CHE221A

Computational Assignment

- **Samanvay Lakhotia**

(190747)

Equation of State – Peng Robinson

Substance – Toluene (C_7H_8)

General Cubic equation of State:

$$P = \frac{NRT}{(V - Nb)} - \frac{N^2 a}{(V + \varepsilon \mathrm{Nb})(V + \sigma \mathrm{Nb})}$$

For Peng Robinson's equation of state:

$\varepsilon = 1 - \sqrt{2}$

$\sigma = 1 + \sqrt{2}$

$\Omega = 0.07780$

$\Psi = 0.45724$

$\alpha(T_r, \omega) = [1 + (0.37464 + 1.54226\,\omega - 0.26992\,\omega^2)(1 - \sqrt{T_r})]^2$,
where $T_r = T/T_c$ (T_c = Critical Temperature)

$a = \frac{\Psi \alpha(T_r) R^2 {T_c}^2}{P_c}$ = Measure of attraction between the particles

$b = \frac{\Omega R T_c}{P_c}$ = Measure of volume excluded by a mole of particles

For Toluene,

$T_c = 591.8$ K

$$P_c = 41.06 \text{ bar}$$

$$\omega = 0.262 = \text{Acentric Factor for Toluene}$$

Source: Toluene (nist.gov)

About the Peng Robinson's Equation of State:

The Peng–Robinson equation of state 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 of the equation should be expressible in terms of the acentric factor and the critical properties.
2. The equation of state should be applicable to the calculations of all fluid properties in the natural gas processes.
3. The mixing rules should not use more than 1 binary interaction parameter, which should be independent of T, P, and composition.
4. The model should have respectable accuracy near the critical point, especially for calculations of Z and the liquid densities.

For the most part the Peng–Robinson equation exhibits performance similar to the Soave equation, though it is generally superior in predicting the liquid densities of many materials, especially nonpolar ones.

(PTO)

For getting an initial guess of Saturation Pressure (as well as for generating a crude form of the Pυ plot), the Antoine's equation has been used.

Below are the coefficients of Antoine's equation (A, B, C) with the temperature ranges where they are well fitted. Since these lines are fitted by the given curve equation to experimental data, these aren't perfect, but are an excellent starting guess value for finding Saturation Pressure for our system at a particular condition.

$\log_{10}(P) = A - (B / (T + C))$

P = vapor pressure (bar)

T = temperature (K)

Temperature (K)	A	B	C	Reference	Comment
273.13 - 297.89	4.23679	1426.448	-45.957	Besley and Bottomley, 1974	Coefficents calculated by NIST from author's data.
303. - 343.	4.08245	1346.382	-53.508	Gaw and Swinton, 1968, 2	Coefficents calculated by NIST from author's data.
420.00 - 580.00	4.54436	1738.123	0.394	Ambrose, Broderick, et al., 1967	Coefficents calculated by NIST from author's data.
308.52 - 384.66	4.07827	1343.943	-53.773	Williamham, Taylor, et al., 1945	
273. - 323.	4.14157	1377.578	-50.507	Pitzer and Scott, 1943	Coefficents calculated by NIST from author's data.

Source: Toluene (nist.gov)

Algorithm for the Code:

We needed to do 2 things:

a) Make the P-υ dome shaped saturation curve.

b) Plot isotherms (T < Tc, T = Tc, T > Tc) within the same plot.

Here is the explanation on how the two tasks were done:

1) Make the P-υ dome shaped saturation curve:

a) For a particular temperature, we want to find the saturation pressure, and the corresponding liquid and gaseous volumes, and plot them on our graph.

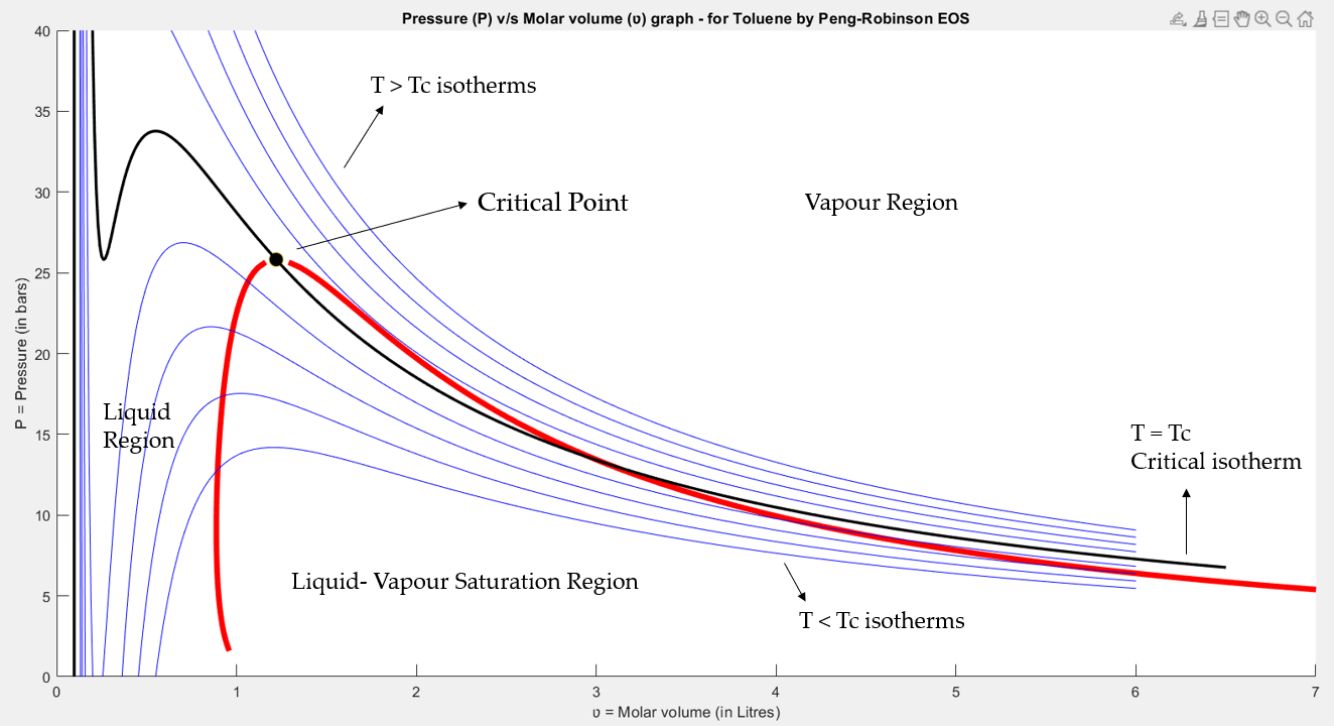
b) This step is repeated for a lot of temperatures with a *for loop* with small increments in temperature so that the graph is smoothly made.

For the a) step, there are basically 3 methods in the code, as follows:

i) **<u>Using Antoine's equation:</u>**

This is a fairly basic method- We use the Antoine's equation to simply obtain the values of Saturation Pressure. This isn't completely accurate, but it does give a working P-υ plot.

Below is the graph obtained in case of using Antoine's equation to get saturation pressure.



Obtained P_c = 25.82 bars, T_c = 570 K, υ_c = 1.22 L/mol

ii) **Convergence of μl - μg to 0 at P = Psat for a given T:**

Algorithm:

1) Take initial guess for Psat from Antoine's equation.
2) Find |μl - μg| (by finding μl and μg wrt ideal gas states at same conditions by the departure function equations).
3) If this is less than a pre-set tolerance value, then we have arrived at the actual Psat.

If the difference is greater than the tolerance value, then we increase and decrease Pressure by a small amount (separately) and calculate the difference again in both cases. We move towards the case that gives us a lower magnitude of difference value, since that is how we slowly converge towards the value of 0 for $|\mu l - \mu g|$.

4) Repeat step 3 again until convergence is reached ($|\mu l - \mu g| <$ tolerance).
5) The Pressure obtained in this case is the Saturation pressure.
   We can use it to obtain vl and vg (liq and gaseous molar volumes) by using the Peng-Robinson cubic EOS and the inbuilt function *roots()* in MATLAB.

   The method provides excellent results but is a bit slow owing to the large amounts of computations required.

   The departure function for finding $\mu(T, P)$ wrt $\mu^{ig}(T, P)$:

$$\frac{G(T,P) - G^{ig}(T,P)}{NRT} = \frac{u(T,P) - u^{ig}(T,P)}{RT} = -\ln\left|1 - \frac{b}{v}\right| - \frac{a}{(\sigma-\varepsilon)}\ln\left|\frac{v+\sigma b}{v+\varepsilon b}\right| + (z-1) - \ln(z)$$

Here, $u^{ig}(T,P)$ is the reference state for both liq. & gas phases

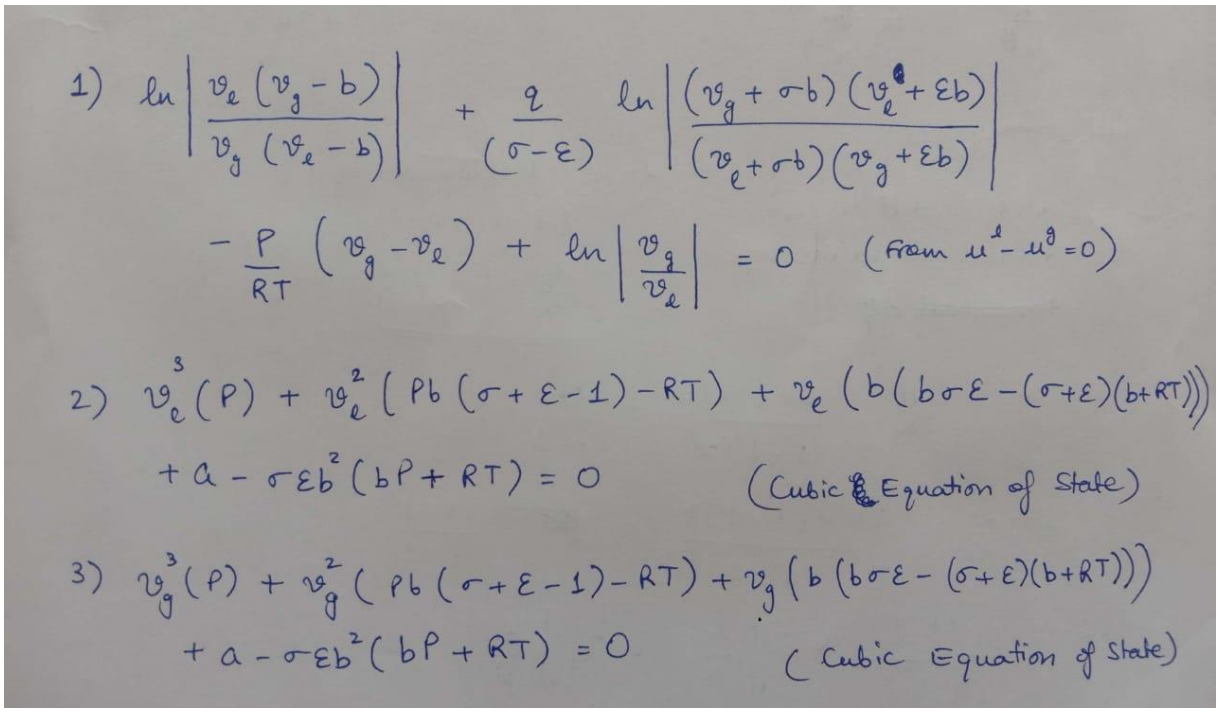
iii) **<u>Solving 3 simultaneous equations to get Psat, vl, vg:</u>**

At equilibrium between 2 different phases, we know that our Saturation Pressure (Psat), Temperature (T) and Chemical potential (μ) is the same for both phases.

Hence, we can make 3 equations: one for $\mu l - \mu g = 0$, and the other two being equations of state- one for liquid and the other for the gaseous phase at equilibrium. Using the fact in the previous paragraph, the Pressure and Temp. in these EOS are the same.

These are then solved simultaneously by the inbuilt *fsolve()* function in MATLAB. Being an inbuilt method, it is quite optimised and offers speedy results.

Hence, the final equations are:



I have created functions for all 3 methods. In the main code, it is very easy to interchange between these methods!

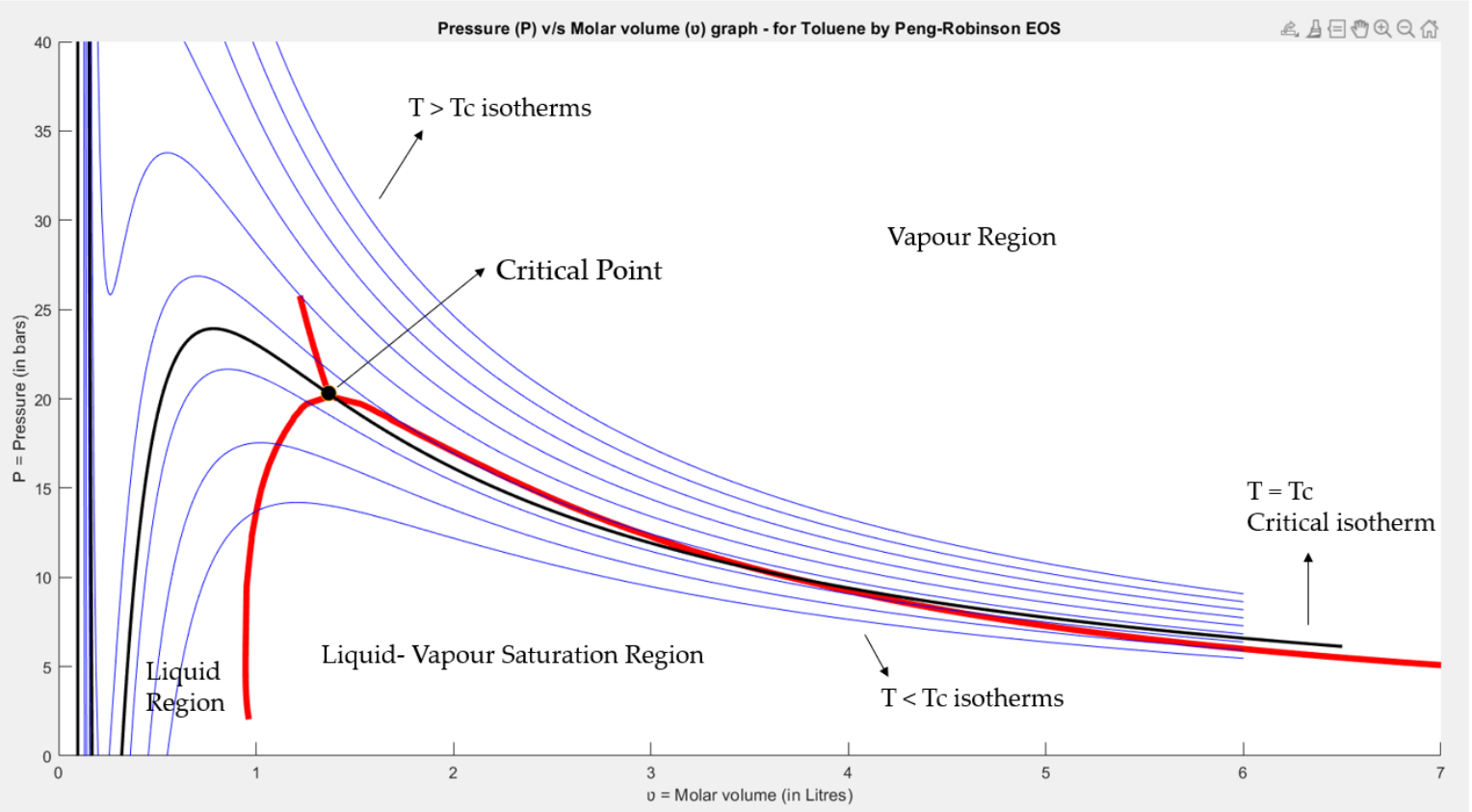
2) Make the isotherms:

The method for creating isotherms is quite simple.

Algorithm:

1) For a constant T, calculate a(T) for Peng-Robinson EOS and Toluene.
2) To plot an isotherm on the P-υ plot, we vary υ (molar volume) on a range and obtain a single P by the Peng-Robinson Equation of state. We append these values in a vector in MATLAB.
3) Plot the final vectors of molar volumes and the corresponding pressures.
4) Repeat steps 3 and 4 for each different temperature to get multiple isotherms.

Final plot:



The obtained values of P_c = 20.31 bar, T_c = 524 K, υ_c = 1.368 L/mol.

The experimentally observed values of P_c, T_c, υ_c are 41.06 bars, 591.8 K, 0.316 L/mol.

Analysis:

The major issue that occurs is that the 3 simultaneous equations don't really go to zero together. For smaller T's (upto 490 K), the *fsolve()* can't decrease the error below it's in-built tolerance levels, causing some errors in the obtained values, and hence in the dome plot. The difference of chemical potential method is a little slow, rendering it infeasible, while the Antoine equation curve is a bit inaccurate, but surprisingly a bit closer to the actual experimental values than the other two methods. The main reason that I believe behind that is that this occurs since I have used multiple Antoine equation coefficients depending on the temperature (taken from Toluene (nist.gov)).

It is also observed that:

i) At $T > T_c$, the Peng-Robinson Equation of State gives only 1 real root of volume (the rest 2 are imaginary and complementary to each other).

ii) At $T = T_c$, we have 3 real roots of volume, all equal to υ_c.

iii) At $T < T_c$, ideally, 3 real roots should be obtained. By code, as we reach $T = T_c$ from the lower side ($T < T_c$), we start having 2 imaginary roots (with the only real root becoming negative too!). This creates issues in the P-υ plot too (One observation is that in the Antoine's plot, we don't obtain the top of the dome if we restrict the molar volume to take only real values (though if we allow it to take imaginary values, the dome gets completed along with a protrusion, this is due to MATLAB ignoring the imaginary coefficients, and plotting only the real coefficients)). Hence, this acts as a source of inaccuracy in the calculated values.

Code:

```
% Substance = Toluene, Equation of State = Peng-Robinson
% All analysis is done for 1 mole of Toluene (N = 1).
clc
clear

% Specifics of Peng-Robinson equation:
global sigma;
sigma = 1 + sqrt(2);
global epsilon;
epsilon = 1 - sqrt(2);
global omega;
omega = 0.07780;
global psi;
psi = 0.45724;
global R
R = 8.314;  %Units = Pa.m^3/(mol.K) => SI units
global R_new_bar;
R_new_bar = 0.0832; %L.bar/mol.K  (0.0821(in
L.atm/mol.K)*1.01325(bar/atm))


% Specifics of Toluene:

global Tc;
Tc = 591.8 ;     %In Kelvins
global Pc;
```

```
Pc = 41.06e5 ;     %In Pascals (Pc = 41.06 bar)
global Pc_bar;
Pc_bar = Pc/1e5;  %Pc in bars.
global w;
w = 0.262 ;        % Omega term in alpha(Tr, w)

%Note that a, being dependent on temperature has a
separate function for
%it, and can't be defined as a single value. We can do
this for b though.
global b;    %The parameter of Toluene
b = omega*R*Tc/Pc;
global b_new;
b_new = b*1e3;  %SI units of m^3/mol to L/mol.

%-------------------------------------------------------
% Characterisation of the system over. Now begins the
main code.

P_sat_vec = []; %Vector for storing pressures
V_vec1 = []; %Vector for storing volumes of liq phase
V_vec2 = []; %Vector for storing volumes of vap phase

xlim([0,7]);
ylim([0, 40]);
xlabel("? = Molar volume (in Litres)");
ylabel("P = Pressure (in bars)");
title("Pressure (P) v/s Molar volume (?) graph - for
Toluene by Peng-Robinson EOS");
hold on;

%PLOTTING THE DOME SHAPED CURVE!
for t = 400: 1: 600

    %Method 1 of finding P_sat- Antoine's eqn.
%     p_sat = Antoine_eqn(t); %The simple case of
calculating by Antoine's eqn.
%     p_sat = p_sat/1e5;

%     p_sat = find_Psat(t, 0.5);    %THE TOLERANCE VALUE
CAN BE CHANGED FROM HERE!!

     [p_sat, vl, vg] = P_by_fsolve(t);

    %These following commented lines are used in case of
using the find_Psat(t, 1) and Antoine's eqn line.

%     alpha_dome = find_alpha(t);
```

```
%     a_dome = psi*alpha_dome*(R_new_bar*Tc)^2/(Pc_bar);
%     eqn = [p_sat, p_sat*b_new*(sigma+epsilon-1)-
R_new_bar*t, b_new*(b_new*sigma*epsilon-
(sigma+epsilon)*(b_new+ R_new_bar*t)), a_dome-
sigma*epsilon*(b_new^2)*(p_sat*b_new+R_new_bar*t)];
%     v = roots(eqn);  %In Litres
%     vg = max(v);
%     vl = min(v(v>0));

%    Check if isreal() can be of any help!
%    display(t);

    if ~isreal(vl)      %Break from the for loop when vl
(and similarly vg) come out to be imaginary
        break
    end

%     display(vl);
%     display(vg);

    V_vec1 = [V_vec1, vg];
    V_vec2 = [V_vec2, vl];
    P_sat_vec = [P_sat_vec, p_sat];

end

plot(V_vec1, P_sat_vec, 'r', 'Linewidth', 4);
plot(V_vec2, P_sat_vec, 'r', 'Linewidth', 4);

%Easily seen from the graph that v(c) = 1.267 L, P(c) =
23.87 bars.
plot(1.368, 20.31, 'o', 'Markersize', 10,
'MarkerFaceColor', 'k');


%PLOTTING THE ISOTHERMS! (Func. definition at the end)
for T = 450: 30: 700
    plot_isotherm(T, 'blue');
end


% This procedure is the same as my function, just that I
wanted to
% emphasize the critical curve more by it's linewidth and
extra volume range. Hence, used the
% function here, with a very slight modification.

T = 524; %Observed from the graph
```

```
a = find_a(T);
hold on;

V_vec = [];
P_vec = [];

