AE 330 Assignment 2

Name : V S Harikrishna

Roll no : 160010054

Problem statement :

1. Write your own python code to design a Rao-nozzle
2. Use ANSYS FLUENT/OpenFOAM to compare the performance of a Conical Nozzle and a Rao Nozzle for optimum expansion conditions inviscid simulations
3. Consider two nozzle lengths : 60% and 80% the length of an equivalent conical nozzle

   Need 1-D plots of :
   a) p, T, M vs x
   b) M, p vs A/At

   Need 2D contour plots of M
4. Are the contours along the radial direction or, is there any curvature?
5. What happens if you increase the back pressure now? Try one or two cases.

1. Python code to generate Rao nozzle.

Implementing a python class 'Bellnozzle( )' with a method to generate nozzle geometry as shown below:

In [1]:
```
import numpy as np
from math import pi
import math
from bisect import bisect_left
import matplotlib.pyplot as plt
```

I will use the below figure for reference.

Nozzle Figure

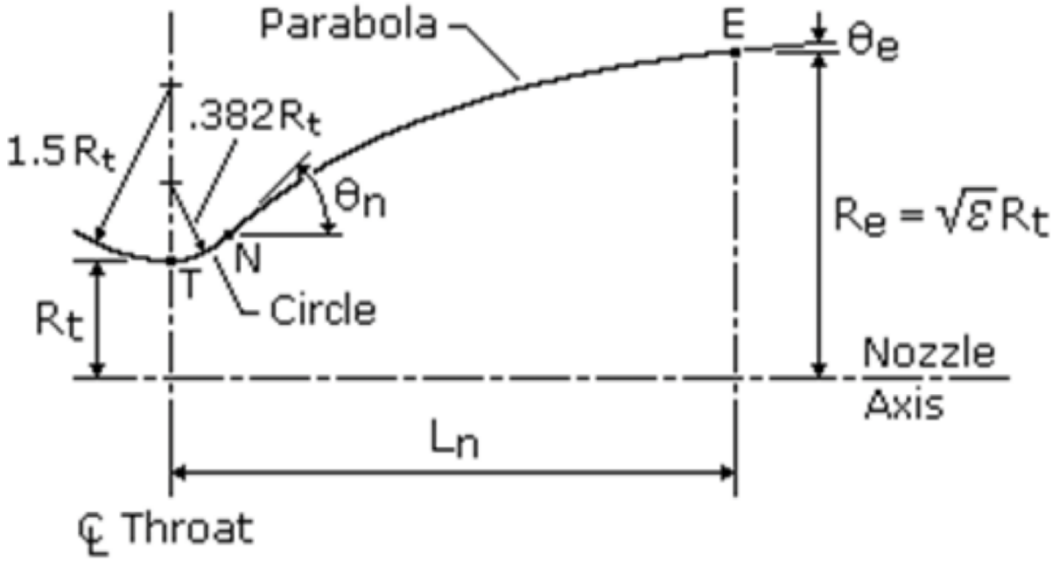


Figure 1: Design of Rao's bell nozzle contour (Sreenath and Mubarak, 2016)

We use the method specified in the given paper here

The following method is used to calculate nozzle geometry:

The user shall specify the following input:

1. Throat radius (R_t)
2. Area ratio of exit to throat (ϵ or e)
3. Length percentage of equivalent conical nozzle (K)
4. Slope angle of tangent at exit i.e. flow deflection angle at exit (θ_e)
5. Slope angle of tangent at point N shown in figure (θ_n)

Calculation

We use the following relations to calculate the curve for nozzle half section:

1. Length of bell section given by:

$$L_n = \frac{K(\sqrt{\epsilon}-1)R_t}{tan(\theta_e)}$$

1. Exit radius R_e given by:

$$R_e = \sqrt{\epsilon}R_t$$

1. To find coordinates of nozzle half-section before throat, we use the relation for x and y on a circle in polar form as follows, where θ_1 is as shown in the figure going from 135^0 to 90^0:

$$x_1 = 1.5R_t cos(\theta_1)$$

$$y_1 = 2.5R_t - 1.5R_t sin(\theta_1)$$

1. To find coordinates of nozzle half-section after throat till N, we use the relation for x and y on a circle in polar form as follows, where θ_2 is as shown in the figure going from 90^0 to $90^0 - \theta_n$:

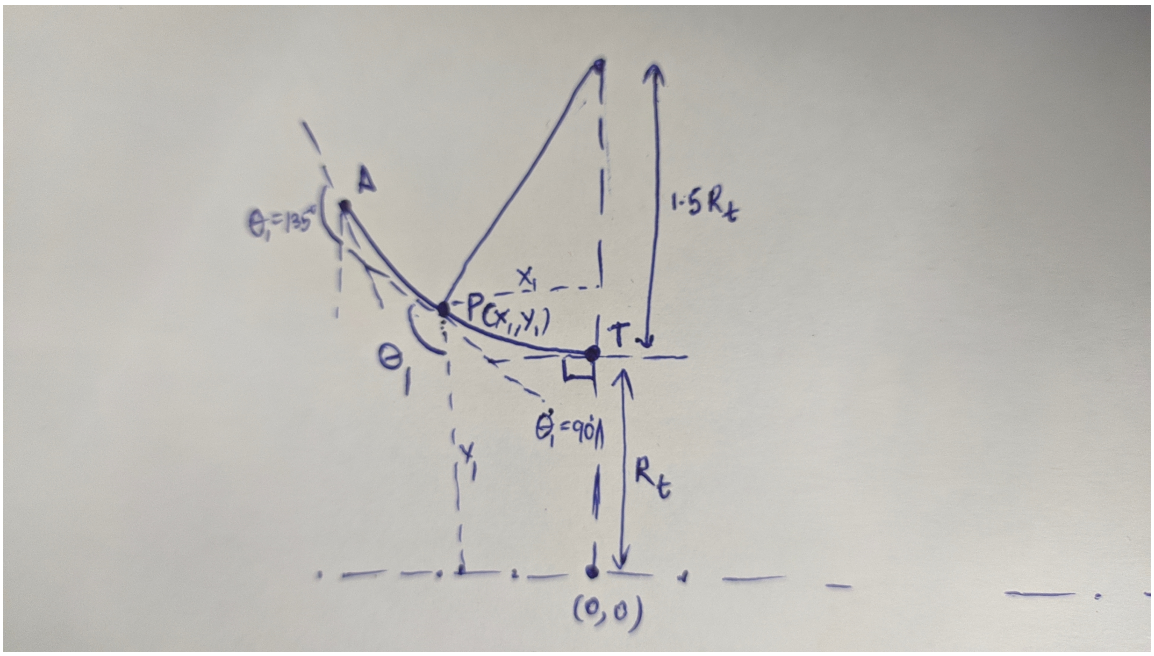
$$x_2 = 0.382R_t cos(\theta_2)$$

$$y_2 = 1.382R_t - 0.382R_t sin(\theta_2)$$

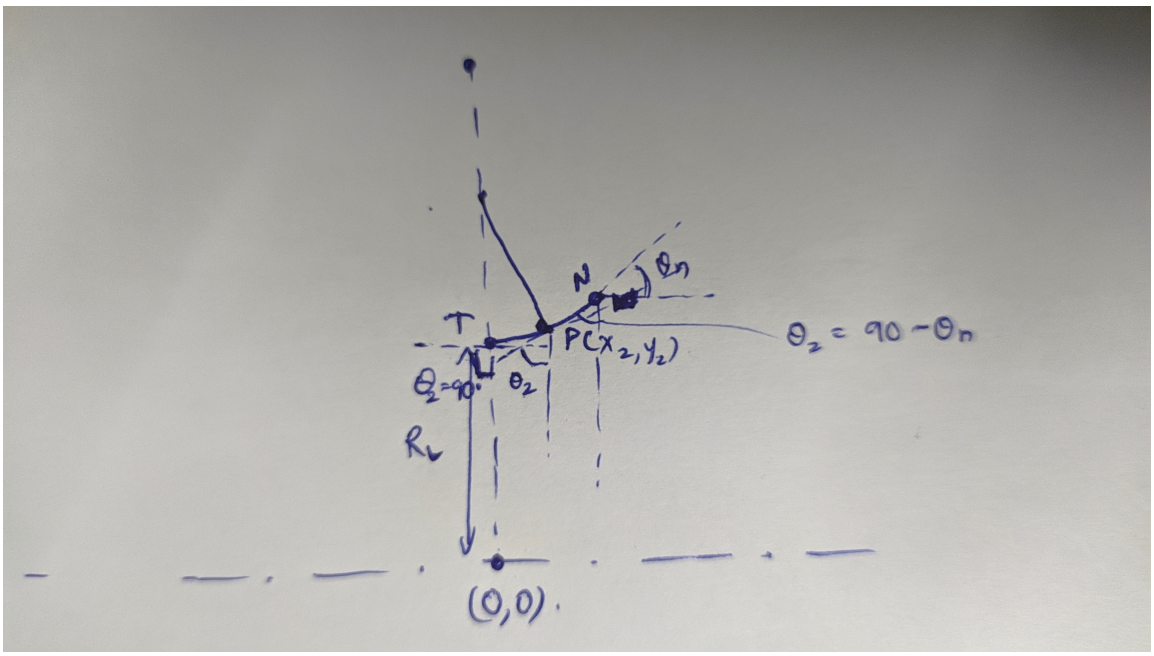
1. To find coordinates of the bell section of the nozzle N, we use the relation for x and y on a parabola as follows (where a, b, c are constants to be solved for):

$$x_b = ay_b^2 + by_b + c$$

θ_1:



θ_2 :



Class defined as below :

In [2]:

```
# Define a class to generate a distribution of x and y coordinates for the
# tranverse half-section of a nozzle with origin at throat where x axis
# is along the nozzle's axis and y'axis is perpendicular to the nozzle axis
# (radial) :
class Bellnozzle (object) :

  # Initializing the class with input parameters K, e, R_t, theta_e and thet
  # where K : fraction of length of equivqlent conical nozzle
  #       e : ratio of exit area to throat area
  #     R_t : throat radius
  # theta_e : slope angle of tangent at exit (i.e. flow deflection angle at
  # theta_n : slope angle of tangent at point N shown in figure
  def __init__(self, K, e, R_t, theta_e):

    self.K = K
    self.e = e
    self.R_t = R_t
    self.theta_e = theta_e
    # The nozzle exit radius R_e is given as:

    #                         R_e = sqrt(e)R_t

    self.R_e = np.sqrt(e)*self.R_t

    # The length of the bell section of nozzle L_n is given as:

    #                         K(sqrt(e)-1)R_t
    #                 L_n = --------------
    #                          tan(theta_e)

    self.L_n, self.theta_n = self.find_wall_angles(self.e, self.R_t, 100*sel

  def find_wall_angles(self, ar, Rt, l_percent = 80 ):
    # wall-angle empirical data
    aratio              = [ 4,    5,    10,   20,   30,   40,   50,   100]
    theta_n_60  = [20.5, 20.5, 16.0, 14.5, 14.0, 13.5, 13.0, 11.2]
    theta_n_80  = [21.5, 23.0, 26.3, 28.8, 30.0, 31.0, 31.5, 33.5]
    theta_n_90  = [20.0, 21.0, 24.0, 27.0, 28.5, 29.5, 30.2, 32.0]









    # nozzle length
    f1 = ( (math.sqrt(ar) - 1) * Rt )/ math.tan(math.radians(15) )

    if l_percent == 60:
      theta_n = theta_n_60
      Ln = 0.6 * f1
    elif l_percent == 80:
      theta_n = theta_n_80
      Ln = 0.8 * f1
    elif l_percent == 90:
      theta_n = theta_n_90
      Ln = 0.9 * f1
    else:
      theta_n = theta_n_80
      Ln = 0.8 * f1
```

```
    # find the nearest ar index in the aratio list
    x_index, x_val = self.find_nearest(aratio, ar)
    # if the value at the index is close to input, return it
    if round(aratio[x_index], 1) == round(ar, 1):
      return Ln, math.radians(theta_n[x_index])

    # check where the index lies, and slice accordingly
    if (x_index>2):
      # slice couple of middle values for interpolation
      ar_slice = aratio[x_index-2:x_index+2]
      tn_slice = theta_n[x_index-2:x_index+2]
      # find the tn_val for given ar
      tn_val = self.interpolate(ar_slice, tn_slice, ar)

    elif( (len(aratio)-x_index) <= 1):
      # slice couple of values initial for interpolation
      ar_slice = aratio[x_index-2:len(x_index)]
      tn_slice = theta_n[x_index-2:len(x_index)]

      # find the tn_val for given ar
      tn_val = self.interpolate(ar_slice, tn_slice, ar)

    else:
      # slice couple of end values for interpolation
      ar_slice = aratio[0:x_index+2]
      tn_slice = theta_n[0:x_index+2]

      # find the tn_val for given ar
      tn_val = self.interpolate(ar_slice, tn_slice, ar)


    return Ln, math.radians(tn_val)


  # Define a plotting method to plot the generated geometry (to be used late
  def plot_nozzle (self, label = "") :
      plt.figure(figsize=(10, 6))
      ax = plt.subplot(121)
      ax.set_aspect('equal')
      plt.plot(self.x_cord, self.y_cord, label = label)
      plt.ylim(-self.y_cord[-1], 2*self.y_cord[-1])
      plt.xlim(self.x_cord[0] - 0.1*self.x_cord[-1], self.x_cord[-1] + \
               0.1*self.x_cord[-1])
      axis_x = np.linspace(self.x_cord[0] - 0.1*self.x_cord[-1], \
                           self.x_cord[-1] + 0.1*self.x_cord[-1], 100)
      axis_y = np.linspace(-self.y_cord[-1], 2*self.y_cord[-1], 100)
      plt.plot(axis_x,np.zeros_like(axis_x), color = "black")
      plt.plot(np.zeros_like(axis_y), axis_y,color = "black")
      plt.xlabel("x (mm)")
      plt.ylabel("y (mm)")
      plt.grid(True)
      plt.title("Rao's bell nozzle contour for e = {}, K = {}".format(self.e
      plt.show()

    # find the nearest index in the list for the given value
  def find_nearest(self, array, value):
      array = np.asarray(array)
      idx = (np.abs(array - value)).argmin()
      return idx, array[idx]
```

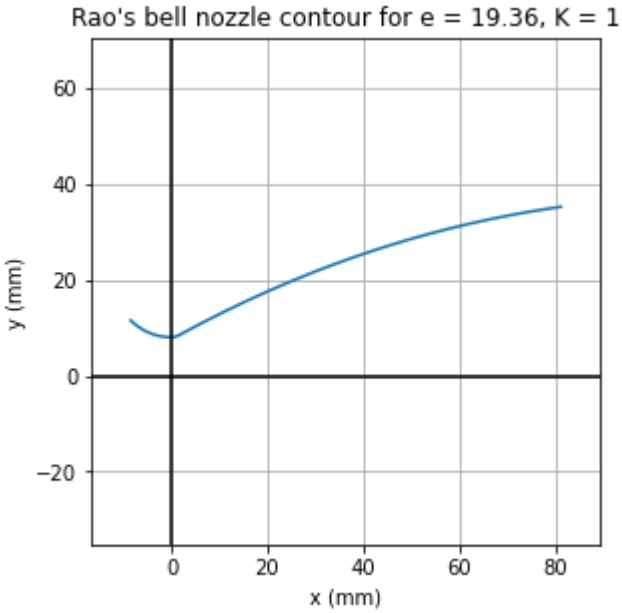
```
  # simple linear interpolation
def interpolate(self, x_list, y_list, x):
  if any(y - x <= 0 for x, y in zip(x_list, x_list[1:])):
    raise ValueError("x_list must be in strictly ascending order!")
  intervals = zip(x_list, x_list[1:], y_list, y_list[1:])
  slopes = [(y2 - y1) / (x2 - x1) for x1, x2, y1, y2 in intervals]

  if x <= x_list[0]:
    return y_list[0]
  elif x >= x_list[-1]:
    return y_list[-1]
  else:
    i = bisect_left(x_list, x) - 1
    return y_list[i] + slopes[i] * (x - x_list[i])
  # Define a method to generate geometry. Takes in an argument 'plot' as a
  # string of "Yes" or "No". This argument decides whether to output a plo
  # 2 numpy arrays of x and y coordinates respectively.


def generatenozzle(self, plot = "No", label = ""):

  self.plot = plot
  self.label = label

  #for section before throat:
  self.x_1 = np.array([])
  self.y_1 = np.array([])
  theta_1_entrance = 135*pi/180
  theta_1_end = pi/2
  theta_1 = np.linspace(theta_1_entrance, theta_1_end, 100)

  for i in theta_1:
    self.x_1 = np.append(self.x_1, 1.5 * self.R_t * np.cos(i))
    self.y_1 = np.append(self.y_1, 2.5 * self.R_t - \
                         1.5 * self.R_t * np.sin(i))
    # end of for loop



  #for section after throat:
  self.x_2 = np.array([])
  self.y_2 = np.array([])
  theta_2_entrance = pi/2
  theta_2_end = pi/2 - self.theta_n
  theta_2 = np.linspace(theta_2_entrance, theta_2_end, 10)

  for i in theta_2:
    self.x_2 = np.append(self.x_2, 0.382 * self.R_t * np.cos(i))
    self.y_2 = np.append(self.y_2, 1.382 * self.R_t - \
                         0.382 * self.R_t * np.sin(i))
    # end of for loop



  # for section after N :


    # To define the parabola of the form

    #                          x = ay^2 + by + c
```

```
    #  we need to find constants a, b and c.

    # To find a, b, c, we use the following 3 equations:

    # 1) x_n and y_n must satisfy the equation of parabola of bell section
    #    at N:

    #    i.e.              x_n = ay_n^2 + by_n + c

    # 2) slope of tangent of parabola of bell section at N must be the tan
    #    function of theta_n:

    #    i.e                              1
    #             tan(theta_n) =    -----------
    #                               (2ay_n + b)




    # 3) slope of tangent of parabola of bell section at exit (E) must be
    #    the tangent function of theta_e:

    #    i.e                              1
    #             tan(theta_e) =    -----------
    #                               (2ay_e + b)

    # But first we need the values coordinates at point N and exit (E):
  self.x_n = 0.382 * self.R_t * np.cos(pi/2 - self.theta_n)
  self.y_n = 1.382 * self.R_t - 0.382 * self.R_t * np.sin(pi/2 - self.thet



  self.x_e = self.L_n    # since L_n is defined as such

  self.y_e = self.R_e    # since at exit, y will be equal to exit radius R

  # using Bezier curve to generate parabolic section:

  # gradient m1,m2 - [Eqn. 8]
  self.m1 = math.tan(self.theta_n);  self.m2 = math.tan(self.theta_e);
  # intercept - [Eqn. 9]
  self.C1 = self.y_n - self.m1*self.x_n;  self.C2 = self.y_e - self.m2*sel
  # intersection of these two lines (at point Q)-[Eqn.10]
  self.Qx = (self.C2 - self.C1)/(self.m1 - self.m2)
  self.Qy = (self.m1*self.C2 - self.m2*self.C1)/(self.m1 - self.m2)

  # We select equally spaced divisions between 0 and 1
  # The bell is a quadratic Bezier curve, which has equations:
  # x(t) = (1 - t)^2 * Nx + 2(1 - t)t * Qx + t^2 * Ex, 0≤t≤1
  # y(t) = (1 - t)^2 * Ny + 2(1 - t)t * Qy + t^2 * Ey, 0≤t≤1 [Eqn. 6]

  int_list = np.linspace(0, 1, 100)
  self.x_b = []; self.y_b = [];
  for t in int_list:
    self.x_b.append( ((1-t)**2)*self.x_n + 2*(1-t)*t*self.Qx + (t**2)*self
    self.y_b.append( ((1-t)**2)*self.y_n + 2*(1-t)*t*self.Qy + (t**2)*self


  # Finally, we combine all three sections to create a single array of
  # x and y coordinates:
  self.x_cord = np.array([])
  self.y_cord = np.array([])
```

```
    self.x_cord = np.append(self.x_cord, self.x_1)
    self.x_cord = np.append(self.x_cord, self.x_2)
    self.x_cord = np.append(self.x_cord, self.x_b)

    self.y_cord = np.append(self.y_cord, self.y_1)
    self.y_cord = np.append(self.y_cord, self.y_2)
    self.y_cord = np.append(self.y_cord, self.y_b)

    # plotting output controlled by argument 'plot' provided by user
    if self.plot == "Yes" :

      return self.plot_nozzle(self.label)



    # returning values of x and y coordinates
    return  self.x_cord, self.y_cord
```

1. We use the above class to generate a nozzle geometry and then use the plot attribute in it to create a plot of the half - section of the nozzle. The class Bellnozzle() takes in the user specified inputs of K, $e(\varepsilon)$, R_t, θ_e and θ_n and returns the x and y coordinates of the nozzle half-section with x along nozzle axis and y along the radial direction as separate 1D numpy arrays.</p> An example case with $K = 1$, $\varepsilon = 19.36$, $R_t = 8mm$, $\theta_e = 8^0$ and $\theta_n = 15^0$ is shown :

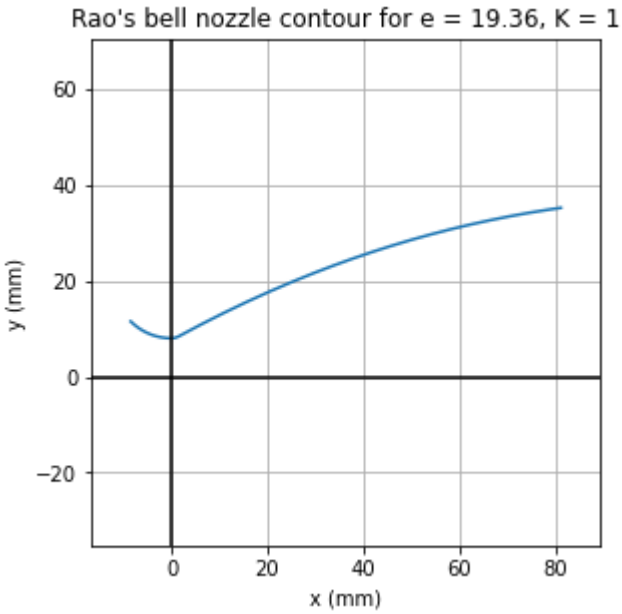
In [3]:
```
Nozzle1 = Bellnozzle(K = 1, e = 19.36, R_t = 8, theta_e = 8*pi/180)

x, y = Nozzle1.generatenozzle()

plt.figure(figsize=(10, 6))
ax = plt.subplot(121)
ax.set_aspect('equal')
plt.plot(x, y)
plt.ylim(-y[-1], 2*y[-1])
plt.xlim(x[0] - 0.1*x[-1], x[-1] + 0.1*x[-1])
axis_x = np.linspace(x[0] - 0.1*x[-1], x[-1] + 0.1*x[-1], 1000)
axis_y = np.linspace(-y[-1], 2*y[-1], 1000)
plt.plot(axis_x,np.zeros_like(axis_x), color = "black")
plt.plot(np.zeros_like(axis_y), axis_y,color = "black")
plt.xlabel("x (mm)")
plt.ylabel("y (mm)")
plt.grid()
plt.title("Rao's bell nozzle contour for e = {}, K = {}".format(Nozzle1.e, \
                                                                Nozzle1.K))
plt.show()
```



I have also implemented a plot function directly into the generatenozzle() method of the which can plot the coordinates directly.

