Generated at Sat Dec 10 04:33:20 2022.

Source at https://jupyterhub-dev.cheme.cmu.edu/user/sgudipud@andrew.cmu.edu/lab/tree/f22-06623/assignments/project/project.ipynb.

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

```
In [ ]: NAME = "Srinivas Gudipudi"
        COLLABORATORS = "None"
```

Final project

Project

Your final project will be a PDF derived from a Jupyter notebook. Your notebook should read like a technical report, with sections to organize the contents, complete sentences, and paragraphs with interspersed code cells and results. The lecture notes are a reasonable model for what I am expecting from you. There is not a hard length; it should be appropriate for your topic.

Your report will be graded in these areas.

Technical content

You should explain what your code is doing, the code should follow the style and conventions used in the lecture notes. You should also explain what your results mean, and provide some context around them. For example, if you are reproducing some analysis, do your results agree? If not, why? If you are developing an algorithm how can you tell if it works? What are the benefits of it compared to others?

Your work should be original. I do not want to see small derivatives of existing work, or copied examples from the internet. If you used existing works, you should take care to cite them (with a URL to access them).

Presentation

I expect full sentences and paragraphs, with limited typos and grammatical mistakes. If you have used external content for something you should have reference (a URL is fine) to indicate that.

The writing should be suitable for a non-expert to read, especially consider what a job recruiter would think about it. Would they be interested in hiring you after reading it? Would a first-year MS student understand what you have written and learn from it.

Typical sections

It is typical to see these kinds of sections in your report. These are not required.

Introduction

Background and Motivation for taking up this project

Hi ! Welcome to my project .

I am going to be talking about a library called Thermo Library.

Ugh!!! One more library to learn! Why do we have to learn about so many Libraries ! We already had enough .

So , I believe Computers if fed with right amount of data and inputs, can processs it and give out much better results with much better precision and accuracy and it just keeps getting better with training more data , which is not the case with humans , any program written by a human from scratch is going to be getting error .So, to make our lives easier lots of Libraries are there , where we can use the functions from them directly.

Thermodynamic properties and Equation of states are used for almost any application in the world, we as a chemical engineers use this widely and we need to have a proper knowledge about the thermodynamics of a process , to troubleshoot any problems in the chemical industries . Bascially any complex problem would eventually leads us to the understanding about the thermodynamics of the process . So it becomes crucial to get to know about the properties and EOS in an industry or academia setting , wherein one wants to have a quick glance of how the whole process is going on , as the Industry 4.0 is approaching , we are trying to get rid of physical form of storing all the thermodynamic data or any records for that instance . So , I want to briefly explain about the Properties and EOS in the Thermo Package

So lets Get Started !

Like for any module installation , we use pip install thermo for python with pip

conda install -c conda-forge thermo # For the conda package manager

git clone git://github.com/CalebBell/thermo.git # for the git version

Thermo Library Introduction

What is Thermo?

Thermo is a open-source software . It facilitates **the retrieval of constants of chemicals**, the calculation of temperature and pressure dependent chemical properties (both thermodynamic and transport), and the calculation of the same for chemical mixtures (including phase equilibria) using various models.

Thermo is quick to install, and is free of charge. Thermo is designed to be easy to use while still providing powerful functionality. If you need to know something about a chemical or mixture.

The Whole documentation can be found in [1]

We import thermo in the following way , we generally import a function called Chemical which gives us information about that particular chemical we are going to use .

```
In [7]: import thermo
        import numpy as np
        import scipy as sp
        from thermo.chemical import Chemical
        import matplotlib.pyplot as plt
```

let's explore some features and constants of Thermo Library

Here in this following example , we can see how we can retrieve the properties of compounds, we could easily retrieve any property we want using the thermo library , We will see it with the code implementation and example to get better insight [2]

We take up 3 chemical compounds and try to see what we can do with the thermo library

```
In [8]: compounds = ['toluene', 'ethane', 'methane', 'ethanol', ]
```

Toluene

```
In [9]: compound  = Chemical(compounds[0])    # the imported Chemical is used here and it finds about the Compound list
```

```
In [10]: print(f'Temperature of the chemical [k]                  ---->  {compound.T}', '\n',
               f'Pressure of the chemical, [Pa]                   ---->  {compound.P}', '\n',
               f'Phase of the chemical                            ---->  {compound.phase}', '\n',
               f'The CAS number of the chemical                   ---->  {compound.CAS}', '\n',
               f'Molecular weight of the compound, [g/mol]------>  {compound.MW}', '\n',
               f'Molecular formula of the compound                ---->  {compound.formula}', '\n',
               f'Critical temperature [K]                         ---->  {compound.Tc}', '\n',
               f'Acentric factor [-]                              ---->  {compound.omega}', '\n',
               )
```

```
Temperature of the chemical [k]                ---->  298.15
 Pressure of the chemical, [Pa]                ---->  101325
 Phase of the chemical                         ---->  l
 The CAS number of the chemical                ---->  108-88-3
 Molecular weight of the compound, [g/mol]---->  92.13842
 Molecular formula of the compound             ---->  C7H8
 Critical temperature [K]                      ---->  591.75
 Acentric factor [-]                           ---->  0.257
```

So, Let us look at the results, we could find out the various properties of the selected chemical , like temperature , pressure , phase etc, which we can easily retrieve through this library and can use this quickly. So, we basically get the major properties for a compound and we can get more as well from the thermo library .

Pretty Cool right!

The next few code blocks are estimating the properties of all the compounds left in the compound list. This is done in a repetitive way to make the user and reader comfortable getting used to terminology

Ethane

```
In [11]: compound  = Chemical(compounds[1])
         print(f'Temperature of the chemical [k]                  ---->  {compound.T}', '\n',
               f'Pressure of the chemical, [Pa]                   ---->  {compound.P}', '\n',
               f'Phase of the chemical                            ---->  {compound.phase}', '\n',
               f'The CAS number of the chemical                   ---->  {compound.CAS}', '\n',
               f'Molecular weight of the compound, [g/mol]------>  {compound.MW}', '\n',
               f'Molecular formula of the compound                ---->  {compound.formula}', '\n',
               f'Critical temperature [K]                         ---->  {compound.Tc}', '\n',
               f'Acentric factor [-]                              ---->  {compound.omega}', '\n',
               )
```

```
Temperature of the chemical [k]                ---->  298.15
 Pressure of the chemical, [Pa]                ---->  101325
 Phase of the chemical                         ---->  g
 The CAS number of the chemical                ---->  74-84-0
 Molecular weight of the compound, [g/mol]---->  30.06904
 Molecular formula of the compound             ---->  C2H6
 Critical temperature [K]                      ---->  305.32
 Acentric factor [-]                           ---->  0.098
```

