Delft University of Technology

Advanced Applied Thermodynamics
ME45160

Assignment: 1

Ilambharathi Govindasamy (5352797) [1]
December 14, 2020

[1] NOTE: *Worked along with Karthik Selvam

1. Select one of the compounds available in the NIST web book. Look up T_c,P_c, and ω – the critical temperature, critical pressure, and acentric factor.These are the only inputs you will need for your EOS calculations. Report your compound and the three inputs to your EOS

   - The chosen component is **Nitrogen**
   - The three inputs to EOS are listed below

Critical temperature (T_c)	126.192 K
Critical pressure (P_c)	3.3958 MPa
Acentric factor	0.0372

Table.1

2. Use Python, Matlab, or your favorite alternative to calculate the critical molar volume. Compare the value predicted by your EOS and the value reported by NIST.

   - The EOS calculations are done using Peng-Robinson equations in **Python**, the code for which can be found in Appendix. The values found and the error are presented in Table.2

	EOS	NIST
Critical volume V_c (m^3/mol)	9.929E-05	8.941E-05

Table.2

   The error is found to be 11.0546%

3. Plot three isotherms of your EOS – one above, one at, and one below the critical temperature. For the subcritical isotherm, include the Maxwell tie line (which you will have to determine numerically). Here and in all subsequent plots, report your results in terms of reduced quantities, i.e. the reduced pressure $p_r = p/p_c$, reduced molar volume $v_r = v/v_c$, and reduced temperature $T_r = T/T_c$. Clearly label your axes and provide legends where needed. Ask yourself two questions: could a classmate understand what I have plotted without having me here to explain? Have I provided enough information that my instructor could reproduce my plot?