It can be accesed by setting the argument plot = "Yes" in the method generatenozzle() as shown:

In [4]:
```
Nozzle1.generatenozzle(plot = "Yes")
```

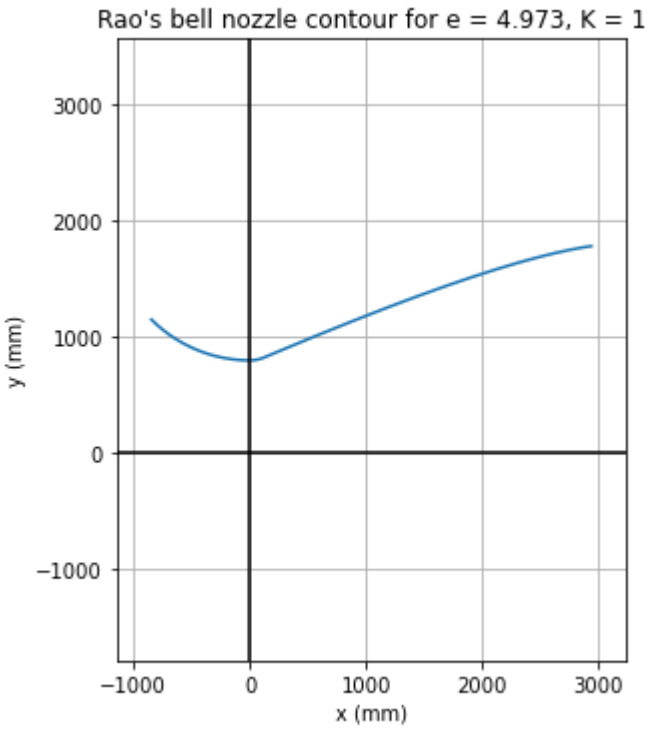


If no input is given for the argument 'plot', the function returns the x and y coordinates as shown in the first case.

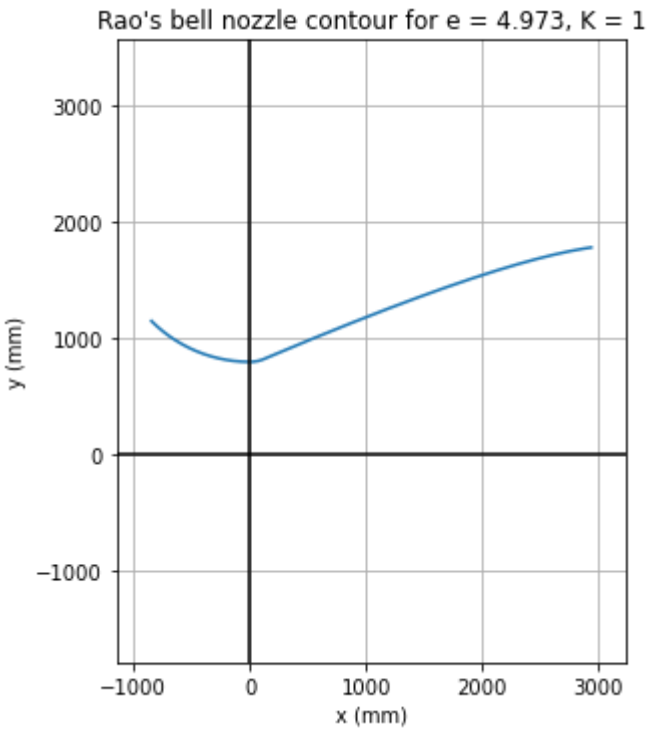
Another example case with $K = 1$, $\varepsilon = 4.973$, $R_t = 8mm$, $\theta_e = 8^0$ and $\theta_n = 15^0$ is shown :

In [5]:
```
Nozzle2 = Bellnozzle(K = 1, e = 4.973, R_t = 800, theta_e = 8*pi/180)

x, y = Nozzle2.generatenozzle()
plt.figure(figsize=(10, 6))
ax = plt.subplot(121)
ax.set_aspect('equal')
```

```
plt.plot(x,y)
plt.ylim(-y[-1], 2*y[-1])
plt.xlim(x[0] - 0.1*x[-1], x[-1] + 0.1*x[-1])
axis_x = np.linspace(x[0] - 0.1*x[-1], x[-1] + 0.1*x[-1], 1000)
axis_y = np.linspace(-y[-1], 2*y[-1], 1000)
plt.plot(axis_x,np.zeros_like(axis_x), color = "black")
plt.plot(np.zeros_like(axis_y), axis_y,color = "black")
plt.xlabel("x (mm)")
plt.ylabel("y (mm)")
plt.grid()
plt.title("Rao's bell nozzle contour for e = {}, K = {}".format(Nozzle2.e, \
                                                                Nozzle2.K))
plt.show()
```

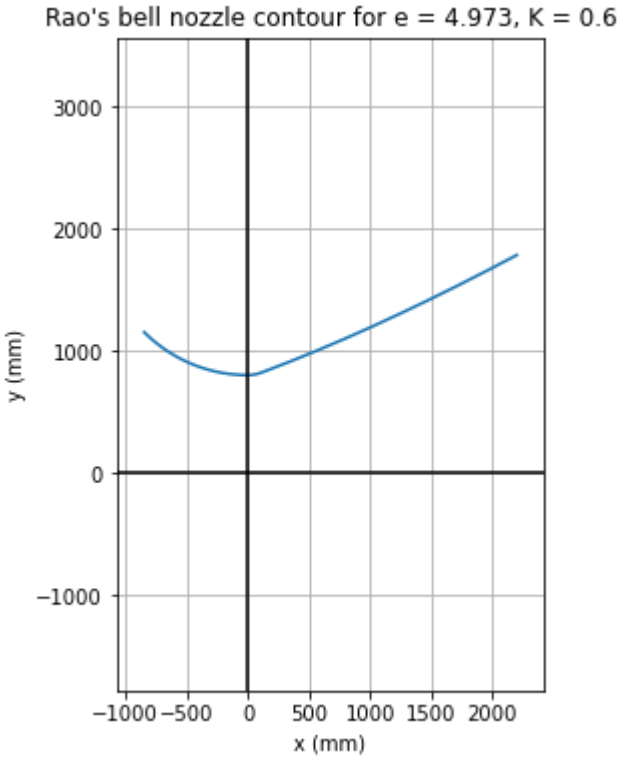


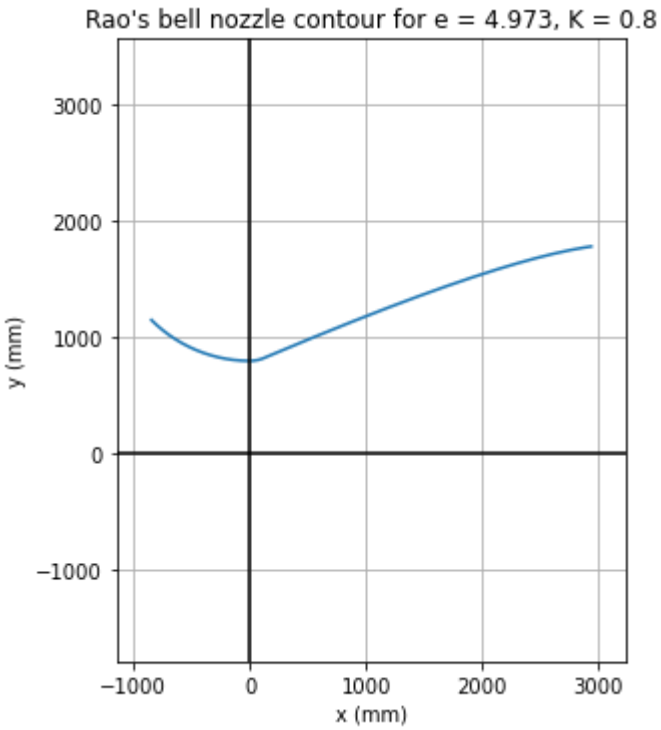
In [6]:
```
Nozzle2.generatenozzle(plot = "Yes")
```



We make 2 nozzles 'Nozzle_K_06' and 'Nozzle_K_08' for the two cases mentioned above.

In [7]:
```
Nozzle_K_06 = Bellnozzle(K = 0.6, e = 4.973, R_t = 800, theta_e = 28*pi/180)

Nozzle_K_08 = Bellnozzle(K = 0.8, e = 4.973, R_t = 800, theta_e = 8*pi/180)

Nozzle_K_06.generatenozzle(plot = "Yes")
Nozzle_K_08.generatenozzle(plot = "Yes")
```





Now to generate equivqlent conical nozzle we use the following python code.

In [8]:
```
# Define a class to generate a distribution of x and y coordinates for the
# tranverse half-section of a nozzle with origin at throat where x axis
```

```
# is along the nozzle's axis and y'axis is perpendicular to the nozzle axis
# (radial) :
class Conicalnozzle (object) :

  # Initializing the class with input parameters K, e, R_t, theta_e and thet
  # where K : fraction of length of equivqlent conical nozzle
  #       e : ratio of exit area to throat area
  #     R_t : throat radius
  # theta_e : slope angle of tangent at exit (i.e. flow deflection angle at
  # theta_n : slope angle of tangent at point N shown in figure
  def __init__(self, K, e, R_t, theta_e):


    self.e = e
    self.R_t = R_t
    self.theta_e = theta_e
    self.theta_n = math.radians(15)
    self.K = K

    self.nozzle = Bellnozzle(K = self.K, e = self.e, R_t = self.R_t, theta_e

    self.nozzle.generatenozzle()


    # The nozzle exit radius R_e is given as:

    #                         R_e = sqrt(e)R_t

    self.R_e = self.nozzle.R_e

    # The length of the bell section of nozzle L_n is given as:

    #                         K(sqrt(e)-1)R_t
    #                  L_n = --------------
    #                          tan(theta_e)

    self.L_n = self.nozzle.L_n/self.K


  # Define a plotting method to plot the generated geometry (to be used late
  def plot_nozzle (self) :
      plt.figure(figsize=(10, 6))
      ax = plt.subplot(121)
      ax.set_aspect('equal')
      plt.plot(self.x_cord, self.y_cord)
      plt.ylim(-self.y_cord[-1], 2*self.y_cord[-1])
      plt.xlim(self.x_cord[0] - 0.1*self.x_cord[-1], self.x_cord[-1] + \
               0.1*self.x_cord[-1])
      axis_x = np.linspace(self.x_cord[0] - 0.1*self.x_cord[-1], \
                           self.x_cord[-1] + 0.1*self.x_cord[-1], 100)
      axis_y = np.linspace(-self.y_cord[-1], 2*self.y_cord[-1], 100)
      plt.plot(axis_x,np.zeros_like(axis_x), color = "black")
      plt.plot(np.zeros_like(axis_y), axis_y,color = "black")
      plt.xlabel("x (mm)")
      plt.ylabel("y (mm)")
      plt.grid(True)
      plt.title("Conical nozzle contour for e = {}".format(self.e))
      plt.show()


  # Define a method to generate geometry. Takes in an argument 'plot' as a
  # string of "Yes" or "No". This argument decides whether to output a plot
  # 2 numpy arrays of x and y coordinates respectively.
  def generatenozzle(self, plot = "No"):
```

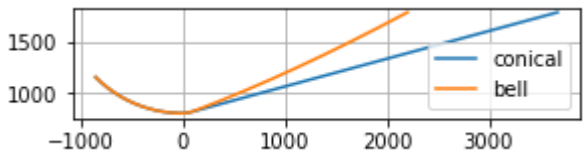
```
    self.plot = plot

    #for section before throat:
    self.x_1 = np.array([])
    self.y_1 = np.array([])
    theta_1_entrance = 135*pi/180
    theta_1_end = pi/2
    theta_1 = np.linspace(theta_1_entrance, theta_1_end, 100)

    for i in theta_1:
      self.x_1 = np.append(self.x_1, 1.5 * self.R_t * np.cos(i))
      self.y_1 = np.append(self.y_1, 2.5 * self.R_t - \
                           1.5 * self.R_t * np.sin(i))
      # end of for loop



    #for section after throat:
    self.x_2 = np.array([])
    self.y_2 = np.array([])
    theta_2_entrance = pi/2
    theta_2_end = pi/2 - self.theta_n
    theta_2 = np.linspace(theta_2_entrance, theta_2_end, 10)

    for i in theta_2:
      self.x_2 = np.append(self.x_2, 0.382 * self.R_t * np.cos(i))
      self.y_2 = np.append(self.y_2, 1.382 * self.R_t - \
                           0.382 * self.R_t * np.sin(i))
      # end of for loop



    # for section after N :


      # To define a line of the form

      #                                  y_e - y_n
      #                      y - y_n =   --------- (x - x_n)
      #                                  x_e - x_n


    self.x_n = 0.382 * self.R_t * np.cos(pi/2 - self.theta_n)
    self.y_n = 1.382 * self.R_t - 0.382 * self.R_t * np.sin(pi/2 - self.thet



    self.x_e = self.L_n   # since L_n is defined as such
    self.y_e = self.R_e    # since at exit, y will be equal to exit radius R




    int_list = np.linspace(0, 1, 100)
    self.x_c = []; self.y_c = [];
    for t in int_list:
      self.x_c.append((1-t)*self.x_n +  t*self.x_e )
      self.y_c.append((1-t)*self.y_n +  t*self.y_e )

    # Finally, we combine all three sections to create a single array of
    # x and y coordinates:
    self.x_cord = np.array([])
```

```
    self.y_cord = np.array([])

    self.x_cord = np.append(self.x_cord, self.x_1)
    self.x_cord = np.append(self.x_cord, self.x_2)
    self.x_cord = np.append(self.x_cord, self.x_c)

    self.y_cord = np.append(self.y_cord, self.y_1)
    self.y_cord = np.append(self.y_cord, self.y_2)
    self.y_cord = np.append(self.y_cord, self.y_c)

    # plotting output controlled by argument 'plot' provided by user
    if self.plot == "Yes" :

      return self.plot_nozzle()




    # returning values of x and y coordinates
    return  self.x_cord, self.y_cord
```

Generating conical nozzle and comparing with a bell nozzle with 60% length.

In [9]:
```
Nozzle_Conical = Conicalnozzle( K = 0.6, e = 4.973, R_t = 800, theta_e = 15*
Nozzle_Conical.generatenozzle()

plt.figure(figsize=(10, 6))
ax = plt.subplot(121)
ax.set_aspect('equal')
plt.plot(Nozzle_Conical.x_cord, Nozzle_Conical.y_cord, label = 'conical' )
plt.plot(Nozzle_K_06.x_cord, Nozzle_K_06.y_cord, label = 'bell')
plt.grid()
plt.legend()
```

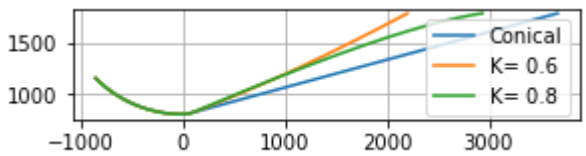
Out[9]:
```
<matplotlib.legend.Legend at 0x7f94307c4bb0>
```



Comparison of Conical nozzle with the two cases of bell nozzles (K = 0.6 and K = 0.8)

In [10]:
```
plt.figure(figsize=(10, 6))
ax = plt.subplot(121)
ax.set_aspect('equal')
plt.plot(Nozzle_Conical.x_cord, Nozzle_Conical.y_cord, label = 'Conical' )
plt.plot(Nozzle_K_06.x_cord, Nozzle_K_06.y_cord, label = 'K= 0.6' )
plt.plot(Nozzle_K_08.x_cord, Nozzle_K_08.y_cord, label = 'K= 0.8' )
plt.grid()
plt.legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x7f9448b3eeb0>
```



To do simulation in Ansys we need to export geometry data into a readable text file. This is done below. Please note that we need to run this code locally in a machine for this part of the code to work, otherwise it will return a "No such directory exists" error.

In [11]:
```
# Saving the coordinates in a .txt file

""" Please note that for the below code, one needs to provide the path to th
    local direcroty in the 'datafile_path' variable.
    Also, to save file locally, this NOTEBOOK NEEDS TO BE RUN LOCALLY as a
    Jupyter notebook"
"""

x_1, y_1 = Nozzle_K_06.generatenozzle()

data = np.column_stack([x_1, y_1])
datafile_path = "/Users/vsharikrishna/Downloads/Nozzle_06.txt"
np.savetxt(datafile_path, data)

x_2, y_2 = Nozzle_K_08.generatenozzle()

data = np.column_stack([x_2, y_2])
datafile_path = "/Users/vsharikrishna/Downloads/Nozzle_08.txt"
np.savetxt(datafile_path, data)


x_3, y_3 = Nozzle_Conical.generatenozzle()

data = np.column_stack([x_3, y_3])
datafile_path = "/Users/vsharikrishna/Downloads/Nozzle_conical.txt"
np.savetxt(datafile_path, data)
```

2. Simulation in Ansys

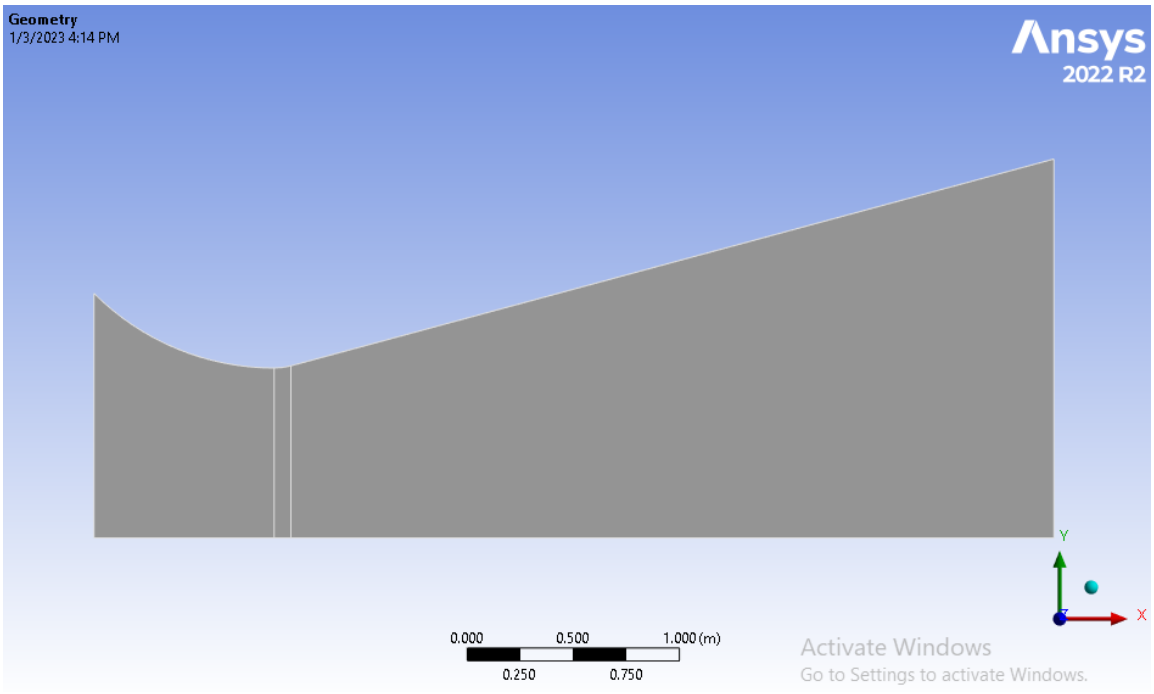
We use Ansys Fluent to simulate nozzle flow and get results. The mesh is generated using the data exported from above. The details of the simulation are given below.

Geometry Details

We have the following images showing the three geometries exported from above.

a) Conical nozzle

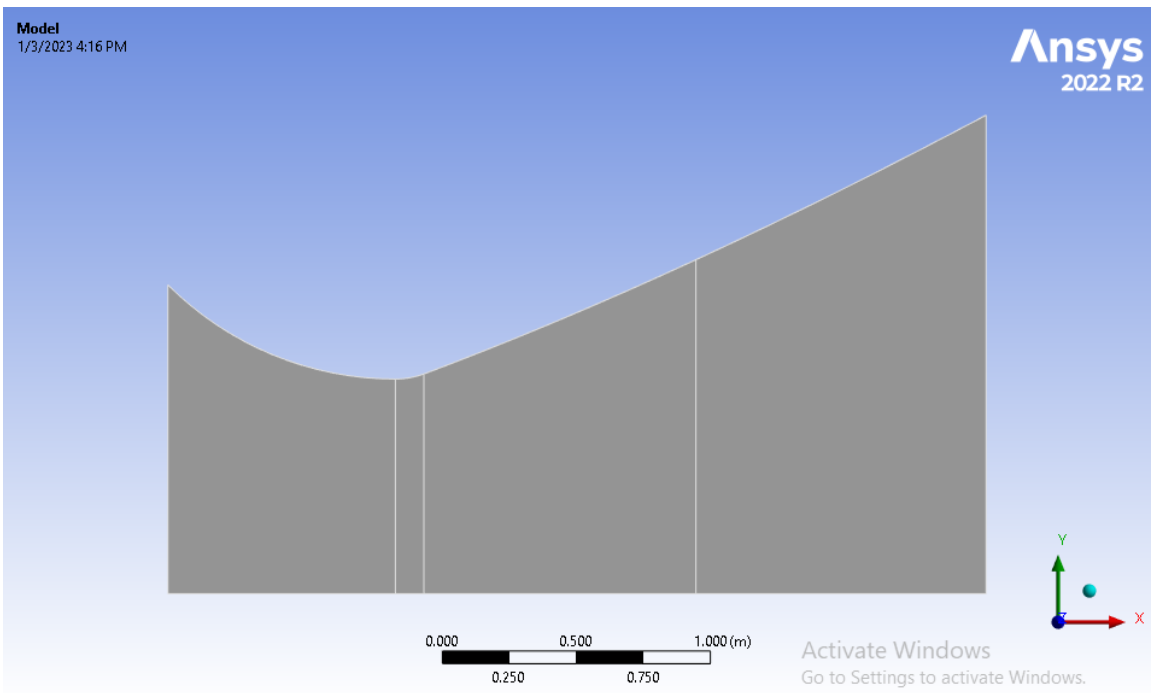
> The geometry is divided into three zones as shown below.



