Table of Contents

.... 1
Problem 1: Decomposition 1
Problem 2: Eigenvectors 2
Problem 3: Power Method 5
Problem 4: Newton-Raphson method 6
Problem 5: Height of water tank 8
Problem 6: Newton-Raphson method using analytical form of Jacobian 8
Problem 7: Bubble and Dew pressures 10
Problem 8: Batch Reactor 12
Problem 9: Pipe Network 13


```
function hmwk2_F15_final()

%{

Description:06-623 Homework 2 Solution
Prepared by: Burcu Karagoz  Sept 28, 2015

%}

clear all %clear stored variables
clc %clear the screen
close all %close all previously created plots
```

Problem 1: Decomposition

residence time tao= flow rate/volume= V/v

```
%dcA/dt=dcA0/tao-cA/tao-k1cA
%dcB/dt=dcB0/tao-cB/tao+k1cA-k2cB+k3cD-k4cB
%dcC/dt=dcC0/tao-cC/tao+k4cB
%dcD/dt=dcD0/tao-cD/tao+k2cB-k3cD
%   A=[(-k1-(1/tao)) 0                0            0;
%          k1       (-k2-k4-(1/tao)) 0            k3;
%     0                k4           (-1/tao)      0;
%     0                k2            0       (-1/tao)-k3]
%css=[cAss cBss cCss cDss]
%cI=1/tao*[cA0 cB0 cC0 cD0]

k=[ 0.1 0.2 0.1 0.8];
V=10; % volume of reactor, L
v=1; %flow rate, L/sec
tao=V/v; %residence time
cI=(1/tao)*[5 0 0 1]'; % feed vector

%A matrix
A=[-k(1)-(1/tao) 0 0 0; k(1) -k(2)-k(4)-(1/tao) 0 k(3); 0 k(4) -(1/
tao) 0; 0 k(2) 0 -(1/tao)-k(3)];
```

```
[L,U,P]=lu(A);
mult= inv(L)*P*-cI; %first multiplication of a triangular matrix by
 the feed vector
css=inv(U)*mult; %steady state concentrations
disp('first multiplication of a triangular matrix by the feed
 vector:')
disp(mult)
disp('steady state concentrations:')
disp(css)
```

```
first multiplication of a triangular matrix by the feed vector:
   -0.5000
   -0.2500
   -0.1818
   -0.1455

steady state concentrations:
    2.5000
    0.3000
    2.4000
    0.8000
```

Problem 2: Eigenvectors

```
A=[3 2 2 1; 2 3 1 2; -1 1 2 0; 2 4 3 5];    % Setting Matrix A
%Part a:
[W1,D1]=eig(A)                        % Determining eigenvalue and
 eigenvectors
                                      % of A, W1 gives the eigenvectors
                                      % diagonal of D1 gives the
 eigenvalues.
norm(W1(:,1))                         % Norm of eigenvector1
norm(W1(:,2))                         % Norm of eigenvector2
norm(W1(:,3))                         % Norm of eigenvector3
norm(W1(:,4))                         % Norm of eigenvector3
% **Normalization Procedure for Matlab**
% Matlab generally uses norm2.
% Matlab multiplies the vector by itself scalarly and takes a square
 root,
% then divides the terms in that vector with this value. For example
 for
% u=[ 3 4], normalization factor=sqrt(3*3+4*4)=5, normalized u=[3/5
 4/5].

%Part b sym function
[W2,D2]= eig(sym(A))                   % Usage of sym function for
 eigenspace of
                                       % A, W2 is eigenvector of A
                                       % D2 is the eigenvalue of A.
```

```
W1 =
```

```
   0.3446 + 0.0000i  -0.1195 - 0.3317i  -0.1195 + 0.3317i  -0.5000 +
 0.0000i
   0.4569 + 0.0000i  -0.5295 + 0.2518i  -0.5295 - 0.2518i  -0.5000 +
 0.0000i
   0.0183 + 0.0000i   0.7213 + 0.0000i   0.7213 + 0.0000i   0.5000 +
 0.0000i
   0.8198 + 0.0000i   0.0723 - 0.0799i   0.0723 + 0.0799i   0.5000 +
 0.0000i


D1 =

   8.1370 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 +
 0.0000i
   0.0000 + 0.0000i   1.4315 + 0.8090i   0.0000 + 0.0000i   0.0000 +
 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.4315 - 0.8090i   0.0000 +
 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   2.0000 +
 0.0000i


ans =

     1


ans =

    1.0000


ans =

    1.0000


ans =

     1


W2 =

[ -1, (43/(9*(454^(1/2)/3 + 341/27)^(1/3)) + (454^(1/2)/3 +
 341/27)^(1/3) + 11/3)^2/2 - 215/(9*(454^(1/2)/3 + 341/27)^(1/3))
 - 5*(454^(1/2)/3 + 341/27)^(1/3) - 31/3, (43/(18*(454^(1/2)/3 +
 341/27)^(1/3)) + (454^(1/2)/3 + 341/27)^(1/3)/2 + (3^(1/2)*(43/
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2
 - 11/3)^2/2 + 215/(18*(454^(1/2)/3 + 341/27)^(1/3)) + (5*(454^(1/2)/3
 + 341/27)^(1/3))/2 + (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3))
 - (454^(1/2)/3 + 341/27)^(1/3))*5i)/2 - 31/3, (43/(18*(454^(1/2)/3
 + 341/27)^(1/3)) + (454^(1/2)/3 + 341/27)^(1/3)/2 - (3^(1/2)*(43/
```

```
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2
 - 11/3)^2/2 + 215/(18*(454^(1/2)/3 + 341/27)^(1/3)) + (5*(454^(1/2)/3
 + 341/27)^(1/3))/2 - (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3)) -
 (454^(1/2)/3 + 341/27)^(1/3))*5i)/2 - 31/3]
[ -1, (43/(9*(454^(1/2)/3 + 341/27)^(1/3)) + (454^(1/2)/3 +
 341/27)^(1/3) + 11/3)^2/2 - 172/(9*(454^(1/2)/3 + 341/27)^(1/3)) -
 4*(454^(1/2)/3 + 341/27)^(1/3) - 44/3,          (43/(18*(454^(1/2)/3
 + 341/27)^(1/3)) + (454^(1/2)/3 + 341/27)^(1/3)/2 + (3^(1/2)*(43/
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2
 - 11/3)^2/2 + 86/(9*(454^(1/2)/3 + 341/27)^(1/3)) + 2*(454^(1/2)/3
 + 341/27)^(1/3) + 3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3))
 - (454^(1/2)/3 + 341/27)^(1/3))*2i - 44/3,          (43/
(18*(454^(1/2)/3 + 341/27)^(1/3)) + (454^(1/2)/3 + 341/27)^(1/3)/2
 - (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3
 + 341/27)^(1/3))*1i)/2 - 11/3)^2/2 + 86/(9*(454^(1/2)/3 +
 341/27)^(1/3)) + 2*(454^(1/2)/3 + 341/27)^(1/3) - 3^(1/2)*(43/
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*2i -
 44/3]
[  1,          43/(454^(1/2)/3 + 341/27)^(1/3) - (43/(9*(454^(1/2)/3
 + 341/27)^(1/3)) + (454^(1/2)/3 + 341/27)^(1/3) + 11/3)^2 +
 9*(454^(1/2)/3 + 341/27)^(1/3) + 26,       26 - 43/(2*(454^(1/2)/3 +
 341/27)^(1/3)) - (9*(454^(1/2)/3 + 341/27)^(1/3))/2 - (3^(1/2)*(43/
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*9i)/2
 - (43/(18*(454^(1/2)/3 + 341/27)^(1/3)) + (454^(1/2)/3 +
 341/27)^(1/3)/2 + (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3))
 - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2 - 11/3)^2,       26 - 43/
(2*(454^(1/2)/3 + 341/27)^(1/3)) - (9*(454^(1/2)/3 + 341/27)^(1/3))/2
 + (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3
 + 341/27)^(1/3))*9i)/2 - (43/(18*(454^(1/2)/3 + 341/27)^(1/3)) +
 (454^(1/2)/3 + 341/27)^(1/3)/2 - (3^(1/2)*(43/(9*(454^(1/2)/3 +
 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2 - 11/3)^2]
[  1,

                         1,



               1,




     1]


D2 =

[ 2,
       0,

                              0,

                                                      0]
```

```
[ 0, 43/(9*(454^(1/2)/3 + 341/27)^(1/3)) + (454^(1/2)/3 +
 341/27)^(1/3) + 11/3,

                                             0,

                                                                 0]
[ 0,
           0, 11/3 - (454^(1/2)/3 + 341/27)^(1/3)/2 - (3^(1/2)*(43/
(9*(454^(1/2)/3 + 341/27)^(1/3)) - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2
 - 43/(18*(454^(1/2)/3 + 341/27)^(1/3)),

                                                           0]
[ 0,
           0,

                                             0, 11/3 - (454^(1/2)/3
 + 341/27)^(1/3)/2 + (3^(1/2)*(43/(9*(454^(1/2)/3 + 341/27)^(1/3))
 - (454^(1/2)/3 + 341/27)^(1/3))*1i)/2 - 43/(18*(454^(1/2)/3 +
 341/27)^(1/3))]
```

Problem 3: Power Method

```
A=[ -2 2 -3; 2 1 -6; -1 -6 0]; % setting A,you can change A from here
[eval, evec]=powerfunc(A)     % use powerfunc


function [eval ,evec] = powerfunc(A) % power method with given A

x = [1 0 0]';                        % setting initial eigenvector
maxIt = 100000;                      % set maximum number of iteration
eps = 1e-6;                          % tolerance
% starting power method
for i=1:maxIt,
    y = A*x;                         % x_i+1=Ax_i
    y = y / sqrt(sum(y.^2));         % normalize new vector
    if sum((x - y).^2) < eps,        % if new vector is almost similar
 to
                                       %previous one then we got the
 evector
           break;
    end
    x = y;
end

eval = max(max(A*x / x));          % calculation of largest eigenvalue
evec = x;                          % eigenvector for largest
 eigenvalue
end


eval =
```

```
    7.3306


evec =

    0.3501
    0.7003
   -0.6221
```

Problem 4: Newton-Raphson method

```
%f(x)= x^3-5x^2+7x-3=(x-1)^2(x-3)--> roots = 1(double root),3
disp('f(x)=(x-1)(x-1)(x-3)');
disp('roots= 1(double root),3');

xStan = zeros(6,1); %generating x for standard update function,
 initial x=0
xMod = zeros(6,1);  %generating x for modified update function,initial
 x=0

f=@(x) (x)^3-5*(x)^2+7*(x)-3;   % f(x)
df=@(x) 3*(x)^2-10*(x)+7;       % first derivative of f(x)
d2f=@(x) 6*(x)-10;              % second derivative of f(x)
Ustan=@(x) -f(x)/df(x);         % standard update function
Umod=@(x) -((f(x)*df(x))/((df(x)^2)-f(x)*d2f(x)));  %modified update
 function
for i=1:5,
    % Newton Raphson using standard update function:
    xStan(i+1) = xStan(i) + Ustan(xStan(i));
    % Newton Raphson using modified update function:
    xMod(i+1) = xMod(i) + Umod(xMod(i));
end


roots = [1 3];
disp(xStan);
disp('error%:')
disp(abs((xStan(6) - roots) ./xStan(6))*100); % calculation of error
 between roots of f(x) and result from NR_standard
disp(xMod);
disp('error%:')
disp(abs((xMod(6) - roots) ./xMod(6))*100);% calculation of error
 between roots of f(x) and result from NR_modified

xStan(1) = 4;       %initial guess for x used in standard update=4
xMod(1) = 4;        %initial guess for x used in modified update=4

f=@(x) (x)^3-5*(x)^2+7*(x)-3; % f(x)
df=@(x) 3*(x)^2-10*(x)+7;     %first derivative of f(x)
d2f=@(x) 6*(x)-10;            % second derivative of f(x)
Ustan=@(x) -f(x)/df(x);       % standard update function
```

```
Umod=@(x) -((f(x)*df(x))/((df(x)^2)-f(x)*d2f(x)));%modified update
 function

for i=1:5,
    % Newton Raphson using standard update function:
    xStan(i+1) = xStan(i) + Ustan(xStan(i));
    % Newton Raphson using modified update function:
    xMod(i+1) = xMod(i) + Umod(xMod(i));
end


roots = [1 3];
disp(xStan);
disp('error%:')
disp(abs((xStan(6) - roots) ./xStan(6))*100);  % calculation of error
 between roots of f(x) and result from NR_standard
disp(xMod);
disp('error%:')
disp(abs((xMod(6) - roots) ./xMod(6))*100); % calculation of error
 between roots of f(x) and result from NR_modified
disp('The two update functions approach the different solutions with
 different initial guess');
```

```
f(x)=(x-1)(x-1)(x-3)
roots= 1(double root),3
         0
    0.4286
    0.6857
    0.8329
    0.9133
    0.9558

error%:
    4.6262  213.8787

         0
    1.1053
    1.0031
    1.0000
    1.0000
    1.0000

error%:
    0.0000  200.0000

    4.0000
    3.4000
    3.1000
    3.0087
    3.0001
    3.0000

error%:
   66.6667    0.0000
```

```
    4.0000
    2.6364
    2.8202
    2.9617
    2.9985
    3.0000

error%:
   66.6666    0.0001

The two update functions approach the different solutions with
 different initial guess
```

Problem 5: Height of water tank

```
L=5; %Length of pipe, m
g=9.81; % gravimetric acceleration, m/s2
V_t= 5; % velocity of water after 3 seconds, m/s
t=3; % time, s
    function F=height(h);
            % make the given equation like f(x)=0. So subtract one
 side
            % from the other make them equal to zero.
            % F(h)=(V_t/sqrt(2*g*h))-tanh((t*sqrt(2*g*h))/(2*L))=0
        F=(V_t/sqrt(2*g*h))-tanh((t*sqrt(2*g*h))/(2*L));
    end
h=fsolve(@height,10) % taking height function with initial guess of
 10, solving F(h)=0, unit is in m.
```

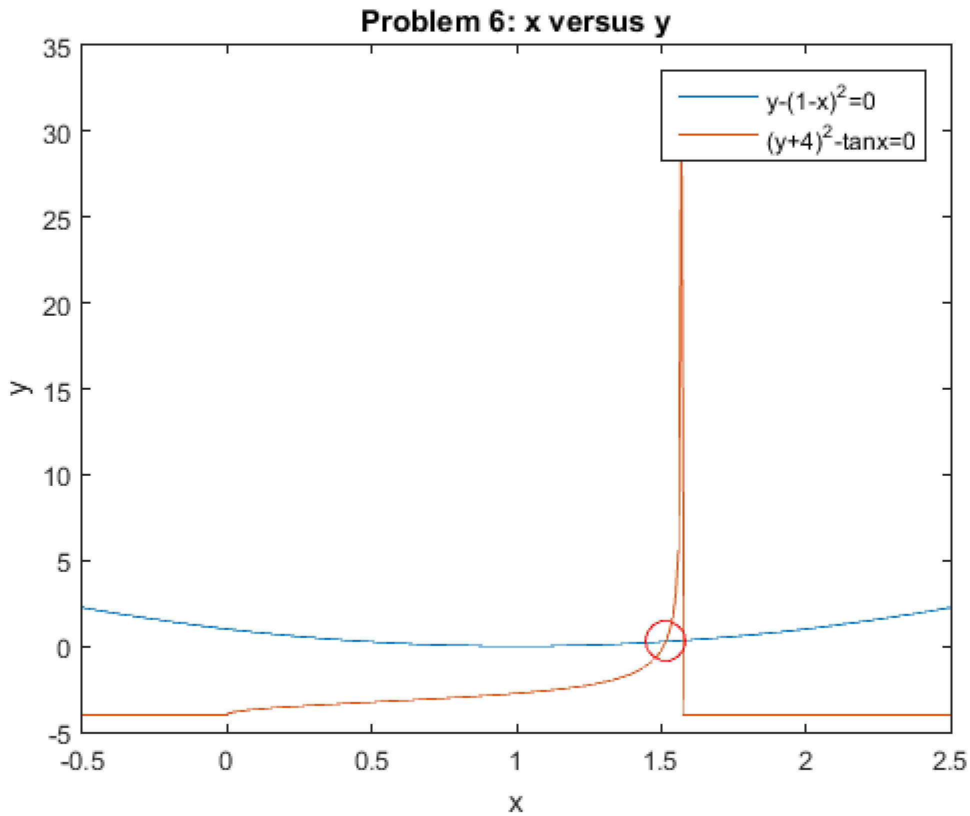
```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.




h =

    1.4896
```

Problem 6: Newton-Raphson method using analytical form of Jacobian

```
f1 = @(x,y) y - (x-1).^2;  % f1(x,y)=0
f2 = @(x,y) (y + 4).^2 - tan(x); % f2(x,y)=0
f = @(x,y) ([f1(x,y);f2(x,y)]); % setting nonlinear equation systems,
 function systems of f
```

```
J = @(x,y) ([-2*x + 2 ,1; -1/(cos(x).^2),2*y+8]); % Jacobian is
 calculated by hand

eps = 1e-6; %Tolerance criteria
maxIt = 100000; % maximum number of iteration
sol = zeros(maxIt,2); % x and y for NR
fval = zeros(maxIt,2); % f entering the NR
%NR method using Jacobian for nonlinear equation systems
for i=2:maxIt,
    fval(i-1,:) = f(sol(i-1,1),sol(i-1,2))';
    sol(i,:) = sol(i-1,:) -
 (inv(J(sol(i-1,1),sol(i-1,2)))*fval(i-1,:)')';

    if sum((sol(i,:) - sol(i-1,:)).^2) < eps,   % convergence criteria
        maxIt = i;
        break;
    end
end
fval(maxIt,:) = f(sol(maxIt,1),sol(maxIt,2))';  % solution for
 f1(x,y)=0 and f2(x,y)=0

X = -0.5:0.01:2.5;  % generating X
plot(X,
 (X-1).^2,X,sqrt(tan(X))-4,sol(maxIt,1),sol(maxIt,2),'or','MarkerSize',15); %
 Problem 6: x versus y
xlabel('x');
ylabel('y');
title('Problem 6: x versus y');
legend('y-(1-x)^2=0','(y+4)^2-tanx=0');
disp('one of the solutions');
disp(sol(maxIt,:));
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
one of the solutions
    1.5159    0.2662
```



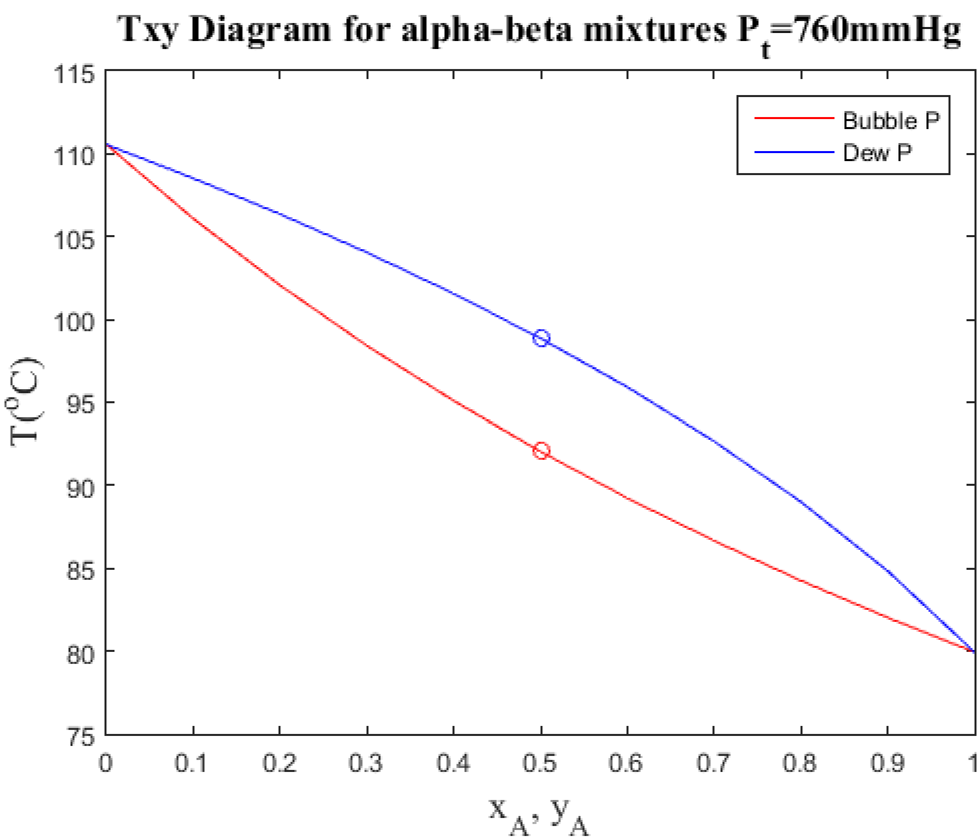
Problem 7: Bubble and Dew pressures

```
Tguess=300;
xinit=0.25;

for i=1:11
    xinit=0.1*(i-1);
    CalcT_bbl=fzero(@(T) bubbleT(T,xinit),Tguess);
    bubT(i,1)=xinit;
    bubT(i,2)=CalcT_bbl;
end

for i=1:11
    yinit=0.1*(i-1);
    CalcT_dew=fzero(@(T) dewT(T,yinit),Tguess);
    dewptT(i,1)=yinit;
    dewptT(i,2)=CalcT_dew;
end

figure
plot(bubT(:,1),bubT(:,2),'r',dewptT(:,1),dewptT(:,2),'b',bubT(6,1),bubT(6,2),'ro',
    dewptT(6,1),dewptT(6,2),'bo')
xlabel('x_A, y_A','fontsize',14,'fontname','times new roman')
ylabel('T(^oC)','fontsize',14,'fontname','times new roman')
```

```
title('Txy Diagram for alpha-beta mixtures
 P_t=760mmHg','fontsize',14,'fontname','times new roman')
legend('Bubble P','Dew P')

str3a=sprintf('The boiling temperature of an equimolar mixture is
 %0.2d C at 1 atm',bubT(6,2));
disp(str3a)

str3b=sprintf('The dew temperature of an equimolar mixture is %0.2d C
 at 1 atm\n',dewptT(6,2));
disp(str3b)

%Comparison
del = dewptT(5,2)-bubT(5,2);

if (abs(del) > 1)
    disp('The temperatures are different, so these two are not in
 equilibrium')
else
    disp('This might just be equilibrium!')
end


    function Pdiff=dewT(T,yinit)
        Ptotal=760; %mm Hg
        yb = yinit;
        A1=-3848.09;
        B1=17.5318;
        A2=-4328.12;
        B2=17.913;
        P1sat=exp(A1/(T+273.15)+B1);
        P2sat=exp(A2/(T+273.15)+B2);
        Pinv=(yb/P1sat+(1-yb)/P2sat);
        Pcalc=1/Pinv;
        xB=yb*Ptotal/P1sat;
        xT=((1-yb)*Ptotal)/P2sat;
        Pdiff=Pcalc-Ptotal;
    end

    function Pdiff=bubbleT(T,xinit)
        Ptotal=760; %mm Hg
        xb = xinit;
        A1=-3848.09;
        B1=17.5318;
        A2=-4328.12;
        B2=17.913;
        P1sat=exp(A1/(T+273.15)+B1);
        P2sat=exp(A2/(T+273.15)+B2);
        Pcalc=(xb*P1sat+(1-xb)*P2sat);
        Pdiff=Pcalc-Ptotal;
    end
```

```
The boiling temperature of an equimolar mixture is 9.20e+01 C at 1 atm
The dew temperature of an equimolar mixture is 9.89e+01 C at 1 atm
```

The temperatures are different, so these two are not in equilibrium



Problem 8: Batch Reactor

```
errorf=1; %Dummy value for error
%Initial guess - solution is sensitive to the guess. Logical choices
% are needed.
xi=[0.1;0.1];
% Call the function f with the two coupled equations
[xf,fval,exitflag]=fsolve(@f_1,xi,optimset('Display','off'));
errorf=norm(fval,2); %Some analysis of error is a good idea
if (exitflag == 1)
disp('Converged')
else
disp('Problem with fsolve')
end
function H=f_1(e) %Function called in batch reactor
H=zeros(2,1);
H(1)= (4*e(1)^2)*((1/3)+e(1)-e(2))/((1/3)-2*e(1))^2-0.1071;
H(2)=(4*e(2)^2)/((1/3+e(1)-e(2))*(1/3-e(2)))-0.01493;
end
display('The solution (x_1 and x_2) is')
disp(xf)
str_err=sprintf('The norm of the error in f(x) is %d',errorf);
str_l=sprintf('The error was left at the standard 1x10^-6');
```

```
disp(str_err)
disp(str_l)
%Part b
xi=[0.5;0.1];
[xf,fval,exitflag]=fsolve(@f_2,xi,optimset('Display','off'));
errorf=norm(fval,2); %Some analysis of error is a good idea
if (exitflag == 1)
disp('Converged')
else
disp('Problem with fsolve')
end
function H=f_2(e) %Function called in batch reactor
H=zeros(2,1);
H(1)= (2*e(1)^2)*((1/3)+e(1)-e(2))/(2-2*e(1))^2-0.1071;
H(2)=(2*e(2)^2)/((1/3+e(1)-e(2))*(1/3-e(2)))-0.01493;
end
display('The solution (x_1 and x_2) is')
disp(xf)
str_err=sprintf('The norm of the error in f(x) is %d',errorf);
str_l=sprintf('The error was left at the standard 1x10^-6');
disp(str_err)
disp(str_l)

Converged
The solution (x_1 and x_2) is
    0.0583
    0.0208

The norm of the error in f(x) is 1.418465e-09
The error was left at the standard 1x10^-6
Converged
The solution (x_1 and x_2) is
    0.3632
    0.0381

The norm of the error in f(x) is 3.049501e-11
The error was left at the standard 1x10^-6
```

Problem 9: Pipe Network

```
L1=100; % set length of pipe line 1
L2=100; % set length of pipe line 2
L3=200; % set length of pipe line 3
L4=75; % set length of pipe line 4
L5=100; % set length of pipe line 5
L6=75; % set length of pipe line 6
L7=50;% set length of pipe line 7
q0=0.1.*ones(7,1); % initial guess for q(i)= 0.1

q=fsolve(@fun,q0(:,1)) % use fun with initial guess of q0, and
 calculates
                      %the each flow rate
    function Q=fun(q),
```

```
        % setting all seven equations into the Q
        Q=[q(1)-q(2)-q(6); q(2)-q(4)-q(3);q(7)-q(6)-q(5);q(5)-q(4)-
q(3);L3*q(3).^2-L4*q(4)^2;
            L2*q(2)^2+L4*q(4)^2+L5*q(5)^2-
L6*q(6)^2;L1*q(1)^2+L6*q(6)^2+L7*q(7)^2-((5.2*10^5*pi()^2*(0.2)^5)/
(8*0.02*998))];
    end
```

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.




q =

    0.2388
    0.0869
    0.0330
    0.0539
    0.0869
    0.1519
    0.2388
```

```
end
```

Published with MATLAB® R2015a