VARMOL

A lab report submitted on partial fulfillment of the course
Modelling and Simulation (**CHEG305**)
by:

Name: **ASHAL ADHIKARI**
Roll no: **02**

Submitted to:

Dr. Kundan Lal Shrestha
Associate Professor

Er. Sagar Ban
Teaching Assistant

DEPARTMENT OF CHEMICAL SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING

KATHMANDU UNIVERSITY

MARCH 2023

TABLE OF CONTENTS

1. INTRODUCTION

Tubular reactors are simply a pipe which is placed in horizontally or vertically at which the fluid flows at a steady state. In this reactor, the single phase flow can be described as reactants flowing from one end, normally left part and the products obtained from the right hand side[1]. The commonly used tubular reactor is a Plug Flow Reactor (PFR) in which the flow is maintained so there is no change in radial direction and travels in axial direction with uniform properties.

Variable Molar Gas-Phase Reaction or **VARMOL** refers to a type of chemical reaction that occurs in the gas phase and involves a change in the number of moles of reactants and products causing a frequent change in volume and volumetric flow rate. This change in number of moles is the effect of temperature and pressure in the gas phase volume[2]. A typical tubular reactor with variable molar flow can be seen in Fig 1.

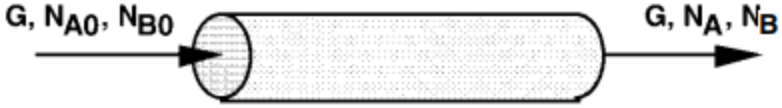


Fig 1: Typical tubular reactor[2]

In this report, the tubular reactor undertaking reaction with variable molar flow is modelled from problem statement and parameters given from the coursebook[2]. The reaction is:

$$A \rightarrow mB \qquad \text{rate constant} = k \tag{1}$$

The steady state balance for the model is written in terms of moles. From the molar flow at each point in the reactor, the Ideal Gas Law is used to calculate the volumetric flowrate. Then, from the steady state mole balance with uniform temperature and pressure, we get:

$$\frac{d(y_A G)}{dZ} = -k * y_A * A \tag{2}$$

$$\frac{d(y_B G)}{dZ} = +m * k * y_A * A \tag{3}$$

$$G = \frac{N_A + N_B + N_I}{P} * R * T \tag{4}$$

$$N_A = \frac{y_A * G * P}{R * T} \tag{5}$$

$$N_B = \frac{y_B * G * P}{R * T} \tag{6}$$

$$Xa = (Na0 - Na) / Na0 \quad (7)$$

Where,

A, **G** and **k** represents cross-sectional area, volumetric flowrate and reaction rate. **m**, **N**, **P**, **R**, **T**, **Xa**, **y** and **Z** represents stoichiometric constant, molar flowrate, pressure, gas constant, temperature, mole fraction and length. Similarly, the indices **0** refers to inlet while $_{A}$, $_{B}$ and $_{I}$ refers to inlet components A, B and inerts I.

1.1 OBJECTIVES:

The objectives of the modelling are as follows:

- To investigate the influence of **m** by varying the stoichiometry
- To observe that G does not change significantly with position by setting a high value for the molar flow rate of inerts
- To obtain a linear temperature profile for the given system
- To obtain a linear pressure profile while accounting for pressure drop across the reactor

2. METHODOLOGY

2.1 PROCEDURE

1. Before starting to code the differential equations, important library packages were imported at the beginning. **NUMPY** is imported operate mathematical arrays and matrices, **MATPLOTLIB** is imported to visualize the solved differential equations and **SCIPY** is imported to integrate the volumetric flow rate differential equations of the tubular reactor.

2. All the initial gas conditions (pressure, temperature) were defined. The initial molar flow rates of each component was also taken from the book. The necessary algebraic equations to find initial value **yaG** and **ybG** was done.

3. A function named **model_f** was created where a single dependent variable, mole fraction of component A(**ya**) and independent variable, length of the reactor(**Z**) is passed and the two differential equations are returned. The function also contains the two ODEs to be solved.

4. The length of the reactor was defined by assigning initial and final time in seconds. An array **Z** was created by using linspace which created equisapced datapoints between initial and final time **Zfinal** at given total interval of **cint**. An array of initial values to be sent to ODE solver to fed to the defined function is made and named **initial**. The elements of the array was **yaG** and **ybG**. An array of solved ODEs datapoints is obtained in variable **sol** at each length interval by using **odeint** library of **SCIPY** package. This library takes the defined function **model_f** containing ODEs to be solved, initial concentration values **initial** and dependent variable **Z**.

5. After solving the ODEs, the molar flow rates of the gas was calculated and stored species wise in an array of **Na** and **Nb**. The new volumetric flow rates was calculated and stored in **G**. For each **G** values, values of new mole fractions of A and B was stored in **yA** and **yB**. Conversion of A is also calcutaled and stored in **Xa**. The specific values of solved ODEs were taken out from array by using **sol[:,i]**. This takes out column values of species '**i**' from all rows.

6. The graph of molar fraction change of component A, B with fractional conversion of A and total volumetric flow rate was plotted along the length of the reactor **Z** using **PYPLOT** library from **MATPLOTLIB** package. Two axis were drawn in the same graph by using **PYPLOT** function **pl.subplots()** which creates sub plots and **ax.twinx()** which makes the graph duplicate the x-axis. The ax in the next subplot that was created by using **pl.subplots()**. The library intakes the x-axis value **Z**, the y-axis value, **yA**, **yB**, **Xa** and **G**, label, color and linestyle of the curve. The left y-axis is for the **yA**, **yB** and **Xa** graph and the right y-axis is for the **G** graph.

7. To measure the influence of **m** has on the volumetric flow rate **G**, we created as an array **mchange** which 5 array elements of various m values. Then, in a **for** loop in range **(0, 5)**, the **m** value changes by taking **i** element of **mchange**. The ODE is solved using **odeint** by passing the **model_f** function with new **m** value. New values of **Na**, **Nb** and **G** values are calculated using solved values stored in **sol**. Five plots with different **m** values in then plotted along the length of the reactor **Z** in a single graph.

8. The influence of initial molar flow rate of inert **Ni0** on the volumetric flow rate **G** was shown in a graph by following the similar approach as **step 7**. Here, the **m** value was reverted to 2 and 5 array elements of various Ni0 values were assigned to the **Ni_change** array. Inside the loop of range **(0, 5)**, the **Ni0** was changed by taking **i** element of **Ni_change.**

9. To see the influence of **G** with the temperature, array **T_change** was introduced and three operating temperatures were stored. Then, a new function was created called **model_new** which has the addition ODE of temperature change with the **Z** with the values of the constant given randomly so that the result best suits with the graph provided at the book[2]. Variable **k1** is also introduced which stores the changed reaction rate constant with the temperature. This new model now returns three ODEs. A subplot was created for plotting the graphs **G** vs **Z** and **Xa** vs **Z**. A **for** loop was runned with range as elements in array **T_change** using **T_change.shape[0]**. Now, the values of the initial was taken as previously and the ODEs were solved by calling **odeint** function. Then, for calculating the values of **G**, **Na** and **Nb**, the temperature used was taken from the **sol** array which stores the solution of solved ODEs rather than the constant **T0**. After that, the graphs were plotted inside the loop to display three curves in a single graph.

