AE602 - Compressible Flow Assignment - 03

Ramkumar S. *
SC22M007, M.Tech. Aerospace - Aerodynamics and Flight Mechanics

The present work is about solving the given supersonic flow over an expansion corner problem using the Method of Characteristics. The inlet Mach number is specified to be 3.0 and the flow deflection angle is given to be $\theta = 15°$, the duct height is 30 cm. and the length of the expansion section is taken to be 120 cm.

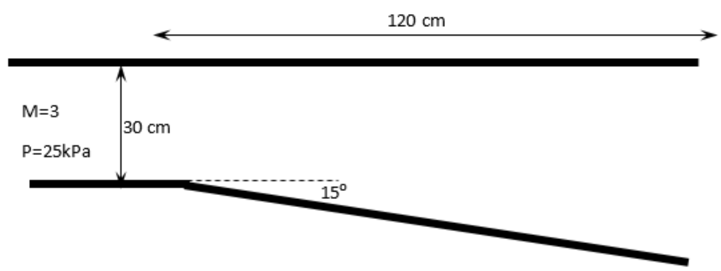
I. Question

Given the inlet conditions and geometry of the test case in the accompanying figure.

Solve the following problem Using the method of characteristics. Provide your answers with

1. Solution based on both point to point method and region to region methods
2. Mach number distribution in the entire domain
3. Computer program of both solutions
4. Graphical representation of the desired flow path using both solutions.

Submit your results as a single PDF file in AIAA format with necessary theory and figures explaining the solution methods.



SOLUTION:

Given data

$$M_1 = 3.0$$
$$h = 0.3\ m$$
$$L = 1.2\ m$$
$$P_1 = 25\ kPa$$

The solution algorithm followed for this problem is given below.

The number of characteristic nodes N, in the inlet plane is first chosen. The value chosen for this problem is 100. An example layout of Characteristic lines with N = 6, in the expansion flow domain of deflection angle 2° is given in Figure 1.But, the actual problem with 15° deflection angle requies N = 100 in order to accurately capture the expansion flow field.

*SC22M007, M.Tech., Aerospace AFM

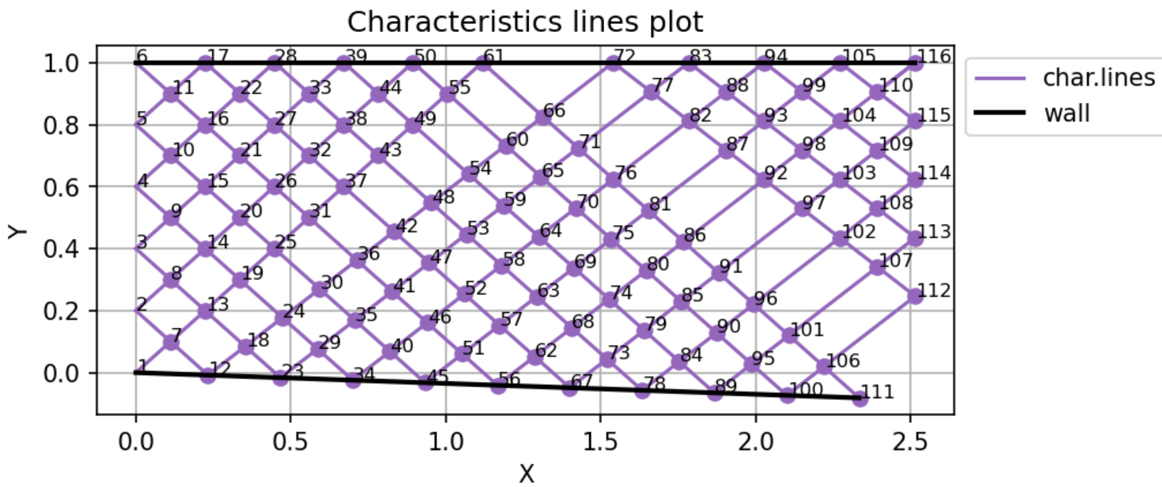


Fig. 1 Coarse characteristic points layout on the 2° **expansion corner**

Then the total number of characteristic nodes that will appear after all interactions is computed using the equation below. Here, n_{wall} is the number of wall bounces requried as it is the one that controls the length of the computation domain in this case.