Methane

```
In [12]: compound  = Chemical(compounds[2])
         print(f'Temperature of the chemical [k]                  ---->  {compound.T}', '\n',
               f'Pressure of the chemical, [Pa]                   ---->  {compound.P}', '\n',
               f'Phase of the chemical                            ---->  {compound.phase}', '\n',
               f'The CAS number of the chemical                   ---->  {compound.CAS}', '\n',
               f'Molecular weight of the compound, [g/mol]------>  {compound.MW}', '\n',
               f'Molecular formula of the compound                ---->  {compound.formula}', '\n',
               f'Critical temperature [K]                         ---->  {compound.Tc}', '\n',
               f'Acentric factor [-]                              ---->  {compound.omega}', '\n',
               )
```

```
Temperature of the chemical [k]                ---->  298.15
 Pressure of the chemical, [Pa]                ---->  101325
 Phase of the chemical                         ---->  g
 The CAS number of the chemical                ---->  74-82-8
 Molecular weight of the compound, [g/mol]---->  16.04246
 Molecular formula of the compound             ---->  CH4
 Critical temperature [K]                      ---->  190.564
 Acentric factor [-]                           ---->  0.008
```

```
In [13]: ###Ethanol
```

```
In [14]: compound  = Chemical(compounds[3])
         print(f'Temperature of the chemical [k]                  ---->  {compound.T}', '\n',
               f'Pressure of the chemical, [Pa]                   ---->  {compound.P}', '\n',
               f'Phase of the chemical                            ---->  {compound.phase}', '\n',
               f'The CAS number of the chemical                   ---->  {compound.CAS}', '\n',
               f'Molecular weight of the compound, [g/mol]------>  {compound.MW}', '\n',
               f'Molecular formula of the compound                ---->  {compound.formula}', '\n',
               f'Critical temperature [K]                         ---->  {compound.Tc}', '\n',
               f'Acentric factor [-]                              ---->  {compound.omega}', '\n',
               )
```

```
Temperature of the chemical [k]                ---->  298.15
 Pressure of the chemical, [Pa]                ---->  101325
 Phase of the chemical                         ---->  l
 The CAS number of the chemical                ---->  64-17-5
 Molecular weight of the compound, [g/mol]---->  46.06844
 Molecular formula of the compound             ---->  C2H6O
 Critical temperature [K]                      ---->  514.0
 Acentric factor [-]                           ---->  0.635
```

That is a good sample of what constant you could retrieve from Thermo library

for more information use the following commands

```
help(thermo),
dir(thermo)
#and for spesific compound
help(Chemical(compound)),
dir(Chemical(compound))
```

Cubic Equation of State

Equation of state is a thermodynamic state variable

Equation of state (EOS)is a equation which relates the state variables which can describe any state of matter under given set of physical conditions such as temperature , Pressure , volume .EOS is generally used to describe the properties of pure substances and mixtures in liquid, solid and gas phase . [3]

The simplest known EOS is the ideal gas equation, which we had used extensively from our childhood. The ideal gas equation tells us that the 4 varibales can not be chnaged independetly , if we know the 3 variables , we can estimate the 4th one . [4]

It is defined by

$$pV = nRT$$

Let V denote the volume of a gas in Litres

Let T denote the Temperature of the gas in

Let R denote the Universal Gas Constant of a gas.

Let P denote the Pressure of the gas.

The problem with the EOS is there is no single EOS that predicts the behavior of all substances under all conditions.

There are various EOS used

1. Peng Robinson
2. Redlich Kwong
3. Soave Redlich Kwong
4. NRTL
5. General NRTL

The ideal gas is a good approximation for non polar gases at low pressures and high temperatures . So , with no universal EOS being present , we should make a documnetation on for what molecules, compounds , which EOS gives the best results , Intially I thought of fitting the same example through all the models , but it wont give me good results , Here I want to explain about each EOS and give an example of each EOS and compare it with other EOS model predictions .

to depict the the state of the desired matter under given conditions like volume, pressure, temperature or internal energy. ALl of those studies could be amazing in authenticating implementation and avoid adultration of pure substances and/or studied mixtures

Peng–Robinson equation PR

The Peng–Robinson equation of state (PR EOS) was developed in 1976 at The University of Alberta by Ding-Yu Peng and Donald Robinson

in order to satisfy the following goals:

1. The parameters should be expressible in terms of the critical properties and the acentric factor.
2. The model should provide reasonable accuracy near the critical point, particularly for calculations of the compressibility factor and liquid density.
3. The mixing rules should not employ more than a single binary interaction parameter, which should be independent of temperature, pressure, and composition.
4. The equation should be applicable to all calculations of all fluid properties in natural gas processes.

![pr equation.png ([3]) (#3)]

Peng Robinson EOS function is used directly from the thermo library , in this case I am trying to estimate the pressures for the compounds mentioned above and using the thermo library , the pressure using the Peng Robinson EOS for each of the compound From calculating the V, T, P_c, T_c, a we can estimate the pressure through peng robinson EOS