10. Similar to **step 9**, the pressure profile with pressure drop was modelled as the linear temperature profile. For the pressure change to observe, a new array called **P_change** was created that stored three different pressure values. Then, inside the for loop, the initial values to be provided were stored in variable **initial** that stored new values using pressure values provided in **P_change**. Inside the function **model_new**, the ODE of pressure change with **Z** (Ergun equation) was introduced with constant values taken as a random number. Similar to **step 9**, the ODEs were solved by calling **odeint** function and the values of **Na**, **Nb** and **G** were calculated using the pressure value **sol[:,2]** which contains the solved ODE of pressure with respect to **Z**. then, the graph of **G** vs **Z** and **P** vs **Z** was plotted in a subplot. A new function was imported from the **MATPLOTLIB** library called **TICKER** which helps to format the plot. Here, the pressure values in y-axis of **P** vs **Z** graph was formatted to be viewed in scientific notation using the command **ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.1e'))** which keeps the decimal number after first digit.

11. To see the change of molar fraction of A, **yA** and volumetric flow rate **G** with the change on initial molar flow rate of inert **Ni0**, arrays **yA_final**, **Ninert** and **G1** were created to store the values for plotting with initialization by using **np.zeros(10)**. For the plotting of inert value from **1** to **10**, the value assign in the newly created array was run inside the **for** loop with range **(0, 10, 1)**. Inside the for loop, the initial values of **yaG** and **ybG** are calculated by varying the inert molar flow and stored in **initial** variable to be sent to **odeint**. The model function **model_f** containing ODEs in solved and stored in **sol**. The new values of **yA** and **G** are calculated and saved in **yA_final** and **G1**. For each iteration, the column values of the last row is taken ie. **sol[-1][i]** into account while calculating **Na**, **Nb** and similarly molar fraction **yA_final** and volumetric flow rate **G1**. The graph of **yA_final** and **G1** is plotted along the initial inert molar flow **Ninert** with different y-axis. This is done in similar way described in **step 6**.

3. RESULTS AND DISCUSSION

3.1) RESPONSE OF MOLAR FRACTION OF COMPONENT 'A' AND 'B' AS WELL AS FRACTIONAL CONVERSION 'Xa' AND VOLUMETRIC FLOW RATE 'G' WITH REACTOR LENGTH 'Z'

The change in the values of molar fraction of A as well as B, conversion of component A and the total volumetric flow rate, G along the reactor length can be seen in Fig 3.1. The mole fraction of A (red curve) can be seen decreasing while the mol fraction of B (blue curve) can be seen increasing. This is due to the fact that reactant A is disappearing and converting to B which is evident from the graph of increasing Xa (dotted orange curve). Since the final value of yA is not equal to zero, we can see that the conversion Xa doesn't reach to value 1. The green dashed curve shows the total volumetric flow rate G which is increasing. This is because the stochiometric value of m is 2, so, the reaction increases the flow rate.

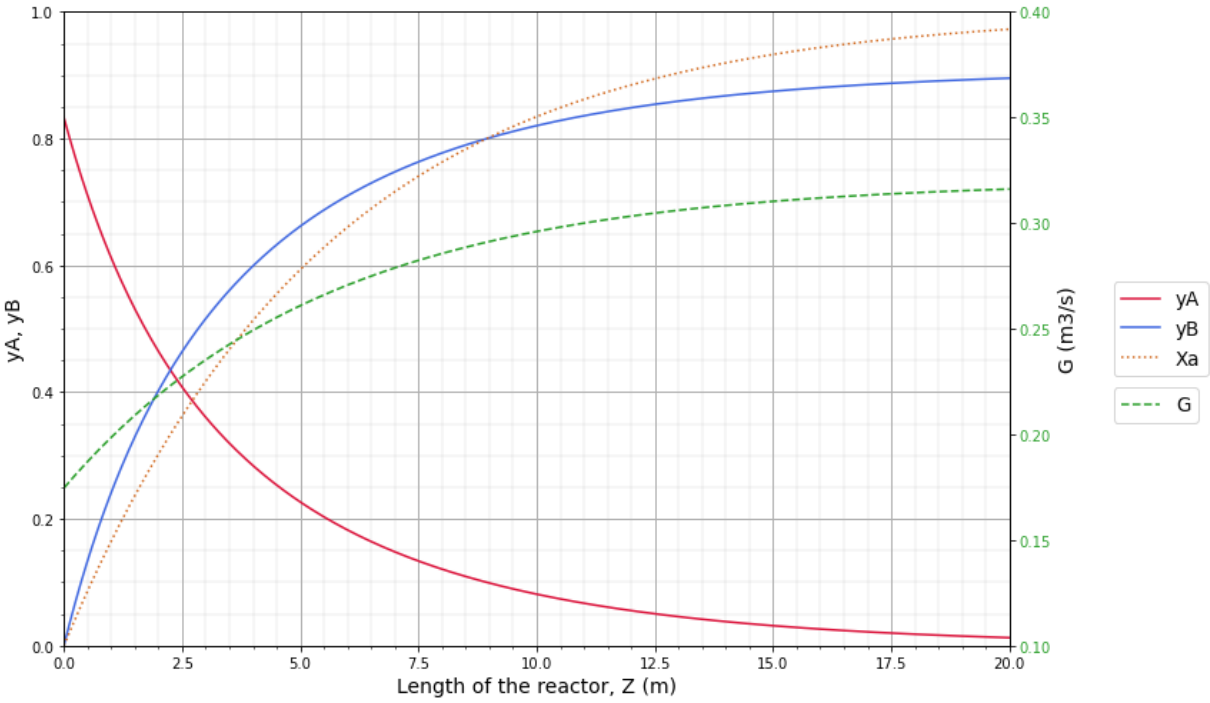


Fig 3.1: Change in yA, yB, Xa and G with the reactor length

3.2) VARIATION OF STOCHIOMETRY OF ‘m’ TO SEE THE INFLUENCE IN G

The volumetric flow rate G is proportional to ‘m’ because G = G0 * (1+ϵXa) where ϵ = m - 1. So increasing the value of ‘m’ increases the flow rate correspondingly which can be observed in Fig 3.2. As stated in the problem, when the value of ‘m’ = 1, there is no change in G throughout the reactor which is shown by the blue line.