> Length (L_n) = 3.67240 m
>
> Throat area (A_t) = 2.01062 m^2
>
> Exit angle (θ_e) = 15^0

b) Rao nozzle with K = 0.6

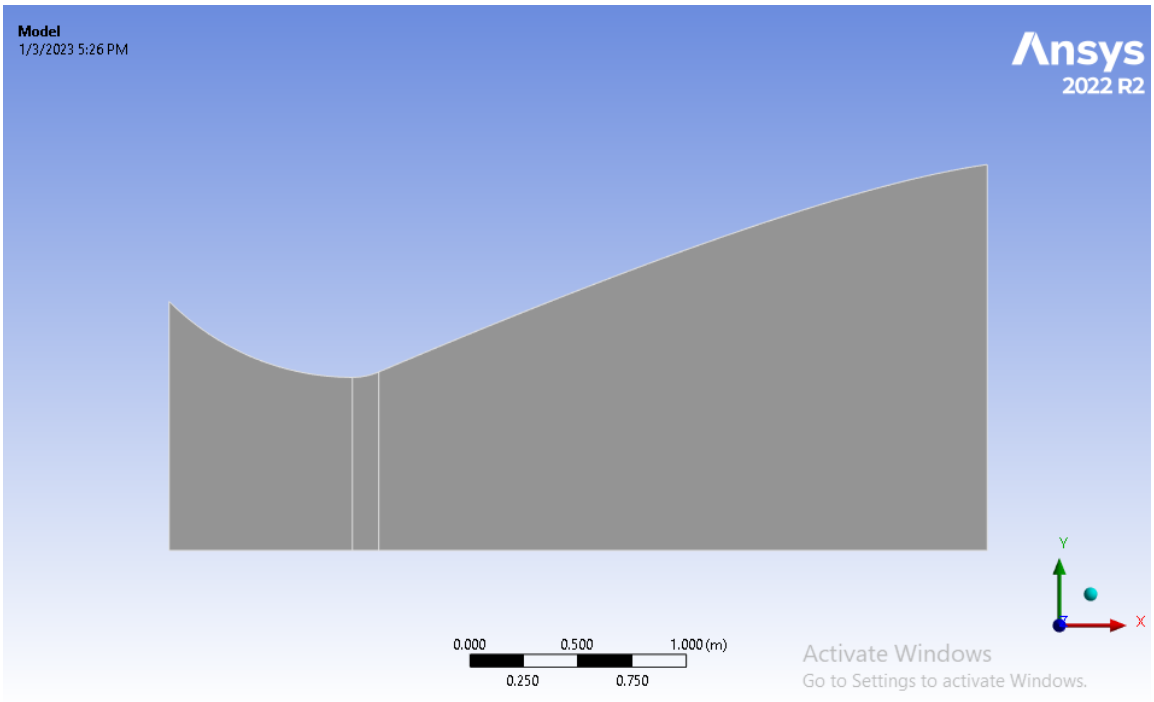
> The geometry is divided into four zones as shown below.



> Length (L_n) = 2.20344 m
>
> Throat area (A_t) = 2.01062 m^2
>
> Exit angle (θ_e) = 28^0

c) Rao nozzle with K = 0.8

> The geometry is divided into three zones as shown below.



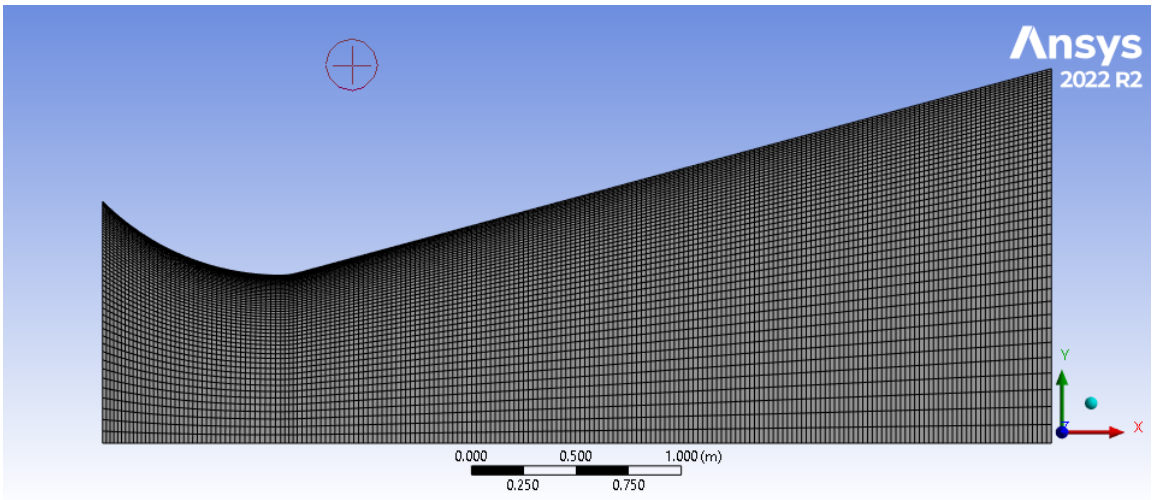
> Length (L_n) = 2.93792 m
>
> Throat area (A_t) = 2.01062 m^2
>
> Exit angle (θ_e) = 8^0

Mesh Details

The mesh has been generated using Fluent's native meshing software

a) Conical nozzle

> The mesh is as shown below



> Sizing:
>
> 47 divisions radially with bias towards walls (bias factor of 8)
>
> 204 divisions (40 + 4 + 160) axially with no bias.

> The mesh metrics are as shown:
>
> Element Quality

Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Element Quality
Min	0.33058
Max	0.99712
Average	0.7491
Standard Deviation	0.13613

> Aspect Ratio

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Aspect Ratio
Min	1.0015
Max	4.7213
Average	1.9594
Standard Deviation	0.7702

> Skewness

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Skewness
Min	8.8746e-004
Max	0.49401
Average	0.1252
Standard Deviation	7.5314e-002

> Orthogonal Quality

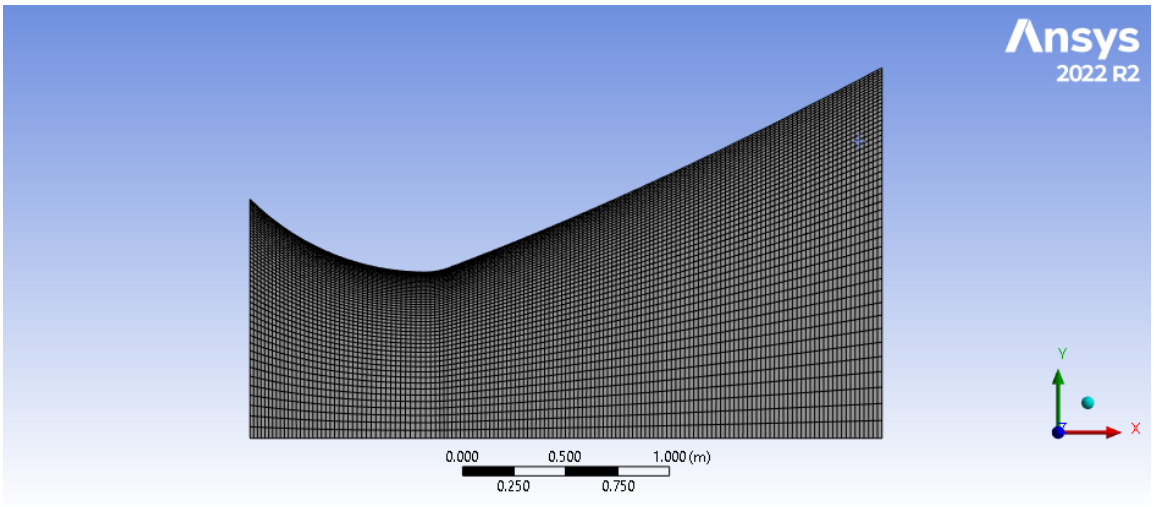
Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Orthogonal Quality
Min	0.71219
Max	1.
Average	0.97411
Standard Deviation	3.5087e-002

> Mesh statistics:

Statistics	
Nodes	9840
Elements	9588

b) Rao nozzle with K = 0.6

> The mesh is as shown below



> Sizing:
>
> 47 divisions radially with bias towards walls (bias factor of 8)
>
> 145 divisions (40 + 5 + 50 + 50) axially with no bias.
>
> The mesh metrics are as shown:
>
> Element Quality

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Element Quality
Min	0.35119
Max	0.99713
Average	0.71523
Standard Deviation	0.12571

> Aspect Ratio

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Aspect Ratio
Min	1.0003
Max	4.2963
Average	1.953
Standard Deviation	0.72054

> Skewness

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Skewness
Min	8.5058e-004
Max	0.49401
Average	0.17927
Standard Deviation	9.605e-002

> Orthogonal Quality

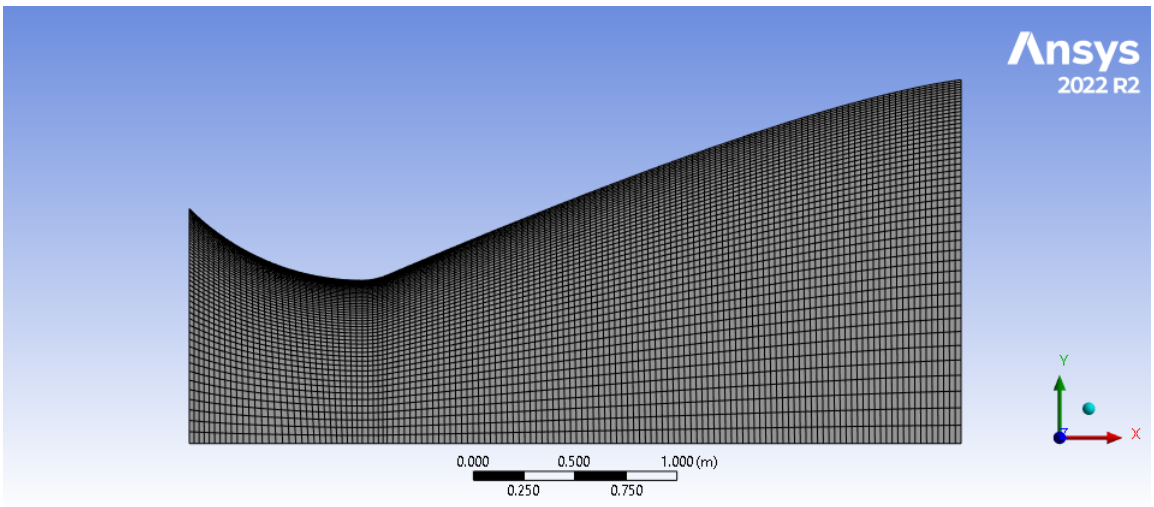
Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Orthogonal Quality
Min	0.71276
Max	1.
Average	0.95003
Standard Deviation	4.4323e-002

> Mesh statistics:

Statistics	
Nodes	7008
Elements	6815

c) Rao nozzle with K = 0.8

> The mesh is as shown below



> Sizing:
>
> 47 divisions radially with bias towards walls (bias factor of 8)
>
> 135 divisions (30 + 5 + 100) axially with no bias.
>
> The mesh metrics are as shown:
>
> Element Quality

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Element Quality
Min	0.26673
Max	0.99178
Average	0.72036
Standard Deviation	0.1698

> Aspect Ratio

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Aspect Ratio
Min	1.0006
Max	6.2618
Average	2.1507
Standard Deviation	1.0794

> Skewness

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Skewness
Min	2.3532e-003
Max	0.49215
Average	0.14635
Standard Deviation	8.456e-002

> Orthogonal Quality

Quality	
Check Mesh Quality	Yes, Errors
Target Skewness	Default (0.9)
Smoothing	Medium
Mesh Metric	Orthogonal Quality
Min	0.7111
Max	1.
Average	0.96532
Standard Deviation	3.9224e-002

> Mesh statistics:

Statistics	
Nodes	6528
Elements	6345

Fluent Solution

In Fluent we solve for 3 cases:

> 1. Optimum expansion condition with atmosperic conditions at exit (P = 1 atm, T = 300 K)
> 2. Higher back pressure with pressure = 2 atm and T = 300 K
> 3. Higher back pressure with pressure = 21.026 atm and T = 300 K

Case 1 : Optimum Expansion Condition

Boundary conditions

The boundary conditions used were as follows:

> Inlet : Pressure inlet
>
> > Total pressure = 4786224.74 Pa
> >
> > Total temperature = 902.58 K
>
> Outlet : Pressure outlet
>
> > Total pressure = 101325 Pa
> >
> > Total temperature = 300 K

Case 2 : Higher Back Pressure (2 atm)

Boundary conditions

The boundary conditions used were as follows:

> Inlet : Pressure inlet
>
> > Total pressure = 4786224.74 Pa
> >
> > Total temperature = 902.58 K
>
> Outlet : Pressure outlet
>
> > Total pressure = 202650 Pa
> >
> > Total temperature = 300 K

Case 3 : Higher Back Pressure (2 atm)

Boundary conditions

The boundary conditions used were as follows:

> Inlet : Pressure inlet
>
> > Total pressure = 4786224.74 Pa
> >
> > Total temperature = 902.58 K

> Outlet : Pressure outlet
>
> > Total pressure = 2130459.94 Pa
> >
> > Total temperature = 300 K

Note that the above values are used such that case 1 should form a perfectly expanded flow at exit, case 2 should form an underexpanded flow at the exit with oblique shocks at exit, and case 3 should form a normal shock at the location where area ratio is 3.0073.

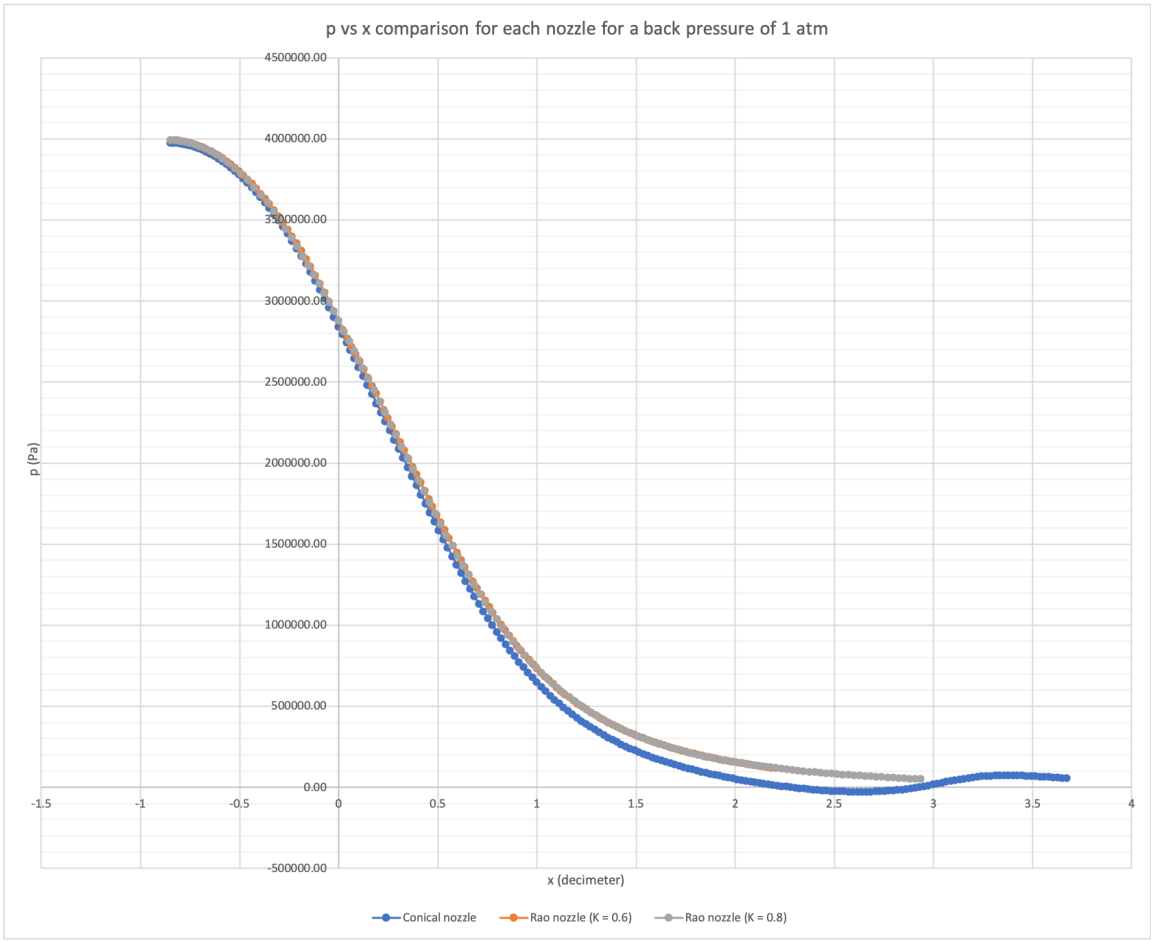
Solver details

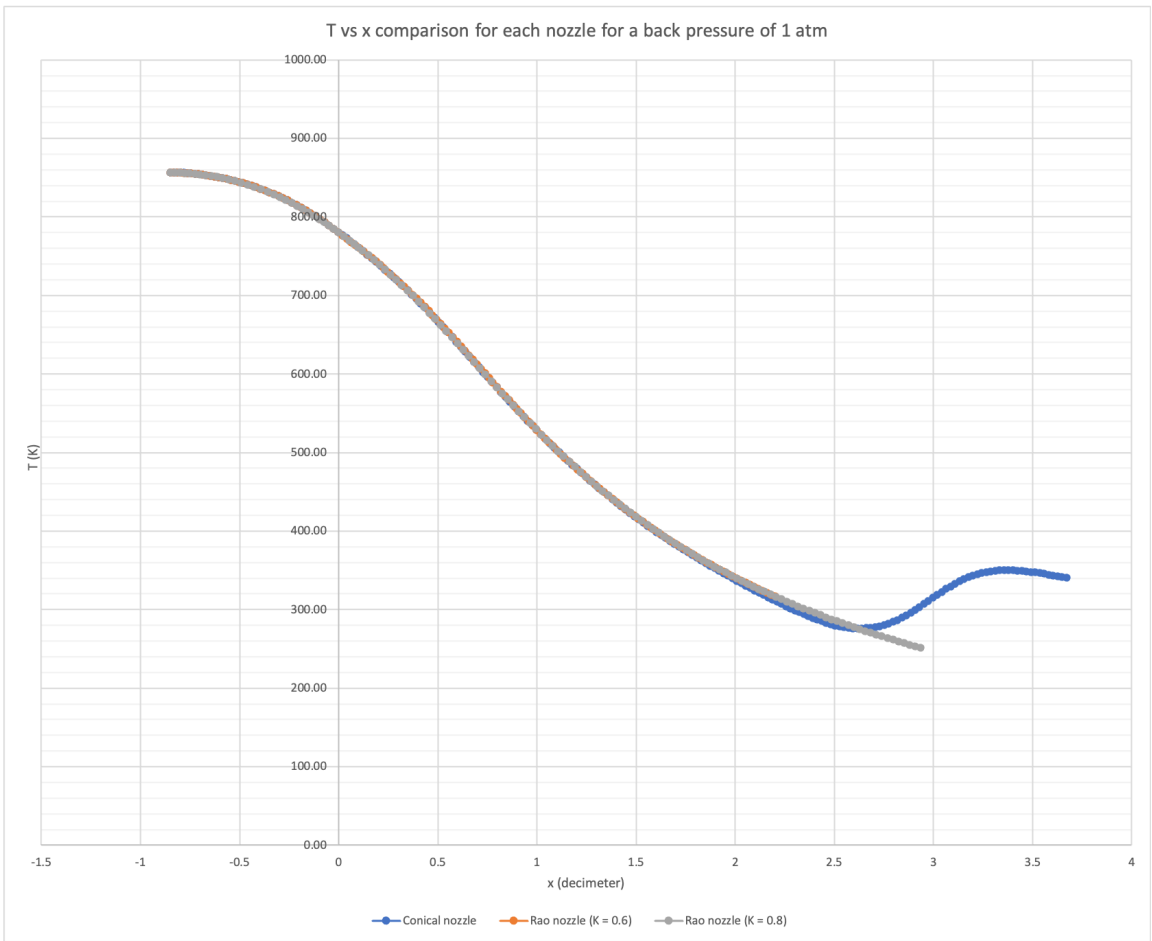
> Density Based
>
> Axisymmetric
>
> Energy : ON
>
> Viscosity : Inviscid
>
> Solution method : Roe FDS
>
> Solution scheme : 2nd order upwind (First order Upwind in case 3)
>
> Residuals : 10^{-9} (10^{-6} for case 3)