The code below is written from scratch , the only purpose of me doing this was to illustrate that things become easier with the direct functions from the library , this is generalised code and can work for any peng robinson model

```
In [15]: import numpy as np
         import matplotlib.pyplot as plt
         from scipy.optimize import root

         R = 8.314e-5  # universal gas constant, m3-bar/K-mol
         class Molecule:
             """
             Store molecule info here
             """
             def __init__(self, name, Tc, Pc, omega):
                 """
                 Pass parameters desribing molecules
                 """
                 #! name
                 self.name = name
                 #! Critical temperature (K)
                 self.Tc = Tc
                 #! Critical pressure (bar)
                 self.Pc = Pc
                 #! Accentric factor
                 self.omega = omega

             def print_params(self):
                 """
                 Print molecule parameters.
                 """
                 print("""Molecule: %s.
                 \tCritical Temperature = %.1f K
                 \tCritical Pressure = %.1f bar.
                 \tAccentric factor = %f""" % (self.name, self.Tc, self.Pc, self.omega))

         def preos(molecule, T, P, plotcubic=True, printresults=True):
             """
             #Returns a Dict() of molecule properties at this T and P.

             # build params in PREOS
             Tr = T / molecule.Tc  # reduced temperature
             a = 0.457235 * R**2 * molecule.Tc**2 / molecule.Pc
             b = 0.0777961 * R * molecule.Tc / molecule.Pc
             kappa = 0.37464 + 1.54226 * molecule.omega - 0.26992 * molecule.omega**2
             alpha = (1 + kappa * (1 - np.sqrt(Tr)))**2

             A = a * alpha * P / R**2 / T**2
             B = b * P / R / T

             # build cubic polynomial
             def g(z):
                 """
                 Cubic polynomial in z from EOS. This should be zero.
                 :param z: float compressibility factor
                 """
                 return z**3 - (1 - B) * z**2 + (A - 2*B - 3*B**2) * z - (
                         A * B - B**2 - B**3)

             # Solve cubic polynomial for the compressibility factor
             z = root(g, 1.0)  # compressibility factor
             rho = P / (R * T * z)  # density

             # fugacity coefficient comes from an integration
             fugacity_coeff = np.exp(z - 1 - np.log(z - B) - A / np.sqrt(8) / B * np.log(
                         (z + (1 + np.sqrt(2)) * B) / (z + (1 - np.sqrt(2)) * B)))

             if printresults:
                 print("""PREOS calculation at
                 \t T = %.2f K
                 \t P = %.2f bar""" % (T, P))
                 print("\tCompressibility factor : ", z)
                 print("\tFugacity coefficient: ", fugacity_coeff)
                 print("\tFugacity at pressure %.3f bar = %.3f bar" % (
                 P, fugacity_coeff * P))
                 print("\tDensity: %f mol/m3" % rho)
                 print("\tMolar volume: %f L/mol" % (1.0 / rho * 1000))
                 print("\tDensity of ideal gas at same conditions: %f mol/m3" % (
                       rho * 22.4 / 1000 * z))
                 print("\tDensity: %f v STP/v" % (rho * 22.4 / 1000))

             if plotcubic:
                 # Plot the cubic equation to visualize the roots
                 zz = np.linspace(0, 1.5)  # array for plotting

                 plt.figure()
                 plt.plot(zz, g(zz), color='k')
                 plt.xlabel('Compressibility, $z$')
                 plt.ylabel('Cubic $g(z)$')
                 plt.axvline(x=z)
                 plt.axhline(y=0)
                 plt.title('Root found @ z = %.2f' % z)
                 plt.show()
             return {"density(mol/m3)": rho, "fugacity_coefficient": fugacity_coeff,
                     "compressibility_factor": z, "fugacity(bar)": fugacity_coeff * P,
                     "molar_volume(L/mol)": 1.0 / rho * 1000.0}

         def preos_reverse(molecule, T, f, plotcubic=False, printresults=True):
             """
             #Returns a Dict() of molecule properties at this T and f.
             # build function to minimize: difference between desired fugacity and that obtained from preos
             def g(P):
                 """
                 :param P: pressure
                 """
                 return (f - preos(molecule, T, P, plotcubic=False, printresults=False)["fugacity(bar)"])

             # Solve preos for the pressure
             P = root(g, f)  # pressure

             # Obtain remaining parameters
             pars = preos(molecule, T, P, plotcubic=plotcubic, printresults=printresults)
             rho = pars["density(mol/m3)"]
             fugacity_coeff = pars["fugacity_coefficient"]
             z = pars["compressibility_factor"]

             return {"density(mol/m3)": rho, "fugacity_coefficient": fugacity_coeff,
                     "compressibility_factor": z, "pressure(bar)": P,
                     "molar_volume(L/mol)": 1.0 / rho * 1000.0}

         def preos_mixture(molecule_a, molecule_b, delta, T, P_total, x, plotcubic=True, printresults=True):
             """
             # build arrays of properties
             Tc = np.array([molecule_a.Tc, molecule_b.Tc])
             Pc = np.array([molecule_a.Pc, molecule_b.Pc])
             omega = np.array([molecule_a.omega, molecule_b.omega])

             # build params in PREOS
             Tr = T / Tc  # reduced temperature
             a0 = 0.457235 * R**2 * Tc**2 / Pc
             b = 0.0777961 * R * Tc / Pc
             kappa = 0.37464 + 1.54226 * omega - 0.26992 * omega**2
             a = a0 * (1 + kappa * (1 - np.sqrt(Tr)))**2

             # apply mixing rules
             aij = (1.0 - delta) * np.sqrt(a[0] * a[1])
             a_mix = a[0] * x[0]**2 + a[1] * x[1]**2 + 2.0 * x[0] * x[1] * aij
             b_mix = x[0] * b[0] + x[1] * b[1]
             A = a_mix * P_total / R**2 / T**2
             B = b_mix * P_total / R / T

             # build cubic polynomial
             def g(z):
                 """
                 Cubic polynomial in z from EOS. This should be zero.
                 :param z: float compressibility factor
                 """
                 return z**3 - (1 - B) * z**2 + (A - 2*B - 3*B**2) * z - (
                         A * B - B**2 - B**3)

             # Solve cubic polynomial for the compressibility factor
             z = root(g, 1.0)  # compressibility factor
             rho_mix = P_total / (R * T * z)  # density

             Lnfug_0 = -np.log(z - B) + (z - 1.0) * b[0] / b_mix - A / np.sqrt(8) / B * (2.0 / a_mix * (x[0] * a[0] + x[
                       np.log((z + (1.0 + np.sqrt(2)) * B) / (z + (1.0 - np.sqrt(2)) * B))
             Lnfug_1 = -np.log(z - B) + (z - 1.0) * b[1] / b_mix - A / np.sqrt(8) / B * (2.0 / a_mix * (x[1] * a[1] + x[
                       np.log((z + (1.0 + np.sqrt(2)) * B) / (z + (1.0 - np.sqrt(2)) * B))

             # fugacity coefficient comes from an integration
             fugacity_coefs = np.exp(np.array([Lnfug_0, Lnfug_1]))

             if printresults:
                 print("""PREOS calculation at
                 \t T = %.2f K
                 \t P, total = %.2f bar""" % (T, P_total))
                 print("\tDensity: %f mol/m3" % rho_mix)
                 print("\tCompressibility factor : %f" % z)
                 print("Component 0, %s:" % molecule_a.name)
                 print("\tFugacity coefficient: %f" % fugacity_coefs[0])
                 print("\tFugacity: %f bar" % (fugacity_coefs[0] * x[0] * P_total))
                 print("Component 1, %s:" % molecule_b.name)
                 print("\tFugacity coefficient: %f" % fugacity_coefs[1])
                 print("\tFugacity: %f bar" % (fugacity_coefs[1] * x[1] * P_total))

             if plotcubic:
                 # Plot the cubic equation to visualize the roots
                 zz = np.linspace(0, 1.5)  # array for plotting

                 plt.figure()
                 plt.plot(zz, g(zz), color='k')
                 plt.xlabel('Compressibility, $z$')
                 plt.ylabel('Cubic $g(z)$')
                 plt.axvline(x=z)
                 plt.axhline(y=0)
                 plt.title('Root found @ z = %.2f' % z)
                 plt.show()
             return {"density(mol/m3)": rho_mix, "fugacity_coefficients": fugacity_coefs,
                     "compressibility_factor": z}
```

```
In [ ]:
```

```
In [16]: from thermo.chemical import Chemical #</a>[1](#2)

         def Peng_Robinson_const(X):
             R = 8.314 #  universal gas constant,J/mol*K
             tt = Chemical(X) # taking the compounds
             p = tt.P/100000              #pressure
             v = tt.Vml_STP
             t = tt.T
             pc = tt.Pc/100000.  # critical pressure
             tc = tt.Tc       # critical temperature

             w = -np.log10(p/pc)-1.0 #accentric factor

             k = 0.37464 + (1.54226 * w) - (0.26992 * (w**2))

             #a =  (1+(k*(1-(np.sqrt(tt/tc)))))**2
             a = tt.omega
             b = 0.07780*((r*tc )/   pc)

             atc = 0.045724 * ( ( a*(r**2)*(tc**2)  ) / pc)

             at = a *atc

             result = [r,t,v,b,at]

             #result  = ( (r*t)/ (v-b))  - (  (at) / ( (v*(v+b)) + (b*(v-b)) ) )

             return result
```

```
In [17]: def Peng_Robinson_calc(x):

             result  = ( (x[0]*x[1])/ (x[2]-x[3])) - ( (x[4]) / ( (x[2]*(x[2]+x[3])) + (x[3]*(x[2]-x[3])) ) )

             return result
```

```
In [18]: for i in compounds:
             print(f'The  Pressure  of {i} is ----->{Peng_Robinson_calc(Peng_Robinson_const(i))}')
```

```
The  Pressure  of toluene is ----->-245.5462549052974
The  Pressure  of ethane is ----->-607.9947661614383
The  Pressure  of methane is ----->-924.8757591665468
The  Pressure  of ethanol is ----->-270.6269564897677
```