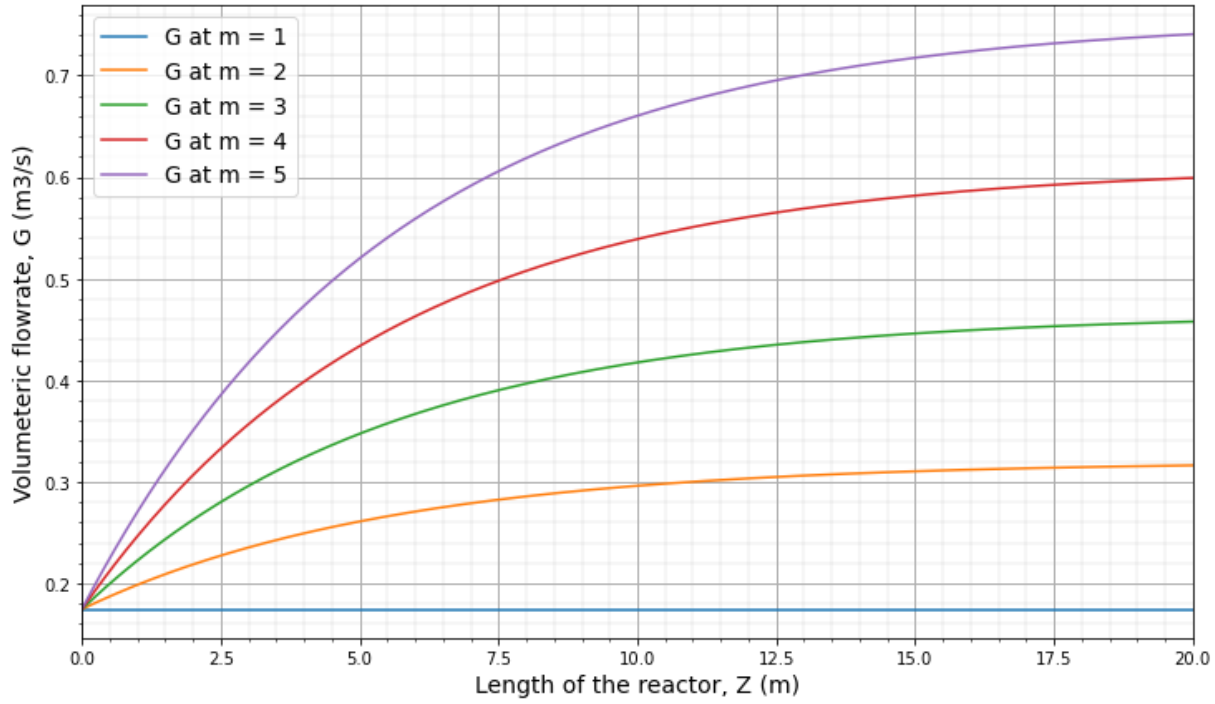


Fig 3.2: Variation in volumetric flow rate G along the reactor length Z with changing stochiomteric ‘m’ value

3.3) EFFECTS OF CHANGING INITIAL INERT MOLAR FLOW TO HIGHER VALUE

The variation in the value of G was observed when setting the initial inert molar flow to higher values from 1 to 100 kmol/s. The observed variation was plotted and shown in Fig 3.3. From the graph, it shows that the value of the G increases but there is no significant change in the slope of the curve while changing Ni0 values. This is because when increasing the N_I, the value of G is dominated by N_I ie. $N_I >>> N_A, N_B$. So, there is no significant change in the slope of G seen along the length of the reactor.

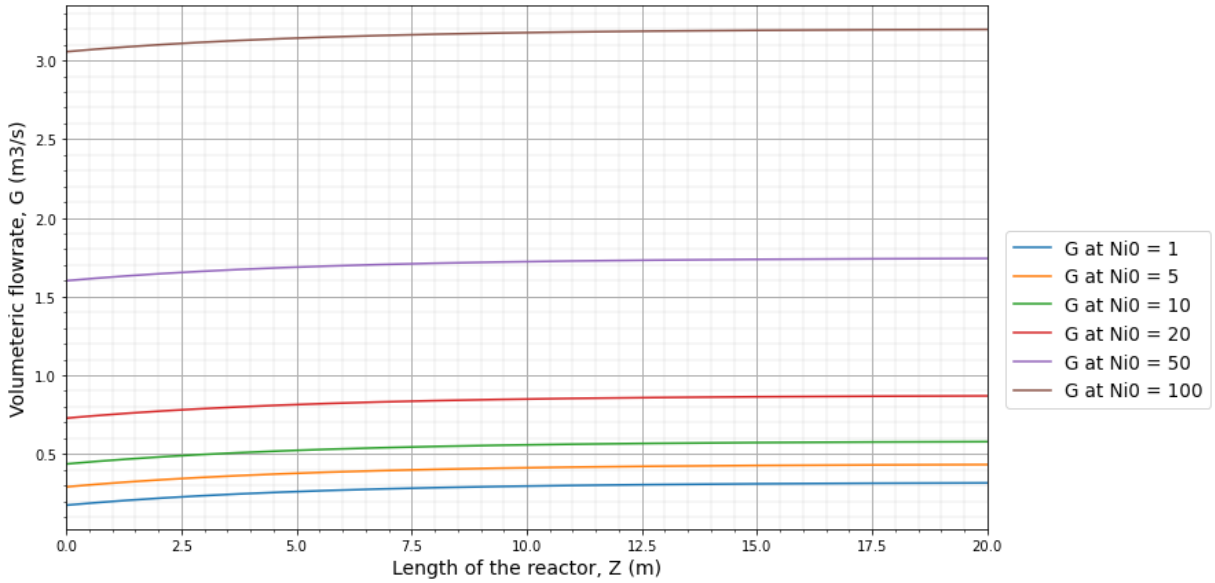


Fig 3.3: Variation in volumetric flow rate G along the reactor length Z with changing initial inert molar flow rate Ni0 to higher values

3.4) TEMPERATURE PROFILE

The change in volumetric flow rate G along the reactor length can be seen increasing with increased initial feed temperature in Fig 3.4. This is because G is directly proportional to T, from the equation 4. However, we observed that no change in the conversion is seen and the conversion curve is constant for initial temperature of 200, 300 and 400K as there is no equation of conversion defined that depends upon the temperature of the reactor. This is usually given by Arrhenius equation in non-isothermal condition. The variation of conversion Xa with temperature is discussed in 3.6.