$$N_{total} = n_{wall}(2N - 1) + N$$

After this, a list of numbers indicating the characteristic points were generated linearly similar to the one shown in Figure 1 and the points on both top and bottom wall is identified using the below equations and were grouped for ease of computation.

$$bottom\ wall : n_b = n_{b,prev} + (2N - 1)$$
$$top\ wall : n_b = n_{b,prev} + (N - 1)$$

Then a table, with the list of dependence upstream characteristic points, to each internal and boundary char.point is made, which will be used during computations. The inlet condition is defined such that, the expansion function values $\nu(M)$ at all inlet char.points is computed for the following specified condition. And the values of K_1 and K_2 were computed using the relations given.

$$M_{inlet} = 3.0$$
$$\theta_{inlet} = 0.0$$
$$K_1 = \nu + \theta$$
$$K_2 = \nu - \theta$$

Then the computation is begun for internal points, where the values of ν and θ were computed by taking the intersected K_1 and K_2 upstream values as shown below.

$$\nu = \frac{K_1 + K_2}{2}$$
$$\theta = \frac{K_1 - K_2}{2}$$

At the wall points, only one upstream characteristics meet, but the wall angle θ_{wall} is known, hence using the upstream characteristic and wall angle values, the expansion function is computed as shown in the example below.

$$bottom\ wall : \nu = K_1 - \theta$$
$$top\ wall : \nu = K_2 + \theta$$

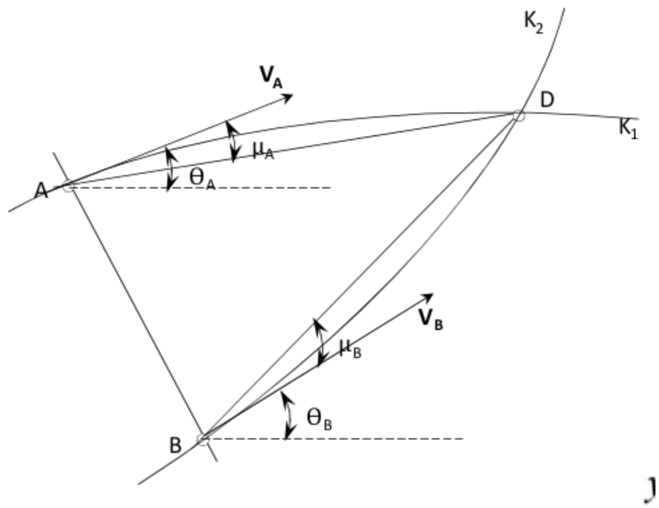


Fig. 2 reference image for the char. point position calculations

Then, the Mach numbers at each characteristic point were computed using Prandtl-Meyer expansion function relation given below.

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

For computing the location of downstream char.point, the following equations that determine the slope of line (with the assumption that the char. curves are straight lines in short length) and the location of the point as shown in the Figure 2.

$$
\begin{aligned}
\left(\frac{dy}{dx}\right)_A &= tan(\theta-\mu)_A \\
\left(\frac{dy}{dx}\right)_B &= tan(\theta+\mu)_B \\
S_1 &= \frac{tan(\theta-\mu)_A + tan(\theta-\mu)_B}{2} \\
S_2 &= \frac{tan(\theta+\mu)_A + tan(\theta+\mu)_B}{2} \\
y_D &= y_A + (x_D - x_A)S_1 \\
y_D &= y_B + (x_D - x_B)S_2 \\
x_D &= \frac{(S_2 x_B - S_1 x_A) + (y_A - y_B)}{S_2 - S_1}
\end{aligned}
$$