The results tell us about the pressure for different compounds using the Peng Robinson EOS.Here we know the 3 parameters and we estimate the 4 th one using Peng Robinson State # [2]

Q) Using Equimolar mixture of nitrogen-methane at 115 K and 1 MPa, with modified acentric factors to show the difference between

```
In [19]: from thermo.eos_mix import PR78MIX, PRMIX # This is one of the import modules used for the mixtures
```

for solving the Peng-Robinson [1]_[2]_ cubic equation of state for a mixture of any number of compounds. Subclasses `PR`. Solves the EOS on initialization and calculates fugacities for all components in all phases.

Two of Temperature [K], Pressure [Pa], and Molar volume [m^3/mol] are needed to solve the EOS.

T-P initialization, nitrogen-methane at 115 K and 1 MPa:

```
In [20]: eos = PR78MIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5], omegas=[0.6, 0.7], zs=[0.5, 0.5], kijs=
```

```
In [21]: pr = [eos.V_l, eos.V_g] #this calculates the saturation liquid and molar volumes
         pr
```

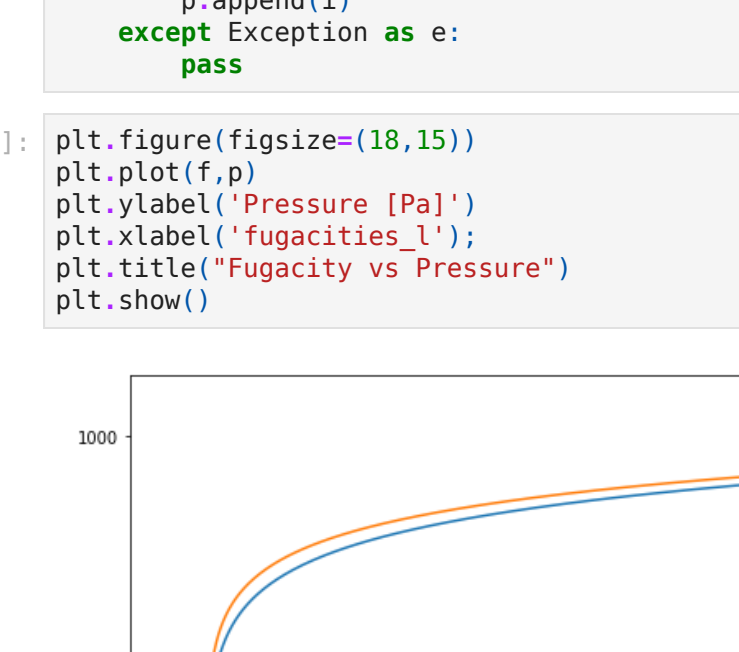
```
Out[21]: [3.2396438915323833e-05, 0.0005043380202435412]
```

With the help of Peng Robinson eos , we can find the volume of liquid and gas , the mixture , at any time we wanr

Let's see the effect of different pressure on molar volume ,

```
In [22]: v =[]
         p = []

         for i in range(-1000,1000):
             try:
                 eos = PR78MIX(T=115, P=i*1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5],  omegas=[0.04, 0.011], zs=[0.
                 v.append(eos.V_l)
                 p.append(i)
             except Exception as e:
                 pass
```