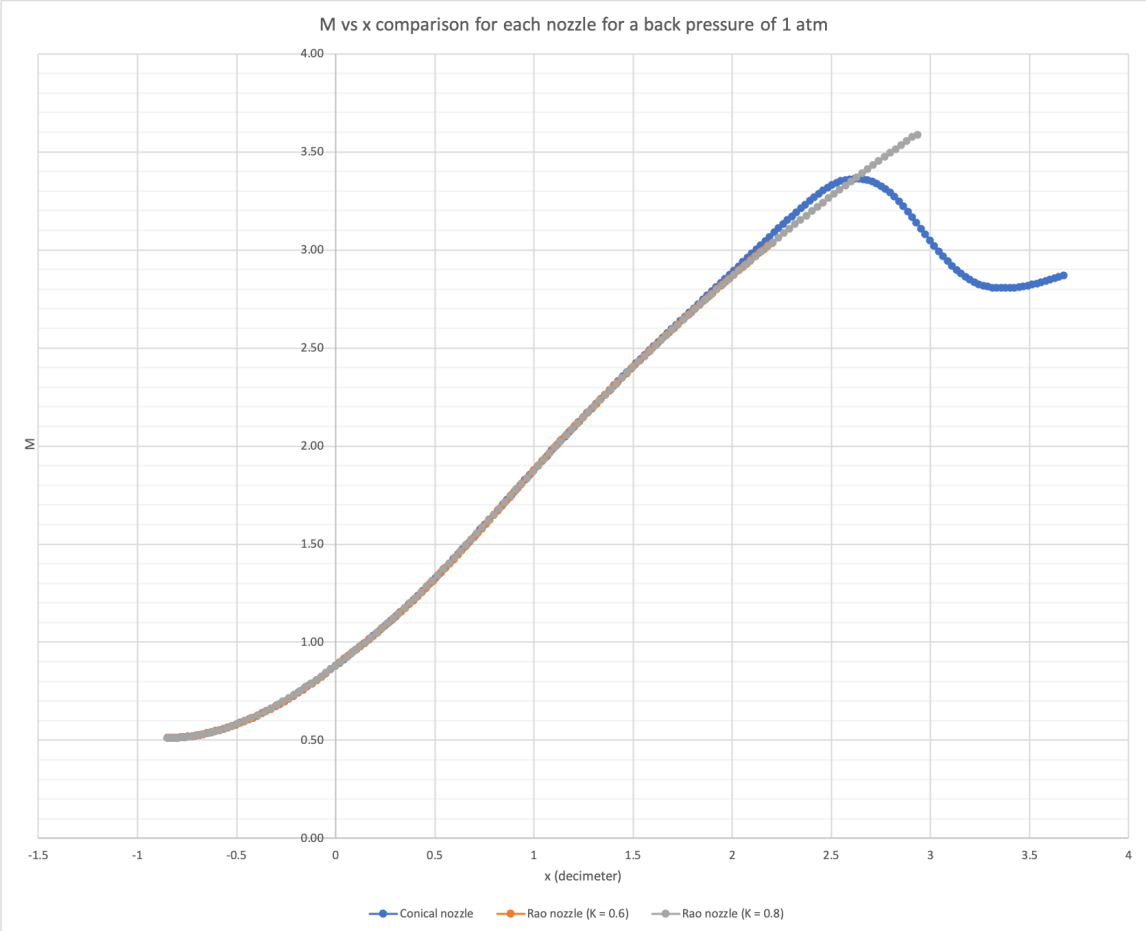
3. Results : 1D contours along the axis of nozzle

Case 1 : Optimum Expansion Condition

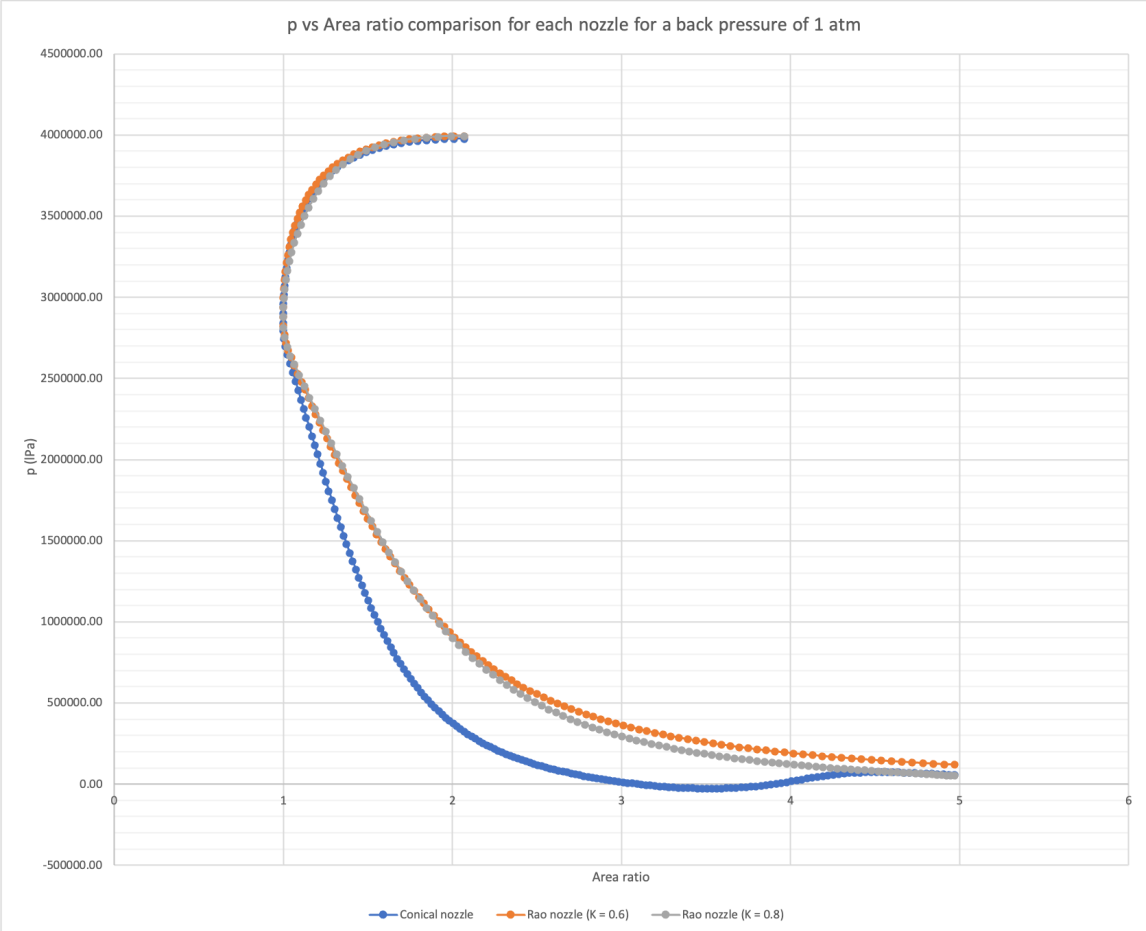
1D plots of p,T,M vs x are as shown for each nozzle (calculated along axis)

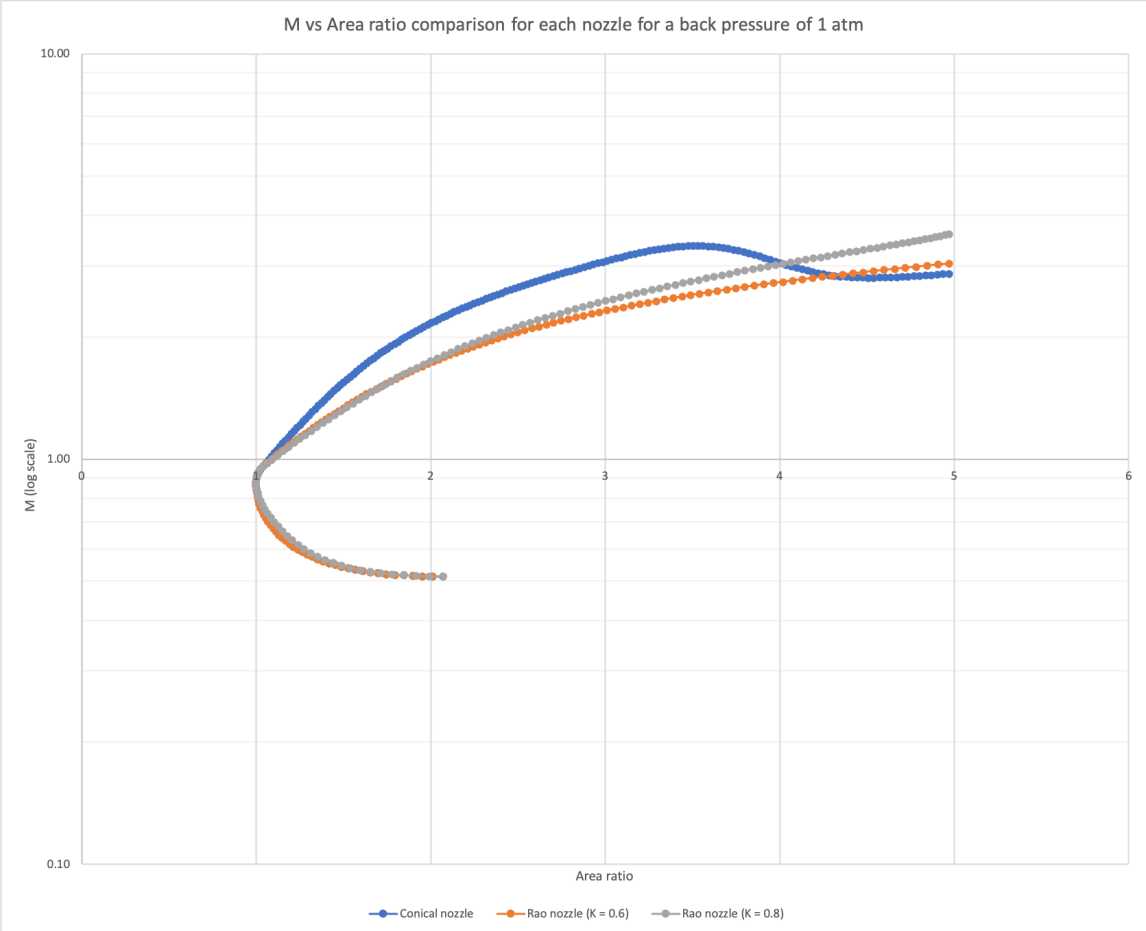

p vs x comparison for each nozzle for a back pressure of 1 atm
4500000.00
4000000.00
3500000.00
3000000.00
2500000.00
2000000.00
1500000.00
1000000.00
500000.00
0.00
-500000.00
p (Pa)
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
x (decimeter)
Conical nozzle
Rao nozzle (K = 0.6)
Rao nozzle (K = 0.8)


T vs x comparison for each nozzle for a back pressure of 1 atm
1000.00
900.00
800.00
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
T (K)
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
x (decimeter)
Conical nozzle
Rao nozzle (K = 0.6)
Rao nozzle (K = 0.8)



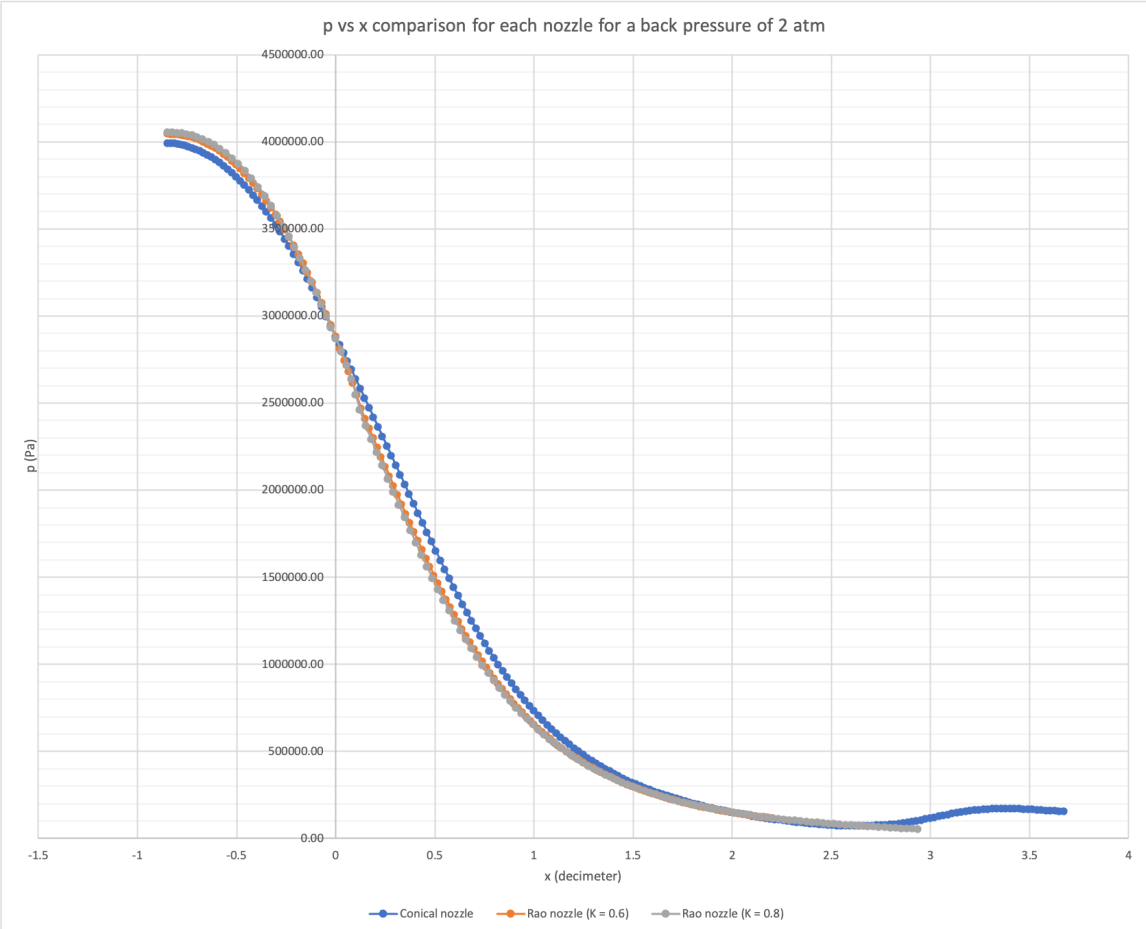
1D plots of p, M vs Area ratio are as shown for each nozzle (calculated along axis)

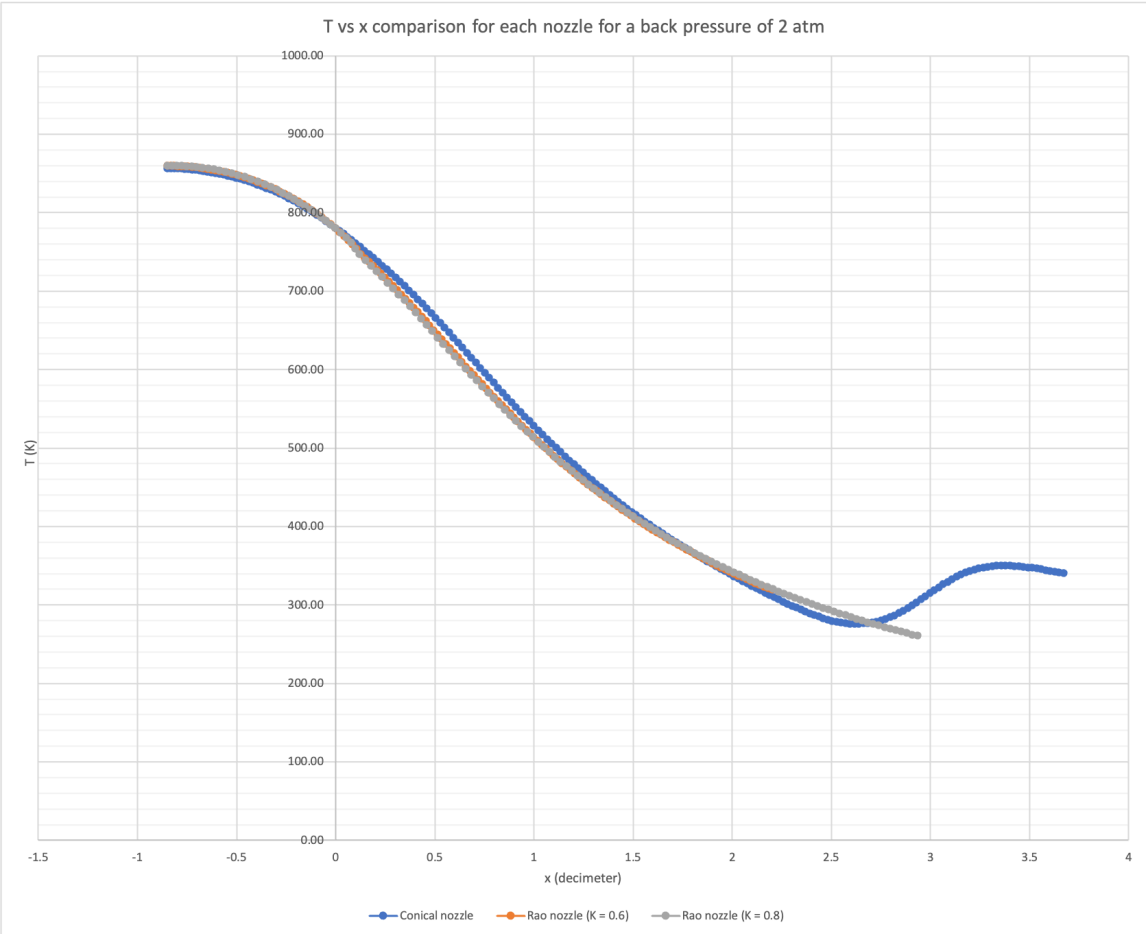


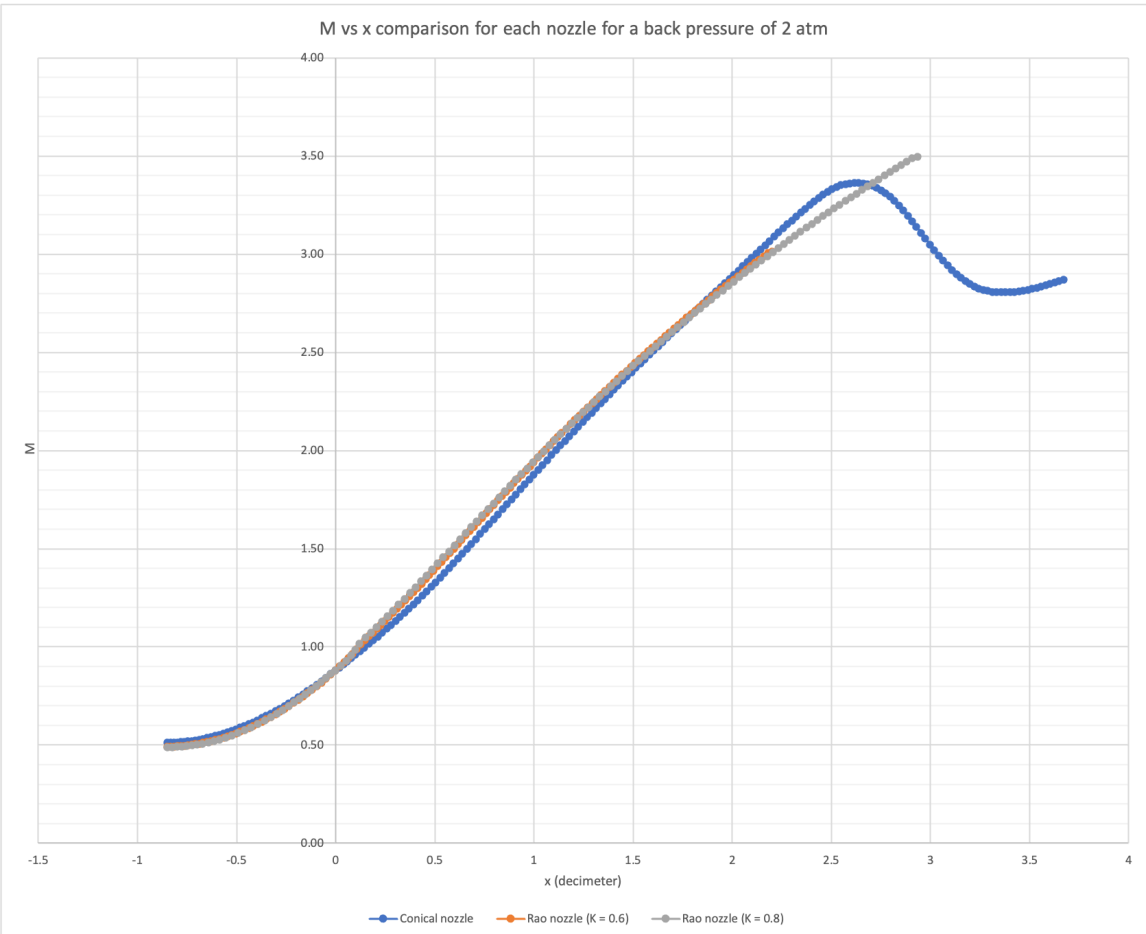


Case 2 : Back pressure of 2 atm

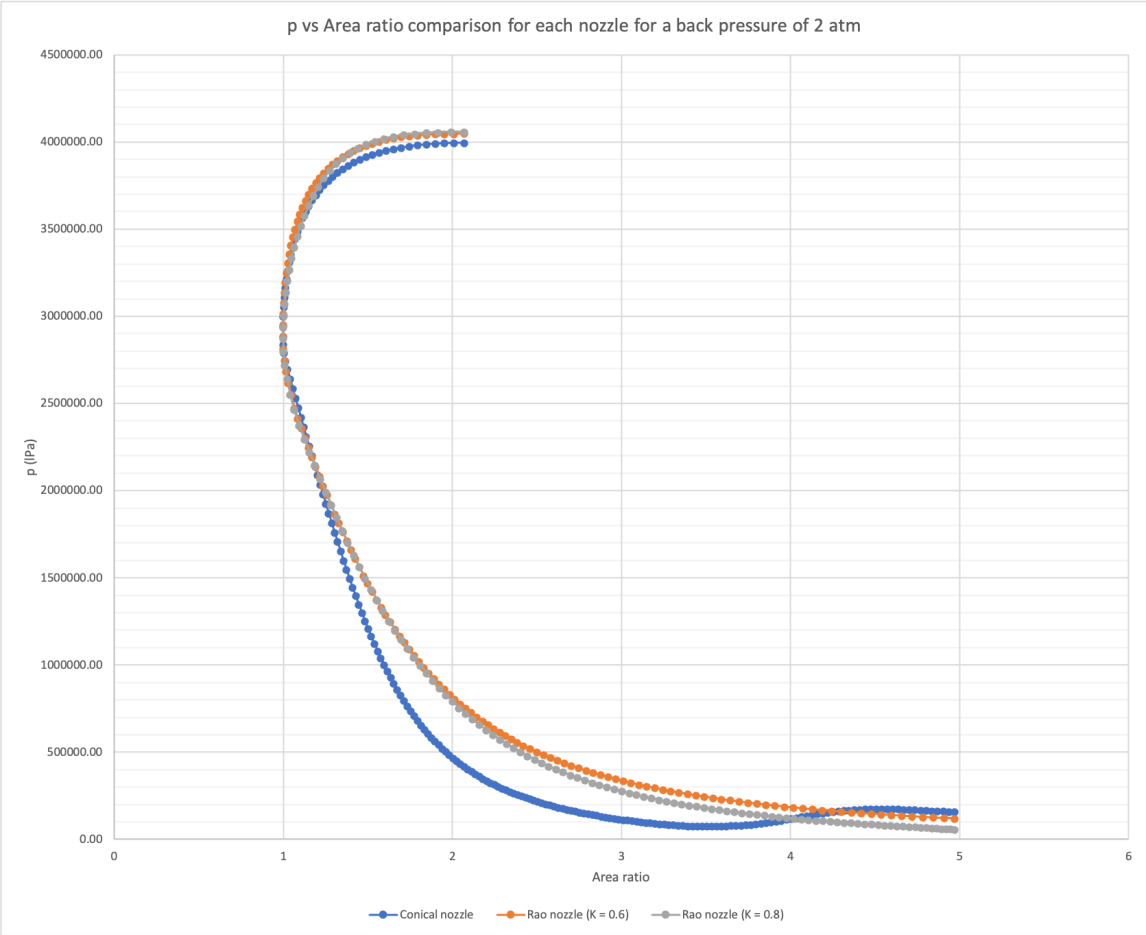
1D plots of p,T,M vs x are as shown for each nozzle (calculated along axis)

