE_x(i,end); intersections.BCE_x(i+1,end);...
                          intersections.DFG_x(1,i+1); intersections.DFG_x(1,i)];
    colormap.CDEF_y{i} = [intersections.BCE_y(i,end); intersections.BCE_y(i+1,end);...
                          intersections.DFG_y(1,i+1); intersections.DFG_y(1,i)];
end

%%%                           REGION EFH                                    %%%
%%%=========================================================================%%%
```

```
colormap.EFH_x = [intersections.BCE_x(end,end); intersections.DFG_x(1,end);...
                  intersections.HIJ_x(1,1)];
colormap.EFH_y = [intersections.BCE_y(end,end); intersections.DFG_y(1,end);...
                  intersections.HIJ_y(1,1)];

%%%                           REGION DFG                                  %%%
%%%=======================================================================%%%
for i = 1:N-1
    for j = i:N-1
        if j == i
        colormap.DFG_x{i,j} = [intersections.DFG_x(i,j);...
        intersections.DFG_x(i,j+1); intersections.DFG_x(i+1,j+1)];
        colormap.DFG_y{i,j} = [intersections.DFG_y(i,j);...
        intersections.DFG_y(i,j+1); intersections.DFG_y(i+1,j+1)];
        else
        colormap.DFG_x{i,j} = [intersections.DFG_x(i,j);...
        intersections.DFG_x(i,j+1); intersections.DFG_x(i+1,j+1);...
        intersections.DFG_x(i+1,j)];
        colormap.DFG_y{i,j} = [intersections.DFG_y(i,j);...
        intersections.DFG_y(i,j+1); intersections.DFG_y(i+1,j+1);...
        intersections.DFG_y(i+1,j)];
        end
    end
end

%%%                           REGION GFHI                                 %%%
%%%=======================================================================%%%
for i = 1:N-1
    colormap.GFHI_x{i} = [intersections.DFG_x(i,end);...
    intersections.DFG_x(i+1,end); intersections.HIJ_x(1,i+1);...
    intersections.HIJ_x(1,i)];
    colormap.GFHI_y{i} = [intersections.DFG_y(i,end);...
    intersections.DFG_y(i+1,end); intersections.HIJ_y(1,i+1);...
    intersections.HIJ_y(1,i)];
end

%%%                           REGION HIJ                                  %%%
%%%=======================================================================%%%
for i = 1:N-1
    for j = i:N-1
        if j == i
        colormap.HIJ_x{i,j} = [intersections.HIJ_x(i,j);...
        intersections.HIJ_x(i,j+1); intersections.HIJ_x(i+1,j+1)];
        colormap.HIJ_y{i,j} = [intersections.HIJ_y(i,j);...
        intersections.HIJ_y(i,j+1); intersections.HIJ_y(i+1,j+1)];
        else
        colormap.HIJ_x{i,j} = [intersections.HIJ_x(i,j);...
        intersections.HIJ_x(i,j+1); intersections.HIJ_x(i+1,j+1);...
        intersections.HIJ_x(i+1,j)];
        colormap.HIJ_y{i,j} = [intersections.HIJ_y(i,j);...
        intersections.HIJ_y(i,j+1); intersections.HIJ_y(i+1,j+1);...
        intersections.HIJ_y(i+1,j)];
        end
    end
end

%%%                           REGION IJK                                  %%%
%%%=======================================================================%%%
```

```
colormap.IJK_x = [intersections.HIJ_x(2,end); intersections.DFG_x(end,end);...
                  x_range];
colormap.IJK_y = [intersections.HIJ_y(2,end); intersections.DFG_y(end,end);...
                  jet_boundary(end)];
end
```

B.5 Create Streamline