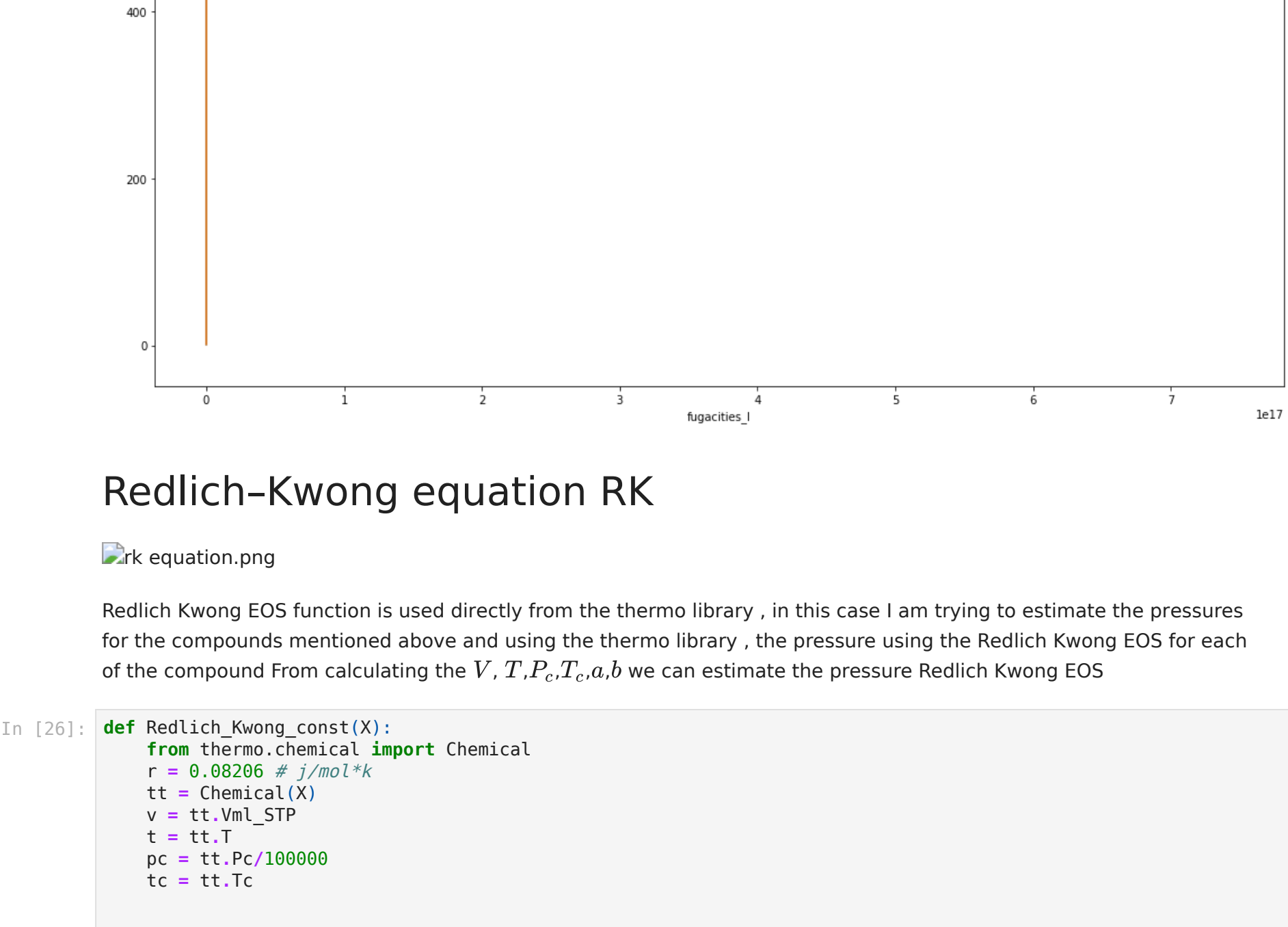
```
In [23]: plt.plot(v,p)
         plt.ylabel('Pressure [Pa]')
         plt.xlabel('Molar volume [m^3/mol]');
         plt.title("Pressure vs Molar Volume")
         plt.show()
```



The pressure vs Molar volume(PV) graph is plotted and can be looked at the process and it is inversely proportional

```
In [24]: f =[] # fugacities
         p = []# pressurer

         for i in range(-1000,1000):
             try:
                 eos = PR78MIX(T=115, P=i*1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5],  omegas=[0.04, 0.011], zs=[0.
                 f.append(eos.fugacities_l)
                 p.append(i)
             except Exception as e:
                 pass
```

```
In [25]: plt.figure(figsize=(18,15))
         plt.plot(f,p)
         plt.ylabel('Pressure [Pa]')
         plt.xlabel('fugacities_l');
         plt.title("Fugacity vs Pressure")
         plt.show()
```



Redlich–Kwong equation RK

![rk equation.png]

Redlich Kwong EOS function is used directly from the thermo library , in this case I am trying to estimate the pressures for the compounds mentioned above and using the thermo library , the pressure using the Redlich Kwong EOS for each of the compound From calculating the V, T, P_c, T_c, a, b we can estimate the pressure Redlich Kwong EOS

```
In [26]: def Redlich_Kwong_const(X):
             from thermo.chemical import Chemical
             r = 0.08206 # j/mol*k
             tt = Chemical(X)
             v = tt.Vml_STP
             t = tt.T
             pc = tt.Pc/100000
             tc = tt.Tc


             a = 0.42748*((  (r**2)*(tc**2.5)  )/   pc )


             b = 0.08664*((   r*tc     )/pc  )

             result = [r, t, v, b, a]
             #result = (   (r*t)  / ( v-b  ) )   -   ( ( a ) /  ( (np.sqrt(t)*v)*(v+b)   )    )  # calculating using red

             return result
```

```
In [27]: def Redlich_Kwong_calc(x):
             result = (   (x[0]*x[1])  / ( x[2]-x[3]  ) )   -   ( ( x[4] ) /  ( ((x[1]**2)*x[2])*(x[2]+x[3])   )    )
             return result
```

```
In [28]: for i in compounds:
             print(f'The Redlich Kwong of {i}is ----->{Redlich_Kwong_calc(Redlich_Kwong_const(i))}')
```

```
The Redlich Kwong of tolueneis ----->-853.2523773084748
The Redlich Kwong of ethaneis ----->-804.5741835699471
The Redlich Kwong of methaneis ----->-953.7005830230746
The Redlich Kwong of ethanolis ----->-1315.8290505801665
```

```
In [29]: from thermo.eos_mix import RKMIX # this is for the mixture
```

T-P initialization, nitrogen-methane at 115 K and 1 MPa: # I want to try taking the same example and try fitting in both Peng and Redlich

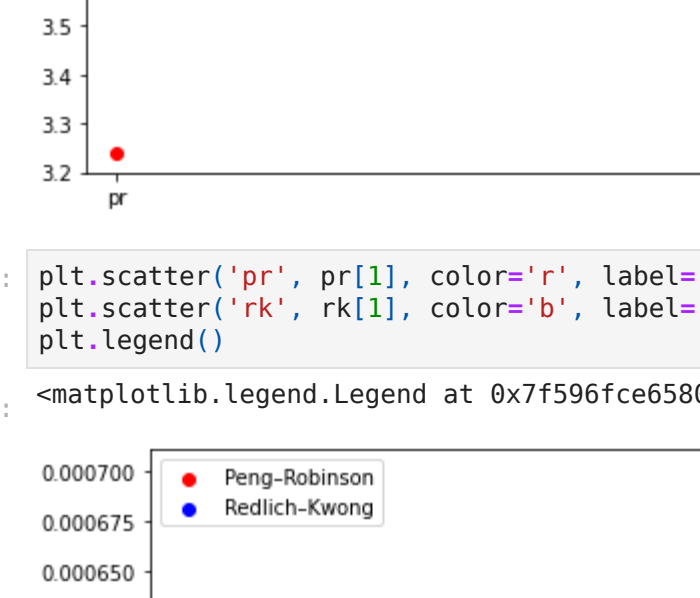
```
In [30]: eos = RKMIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5], zs=[0.5, 0.5], kijs=[[0,0],[0,0]])
```

```
In [31]: rk = [eos.V_l, eos.V_g]
         rk
```

```
Out[31]: [4.04841478191211e-05, 0.0007006060586399438]
```