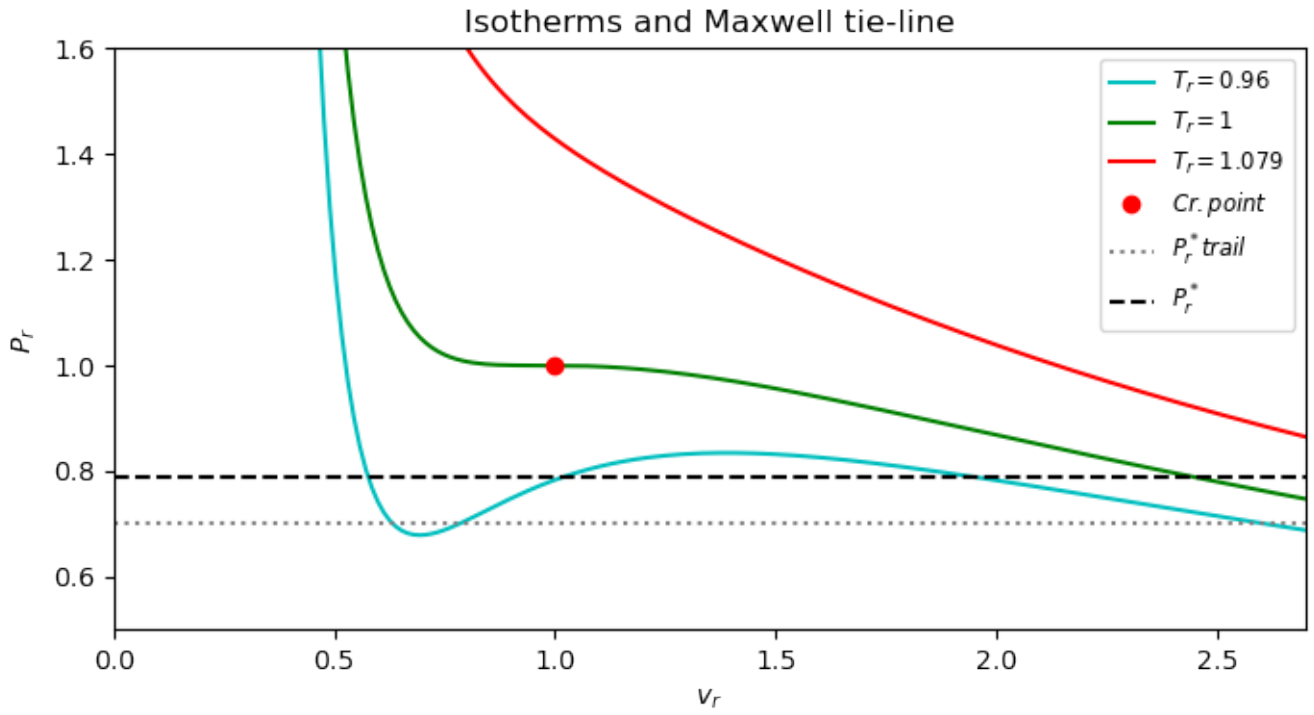

Figure 1: Isotherms and Maxwell Tie-line

4. Repeat your calculation of the coexistence pressure for five additional temperatures. Plot the boundaries of the coexistence region predicted by your EOS in p_r-v_r space. Calculate v_r using the critical molar volume predicted by your EOS. In the same plot, plot the boundaries of the coexistence data from experimental data, as reported by NIST.

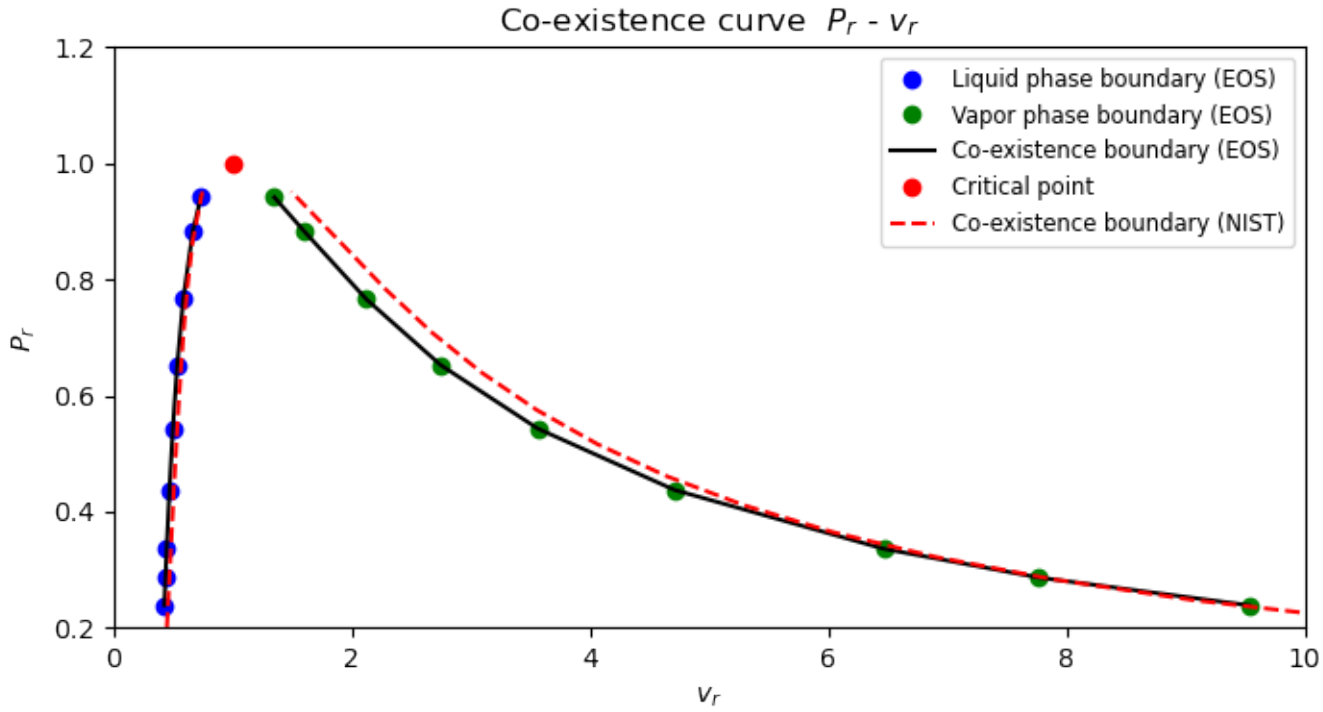

Figure 2: Co-existence curve

5. Plot the coexistence curve predicted by your EOS in P_r-T_r space. In thesame plot, include the coexistence curve from NIST