% for v = 0.4: 0.05: 6.5
for v = 0.04: 0.01: 6.5
    P = R_new_bar*T/(v - b_new) -
a/((v+sigma*b_new)*(v+epsilon*b_new));
    P_vec = [P_vec, P];
    V_vec = [V_vec, v];
end

plot(V_vec, P_vec, 'color', 'k', 'Linewidth', 2);
hold off

%-------------------------------------------------------
% All Functions below

function Treduced = Tred(T)  %To simply calculate the
reduced temperature from T, Tc
    global Tc;
    Treduced = T/Tc;
end

function alpha = find_alpha(T)
    Tr = Tred(T);
    global w;
    alpha = (1 + (0.37464 + 1.54226*w - 0.26992*(w^2))*(1
- sqrt(Tr)))^2 ;
end

function a = find_a(T)
    alpha_Tr = find_alpha(T);
    global Tc Pc_bar R_new_bar psi;
    a = psi*alpha_Tr*(R_new_bar^2)*(Tc^2)/(Pc_bar);
end

function P = Antoine_eqn(T)
%The following values correspond to P in bars, and T in
K.
    if (T < 308)   %Lower limit is 273K, but don't need
to put that.
%We won't go that low, and even if we do, the Antoine
eqns. will still be approximately valid
        A = 4.14157;
        B = 1377.578;
```

```
        C = -50.507;
    elseif (T >= 308 &&  T < 384)
        A = 4.07827;
        B = 1343.943;
        C = -53.773;
    elseif (T >= 384)  % We use these parameters to get
the guess value of P_sat for T > Tc too.
        A = 4.54436;
        B = 1738.123;
        C = 0.394; % A big change from the -ve values...
    end

    P = 10^(A - B/(T + C)); % Gives P in bars
    P = P*1e5; % Gives P in Pa

end

function mu = find_mu(T, v)
    global epsilon sigma b_new R_new_bar;
    a = find_a(T);

    Z = v/(v-b_new) - a*v/(R_new_bar*T*(v +
epsilon*b_new)*(v + sigma*b_new)) ;
    %If P is given, then Z = Pv/RT.
    q = a/(b_new*R_new_bar*T);
    mu = -v*log(1 - b_new/v) - (q/sigma-
epsilon)*log((v+sigma*b_new)/(v+epsilon*b_new)) + (Z-1) -
log(Z);
    %This actually calculates mu wrt the ideal gas at
same conditions of
    %T, P (i.e. by the departure function).
    mu = mu*R_new_bar*T;

end

% Calculating P_sat by equating chemical potentials of
liq and vap phases
% as taught to us in DH and in Assg 3, Q.4 : abs(mu_l -
mu_g) < tolerance.
% Works perfectly well, is just a little slow to compute
owing to the large
% amounts of computations.
function P_sat = find_Psat(T, tolerance)

    global R_new_bar b_new sigma epsilon
    P = Antoine_eqn(T);  %This is our initial guess.
    P = P/1e5;
```

```
    P_ini = P;
    a = find_a(T);
    eqn = [P, P*b_new*(sigma+epsilon-1)-R_new_bar*T,
b_new*(b_new*sigma*epsilon-(sigma+epsilon)*(b_new+
R_new_bar*T)), a-
sigma*epsilon*(b_new^2)*(P*b_new+R_new_bar*T)];
    v = roots(eqn);
    v_l = min(v(v>0));
    v_g = max(v);
    mu_l = find_mu(T, v_l);
    mu_g = find_mu(T, v_g);
    diff = abs(mu_g - mu_l);

    while(diff > tolerance)

        P1 = P_ini + 0.05; %Computing at a slightly
higher pressure. 0.05 can be changed for convergence
        eqn = [P1, P1*b_new*(sigma+epsilon-1)-
R_new_bar*T, b_new*(b_new*sigma*epsilon-
(sigma+epsilon)*(b_new+ R_new_bar*T)), a-
sigma*epsilon*(b_new^2)*(P1*b_new+R_new_bar*T)];
        v = roots(eqn);
        v_l1 = min(v(v>0));
        v_g1 = max(v);
        mu_l1 = find_mu(T, v_l1);
        mu_g1 = find_mu(T, v_g1);
        diff1 = abs(mu_g1 - mu_l1);