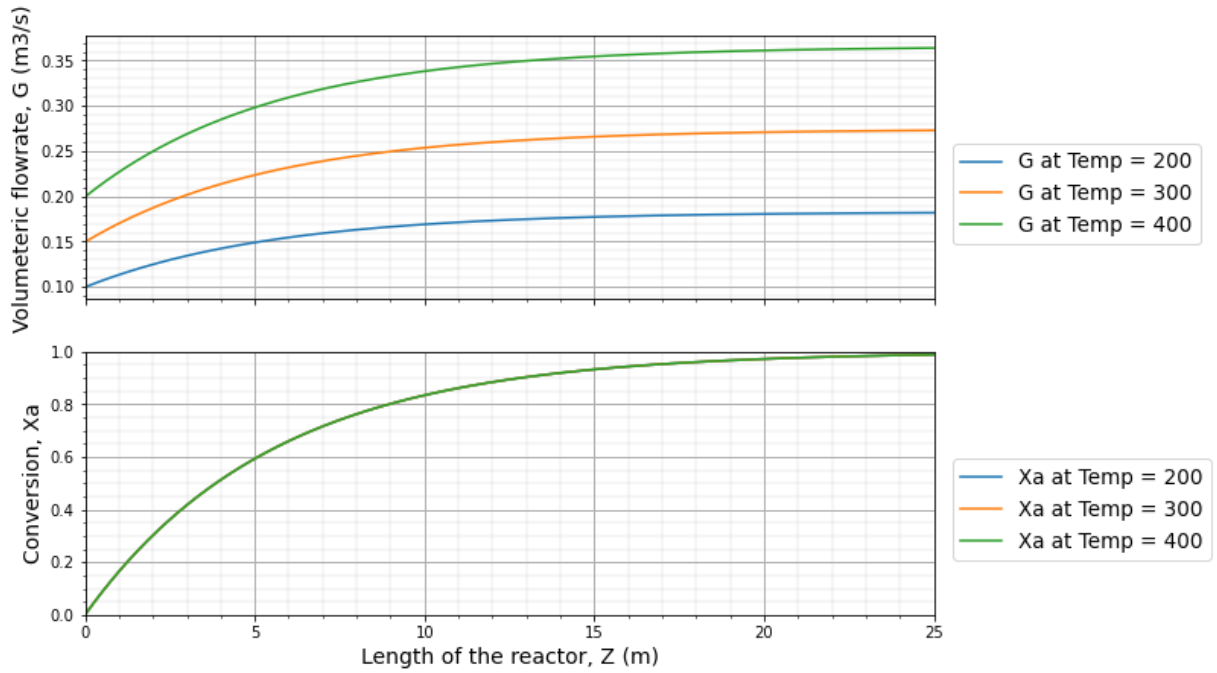


Fig 3.4: Variation in volumetric flow rate G and the fractional conversion Xa along the reactor length Z with changing initial feed temperature to 200, 300 and 400K

3.5) PRESSURE PROFILE

From Fig 3.5, we can deduce that the total volumetric flow rate G of the reactor increases as the inlet feed pressure is decreased. This is expected as G is inversely proportional to the pressure P which can be seen in equation 4. For smaller pressure value of 50,000 Pa, it shows that there is significant change in volumetric flow G along the reactor position than at higher pressures of 200,000 and 500,000 Pa. In the second graph of Fig 3.5, we can observe the pressure drop across the reactor.

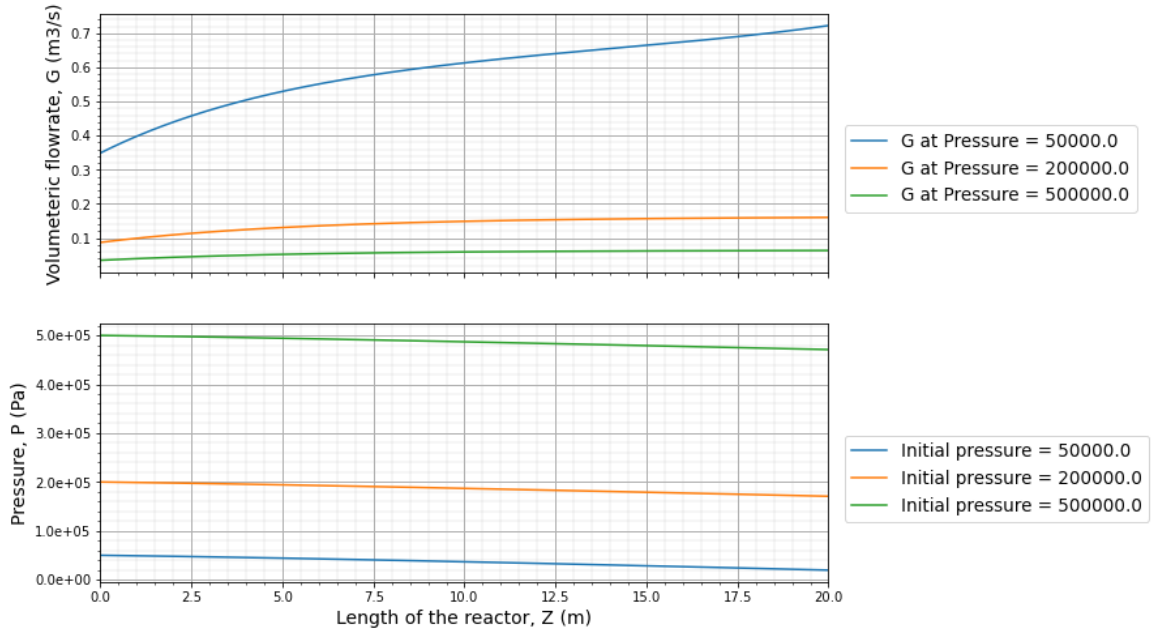
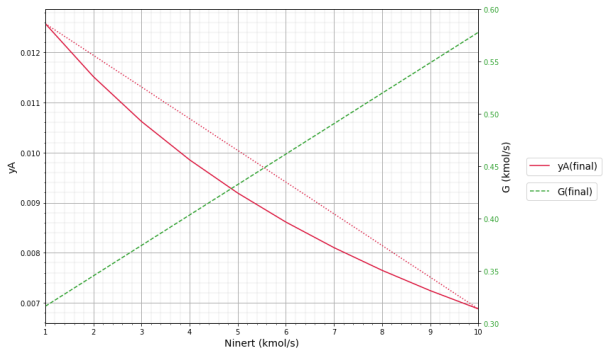


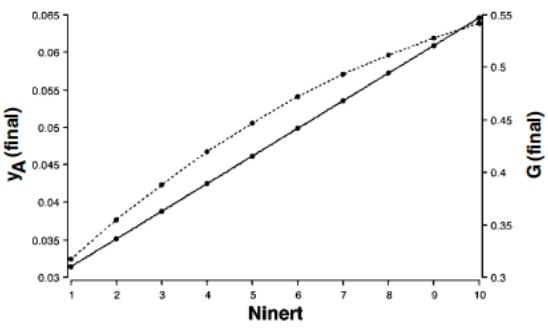
Fig 3.5: Variation of volumetric flow rate G and the pressure drop across the reactor length Z at changing initial pressure of reactor at 50000, 200000 and 500000 Pa

3.6) COMPARISON OF RESULTS WITH THE BOOK

The graph of yA and G is plotted as a function of molar flow rate of inerts Ninert in Fig 3.6. The graph obtained from modelling (a) shows that with increases in molar flow rate of inerts the total volumetric flow rate also increases as molar flow rate of inert is added while calculating G. Similarly, value of yA decreases as yA is obtained from dividing by G. So, as G increases, yA must decreases. However in the graph from book (b), it is shown both values of yA and G increasing with increasing molar flow rate of inerts.



(a)



(b)

Fig 3.6: yA and G as a function of molar flow rate of inerts, (a) graph obtained modelling (b) graph snipped from the book[2]

From the Fig 3.7, we can see that the total volumetric flow rate increases as increase in T which is discussed in 3.5. However, the graph of X vs Z shows that with increasing temperature, the conversion decreases. This indicates that the gas-pahse reaction is an exothermic reaction. From the graph, we can estimate that the length of the reactor can be reduced to 12.5 m to obtain the total conversion of A to B in all operating conditions of 200, 300 and 400K with temperature of 200K having the shortest length of reactor required.

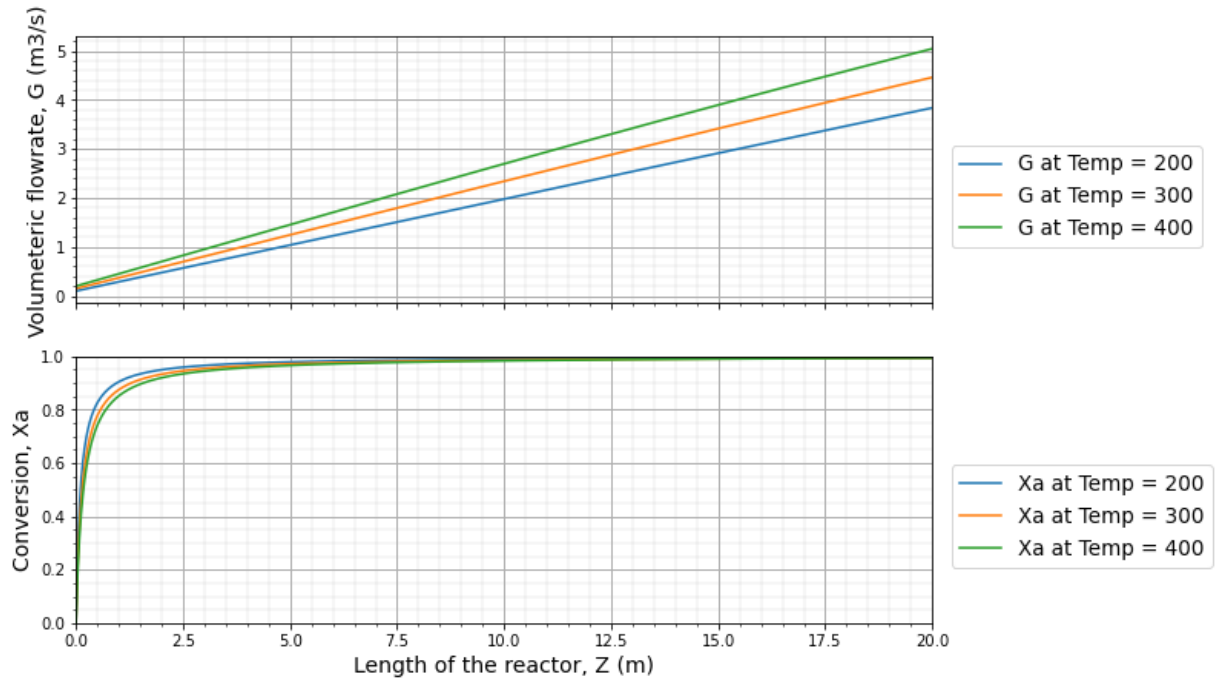


Fig 3.7: change in G and X values along the reactor length Z in a non-isothermal condition in the reactor

4. CONCLUSION

The conclusion to be drawn from this modelling report is listed as:

- The volumetric flowrate remained constant throughout the system for stoichiometry m = 1 and the slope of curve gets steeper with increasing ‘m’
- There is no spatial/slope change but just initial and final value change while setting the molar flow of inerts to higher values
- The volumetric flow rate increases as the temperature increases and the conversion decreases with increasing temperature, hinting the reaction as an exothermic reaction
- There is a pressure drop across the reactor and volumetric flow rate decreases with the increasing initial pressure

5. REFERENCE

1. Nikolay (2021) *Plug flow (tubular) reactor*, *Stoli Chem*. Available at: https://stolichem.com/plug-flow-tubular-reactor/ (Accessed: March 13, 2023).

2. Ingham, J. *et al.* (1994) *Chemical Engineering Dynamics: Modelling with PC simulation*. Weinheim, Basel: VCH.

APPENDIX

The codes are attached in the following page.

VARMOL

RESPONSE OF **yA**, **yB**, **Xa** AND **G** ALONG THE REACTOR LENGTH **Z**

In [14]:
```
# importing the necessary libraries for equations solving and visualizing
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as pl
```

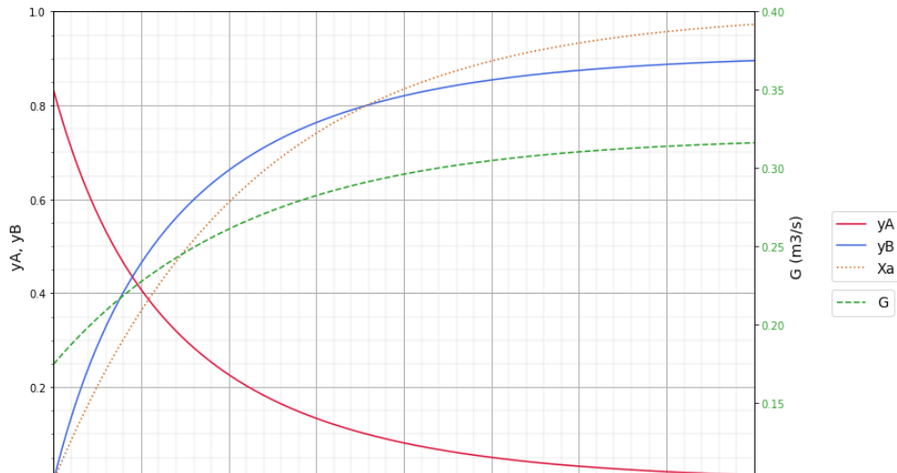
In [2]:
```
# initial parameter values
R = 8.3144                      # gas constant, (J/mol.K)
T = 350                         # Reaction temperature, (K)
P = 1.e5                        # Pressure in PFR, (N/m^2)
k = 15                          # Reaction rate constant, (1/s)
m = 2                           # stoichiometric constant
Na0 = 5                         # initial molar concentration of component A, (mol/s)
Nb0 = 0                         # initial molar concentration of component B, (mol/s)
Ni0 = 1                         # initial molar concentration of inert I, (mol/s)
A = 0.012                       # cross-sectional area, (m^2)

# formulas to calculate initial value of ODEs
NT0 = Na0 + Nb0 + Ni0
G0 = (NT0 * R * T) / P          # initial volumetric flow rate
ya0 = Na0 / NT0                 # mole fraction of component A
yb0 = Nb0 / NT0                 # mole fraction of component B
yi0 = Ni0 / NT0                 # mole fraction of inert I
yaG = ya0 * G0
ybG = yb0 * G0

# number of intervals and final length of the reactor
cint = 500
Zfinal = 20
```

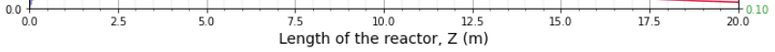
In [3]:
```
# function created to feed into ode solver that returns the intergrated value of the ODEs
def model_f(x, Z):
    ya = x[0]

    # ODEs to be solved
    dyaGdZ = - k * ya * A
    dybGdz = m * k * ya * A

    #returns the integrated value when used in ode solver at given 'Z'
    return [dyaGdZ, dybGdz]
```

In [4]:
```
#equal spaced length intervals created to use while plotting the intergrated data
Z = np.linspace(0, Zfinal, cint)

#initial value array
initial = [yaG, ybG]

"""python ode solver that takes differential equations, initital guess and the interval of independent variable at which
the solver solves the equations and returns an array of solutions"""
sol = odeint(model_f, initial, Z)
```