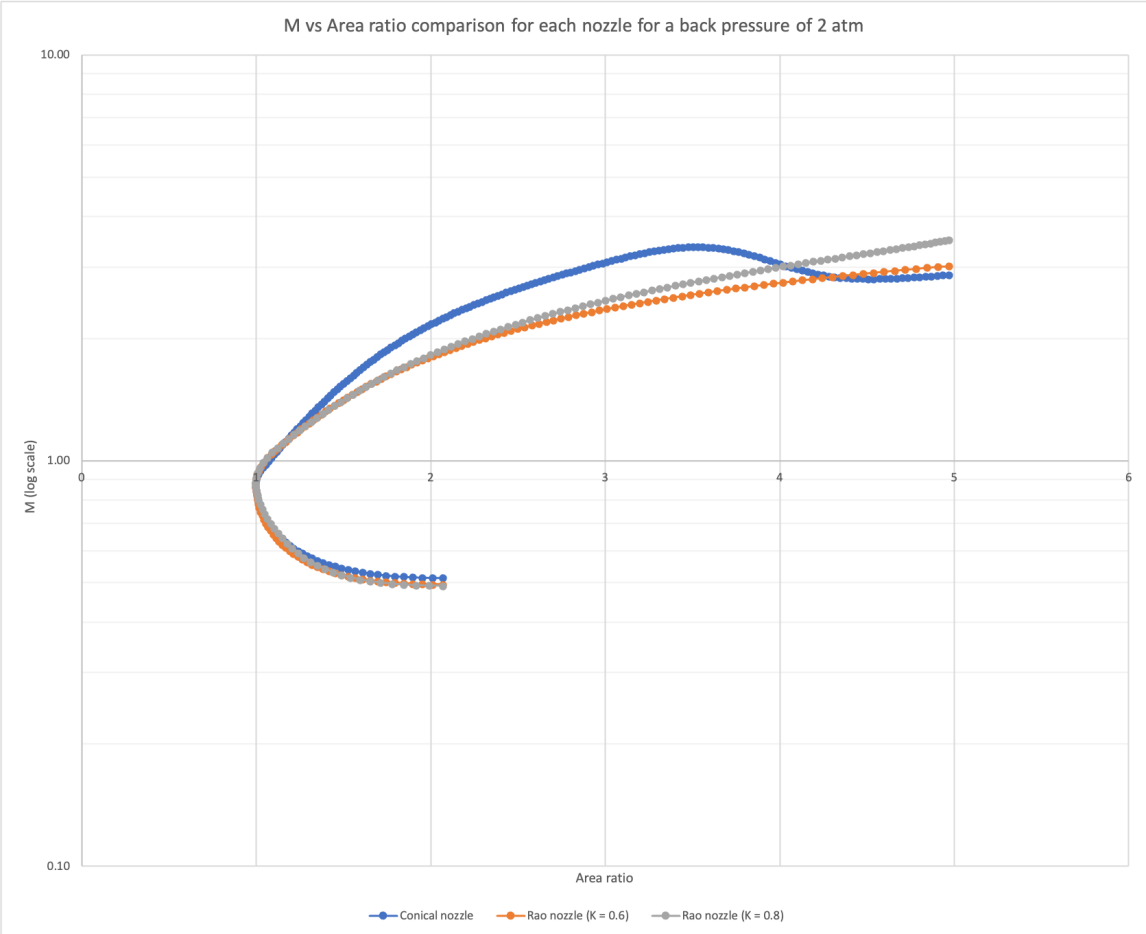






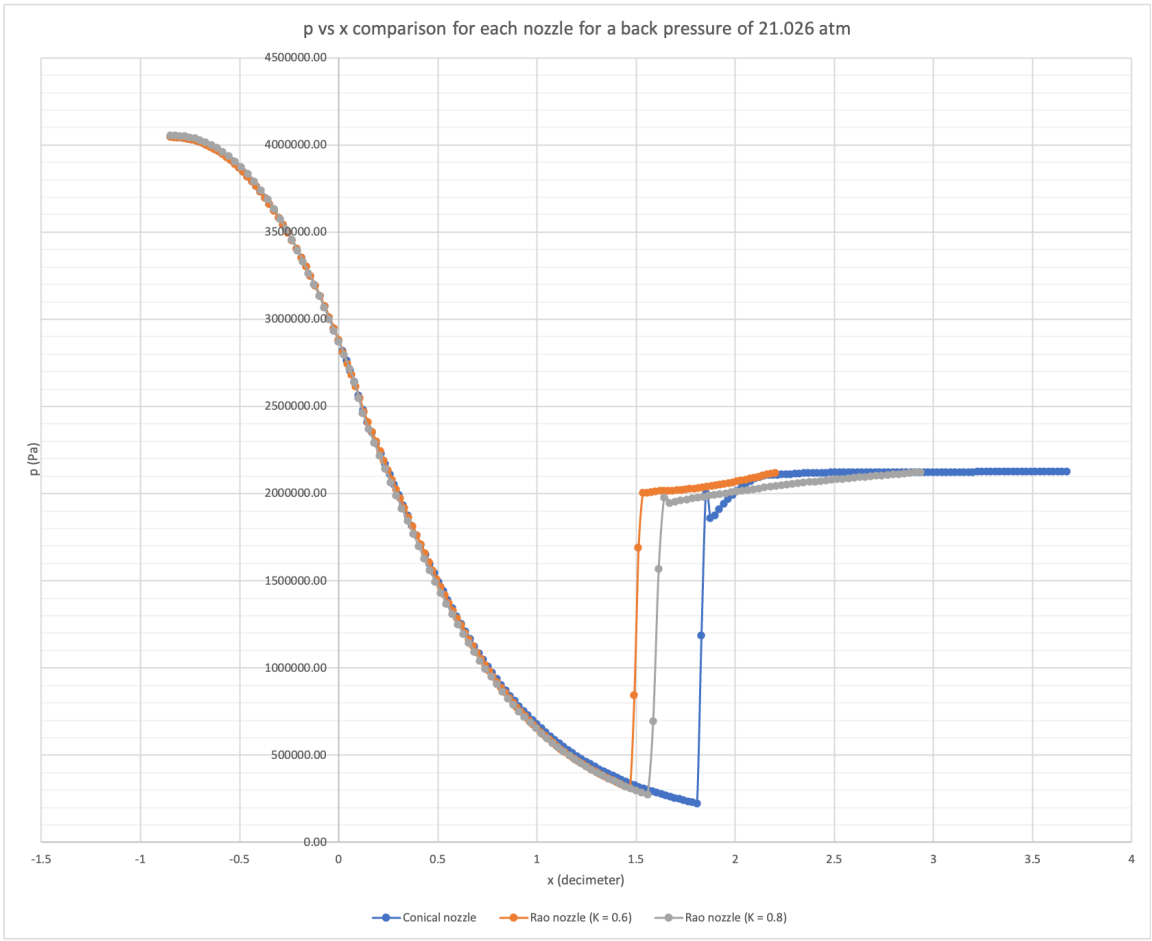
1D plots of p, M vs Area ratio are as shown for each nozzle (calculated along axis)

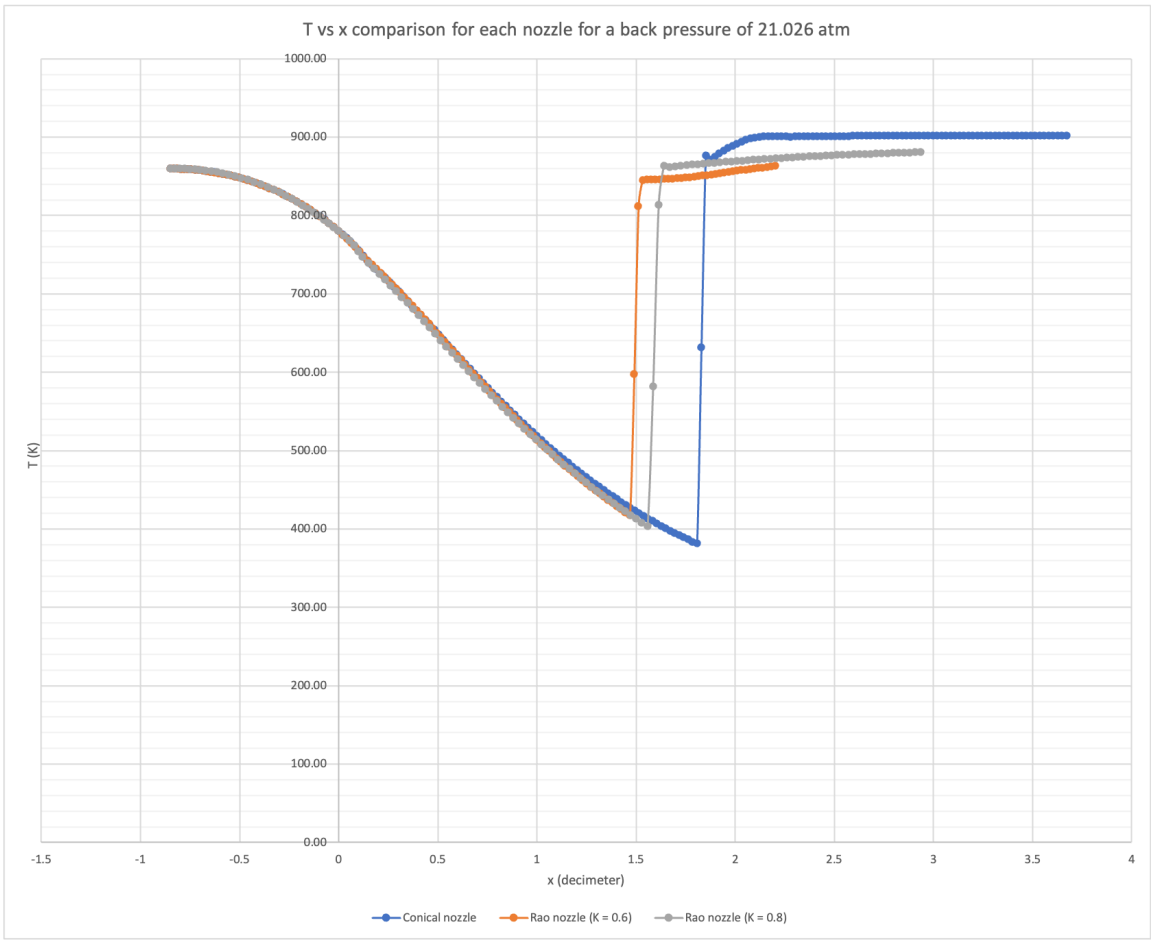


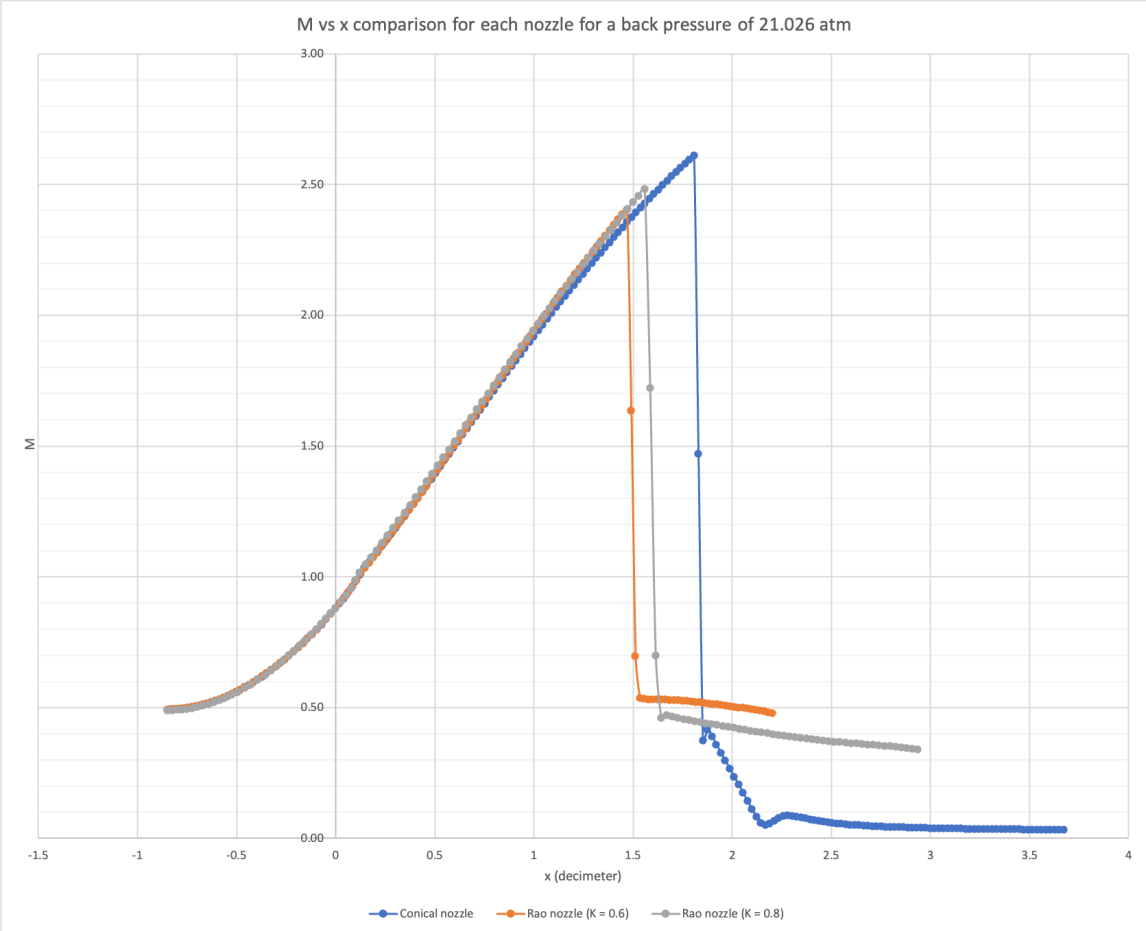


Case 3 : Back pressure of 21.026 atm

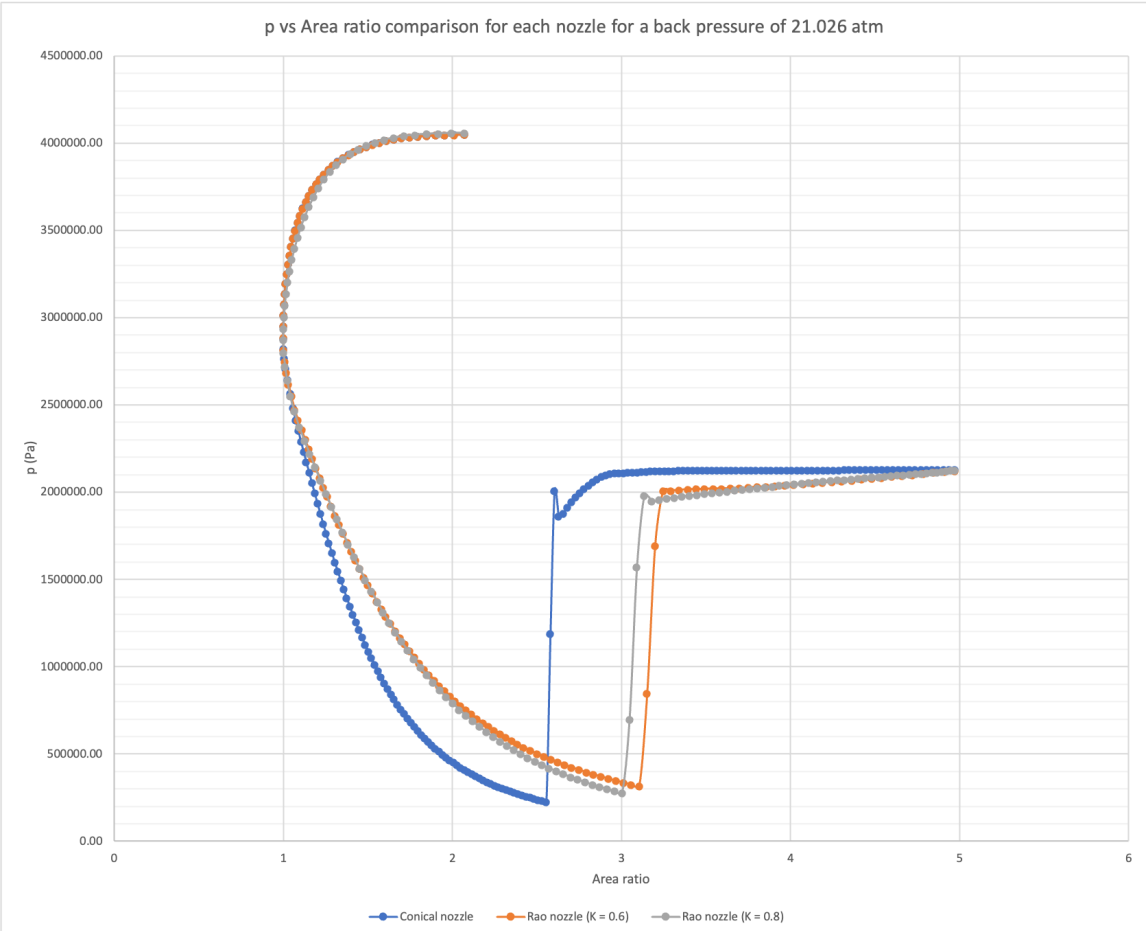
1D plots of p,T,M vs x are as shown for each nozzle (calculated along axis)

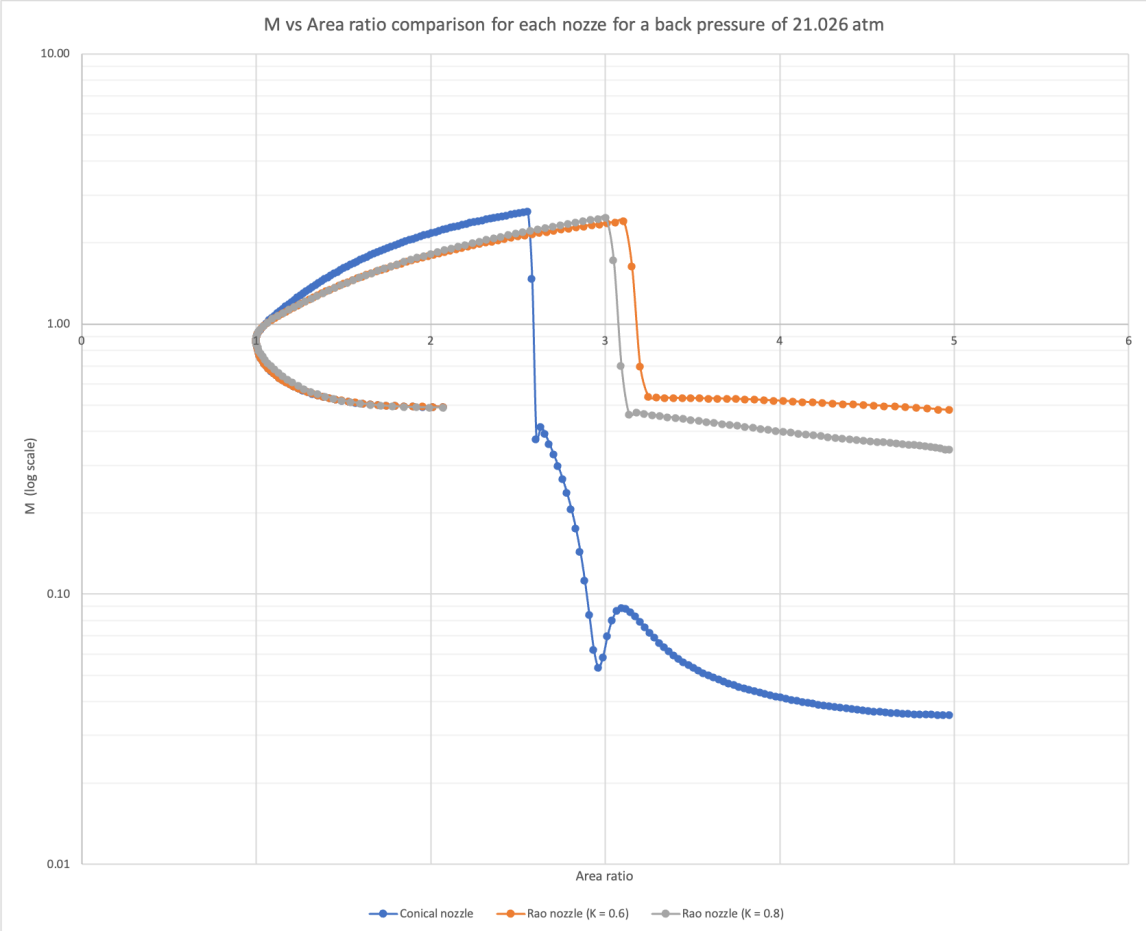

p vs x comparison for each nozzle for a back pressure of 21.026 atm
p (Pa)
4500000.00
4000000.00
3500000.00
3000000.00
2500000.00
2000000.00
1500000.00
1000000.00
500000.00
0.00
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
x (decimeter)
Conical nozzle
Rao nozzle (K = 0.6)
Rao nozzle (K = 0.8)


T vs x comparison for each nozzle for a back pressure of 21.026 atm
T (K)
1000.00
900.00
800.00
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
x (decimeter)
Conical nozzle
Rao nozzle (K = 0.6)
Rao nozzle (K = 0.8)



1D plots of p, M vs Area ratio are as shown for each nozzle (calculated along axis)





Inference

For Optimum expansion condition the variation of all quantities are observed to be smooth and this shows the absence of shock in the nozzle for this case.