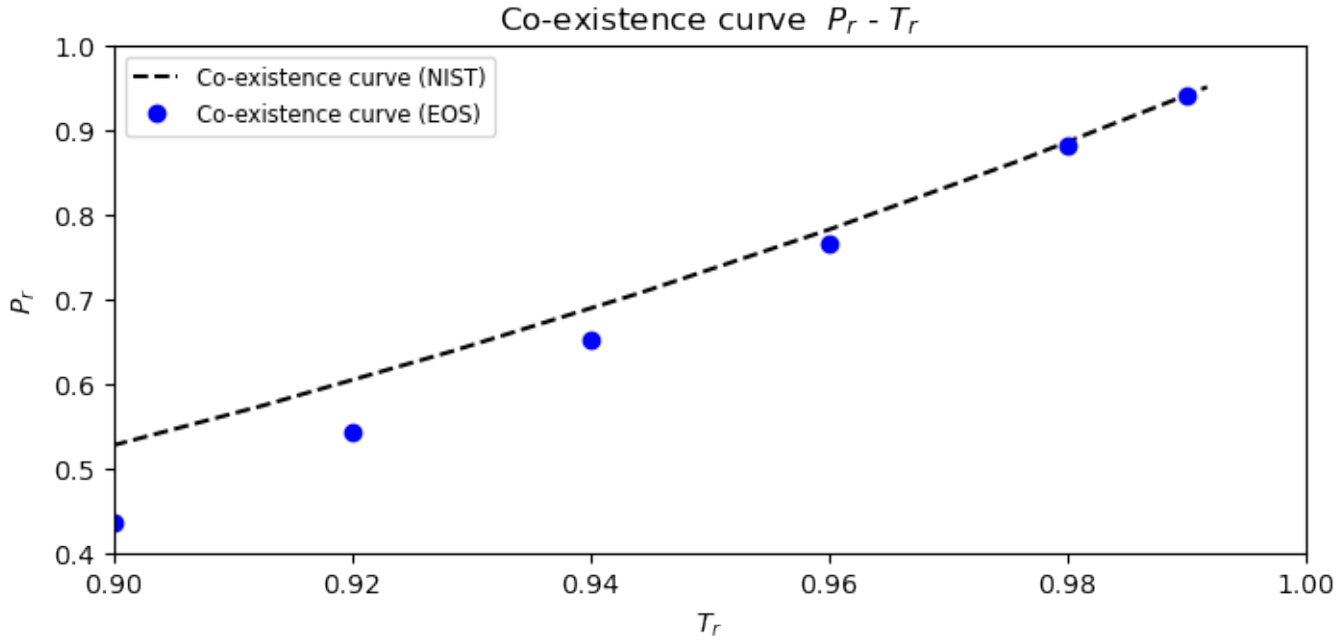

Figure 3: Co-existence curve

6. Plot the coexistence curve predicted by your EOS as $\ln(p_r)$ versus $1/T_r$. Fita line to the data and report the result of the fit. Use your fit coefficients to estimate the enthalpy of vaporization. Compare your result to the value reported by NIST.