```
function [streamline, press_dist, x_crossings] = ...
Create_Streamline(initial_height_streamline, x_mesh, mach_initial,...
gamma, pressure_total, N, x_steps, intersections, gammas, Wave)
%CREATE_STREAMLINE
streamline = initial_height_streamline + 0 * x_mesh;
press_dist = (Pressure_From_Total_Pressure(mach_initial, gamma, pressure_total));
x_crossings = x_mesh(1);
if initial_height_streamline > 0
    end_reached = true;
else
    end_reached = false;
    [press_dist, x_crossings] = Center_Pressure_Distribution(press_dist, x_crossings);
end
prev_i = 1;
slope = 0;
while end_reached
    streamline_indices = Streamline_Crossing(streamline, prev_i, N, gammas);
    [prev_i, Gamma] = Find_Crossing(streamline_indices, prev_i, x_steps, N);
    if x_mesh(prev_i) < intersections.BCE_x(Gamma,Gamma)
            slope = tan(Wave.ABC_phi(Gamma));
            local_mach = Wave.ABC_mach(Gamma);
    elseif (x_mesh(prev_i) > intersections.BCE_x(Gamma,Gamma) && x_mesh(prev_i)...
            < intersections.BCE_x(Gamma,end))
        for i = 1:N
            if i == 1
                if (streamline(prev_i) < intersections.BCE_y(Gamma,i)...
                    && streamline(prev_i) > intersections.BCE_y(Gamma,end))
                    slope = tan((Wave.BCE_phi(Gamma,i) + Wave.BCE_phi(Gamma,end)) / 2);
                    local_mach = Wave.BCE_mach(Gamma,i);
                end
            else
                if (streamline(prev_i) < intersections.BCE_y(Gamma,i)...
                    && streamline(prev_i) > intersections.BCE_y(Gamma,i-1))
                    slope = tan((Wave.BCE_phi(Gamma,i) + Wave.BCE_phi(Gamma,i-1)) / 2);
                    local_mach = Wave.BCE_mach(Gamma,i);
                end
            end
        end
    elseif (x_mesh(prev_i) > intersections.BCE_x(Gamma,end) && x_mesh(prev_i)...
            < intersections.DFG_x(Gamma,Gamma))
            slope = tan(Wave.BCE_phi(Gamma,end));
            local_mach = Wave.BCE_mach(Gamma,end);
    elseif (x_mesh(prev_i) > intersections.DFG_x(Gamma,Gamma) && x_mesh(prev_i)...
            < intersections.DFG_x(Gamma,end))
        for i = 1:N
            if i == 1
                if (streamline(prev_i) < intersections.DFG_y(Gamma,i)...
                    && streamline(prev_i) > intersections.DFG_y(Gamma,end)...
                    || streamline(prev_i) > intersections.DFG_y(Gamma,i) &&...
                    streamline(prev_i) < intersections.DFG_y(Gamma,end))
```

```
                    slope = tan((Wave.DFG_phi(Gamma,i) + Wave.DFG_phi(Gamma,end))/2);
                    local_mach = Wave.DFG_mach(Gamma,i);
                end
            else
                if (streamline(prev_i) < intersections.DFG_y(Gamma,i)...
                   && streamline(prev_i) > intersections.DFG_y(Gamma,i-1)...
                   || streamline(prev_i) > intersections.DFG_y(Gamma,i)...
                   && streamline(prev_i) < intersections.DFG_y(Gamma,i-1))
                   slope = tan((Wave.DFG_phi(Gamma,i) + Wave.DFG_phi(Gamma,i-1))/2);
                   local_mach = Wave.DFG_mach(Gamma,i);
                end
            end
        end
    elseif (x_mesh(prev_i) > intersections.DFG_x(Gamma,end)...
            && x_mesh(prev_i) < intersections.HIJ_x(Gamma,Gamma))
            slope = tan(Wave.DFG_phi(Gamma,end));
            local_mach = Wave.DFG_mach(Gamma,end);
    elseif (x_mesh(prev_i) > intersections.HIJ_x(Gamma,Gamma)...
            && x_mesh(prev_i) < intersections.HIJ_x(Gamma,end))
        for i = 1:N
            if i == 1
                if (streamline(prev_i) < intersections.HIJ_y(Gamma,i)...
                    && streamline(prev_i) > intersections.HIJ_y(Gamma,end))
                    slope = tan((Wave.HIJ_phi(Gamma,i) + Wave.HIJ_phi(Gamma,end))/2);
                    local_mach = Wave.HIJ_mach(Gamma,i);
                end
            else
                if (streamline(prev_i) < intersections.HIJ_y(Gamma,i)...
                    && streamline(prev_i) > intersections.HIJ_y(Gamma,i-1))
                    slope = tan((Wave.HIJ_phi(Gamma,i) + Wave.HIJ_phi(Gamma,i-1))/2);
                    local_mach = Wave.HIJ_mach(Gamma,i);
                end
            end
        end
    elseif x_mesh(prev_i) > intersections.HIJ_x(Gamma,end)
            slope = tan(Wave.HIJ_phi(Gamma,end));
            end_reached = false;
    end

    streamline(prev_i:end) = streamline(prev_i) + slope...
                           * (x_mesh(prev_i:end) - x_mesh(prev_i));

    x_crossings = [x_crossings, x_mesh(prev_i)];
    press_dist = [press_dist, Pressure_From_Total_Pressure(local_mach,...
                 gamma, pressure_total)];
end

end
```

B.6 Find Intersections

```
function [lowest_idx, gamma] = Find_Crossing(streamline_indices, prev_idx, x_steps, N)
%FIND_CROSSING
lowest_idx = x_steps;
gamma = 0;
for i = 1:N
    if  (streamline_indices.index(i) > prev_idx && streamline_indices.index(i) < lowest_idx)
        lowest_idx = streamline_indices.index(i);
```

```
        gamma = streamline_indices.gamma(i);
    end
end
```

B.7 Forward Phi Gamma Plus

```
function [phi_forward] = Forward_Phi_Gamma_Plus(nu_backward,nu_forward,phi_backward)
%FORWARD_PHI_GAMMA_PLUS Calculates forward phi
phi_forward = nu_forward + phi_backward - nu_backward;
end
```

B.8 Mach For Prandtl-Meyer

```
function [mach_for_prandtl_meyer] = Mach_For_Prandtl_Meyer(mach,nu)
%MACH_FOR_PRANDTL_MEYER
mach_for_prandtl_meyer = Prandtl_Meyer_From_Mach(mach) - nu;
end
```

B.9 Mu From Mach

```
function [mu] = Mu_From_Mach(mach)
%MU_FROM_MACH Calculates angle mu in radians from mach number
mu = asin(1/mach);
end
```

B.10 Mu From Nu Bisection

```
function [mu, mach_i] = Mu_From_Nu_Bisection(nu, mach_left, mach_right, tolerance)
%MU_FROM_NU_BISECTION
    while (abs(mach_left - mach_right) >= tolerance)
        mach_i = (mach_left + mach_right)/2;
        step = Mach_For_Prandtl_Meyer(mach_i, nu);
        if (abs(step) < abs(tolerance))
            break
        elseif (step * Mach_For_Prandtl_Meyer(mach_left, nu) < tolerance)
            mach_right = mach_i;
        else
            mach_left = mach_i;
            mu = Mu_From_Mach(mach_i);
        end
    end
end
```

B.11 Prandtl-Meyer From Mach

```
function [prandtl_meyer] = Prandtl_Meyer_From_Mach(mach)
%Prandtl_Meyer_From_Mach: Calculates the Prandtl-Meyer in radians from given mach number
gamma = 7/5;
prandtl_meyer = sqrt((gamma+1)/(gamma-1)) * atan( sqrt(((gamma-1)/(gamma+1))...
                * (mach^2-1)) ) - atan(sqrt(mach^2 -1));
end
```

B.12 Pressure From Total Pressure

```
function [prandtl_meyer] = Prandtl_Meyer_From_Mach(mach)
%Prandtl_Meyer_From_Mach: Calculates the Prandtl-Meyer in radians from given mach number
gamma = 7/5;
prandtl_meyer = sqrt((gamma+1)/(gamma-1)) * atan( sqrt(((gamma-1)/(gamma+1))...
                * (mach^2-1)) ) - atan(sqrt(mach^2 -1));
```

```
end
```

B.13 Streamline Intersections

```
function [streamline_indices] = Streamline_Intersections(streamline, prev_idx, N, gammas)
%STREAMLINE_INTERSECTIONS
streamline_indices = struct('index',zeros(N,1),'gamma',zeros(N,1));
for i = 1:N
    crossing_index = find(diff(sign(streamline(prev_idx:end) - gammas(i,prev_idx:end))));
    if (isempty(streamline_indices) && size(crossing_index) > 0)
        streamline_indices.index(i) = prev_idx + min(crossing_index);
        streamline_indices.gamma(i) = i;
    elseif (size(crossing_index) > 0)
        streamline_indices.index(i) = prev_idx + min(crossing_index);
        streamline_indices.gamma(i) = i;
    end
end
end
```