Although, a small decrease in Mach number is oberseved in the diverging section for the case of conical nozzle which is due to improper expansion of flow due to the conical geometry.

We are hypothesising that this is due to the presence of a weak shock in the diverging section of the conical nozzle. We can verify this using the 2D contours we obtained from Ansys.

Thus the performance of a conical nozzle is is observed to be inferior to that of a Rao nozzle (Bell nozzle).

In case of 2 atm back pressure, the behavior is expected to be similar to that of optimum expansion condition as the shock is supposed to form outside nozzle after the exit plane so flow peroperties must behave similarly to previous case inside the nozzle section. This can be observed in the plots.

In case of 21.026 atm back pressure, we expect a normal shock to form at a location corresponding to an area ratio of 3.0076. From the plots, it is observed that the shock indeed forms close to area ratio of 3 as there is a sharp variation in p, T and M at this location.

One can also note that the exit mach number observed for conical nozzle is much lower than that for bell nozzles, for both prefectly expanded conditions as well as flow with normal shock at exit plane.

Comparing between 60% and 80% length bell nozzles, we obsrve that the 60% length bell nozzle performs slightly better than the 80% length in case where there is shock at diverging section. Since the simulation is inviscid, we can ignore the wall friction effect. So this performance variation is purely based on geometrical influence on the flow.

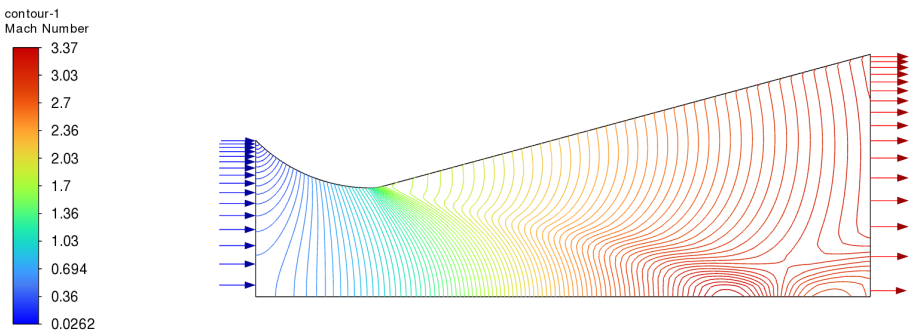
On the other hand, in case of perfect expansion case (1 atm), the 80% nozzle performs slightly better than the 60% nozzle.

4. Results : 2D contours

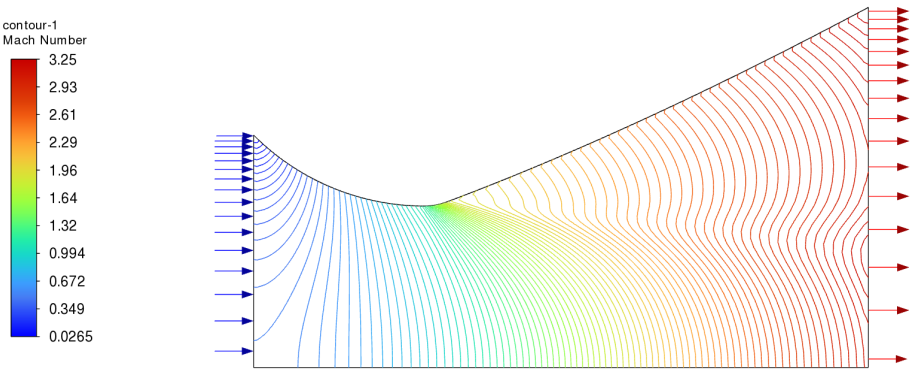
Case 1 : Optimum expansion condition (Back pressure of 1 atm)

2D contours of M is shown for each nozzle

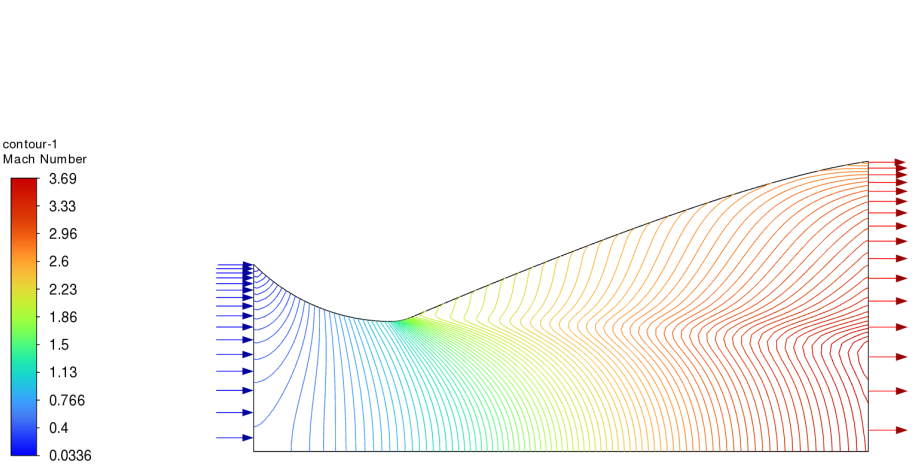
a. conical nozzle



b. Rao nozzle (K = 0.6)



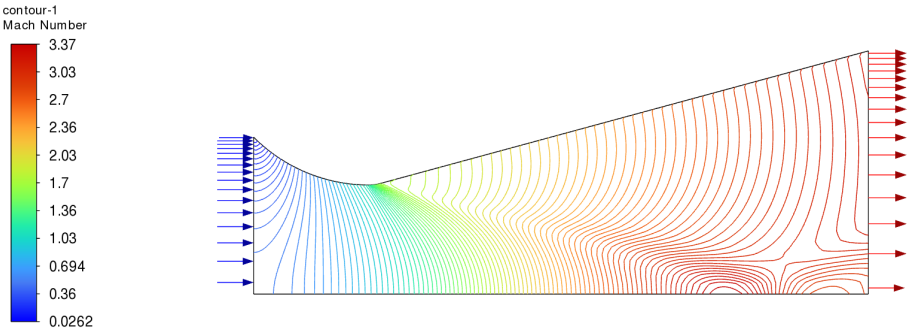
c. Rao nozzle (K = 0.8)



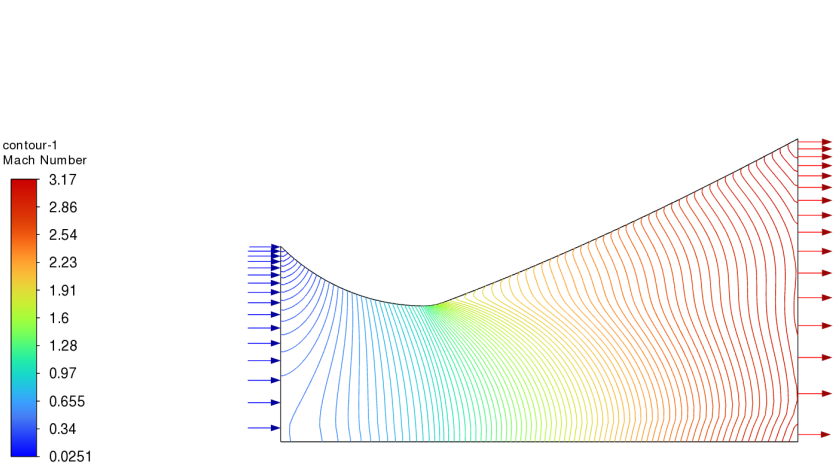
Case 2 : Back pressure of 2 atm

2D contours of M is shown for each nozzle

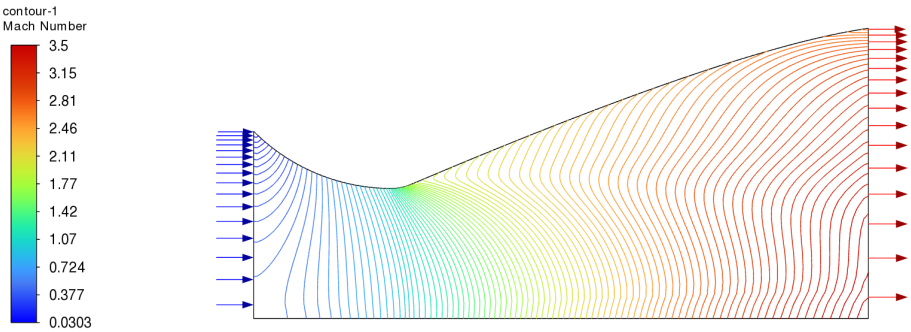
a. conical nozzle



b. Rao nozzle (K = 0.6)



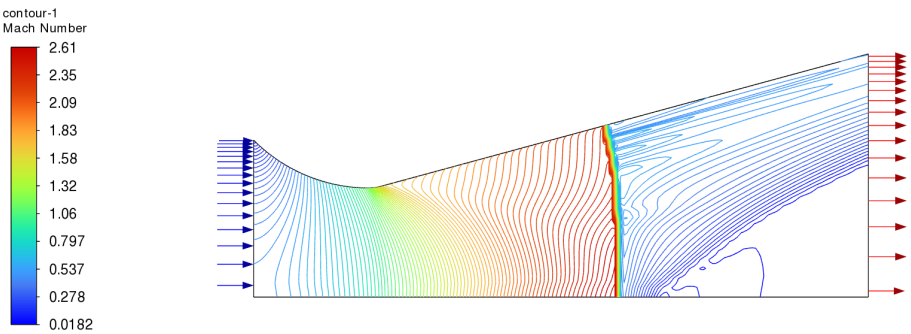
c. Rao nozzle (K = 0.8)



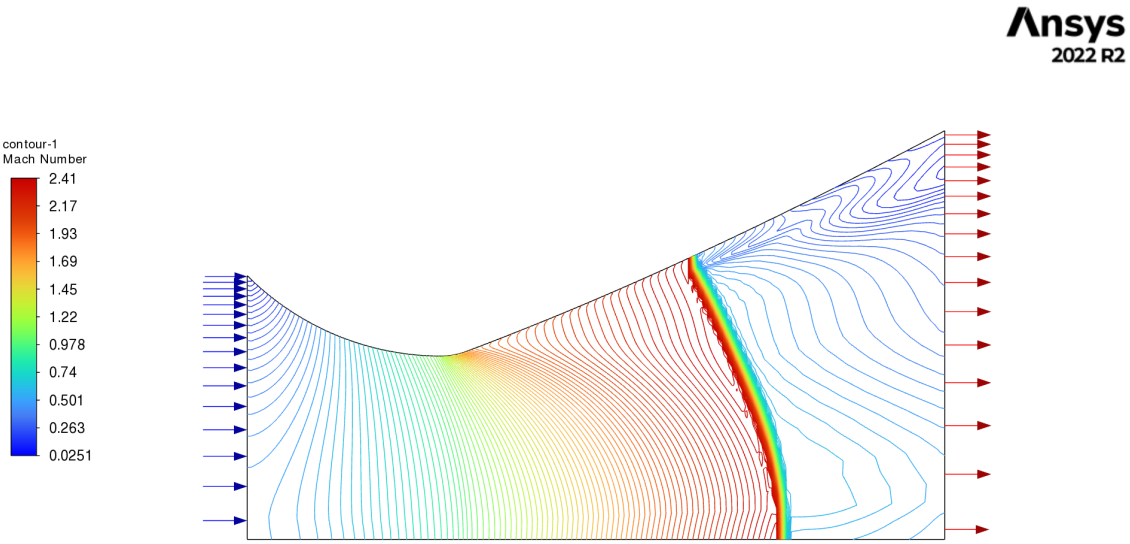
Case 3 : Back pressure of 21.026 atm

2D contours of M is shown for each nozzle

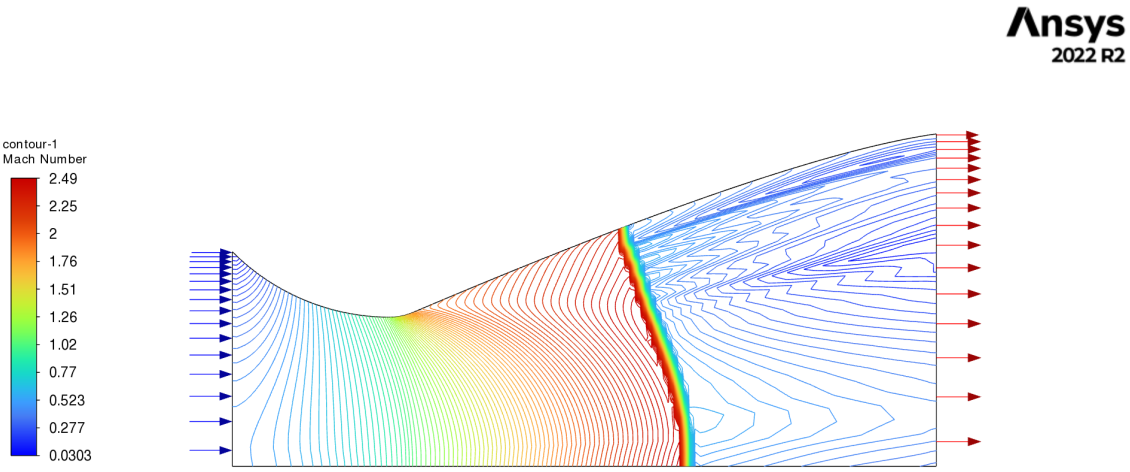
a. conical nozzle



b. Rao nozzle (K = 0.6)



c. Rao nozzle (K = 0.8)



Inference

For Optimum expansion case and higher back pressure of 2 atm, our obervations are in agreement with the hypothesis before, as we observe that there is a weak shock in the diverging section in case of the conical nozzle apparent by the high curvature of the iso-contours of Mach number in case of conical nozzle. This curvature is also present in case of the bell nozzle but is not as prominent. Thus we can conclude that in order to avoid such flow features, it is better to utilise a bell nozzle as opposed to a conical nozzle.

For the case where there is normal shock at diverging section we observe that again there is a presence of high curvature after the normal shock for conical nozzle as opposed to bell nozzles.

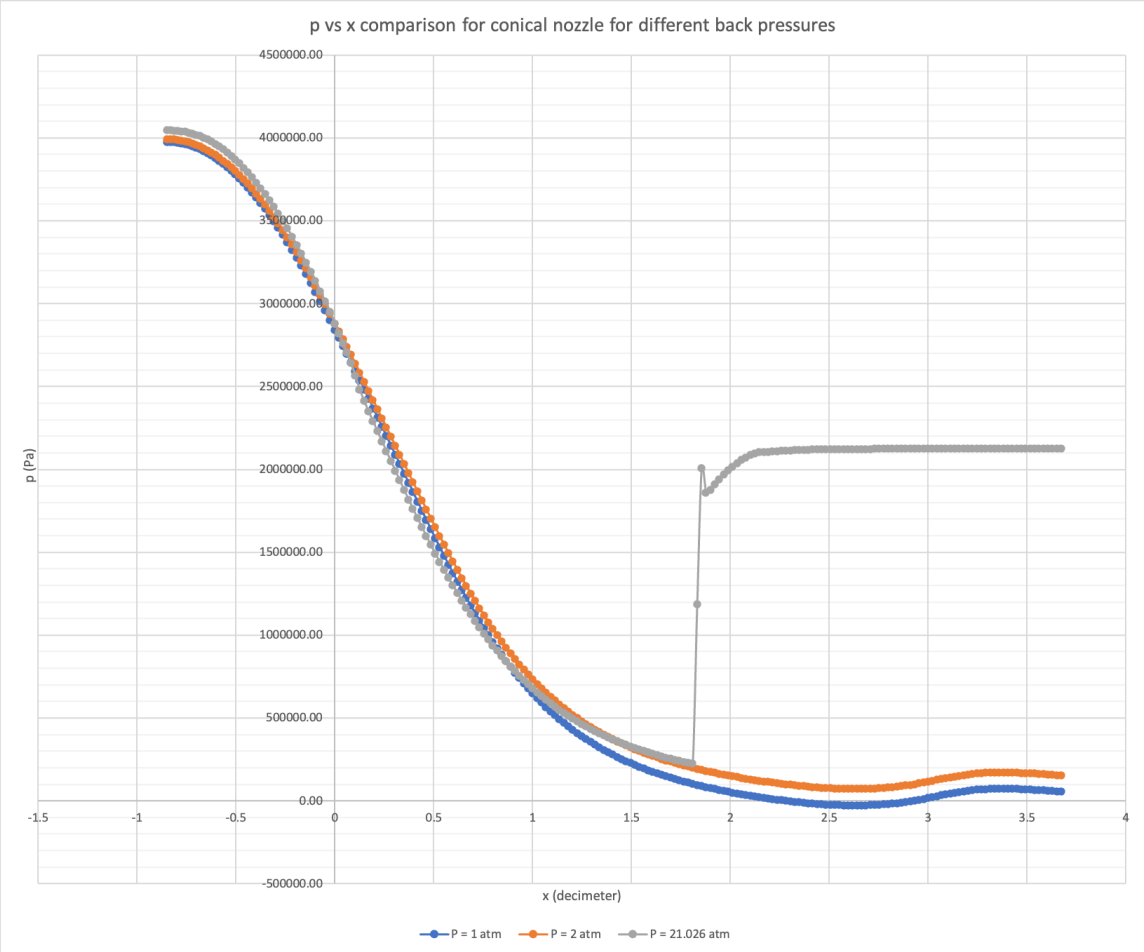
Also it can be noted that the 2D contours of M are mostly along radial direction near the axis of the nozzle while they curve as one moves radially away from axis. This behaviour can be attributed to the effect of geometry as the flow is choked at throat and any geomtrical changes after throat has to be accomodated by the flow accordingly by adjusting the radial flow. This is evident from the fact that the iso-contours are identical for every nozzle before the throat while it changes accodingly after throat due to different geomtry after the throat in each nozzle case.