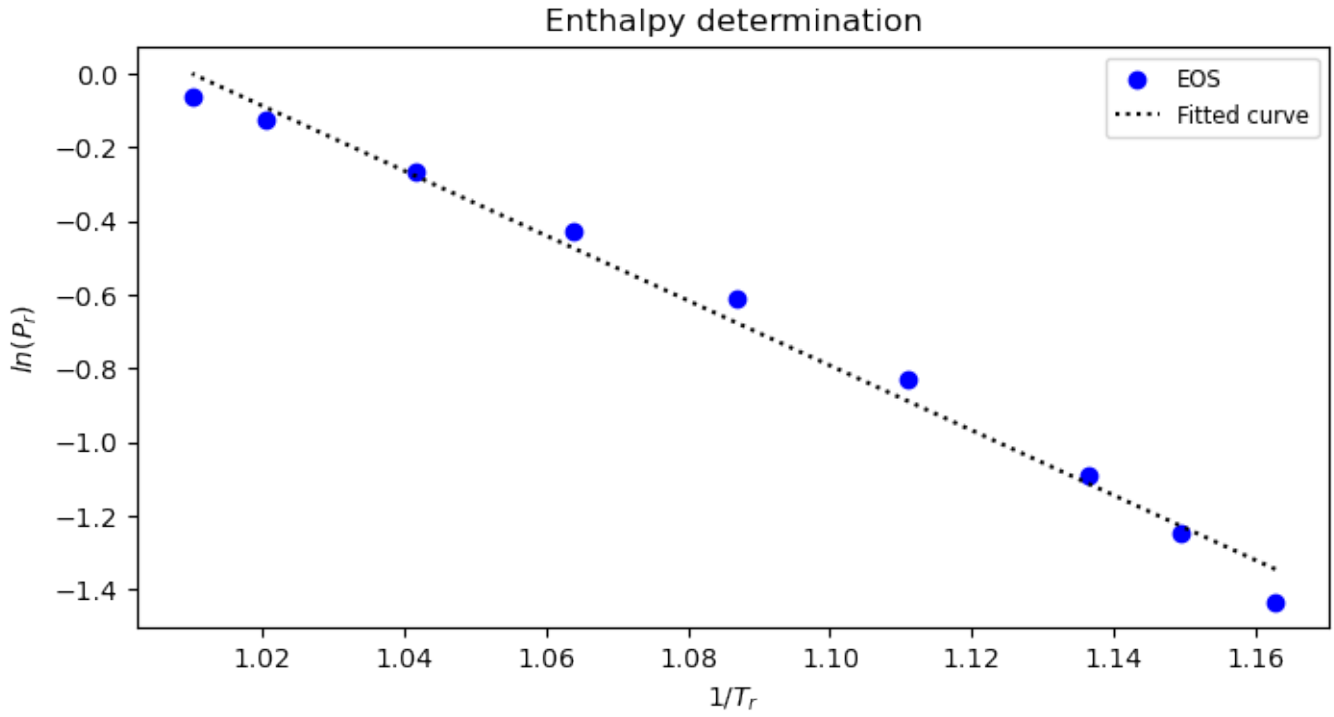

Figure 4: $\ln(p_r)$ vs $1/T_r$

Enthalpy derived by August relation from EOS is 9.24 kJ/mol , from NIST is 5.85 kJ/mol

Appendix:

The following Python code is used for this assignment

```
#Peng-Robinson
#Nitrogen
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
from scipy.signal import argrelextrema

#1
#All in SI units
Pc=3.3958*10**6 #in pascal
Tc=126.192 #in Kelvin
R=8.3145
q=0.0372
k=0.37464+(1.54226*q)-(0.26992*q)**2

#2
#By cubic equation with reduced parameters
i=Pc/(R*Tc) # The coefficient of critical volume in Z
alpha=(1+k*(1-np.sqrt(1)))**2
a=0.45724*alpha*((R*Tc)**2)/Pc
b=0.07780*R*Tc/Pc
```

```
A=(a*Pc)/(R*Tc)**2
B=(b*Pc)/(R*Tc)
al=B-1
be=A-2*B-3*B*B
ga=-A*B+(B*B)+(B*B*B)
e=[(i**3),(al*(i**2)),(be*i),ga] #cubic equation in reduced form
v=np.roots(e) #Roots of volume
vc=v.real[v.imag==0] #Only the roots without imaginary values are chosen
print("The critical volume is found to be "+str(vc))
u=np.reciprocal(11183.9) # From crtical density
print("The error in critical volume is found to be  "+str((vc-u)*100/u)+" %")

#3.1
T=np.array((Tc-5,Tc,Tc+10)) #sub-critical, critical and super-critical
Rr=R*Tc/(Pc*vc) # Gas constant for reduced form
n=100
br=b/vc
vr=np.linspace(0.3,3,200)
fig1=plt.figure(figsize=(8,4),dpi=100)
Ap=[np.zeros(n),np.zeros(n),np.zeros(n)]
co=['c','g','r']
for i in range(3):
    Tr=T[i]/Tc
    alpha=(1+k*(1-np.sqrt(Tr)))**2
    a=0.45724*((R*Tc)**2)*alpha/Pc
    ar=a/(Pc*vc**2)
    Pr= (Rr*Tr/(vr-br))-ar/(vr*(vr+br)+br*(vr-br))
    Ap[i]=Pr
    plt.plot(vr,Pr,co[i]) #Plotting vr vs Pr
    plt.ylim([0.5,1.6])
    plt.xlim([0,2.7])
    plt.ylabel(r'$P_r$')
    plt.xlabel(r'$v_r$')
plt.plot(1.0,1.0,'ro')
pa=Ap[0]

#3.2
#chosen 0.7
def root(pr,Tr,k,Rr,Pc,vc,Tc): #Finding the roots
    i=pr/(Rr*Tr)
    alpha=(1+k*(1-np.sqrt(Tr)))**2
    a=0.45724*((R*Tc)**2)*alpha/Pc
    ar=a/(Pc*vc**2)
    b=0.07780*R*Tc/Pc
    br=b/vc
    A=(ar*pr)/(Rr*Tr)**2
    B=(br*pr)/(Rr*Tr)
    al=B-1
    be=A-2*B-3*B*B
    ga=-A*B+(B*B)+(B*B*B)
    x=i**3
    y=al*(i**2)
    z=be*i
    e= [x[0],y[0],z[0],ga[0]]
    vra=np.roots(e) #finding roots
    va=min(vra) #On liquid side intersection
    vb=max(vra) #On vapor side intersection
    return(va,vb)
```

```
def PR(Tr,vr,ar,br,Rr):
        Pr= (Rr*Tr/(vr-br))-ar/(vr*(vr+br)+br*(vr-br)) #Peng-Robinson reduced form
        return(Pr)
#Iterating
def prco(pr,Tr,k,Rr,Pc,vc,Tc,R,br):
    delpr=0.01
    Fx=1
    while Fx>0.00001: #secant method
        prxx=pr+delpr
        vax,vbx=root(pr,Tr,k,Rr,Pc,vc,Tc)
        vaxx,vbxx=root(prxx,Tr,k,Rr,Pc,vc,Tc)
        alpha=(1+k*(1-np.sqrt(Tr)))**2
        a=0.45724*((R*Tc)**2)*alpha/Pc
        ar=a/(Pc*vc**2)
        Fx = integrate.quad(lambda vr: PR(Tr,vr,ar,br,Rr)-pr,vax,vbx)[0] #integrating
        Fxx=integrate.quad(lambda vr: PR(Tr,vr,ar,br,Rr)-prxx,vaxx,vbxx)[0] #integrating
        Fd=(Fxx-Fx)/delpr
        pr=pr-(Fx/Fd)
    return pr
pr = 0.7
plt.plot([0,2.7],[pr,pr],color='grey',linestyle=':')
Tr = T[0]/Tc
pr=prco(pr,Tr,k,Rr,Pc,vc,Tc,R,br)
plt.plot([0,2.7],[pr,pr],'k--')