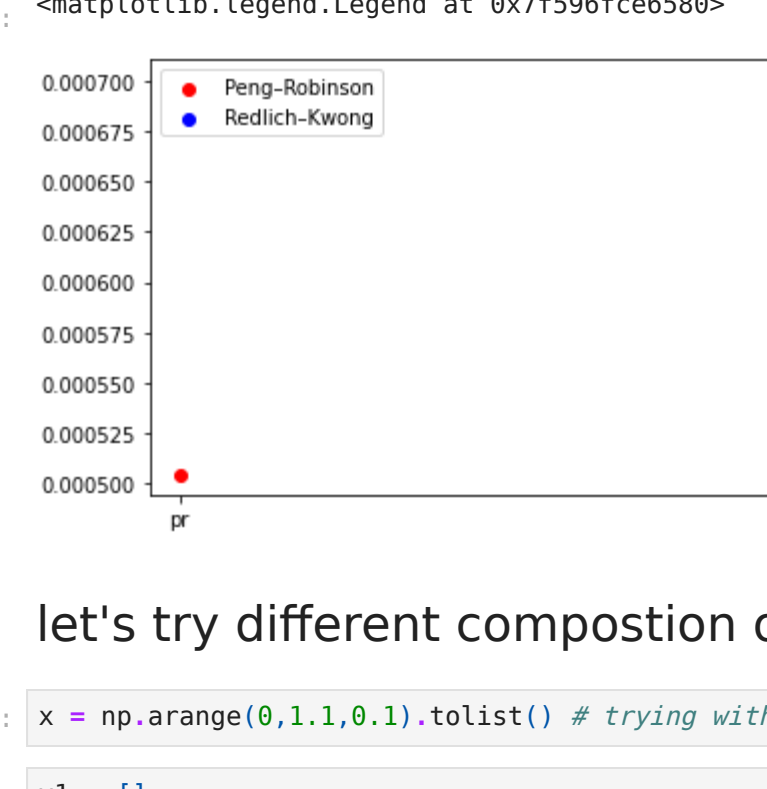
```
In [35]: # The difference between the two models
         plt.scatter('pr', pr[0], color='r', label='Peng–Robinson')
         plt.scatter('rk', rk[0], color='b', label='Redlich–Kwong')
         plt.title("Comparing two models")


         plt.legend()
```

```
Out[35]: <matplotlib.legend.Legend at 0x7f597891c880>
```




```
In [36]: plt.scatter('pr', pr[1], color='r', label='Peng–Robinson')
         plt.scatter('rk', rk[1], color='b', label='Redlich–Kwong')
         plt.legend()
```

```
Out[36]: <matplotlib.legend.Legend at 0x7f596fce6580>
```