        P2 = P_ini - 0.05;  %Computing at a slightly
lower pressure. 0.05 can be changed for convergence
        eqn = [P2, P2*b_new*(sigma+epsilon-1)-
R_new_bar*T, b_new*(b_new*sigma*epsilon-
(sigma+epsilon)*(b_new+ R_new_bar*T)), a-
sigma*epsilon*(b_new^2)*(P2*b_new+R_new_bar*T)];
        v = roots(eqn);
        v_l2 = min(v(v>0));
        v_g2 = max(v);
        mu_l2 = find_mu(T, v_l2);
        mu_g2 = find_mu(T, v_g2);
        diff2 = abs(mu_g2 - mu_l2);

        if(diff1 < diff2)
            diff = diff1;
            P_ini = P1;
        else
            diff = diff2;
            P_ini = P2;
```

```
        end

    end
    P_sat = P_ini; %This P_ini has actually changed to
the correct final P_sat.
end

function [P, vl, vg] = P_by_fsolve(T)
    global b_new R_new_bar sigma epsilon

    a = find_a(T);
    q = a/(b_new*R_new_bar*T);
    P_guess = Antoine_eqn(T);
    P_guess = P_guess/1e5;

    eqn = [P_guess, P_guess*b_new*(sigma+epsilon-1)-
R_new_bar*T, b_new*(b_new*sigma*epsilon-
(sigma+epsilon)*(b_new+ R_new_bar*T)), a-
sigma*epsilon*(b_new^2)*(P_guess*b_new+R_new_bar*T)];
    v = roots(eqn);
    vl = min(v(v>0));
    vg = max(v);

        %F = @(x) [log((x(2)/x(3))*(x(3)- b_new)/(x(2)-
b_new)) - x(1)*(x(3)-x(2))/(R_new_bar*T) + (q/sigma-
epsilon)*log(((x(3)+sigma*b_new)/(x(2)+sigma*b_new))*((x(
2)+epsilon*b_new)/(x(3)+epsilon*b_new))) +
log(x(3)/x(2)); (x(2)^3)*x(1) +
(x(2)^2)*(x(1)*b_new*(sigma+epsilon-1) - R_new_bar*T) +
x(2)*(b_new*(b_new*sigma*epsilon - (sigma+epsilon)*(b_new
+ R_new_bar*T))) + a-sigma*epsilon*(b_new^2)*(b_new*x(1)
+ R_new_bar*T); (x(3)^3)*x(1) +
(x(3)^2)*(x(1)*b_new*(sigma+epsilon-1) - R_new_bar*T) +
x(3)*(b_new*(b_new*sigma*epsilon - (sigma+epsilon)*(b_new
+ R_new_bar*T))) + a-sigma*epsilon*(b_new^2)*(b_new*x(1)
+ R_new_bar*T)];
    F = @(x) [log(abs((x(2)/x(3))*((x(3)- b_new)/(x(2)-
b_new)))) - ((R_new_bar*T/(x(2)-b_new)) -
(a/((x(2)+sigma*b_new)*(x(2)+epsilon*b_new))))*(x(3)-
x(2))/(R_new_bar*T) + (q/sigma-
epsilon)*log(abs(((x(3)+sigma*b_new)/(x(2)+sigma*b_new))*
((x(2)+epsilon*b_new)/(x(3)+epsilon*b_new)))) +
log(abs(x(3)/x(2)));
        (x(2)^3)*x(1) +
(x(2)^2)*(x(1)*b_new*(sigma+epsilon-1) - R_new_bar*T) +
x(2)*(b_new*(b_new*sigma*epsilon - (sigma+epsilon)*(b_new
```

```
+ R_new_bar*T))) + a-sigma*epsilon*(b_new^2)*(b_new*x(1)
+ R_new_bar*T);
        (x(3)^3)*x(1) +
(x(3)^2)*(x(1)*b_new*(sigma+epsilon-1) - R_new_bar*T) +
x(3)*(b_new*(b_new*sigma*epsilon - (sigma+epsilon)*(b_new
+ R_new_bar*T))) + a-sigma*epsilon*(b_new^2)*(b_new*x(1)
+ R_new_bar*T)];

    ini_guesses = [P_guess, vl, vg];
    [x_ans, func_val_at_roots] = fsolve(F, ini_guesses);


    %IMP!!!!!!!!
    % x, x_ans is a vector with elements: P_sat, vl, vg.

    P = x_ans(1);
    vl = x_ans(2);
    vg = x_ans(3);
end

function plot_isotherm(T, col)

    global R_new_bar b_new sigma epsilon
    a = find_a(T);
    hold on;

    V_vec = [];
    P_vec = [];
        for v = 0.05: 0.01: 6
        P = R_new_bar*T/(v - b_new) -
a/((v+sigma*b_new)*(v+epsilon*b_new));
        P_vec = [P_vec, P];
        V_vec = [V_vec, v];
    end

    plot(V_vec, P_vec, 'color', col);
end

% For our system, by the code, vc obtained = 1.368 L/mol,
Pc = 20.31 bar, Tc = 524 K.
```

----------THANK YOU----------