plt.legend([r'$T_r=0.96$',r'$T_r=1$',r'$T_r=1.079$',r'$Cr.point$',r'$P^*_r trail$',r'$P^*_r$'],loc='upp
plt.title('Isotherms and Maxwell tie-line')
plt.show()
plt.figure(figsize=(8,4),dpi=100)
#4
Trk=[0.86,0.87,0.88,0.90,0.92,0.94,0.96,0.98,0.99]
z=len(Trk)
vl=np.zeros(z)
pf=np.zeros(z)
pj=np.zeros(z)
vg=np.zeros(z)
Tv=np.zeros(z)
Pv=np.zeros(z)
vr=np.linspace(0.3,3,200)
for i in range(z):
    Tr=Trk[i]
    alpha=(1+k*(1-np.sqrt(Tr)))**2
    a=0.45724*((R*Tc)**2)*alpha/Pc
    ar=a/(Pc*vc**2)
    vr=vr.reshape(200,1)
    P=(Rr*Tr/(vr-br))-ar/(vr*(vr+br)+br*(vr-br))
    ipmin = argrelextrema(P, np.less) #Finding the index of local minimum
    ipmax = argrelextrema(P, np.greater) #Finding the index of local maximum
    pr=np.mean([P[ipmax],P[ipmax]]) #Choosing the trail pressure ratio as a mean between extrema to ens
    pf[i]=prco(pr,Tr,k,Rr,Pc,vc,Tc,R,br)
    vl[i],vg[i]=root(pf[i],Tr,k,Rr,Pc,vc,Tc)
    Tv[i]=Tr*Tc
    Pv[i]=pf[i]*Pc
plt.plot(vl,pf,'bo')
plt.plot(vg,pf,'go')
plt.plot(vl,pf,'k-',label='_nolegend_')
plt.plot(vg,pf,'k-')
plt.plot(1.0,1.0,'ro')
```

```
plt.ylim([0.2,1.2])
plt.xlim([0,10])
plt.ylabel(r'$P_r$')
plt.xlabel(r'$v_r$')

Pn=np.array([0.012520,0.017863,0.024898,0.033978,0.045490,0.059850,0.077500,0.098911,0.12457,0.15499,0.
Pnr=Pn*10**6/(3.3958*10**6)
Vn=np.array([3.2303e-05,3.2613e-05,3.2933e-05,3.3264e-05,3.3607e-05,3.3962e-05,3.4330e-05,3.4713e-05,3.5
Vnr=Vn*11183.9
Vng=np.array([0.041546,0.029955,0.022075,0.016589,0.012689,0.0098618,0.0077753,0.0062105,0.0050192,0.004
Vnrg=Vng*11183.9
plt.plot(Vnr,Pnr,'r--')
plt.plot(Vnrg,Pnr,'r--')
plt.title('Co-existence curve  ' +r'$P_r$'+' - '+r'$v_r$')
plt.legend(['Liquid phase boundary (EOS)','Vapor phase boundary (EOS)','Co-existence boundary (EOS)','C
plt.show()

#5
Tn=np.array([63.151,65.151,67.151,69.151,71.151,73.151,75.151,77.151,79.151,81.151,83.151,85.151,87.151
Tnr=Tn/Tc
plt.figure(figsize=(8,3.5),dpi=100)
plt.plot(Tnr,Pnr,'k--') #Plotting NIST Values
plt.plot(Trk,pf,'bo') #Plotting EOS values
plt.xlim(0.90,1)
plt.ylim(0.4,1)
plt.ylabel(r'$P_r$')
plt.xlabel(r'$T_r$')
plt.legend(['Co-existence curve (NIST)','Co-existence curve (EOS)'],loc='upper left',fontsize='small')
plt.title('Co-existence curve  ' +r'$P_r$'+' - '+r'$T_r$')
plt.show()

#6
plt.figure(figsize=(8,4),dpi=100)
x=np.reciprocal(Trk)
y=np.log(pf)
xn=np.reciprocal(Tnr)
yn=np.log(Pnr)
plt.plot(x,y,'bo') #plotting EOS values
plt.ylabel(r'$ln(P_r)$')
plt.xlabel(r'$1/T_r$')
b,a = np.polyfit(x, y, 1) #b - slope, a- intercept for EOS
c,d = np.polyfit(xn, yn, 1) # similarly for NIST
entp=(-b*R*Tc) #Enthalpy from EOS (by using dp*/dT)
entpni=(-c*R*Tc) # Enthalpy from NIST

plt.plot(x,b*x+a,'k:') #Plotting fitted values
plt.legend(['EOS','Fitted curve'],loc='upper right',fontsize='small')
plt.title("Enthalpy determination")
print("Enthalpy derived by August relation from EOS is  "+ str(entp/1000)+ " kJ/mol , from NIST is "+ s
plt.show()
```