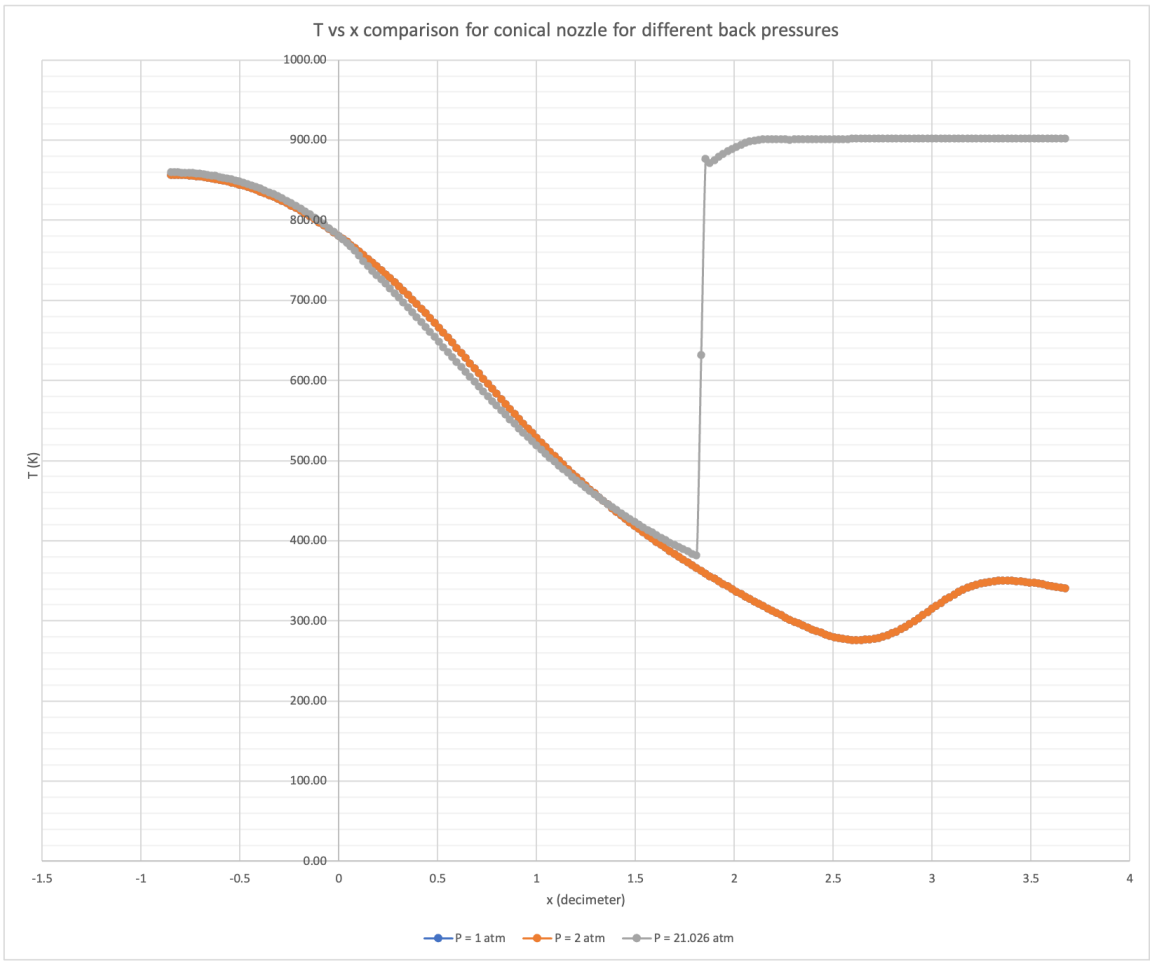
5. Effect of increasing back pressure

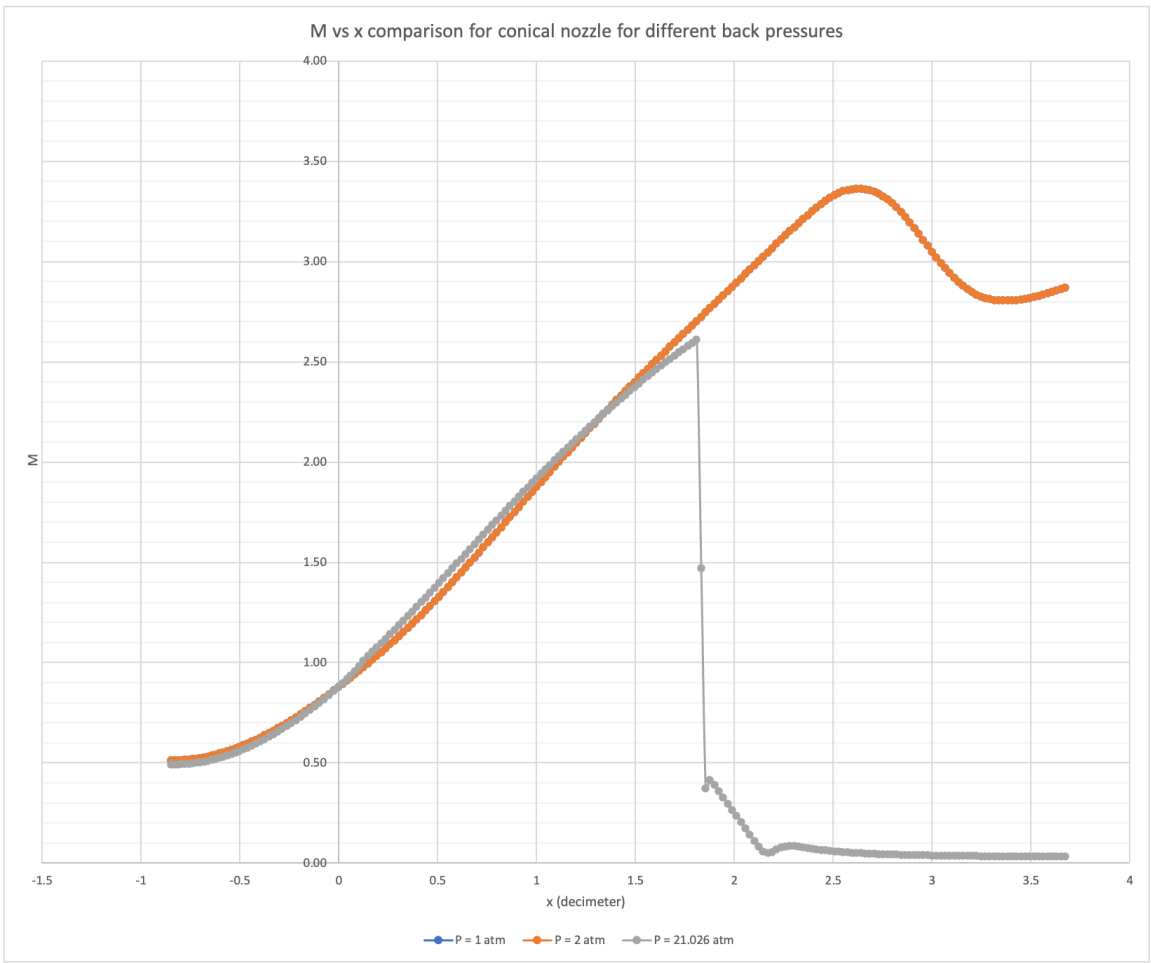
Comparison of 1D plots

The following plots show the comparison of p, T, M vs x values for each nozzle for the 3 different back pressure cases we considered.

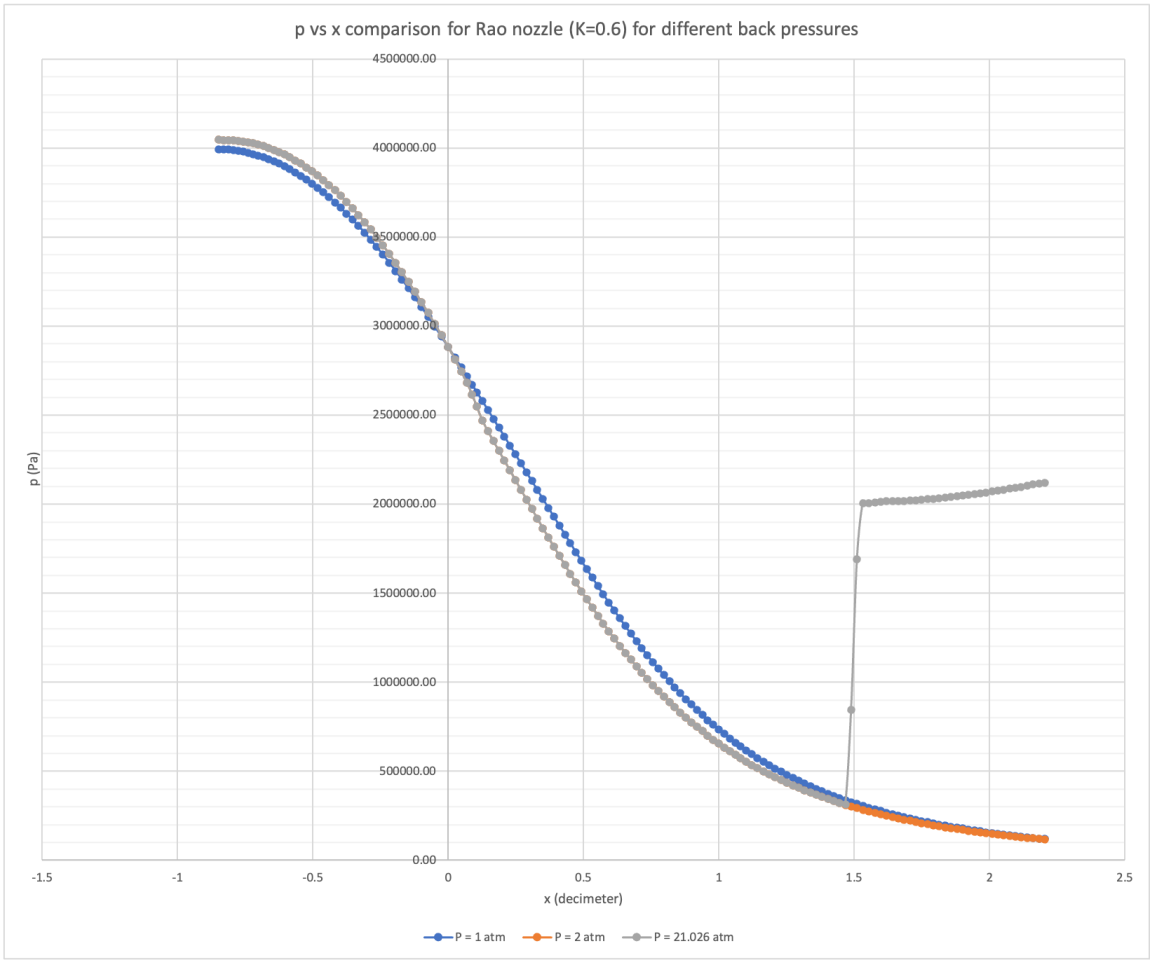
a. Conical nozzle

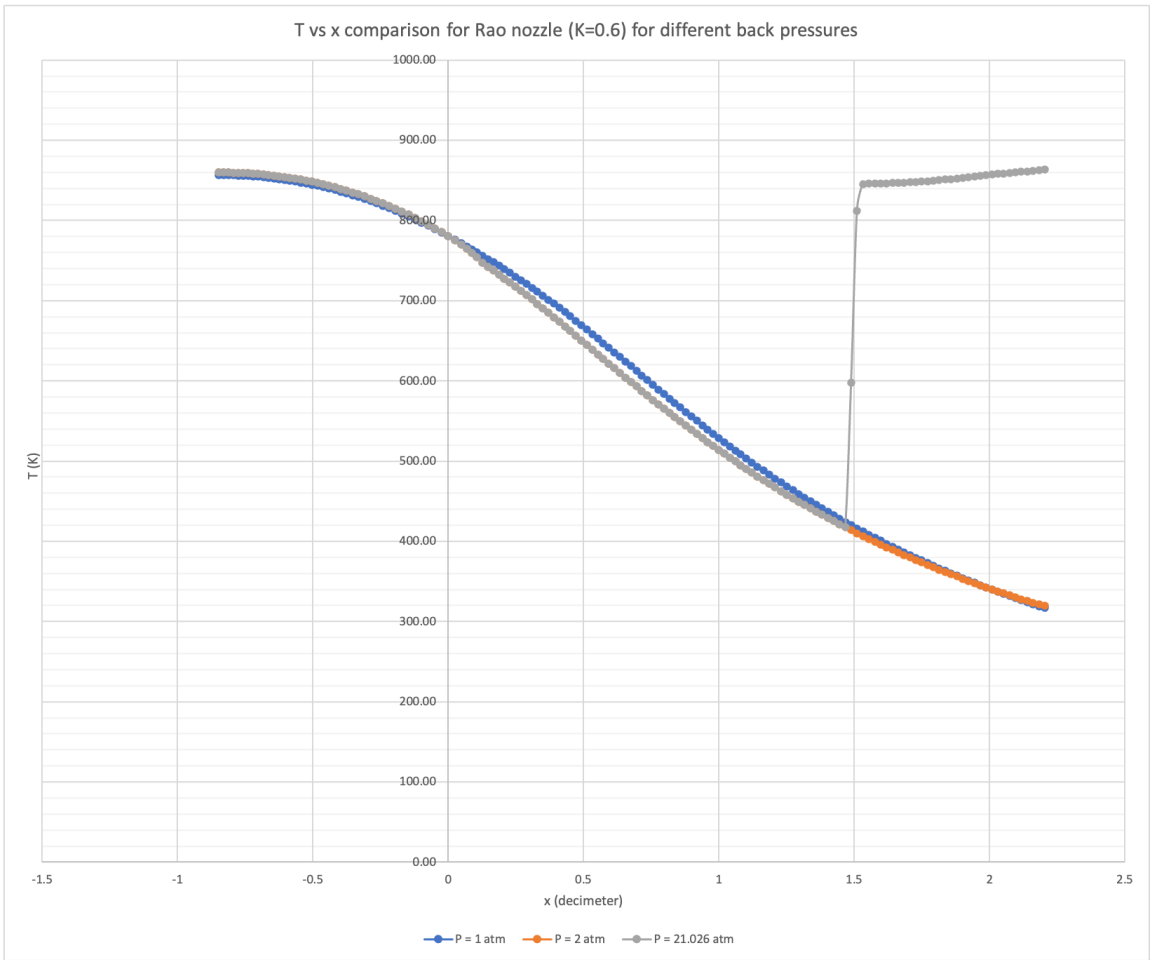


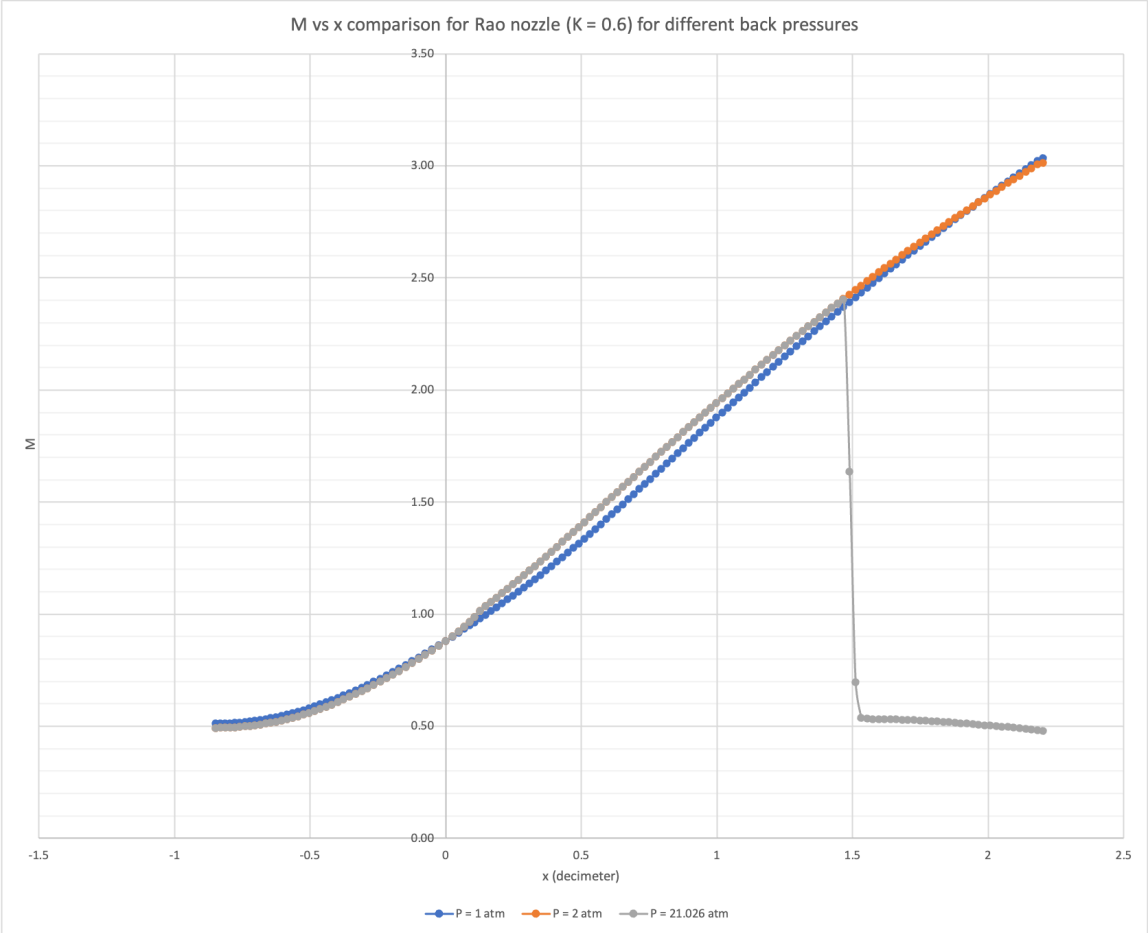




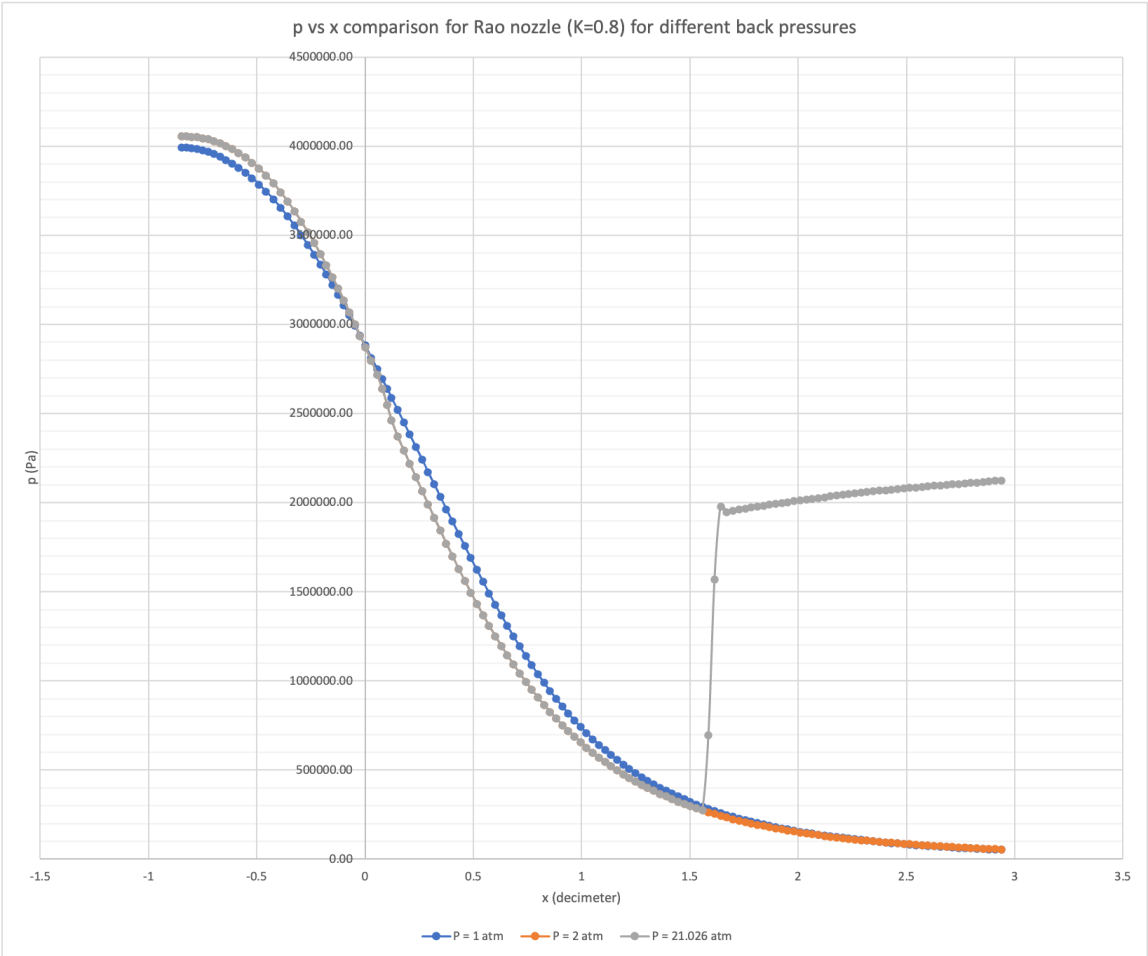
b. Rao nozzle (K = 0.6)

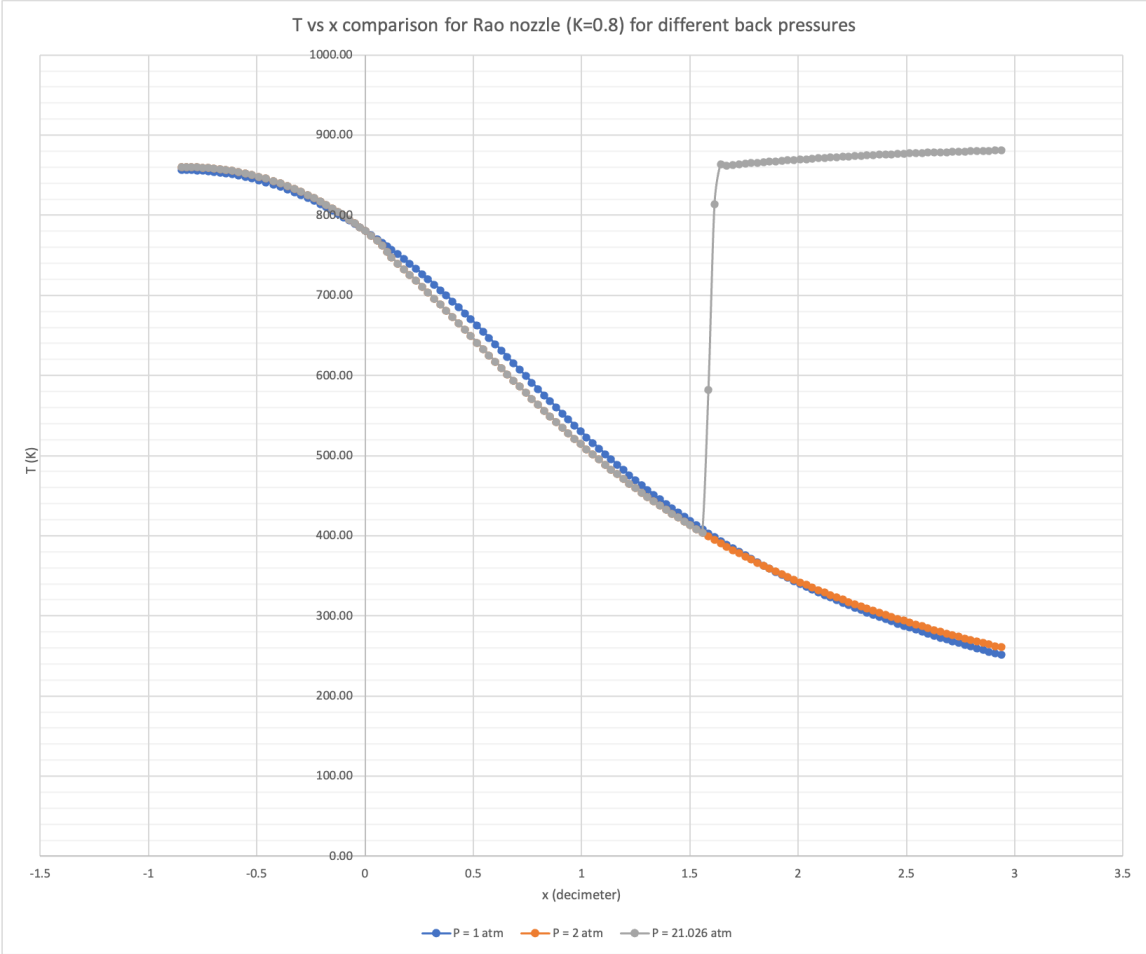

p vs x comparison for Rao nozzle (K=0.6) for different back pressures
4500000.00
4000000.00
3500000.00
3000000.00
2500000.00
2000000.00
1500000.00
1000000.00
500000.00
0.00
p (Pa)
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
x (decimeter)
P = 1 atm
P = 2 atm
P = 21.026 atm