In [5]:
```
# addition equations to find the value of required G
Na = (sol[:,0] * P) / (R * T)
Nb = (sol[:,1] * P) / (R * T)
G = ((Na + Nb + Ni0) * R * T) / P
yA = sol[:,0] / G
yB = sol[:,1] / G
yI = Ni0 / G
Xa = (Na0 - Na) / Na0
```

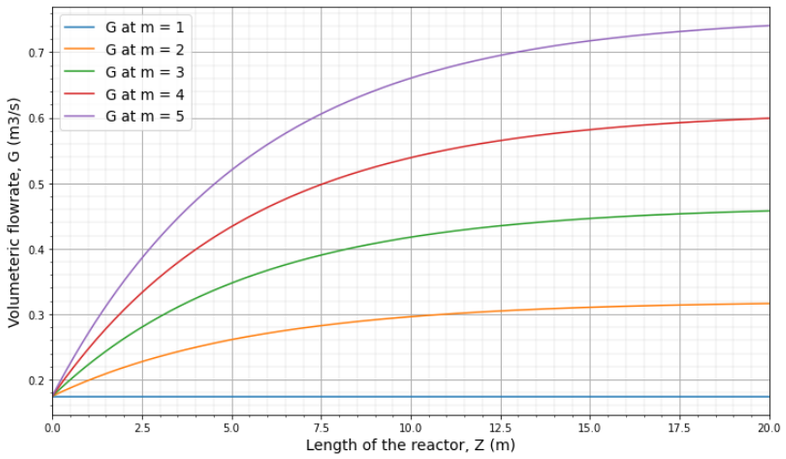
In [6]:
```
# creating a subplot to fit two y-axis and increasing the output figure size
fig, plt = pl.subplots(figsize=(12,7))

#codes to plot the time and integrated values
plt.plot(Z, yA, color = "crimson", label = "yA")
plt.plot(Z, yB, color = "royalblue", label = "yB")
plt.plot(Z, Xa, color = "chocolate", label = "Xa", ls=":")

# plot description
plt.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.1,0.5))
plt.set_xlim(0,Zfinal)
plt.set_ylim(0,1)
plt.set_xlabel("Length of the reactor, Z (m)", fontsize=14)
plt.set_ylabel("yA, yB", fontsize=14)

# subplot with the same x-axis as the above one
ax = plt.twinx()
color = "tab:green"

# plotting the G vs Z graph in the above graph
ax.plot(Z, G, color = color, linestyle = "--", label = "G")

# to make the line color same as y-axis data color
ax.tick_params(axis='y', labelcolor=color)
ax.set_ylim(0.1,0.4)
ax.set_ylabel("G (m3/s)", fontsize=14)
ax.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.1,0.38))

# plot description
plt.grid(which='minor', linewidth=0.2)
plt.grid(which='major', linewidth=1)
plt.minorticks_on()
pl.tight_layout()
pl.show()
```





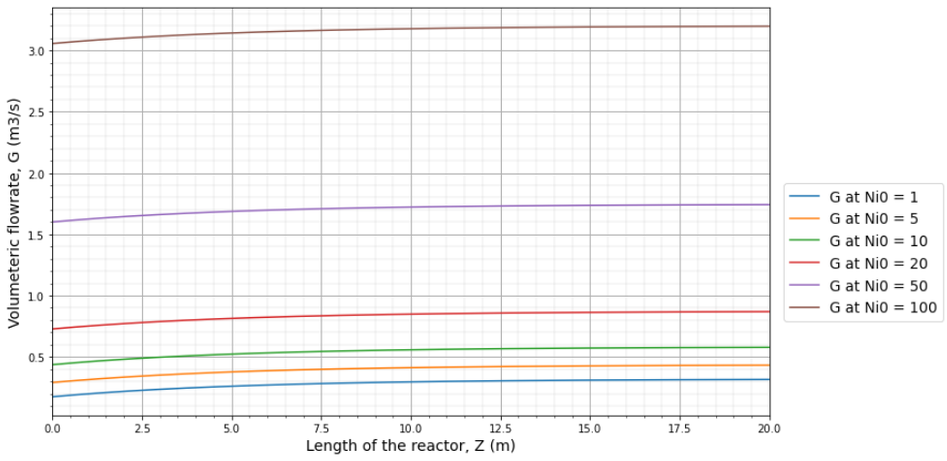
1. Vary the stoichiometry to see the influence of m. Note that if m=1, G must be constant through the reactor.

In [7]:
```
# various values of m to see the influence in G
mchange = [1, 2, 3, 4, 5]

# increases the output plot size
fig = pl.figure(figsize=(12,7))

# loop to plot the in a same graph of various G values with changing m values
for i in range(5):
    m = mchange[i]
    sol = odeint(model_f, initial, Z)
    Na = (sol[:,0] * P) / (R * T)
    Nb = (sol[:,1] * P) / (R * T)
    G = ((Na + Nb + Ni0) * R * T) / P
    pl.plot(Z, G, label = "G at m = {}".format(m))
    pl.grid(which='minor', linewidth=0.2)
    pl.grid(which='major', linewidth=1)
    pl.minorticks_on()
    pl.legend(fontsize=13.5)
    pl.xlim(0, Zfinal)

pl.ylabel("Volumeteric flowrate, G (m3/s)", fontsize=14)
pl.xlabel("Length of the reactor, Z (m)", fontsize=14)
pl.show()
```



2. Set the molar feed rate of inerts to a high value, and note that G does not change much with position.

In [8]:
```
m = 2
Ni_change = np.array([1, 5, 10, 20, 50, 100])
fig = pl.figure(figsize=(12,7))

for i in range(Ni_change.shape[0]):
    Ni0 = Ni_change[i]
    sol = odeint(model_f, initial, Z)
    Na = (sol[:,0] * P) / (R * T)
    Nb = (sol[:,1] * P) / (R * T)
    G = ((Na + Nb + Ni0) * R * T) / P

    pl.plot(Z, G, label = "G at Ni0 = {}".format(Ni0))
    pl.grid(which='minor', linewidth=0.2)
    pl.grid(which='major', linewidth=1)
    pl.minorticks_on()
    pl.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    pl.xlim(0, Zfinal)

pl.ylabel("Volumeteric flowrate, G (m3/s)", fontsize=14)
pl.xlabel("Length of the reactor, Z (m)", fontsize=14)
pl.show()
```

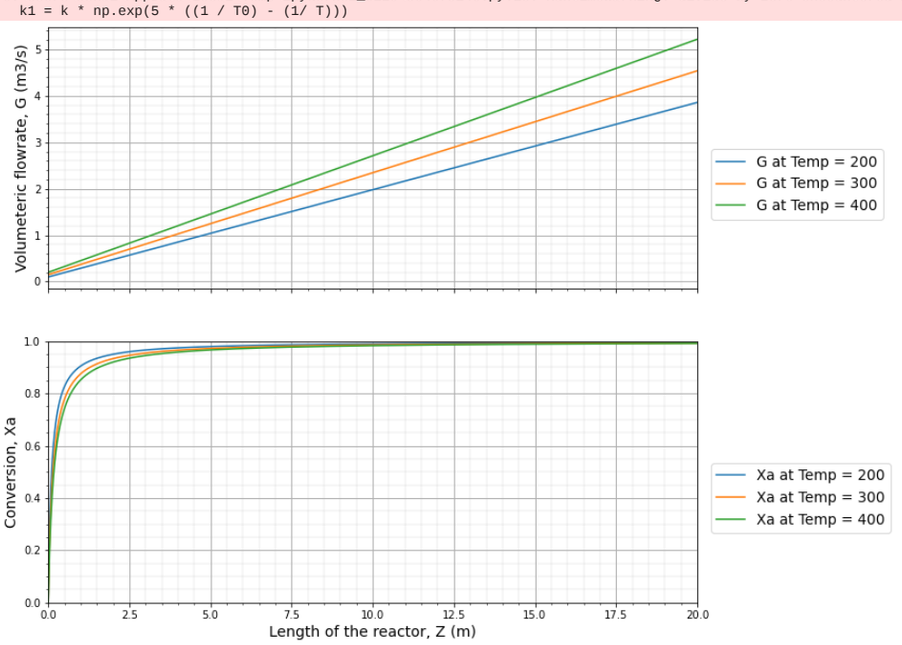


3. Change the model and program to account for a linear temperature profile.

In [9]:
```
Ni0 = 1
m = 2

T_change = np.array([200, 300, 400])

cint = 500
Zfinal = 20

def model_new(x, Z):
    ya0 = x[0]
    T = x[1]
```

```
    k1 = k * np.exp(5 * ((1 / T0) - (1/ T)))
    dyaGdZ = (- k1 * ya0 * A)
    dybGdZ = m * k * ya0 * A
    dTdZ = 1500 - 2 * (T - T0)

    return [dyaGdZ, dybGdZ, dTdZ]

Z = np.linspace(0, Zfinal, cint)


fig, (plt, ax) = pl.subplots(2, 1, sharex = True, figsize=(11,10))


for i in range(T_change.shape[0]):
    T0 = T_change[i]
    G0 = (NT0 * R * T0) / P        # initial volumetric flow rate
    ya0 = Na0 / NT0               # mole fraction of component A
    yb0 = Nb0 / NT0               # mole fraction of component B
    yi0 = Ni0 / NT0               # mole fraction of inert I
    yaG = ya0 * G0
    ybG = yb0 * G0
    initial = [yaG, ybG, T0]
    sol = odeint(model_new, initial, Z)
    Na = (sol[:,0] * P) / (R * sol[:,2])
    Nb = (sol[:,1] * P) / (R * sol[:,2])
    G = ((Na + Nb + Ni0) * R * sol[:,2]) / P
    Xa = (Na0 - Na) / Na0
    plt.plot(Z, G, label = "G at Temp = {}".format(T_change[i]))
    plt.grid(which='minor', linewidth=0.2)
    plt.grid(which='major', linewidth=1)
    plt.minorticks_on()
    plt.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    plt.set_xlim(0, Zfinal)
    plt.set_ylabel("Volumeteric flowrate, G (m3/s)", fontsize=14)

    ax.plot(Z, Xa, label = "Xa at Temp = {}".format(T_change[i]))
    ax.grid(which='minor', linewidth=0.2)
    ax.grid(which='major', linewidth=1)
    ax.minorticks_on()
    ax.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    ax.set_xlim(0, Zfinal)
    ax.set_ylim(0, 1)


ax.set_ylabel("Conversion, Xa", fontsize=14)
ax.set_xlabel("Length of the reactor, Z (m)", fontsize=14)
pl.show()
```

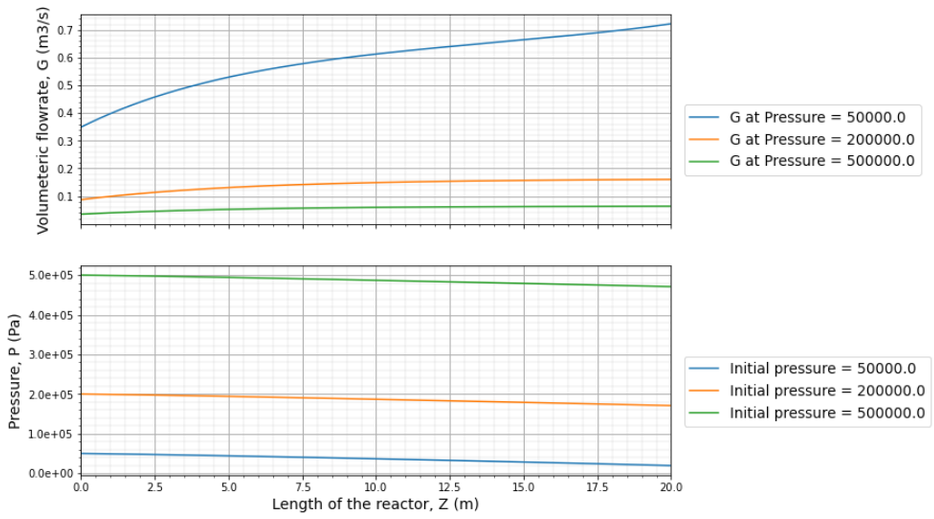
```
C:\Users\ashal\AppData\Local\Temp\ipykernel_6120\564675146.py:13: RuntimeWarning: divide by zero encountered in scalar divide
  k1 = k * np.exp(5 * ((1 / T0) - (1/ T)))
C:\Users\ashal\AppData\Local\Temp\ipykernel_6120\564675146.py:13: RuntimeWarning: divide by zero encountered in scalar divide
  k1 = k * np.exp(5 * ((1 / T0) - (1/ T)))
C:\Users\ashal\AppData\Local\Temp\ipykernel_6120\564675146.py:13: RuntimeWarning: divide by zero encountered in scalar divide
  k1 = k * np.exp(5 * ((1 / T0) - (1/ T)))
```



4. Change the model and program to account for a linear pressure profile, allowing for pressure drop through the reactor.

In [10]:
```
import matplotlib.ticker as mtick

Ni0 = 1
m = 2
T0 = 350
P0 = 1e5
P_change = np.array([0.5e5, 2e5, 5e5])
beta = 900

cint = 500
Zfinal = 20

Z = np.linspace(0, Zfinal, cint)

fig, (plt, ax) = pl.subplots(2, 1, sharex = True, figsize=(10,8))

for i in range(P_change.shape[0]):

    P0 = P_change[i]
    G0 = (NT0 * R * T0) / P0
    ya0 = Na0 / NT0
    yb0 = Nb0 / NT0
    yi0 = Ni0 / NT0
    yaG = ya0 * G0
    ybG = yb0 * G0
    initial = [yaG, ybG, P0]

    def model_new(x, Z):
        ya0 = x[0]
        yb0 = x[1]
        P = x[2]

        Na = (ya0 * P) / (R * T0)
        Nb = (yb0 * P) / (R * T0)
        NT = Na + Nb + Ni0
```

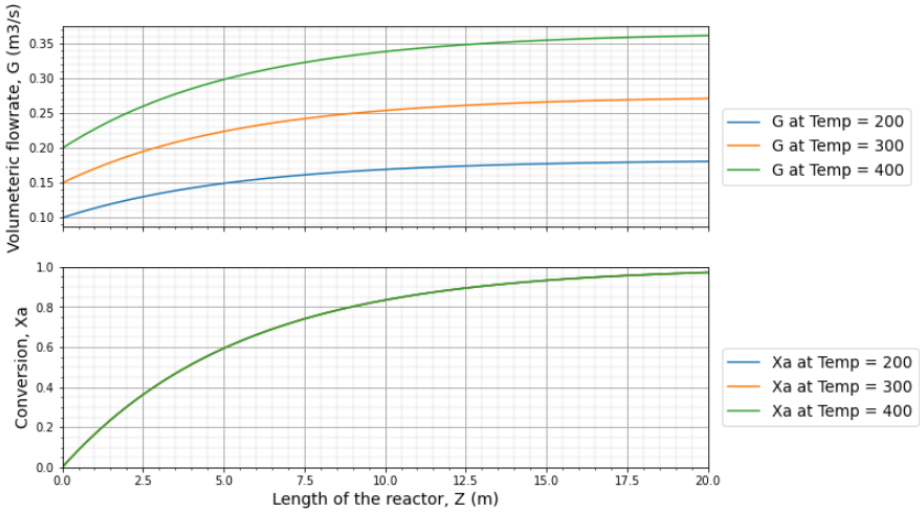
```
        dyaGdZ = (- k * ya0 * A)
        dybGdZ = m * k * ya0 * A
        dpdZ = -beta * (P0 / P) * (NT / NT0)

        return [dyaGdZ, dybGdZ, dpdZ]

    sol = odeint(model_new, initial, Z)
    Na = (sol[:,0] * sol[:,2]) / (R * T0)
    Nb = (sol[:,1] * sol[:,2]) / (R * T0)
    NT = Na + Nb + Ni0
    G = (NT * R * T0) / sol[:,2]
    Xa = (Na0 - Na) / Na0
    plt.plot(Z, G, label = "G at Pressure = {}".format(P_change[i]))
    plt.grid(which='minor', linewidth=0.2)
    plt.grid(which='major', linewidth=1)
    plt.minorticks_on()
    plt.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    plt.set_xlim(0, Zfinal)
    plt.set_ylabel("Volumeteric flowrate, G (m3/s)", fontsize=14)

    ax.plot(Z, sol[:,2], label = "Initial pressure = {}".format(P_change[i]))
    ax.grid(which='minor', linewidth=0.2)
    ax.grid(which='major', linewidth=1)
    ax.minorticks_on()
    ax.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    ax.set_xlim(0, Zfinal)

ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.1e'))
ax.set_ylabel("Pressure, P (Pa)", fontsize=14)
ax.set_xlabel("Length of the reactor, Z (m)", fontsize=14)
pl.show()
```



Compare the results obtained by solving the problems to results presented in Ingham (1994 & 2007 ed.) textbooks.

In [13]:

```
R = 8.3144                      # gas constant, (J/mol.K)
P = 1.e5                        # Pressure in PFR, (N/m^2)
k = 15                          # Reaction rate constant, (1/s)
m = 2                           # stoichiometric constant
Na0 = 5                         # initial molar concentration of component A, (mol/s)
Nb0 = 0                         # initial molar concentration of component B, (mol/s)
Ni0 = 1                         # initial molar concentration of inert I, (mol/s)
A = 0.012                       # cross-sectional area, (m^2)


T_change = np.array([200, 300, 400])

cint = 500
Zfinal = 20

def model_f(x, Z):
    ya0 = x[0]

    dyaGdZ = - k * ya0 * A
    dybGdz = m * k * ya0 * A

    return [dyaGdZ, dybGdz]

Z = np.linspace(0, Zfinal, cint)


fig, (plt, ax) = pl.subplots(2, 1, sharex = True, figsize=(10,7))


for i in range(T_change.shape[0]):
    T = T_change[i]
    G0 = (NT0 * R * T) / P        # initial volumetric flow rate
    ya0 = Na0 / NT0               # mole fraction of component A
    yb0 = Nb0 / NT0               # mole fraction of component B
    yi0 = Ni0 / NT0               # mole fraction of inert I
    yaG = ya0 * G0
    ybG = yb0 * G0
    initial = [yaG, ybG]
    sol = odeint(model_f, initial, Z)
    Na = (sol[:,0] * P) / (R * T)
    Nb = (sol[:,1] * P) / (R * T)
    G = ((Na + Nb + Ni0) * R * T) / P
    Xa = (Na0 - Na) / Na0
    plt.plot(Z, G, label = "G at Temp = {}".format(T))
    plt.grid(which='minor', linewidth=0.2)
    plt.grid(which='major', linewidth=1)
    plt.minorticks_on()
    plt.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    plt.set_xlim(0, Zfinal)
    plt.set_ylabel("Volumeteric flowrate, G (m3/s)", fontsize=14)

    ax.plot(Z, Xa, label = "Xa at Temp = {}".format(T))
    ax.grid(which='minor', linewidth=0.2)
    ax.grid(which='major', linewidth=1)
    ax.minorticks_on()
    ax.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.01,0.4))
    ax.set_xlim(0, Zfinal)
    ax.set_ylim(0, 1)


ax.set_ylabel("Conversion, Xa", fontsize=14)
ax.set_xlabel("Length of the reactor, Z (m)", fontsize=14)
pl.show()
```



In [12]:

```
m = 2
T = 350
G1 = np.zeros(10)
yA_final = np.zeros(10)
Ninert = np.zeros(10)

for i in range(0, 10, 1):
    Ninert[i] = i+1
    NT0 = Na0 + Nb0 + Ninert[i]
    G0 = (NT0 * R * T) / P
    ya0 = Na0 / NT0
    yb0 = Nb0 / NT0
    yi0 = Ninert / NT0
    yaG = ya0 * G0
    ybG = yb0 * G0
    initial = [yaG, ybG]
    sol = odeint(model_f, initial, Z)
    Na1 = (sol[-1][0] * P) / (R * T)
    Nb1 = (sol[-1][1] * P) / (R * T)
    G1[i] = ((Na1 + Nb1 + Ninert[i]) * R * T) / P
    yA_final[i] = sol[-1][0] / G1[i]

fig, plt = pl.subplots(figsize=(12,7))
appl = np.linspace(max(yA_final), min(yA_final), 10)
plt.plot(Ninert, yA_final, color = "crimson", label = "yA(final)")
plt.plot(Ninert, appl, color = "crimson", ls = ":")
plt.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.1,0.5))
plt.set_xlim(1,10)
plt.set_xlabel("Ninert (kmol/s)", fontsize=14)
plt.set_ylabel("yA", fontsize=14)

ax = plt.twinx()
color = "tab:green"
ax.plot(Ninert, G1, color = color, linestyle = "--", label = "G(final)")
ax.tick_params(axis='y', labelcolor=color)
ax.set_ylim(0.3,0.6)
ax.set_ylabel("G (kmol/s)", fontsize=14)
ax.legend(fontsize=13.5, loc = "center left", bbox_to_anchor=(1.1,0.42))

plt.grid(which='minor', linewidth=0.2)
plt.grid(which='major', linewidth=1)
plt.minorticks_on()
pl.tight_layout()
pl.show()
```