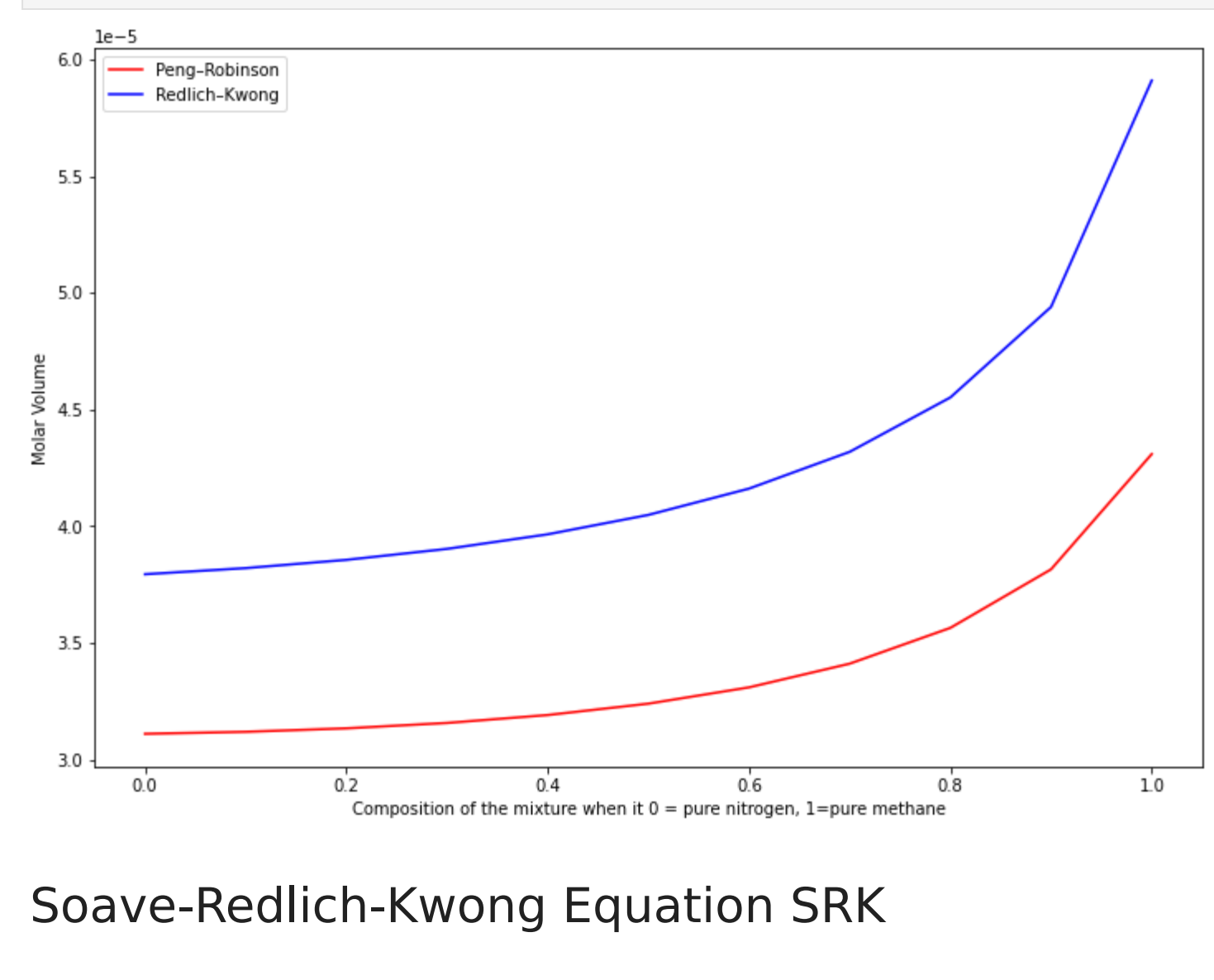


let's try different compostion of mixture

```
In [39]: x = np.arange(0,1.1,0.1).tolist() # trying with different composition of mixture
```

```
In [40]: v1 = []
         v2 = []
         for i in range(len(x)):
             eos1 = PR78MIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5], omegas=[0.6, 0.7], zs=[x[i], 1-x[i
             eos2 = RKMIX(T=115, P=1E6, Tcs=[126.1, 190.6], Pcs=[33.94E5, 46.04E5], zs=[x[i], 1-x[i]], kijs=[[0,0],[0,0
             v1.append(eos1.V_l)
             v2.append(eos2.V_l)
```

```
In [41]: plt.figure(figsize=(12,8))
         plt.plot(x,v1, color='r', label='Peng–Robinson')
         plt.plot(x,v2, color='b', label='Redlich–Kwong')
         plt.xlabel('Composition of the mixture when it 0 = pure nitrogen, 1=pure methane');
         plt.ylabel('Molar Volume')
         plt.legend();
```



Soave-Redlich-Kwong Equation SRK

![srk equation.png]

Soave modified the Redlich-Kwong equation to by replacing the temperature in th denominator with more complex term of temerature. That modification gives better estimations of liquid phase.

In [42]:
```python
from thermo.eos_mix import SRK
```

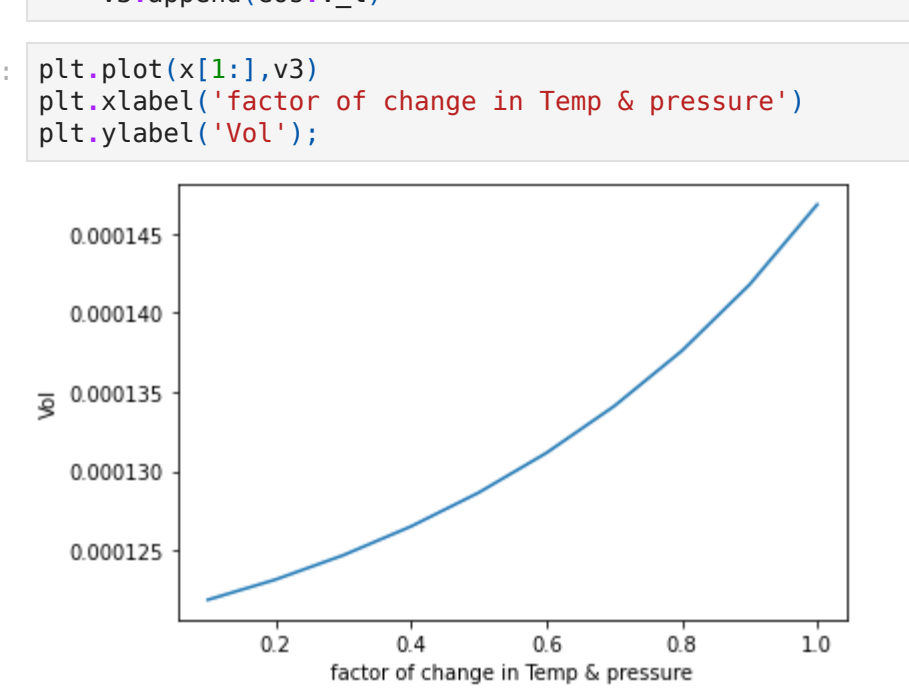
In [43]:
```python
# P-T initialization (methanol), liquid phase at temperature 299
eos = SRK(Tc=507.6, Pc=3025000, omega=0.2975, T=299., P=1E6)
```

In [44]:
```python
eos.phase, eos.V_l, eos.H_dep_l, eos.S_dep_l # this gives the phase , Volume , Heat of vaporization and
```

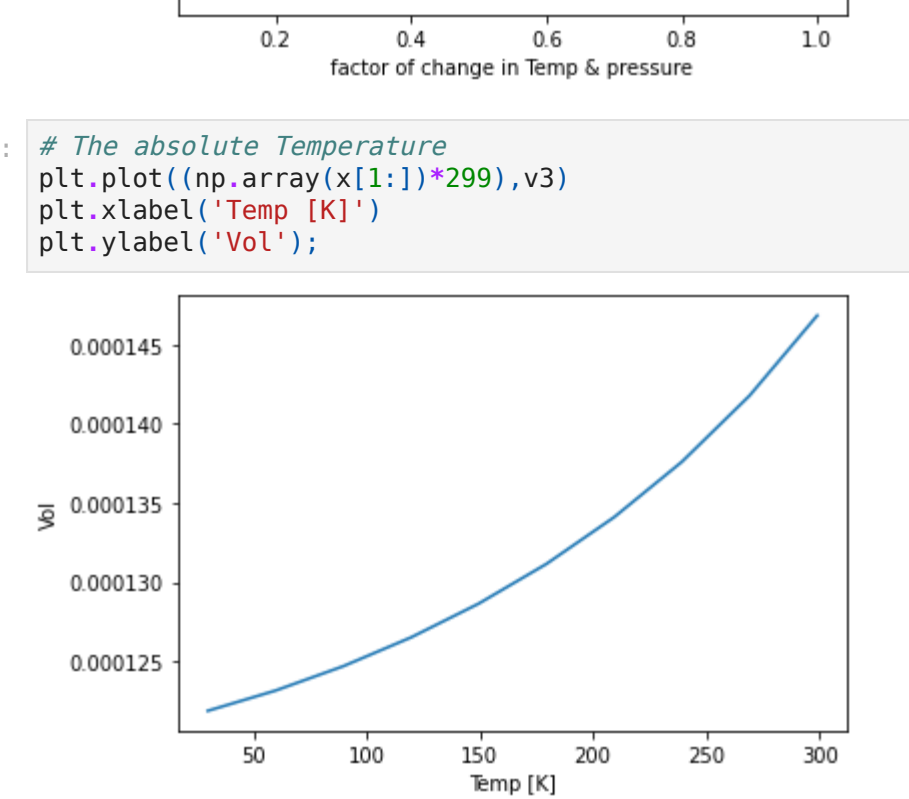
Out[44]:
```
('l', 0.00014682107735472652, -31754.66385964965, -74.37327204446999)
```