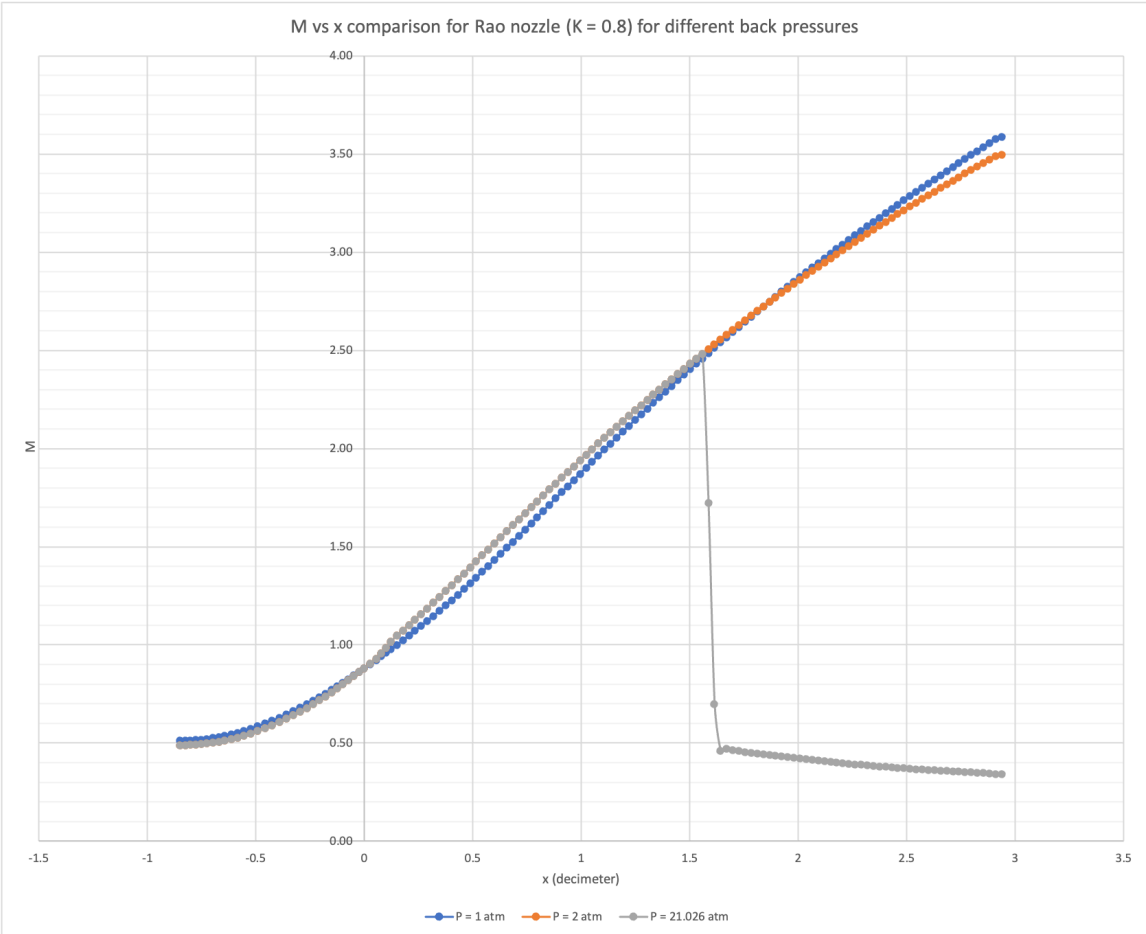

T vs x comparison for Rao nozzle (K=0.6) for different back pressures
1000.00
900.00
800.00
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
T (K)
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
x (decimeter)
P = 1 atm
P = 2 atm
P = 21.026 atm



c. Rao nozzle (K = 0.8)

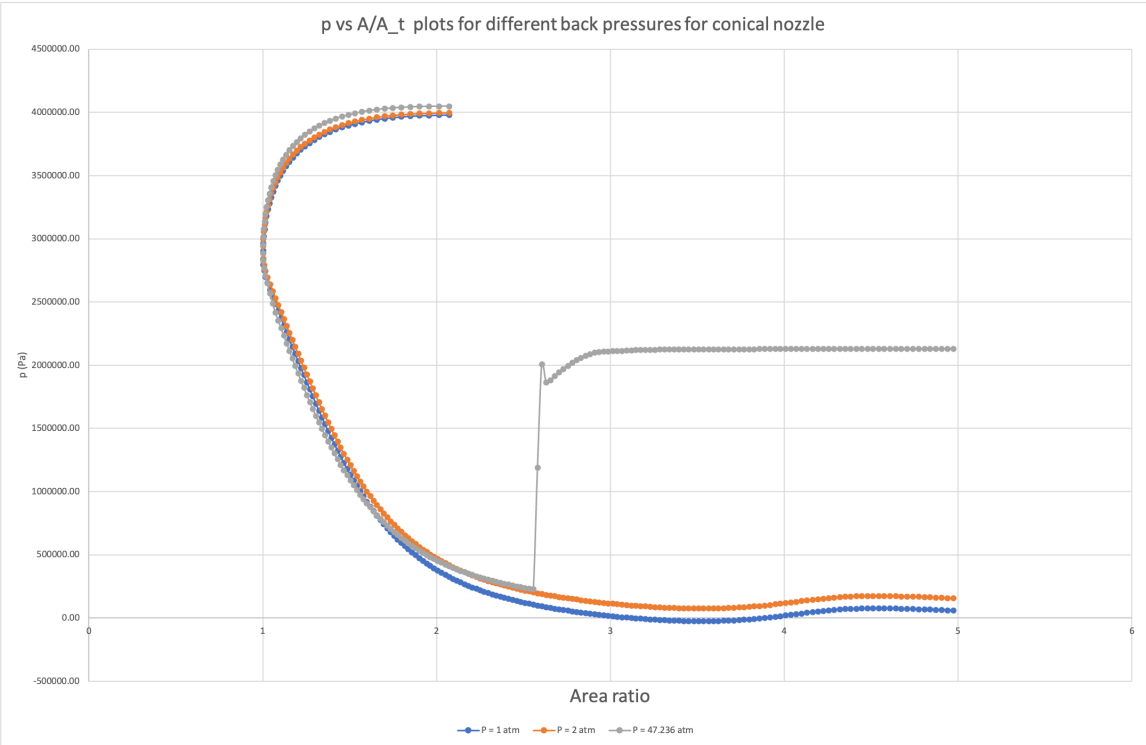


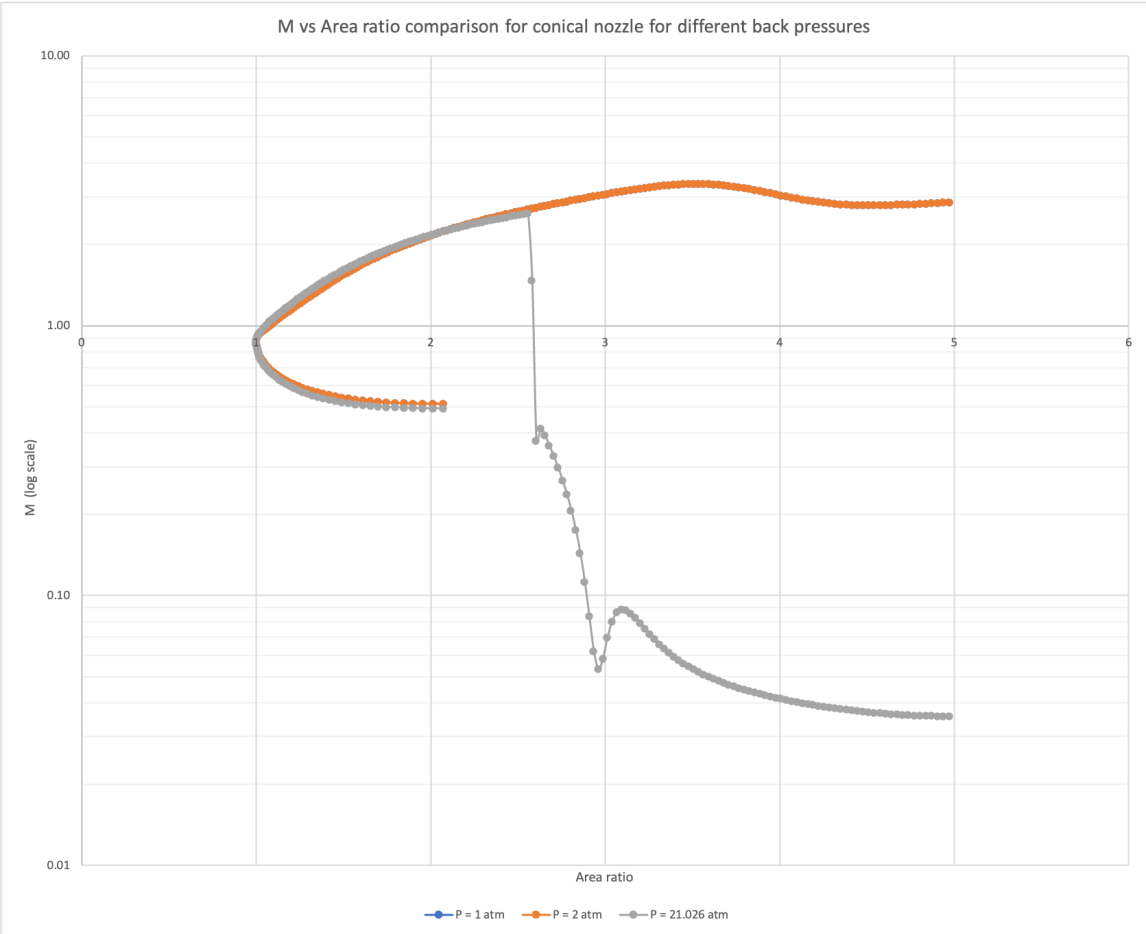




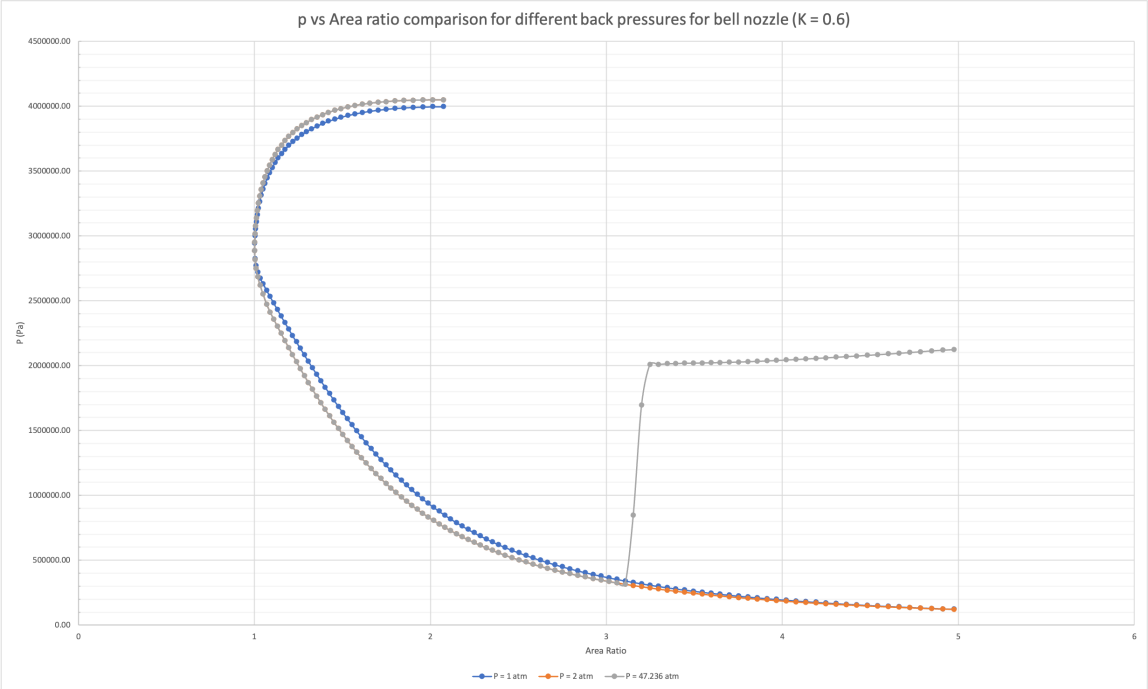
The following plots show the comparison of p, M vs Area ratio values for each nozzle for the 3 different back pressure cases we considered.

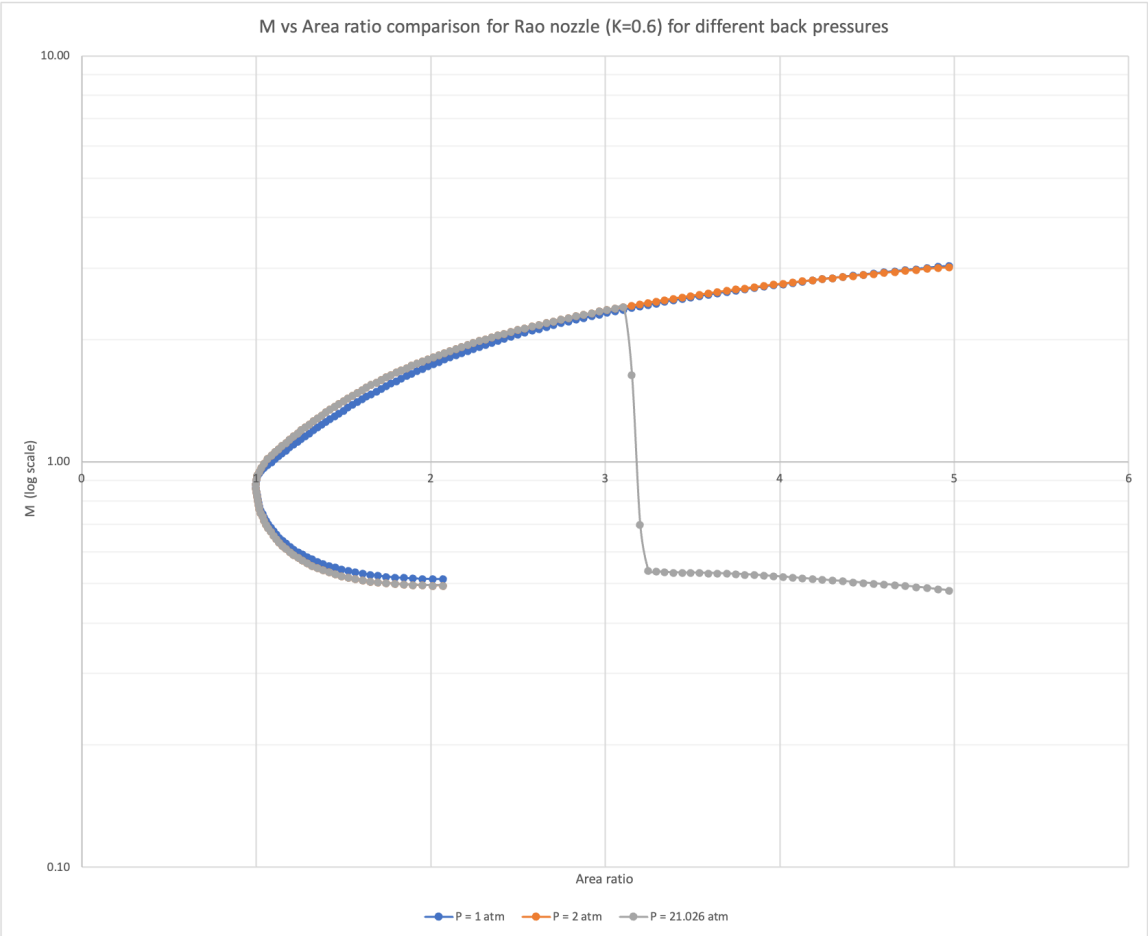
a. Conical nozzle



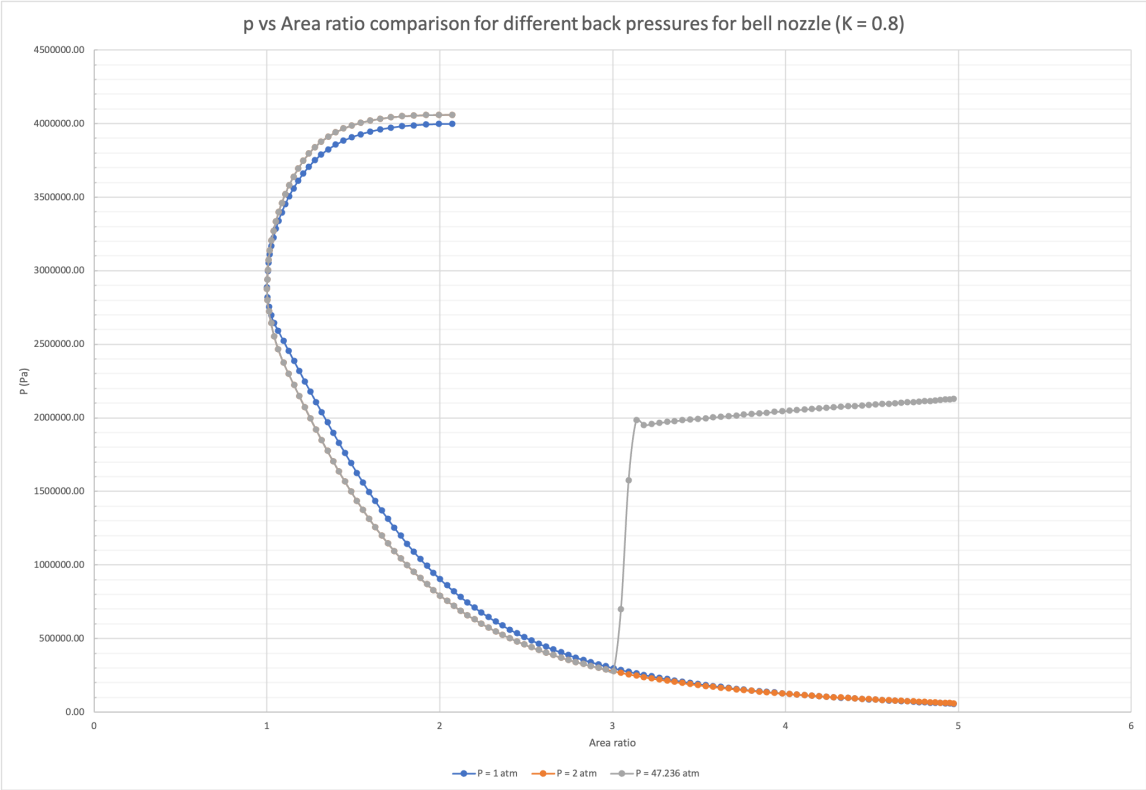


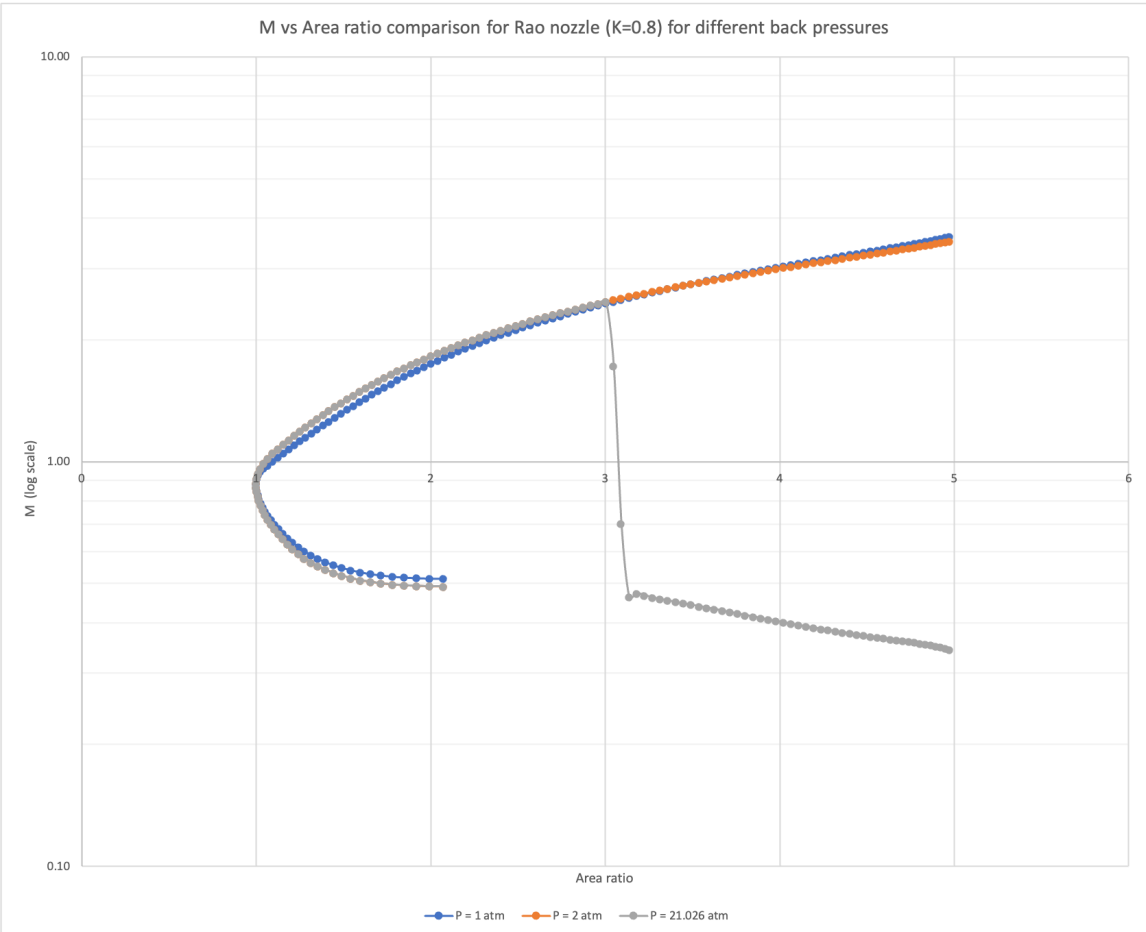
b. Rao nozzle (K = 0.6)





c. Rao nozzle (K = 0.8)





Inference

Again, as dicussed before, we can see that changing back pressure will not affect the flow until the the case where normal shock is generated at exit plane. Thus the plots for 1 atm back pressure (optimum expansion condition) and 2 atm back pressure are more or less identical.

From calculation, it is observed that for area ratio of 4.973, until the back pressure is increased more than 13.4025 atm, there will not be any change in the contours of diverging section. As we further increase back pressure, there is the formation of normal shock at exit plane and thus properties (P,T,M) change at exit plane from the optimum expansion case.

In order to capture the effect of oblique shocks after nozzle exit plane, the flow domain must also include the region after nozzle exit plane which is not the case for the above simulation.

Conclusion

We can conclude from our analysis and comparisons that compared to a conical nozzle, a Rao nozzle has better performance in both perfectly expanded operation and in case of operation with normal shock at diverging section.