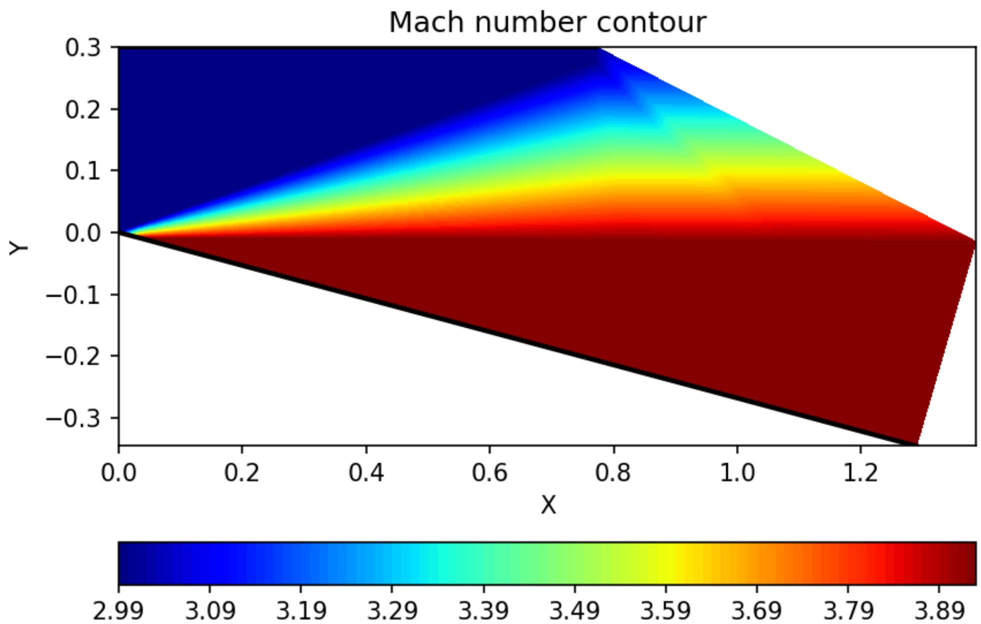


Fig. 3 Mach number contour output from computation

Results:

The Python code was developed for this computation and given in Section A. The contour of Mach number distribution and the characteristic points obtained for this problem is given in Figures 3 and 4, respectively.

Further, the Mach number of downstream section after expansion, obtained from the computation is compared with the theoretical value obtained from [1]. The following gives the values obtained and they found to be in agreement with the theory.

$$M_{2,computation} = 3.92329$$
$$M_{2,theory} = 3.923$$

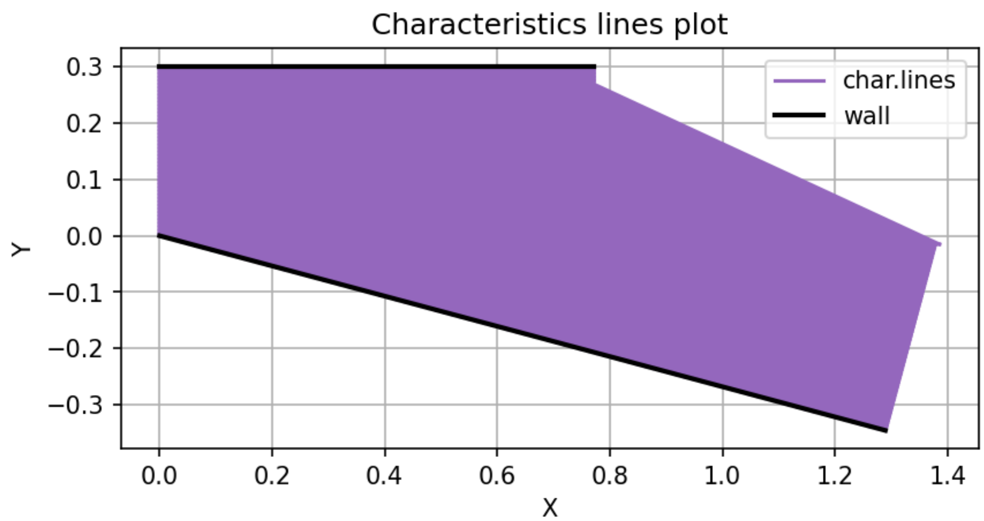


Fig. 4 characteristic points distribution (conjested due to 18k points for N = 100)

References

[1] Online Prandtl-Meyer expansion flow calculator; https://www.omnicalculator.com/physics/prandtl-meyer-expansion

[2] Michel A. Saad; Compressible fluid flow

A. Appendix - Python code of Question 1

This section contains the *Python* code developed for the MOC computation

```
#!/bin/python3
"""
AE602 - Compressible Flow
Assignment - 03

Expansion corner flow field computation using Method of Characteristics

Ramkumar S.
SC22M007
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# disabling setting with a copy warning
pd.options.mode.chained_assignment = None  # default='warn'

# input data-------------------------------------------------------------
# number of char points in the inlet
N = 100

# number of wall bounces required
n_wall = 90

# inlet Mach number
M_inlet = 3

# ratio of specific heats
g = 1.4

# lower wall angle
theta_lower = np.radians(-15.0)

# upper wall angle
theta_upper = np.radians(0.0)

# channel height in m
height = 0.3

# initializing dataframe with char points---------------------------------
# getting total number of char points including inlet ones
N_total = n_wall*(2*N-1)+N

# preparing a list of char points and making a dataframe
fid = pd.DataFrame(np.transpose(np.linspace(1,N_total,N_total,dtype=int)),
        columns = ["N"])
fid["N1"] = 0
fid["N2"] = 0

# preparing list of bottom wall points
pnts_bottom = [1]
pnts_top = [N]
while True:
    val = pnts_bottom[-1] + (2*N-1)
    if val > N_total:
        break
    pnts_bottom.append(val)
    pnts_top.append(val + N-1)

# assigning dependence char points
for i in range(N,N_total):
    # getting current char point
    N_curr = fid["N"].iloc[i]
```

```python
    # checking if the current point is on a wall
    if N_curr in pnts_bottom:
        fid["N1"].iloc[i] = N_curr - N + 1
        fid["N2"].iloc[i] = 0
    elif N_curr in pnts_top:
        fid["N1"].iloc[i] = 0
        fid["N2"].iloc[i] = N_curr - N
    else:
        fid["N1"].iloc[i] = N_curr - N
        fid["N2"].iloc[i] = N_curr - N + 1

# adding needed columns for further computation
fid["M"] = 0
fid["K1"] = 0
fid["K2"] = 0
fid["theta"] = 0
fid["nu"] = 0
fid["mu"] = 0
fid["X"] = 0
fid["Y"] = 0

# function definitions-----------------------------------------------
# prandtl-meyer function
def PM_nu(Mach):
    val = (np.sqrt((g+1)/(g-1))*np.arctan(np.sqrt((g-1)/(g+1)*(Mach**2-1)))
           - np.arctan(np.sqrt(Mach**2-1)))
    return val

# inverse prandtl-meyer function
def inv_PM(nu):
    # using bisection method

    # initial values of mach numbers
    Ma = 1
    Mb = 200
    Mc = (Ma+Mb)/2

    # begin loop
    while abs(nu - PM_nu(Mc)) > 1e-6:
        # computing residual
        res = nu - PM_nu(Mc)

        # deciding the offset side
        if res > 0:
            Ma = Mc
        else:
            Mb = Mc

        # updating Mc value
        Mc = (Ma+Mb)/2

    return Mc


# begin computation---------------------------------------------------
# initializing inlet data in the dataframe
fid["M"].iloc[0:N] = M_inlet #  Mach no
fid["theta"].iloc[0:N] = 0.0 #  flow deflection angle
fid["nu"].iloc[0:N] = PM_nu(M_inlet) # PM function value
fid["mu"].iloc[0:N] = np.arcsin(1/M_inlet) # Mach angle

# computing K1 and K2 for inlet char points
for i in range(N):
    # getting curr char point no.
    N_curr = fid["N"].iloc[i]

    # checking if the current point is on a wall
    if N_curr in pnts_bottom:
```

```python
            fid["K1"].iloc[i] = 0
            fid["K2"].iloc[i] = fid["nu"].iloc[i] - fid["theta"].iloc[i]
        elif N_curr in pnts_top:
            fid["K1"].iloc[i] = fid["nu"].iloc[i] + fid["theta"].iloc[i]
            fid["K2"].iloc[i] = 0
        else:
            fid["K1"].iloc[i] = fid["nu"].iloc[i] + fid["theta"].iloc[i]
            fid["K2"].iloc[i] = fid["nu"].iloc[i] - fid["theta"].iloc[i]

# initializing x and y coordinates for inlet char points
dy = height/(N-1)
for i in range(N-1):
    fid["Y"].iloc[i+1] = fid["Y"].iloc[i] + dy
    fid["X"].iloc[i+1] = 0

# begining loop over all internal char points
for i in range(N,N_total):
    # getting current char point id
    N_curr = fid["N"].iloc[i]

    # checking if it is on bottom wall
    if N_curr in pnts_bottom:
        # it has only K1 characteristics and theta as theta_lower
        theta = theta_lower

        # getting index of dependence node
        n1 = fid["N1"].iloc[i] - 1

        # getting K1 value
        K1 = fid["K1"].iloc[n1]

        # computing nu from K1 characteristics: nu + theta = K1
        nu = K1 - theta

        # computing Mach number for the given nu value
        M = inv_PM(nu)

        # computing Mach angle
        mu = np.arcsin(1/M)

        # computing K2
        K2 = nu - theta

        # computing slope S1 for K1 characteristics
        S1 = np.tan(fid["theta"].iloc[n1]-fid["mu"].iloc[n1])

        # computing the x position of current char pnt
        x = (fid["Y"].iloc[n1] - fid["X"].iloc[n1]*S1)/(np.tan(theta_lower)-S1)

        # computing the y position of current char pnt
        y = x*np.tan(theta_lower)

    # checking if it is on top wall
    elif N_curr in pnts_top:
        # it has only K2 characteristics and theta as theta_upper
        theta = theta_upper

        # getting index of dependence node
        n2 = fid["N2"].iloc[i] - 1

        # getting K2 value
        K2 = fid["K2"].iloc[n2]

        # computing nu from K2 characteristics: nu - theta = K2
        nu = K2 + theta

        # computing Mach number for the given nu value
        M = inv_PM(nu)
```

```
        # computing Mach angle
        mu = np.arcsin(1/M)

        # computing K1
        K1 = nu + theta

        # computing slope S2 for K2 characteristics
        S2 = np.tan(fid["theta"].iloc[n2]+fid["mu"].iloc[n2])

        # computing the x position of current char pnt
        x = (fid["Y"].iloc[n2] - fid["X"].iloc[n2]*S2 - height)/(np.tan(theta_upper)-S2)

        # computing the y position of current char pnt
        y = x*np.tan(theta_upper) + height


    # working on internal char points
    else:
        # it has both K1 and K2 characteristics

        # getting index of dependence nodes
        n1 = fid["N1"].iloc[i] - 1
        n2 = fid["N2"].iloc[i] - 1

        # getting K1 and K2 value
        K1 = fid["K1"].iloc[n2]
        K2 = fid["K2"].iloc[n1]

        # computing nu and theta
        nu = (K1+K2)/2
        theta = (K1-K2)/2

        # computing Mach number for the given nu value
        M = inv_PM(nu)

        # computing Mach angle
        mu = np.arcsin(1/M)

        # computing slope S1 for K1 characteristics
        S1 = (np.tan(theta + mu) +
                np.tan(fid["theta"].iloc[n1]+fid["mu"].iloc[n1]))/2

        # computing slope S2 for K2 characteristics
        S2 = (np.tan(theta - mu) +
                np.tan(fid["theta"].iloc[n2]-fid["mu"].iloc[n2]))/2

        # computing the x position of current char pnt
        x = ((S2*fid["X"].iloc[n2] - S1*fid["X"].iloc[n1]) +
                (fid["Y"].iloc[n1]-fid["Y"].iloc[n2]))/(S2-S1)

        # computing the y position of current char pnt
        y = fid["Y"].iloc[n1] + (x - fid["X"].iloc[n1])*S1

    # updating the data frame
    fid["nu"].iloc[i] = nu
    fid["M"].iloc[i] = M
    fid["mu"].iloc[i] = mu
    fid["theta"].iloc[i] = theta
    fid["X"].iloc[i] = x
    fid["Y"].iloc[i] = y
    fid["K1"].iloc[i] = K1
    fid["K2"].iloc[i] = K2


print(fid)

fid.to_csv("computation_data.csv", index = None)
```

```python

# characteristics plot--------------------------------------------------------------

plt.figure()
for i in range(N,N_total):
    # getting current char number
    N_curr = fid["N"].iloc[i]

    # checking if it is a wall pnt or not
    if N_curr in pnts_bottom:
        # getting dependence char point index
        n1 = fid["N1"].iloc[i] - 1

        # getting coordinates
        x1 = fid["X"].iloc[i]
        y1 = fid["Y"].iloc[i]
        x2 = fid["X"].iloc[n1]
        y2 = fid["Y"].iloc[n1]

        # plotting line
        plt.plot([x1,x2],[y1,y2],'--',color='tab:purple')
        #  plt.plot(x1,y1,'o',color='tab:purple')
    elif N_curr in pnts_top:
        # getting dependence char point index
        n2 = fid["N2"].iloc[i] - 1

        # getting coordinates
        x1 = fid["X"].iloc[i]
        y1 = fid["Y"].iloc[i]
        x2 = fid["X"].iloc[n2]
        y2 = fid["Y"].iloc[n2]

        # plotting line
        plt.plot([x1,x2],[y1,y2],'--',color='tab:purple')
        #  plt.plot(x1,y1,'o',color='tab:purple')
    else: #  internal point
        # getting dependence char point index
        n1 = fid["N1"].iloc[i] - 1
        n2 = fid["N2"].iloc[i] - 1

        # getting coordinates
        x1 = fid["X"].iloc[i]
        y1 = fid["Y"].iloc[i]
        x2 = fid["X"].iloc[n1]
        y2 = fid["Y"].iloc[n1]
        x3 = fid["X"].iloc[n2]
        y3 = fid["Y"].iloc[n2]

        # plotting line
        plt.plot([x1,x2],[y1,y2],'--',color='tab:purple')
        plt.plot([x1,x3],[y1,y3],'--',color='tab:purple')
        #  plt.plot(x1,y1,'o',color='tab:purple')
# for legend purpose
plt.plot([x1,x2],[y1,y2],'--',color='tab:purple',label='char.lines')

# plotting border wall lines
X_lower = fid["X"].iloc[np.array(pnts_bottom)-1]
Y_lower = fid["Y"].iloc[np.array(pnts_bottom)-1]
X_upper = fid["X"].iloc[np.array(pnts_top)-1]
Y_upper = fid["Y"].iloc[np.array(pnts_top)-1]
plt.plot(X_lower,Y_lower,'-k',linewidth=2,label='wall')
plt.plot(X_upper,Y_upper,'-k',linewidth=2)

plt.grid()
plt.xlabel("X")
plt.ylabel("Y")
plt.legend(bbox_to_anchor=(1,1))
plt.axis("image")
```

```
plt.title("Characteristics lines plot")
plt.savefig("char_map.png", dpi = 150, bbox_inches = "tight")

# Mach number contour plot
plt.figure()
#   plt.tricontourf(fid["X"],fid["Y"],fid["M"],100, cmap='Purples')
plt.tricontourf(fid["X"],fid["Y"],fid["M"],100, cmap='jet')
plt.axis("image")
plt.colorbar(location='bottom')
plt.plot(X_lower,Y_lower,'--k',linewidth=2)
plt.plot(X_upper,Y_upper,'--k',linewidth=2)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Mach number contour")
plt.savefig("M_contour.png", dpi = 150)

plt.show()
```

****