In [45]:
```python
# The effect of changing the Temperature and pressure by small change
v3 = []
for i in x[1:]:
    eos = SRK(Tc=507.6, Pc=3025000, omega=0.2975, T=299*i, P=1E6*i)
    v3.append(eos.V_l)
```

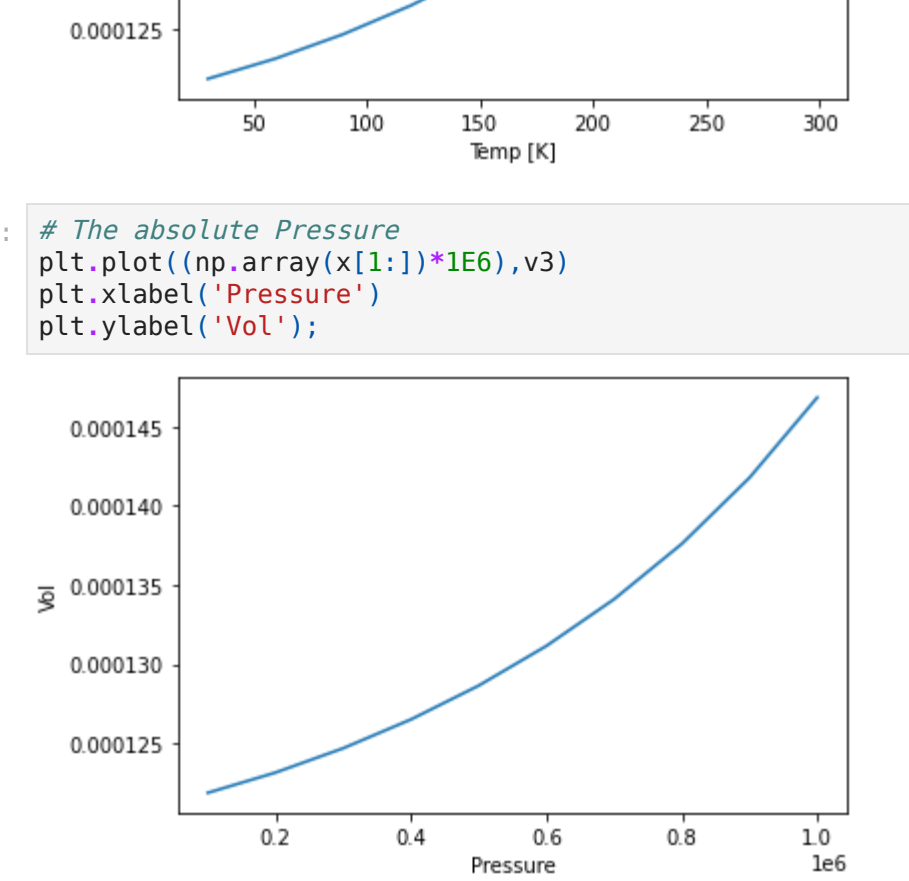
In [46]:
```python
plt.plot(x[1:],v3)
plt.xlabel('factor of change in Temp & pressure')
plt.ylabel('Vol');
```



In [47]:
```python
# The absolute Temperature
plt.plot((np.array(x[1:])*299),v3)
plt.xlabel('Temp [K]')
plt.ylabel('Vol');
```

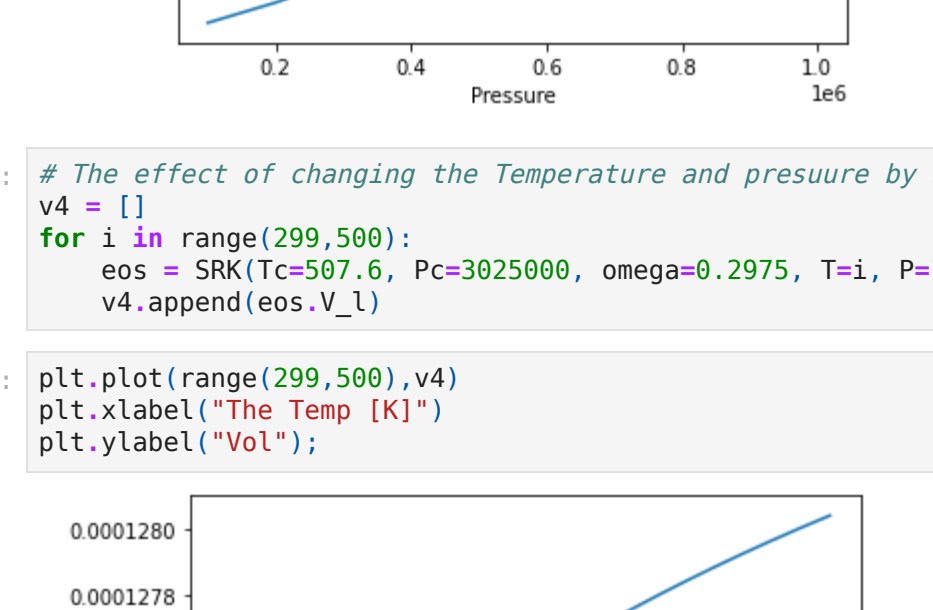


In [48]:
```python
# The absolute Pressure
plt.plot(np.array(x[1:])*1E6),v3)
plt.xlabel('Pressure')
plt.ylabel('Vol');
```



In [49]:
```python
# The effect of changing the Temperature and pressure by big change
v4 = []
for i in range(299,500):
    eos = SRK(Tc=507.6, Pc=3025000, omega=0.2975, T=i, P=1E6*i)
    v4.append(eos.V_l)
```

In [50]:
```python
plt.plot(range(299,500),v4)
plt.xlabel("The Temp [K]")
plt.ylabel("Vol");
```



In [93]:
```
###References
```


1 https://pypi.org/project/thermo/[[1]]

2 https://thermo.readthedocs.io[[2]] # All the codes were source from here

3 https://en.wikipedia.org/wiki/Equation_of_state/[[3]]

4 https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Book%3A_Mathematical_Meth

In [ ]:

In [ ]:

In [ ]:

In [51]:
```python
# Run this cell to generate a pdf from this notebook
# Click the generated links to preview and download it.
# Report errors to Professor Kitchin
from f22_06623 import pdf
%pdf
```

Open project.pdf

download project.pdf

In [ ]: