**ChE352: Final Project**

**Mickey Huang**

**May 9, 2018**

**Prof. Okorafor**

**Number of hours spent on this project: 48 hours**

AspenTech – 100 Fake Road, Newtown, PA Sent: 5/9/2018

To: Sean F. Askebom

From: Mickey Huang

Subject: Vapor Pressure Curve Fitting

Dear Mr. Askebom,

Over the past few days, I have used several numerical methods to approximate the vapor pressures over a range of temperatures for several reagents we use in our flash reactors: C3H6, 1-C4H8, n-C4H10, and cis-2-C4H8. I used numerical method to curve fit a reliable experimental data points from Perry’s Handbook for Chemical Engineers.

The data was determined to be exponential in nature using graph. To model this relationship, I modeled the relationship using Antoine’s method. Antoine is a semi-experimental method, the model generally works decently well, the accuracy of the model depends on how accurate the Antoine parameters A, B, and C. To determine the Antoine parameters, I first linearized the equation by taking the natural log of the equation and then use multiple linear regression to determine the A, B, and C values. A pseudo code of the algorithm is attached in the next page.

Regression is a reliable method to determine the parameters. However, this method tend to suffers greater error at small temperature and pressure. Since the data in the Handbook is accurate to the last digit ±1, therefore an improvement must be implemented for this model to work. One of the disadvantage Antoine equation suffers is it cannot be used to describe the entire saturated vapor pressure curve from the [triple point](https://en.wikipedia.org/wiki/Triple_point) to the [critical point](https://en.wikipedia.org/wiki/Critical_point_(thermodynamics)) because it lacks the flexibility. To solve this problem, multiple parameter sets for a single component are used. A low-pressure parameter set is used to describe the vapor pressure curve up to the [normal boiling point](https://en.wikipedia.org/wiki/Normal_boiling_point) and the second set of parameters is used for the range from the normal boiling point to the critical point. This gives us a piecewise-defined exponential function to approximate the relation of temperature and vapor pressure while minimize the error at smaller pressure.

Antoine and regression method saves computational cost and time. Methods like Lagrange interpolation require up to 14th order polynomials to get the right answer; many interpolation method’s range did not always extend to range of 170K to the critical temperature, where there is an equilibrium between liquid and gas. Using a single set parameters for Antoine can yield a percent error of 20-64% at low temperature. However, by defining a low pressure and high pressure parameter the error can be reduced up to 5-8%. Since Perry’s Handbook is accurate until last digit ±1, this inaccuracy is a source of error in our approximation. Other sources of error include truncation error and round-off error generated by the computing device’s inability to store highly precise numbers.

Attached is a script of my work in this approximation. I believe this method exceeds your specifications. Please let me know if you require any further question

Sincerely,

Mickey Huang

%This pseudo code is for determining Antoine parameter

%A,B,C Antoine parameter

log(P) = A - B / ( T + C )

log(P) = A - B / ( T + C )

T.log(P) + C.log(P) = A.T + A.C - B

log(P) = A + (A.C-B)/T - C.log(P)/T

%a0 = A a1=A\*C-B a2=-C

%x1=1/T x2=log(P)/T

y = a0 + a1.x1 + a2.x2

X = [ones(size(x1)) x1 x2];

b = regress(y,X) %multiple linear regression return to 3 coefficients

b(1)=a0=A %A

-b(3)=a2=C %C

b(2)=a1

B=A\*C-b(2)

%Psat1

%Mickey Huang

%Find Vapor Pressure of C3H6 at different temperatures

%Based on Antoine fit of tabulated data

function [VaporPressure] = Psat1(T)

format long

%Critical Temperature

CritTemp = 365.6;

%CASE 1: INVALID T

%Throw errors and complain a lot if you get a bad (high/low) T input

if(T<170 || T>CritTemp)

disp('ERROR: INVALID TEMPERATURE');

VaporPressure = NaN;

%CASE 2: From 170K to BP

elseif(T>=170 && T<=225.46)

Y=9.1667-1838.8271/(T-24.5651);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure);

%CASE 3: BP to CP

elseif(T>225.46 && T<=356.6)

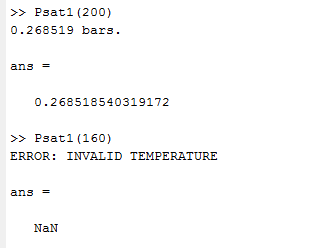
Y=11.4006-2972.6296/(T+35.8244);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure)

end

end



%Psat2

%Mickey Huang

%Find Vapor Pressure of 1-C4H8 at different temperatures

%Based on Antoine fit of tabulated data

function [VaporPressure] = Psat2(T)

format long

%Critical Temperature

CritTemp = 418.6;

%CASE 1: INVALID T

%Throw errors and complain a lot if you get a bad (high/low) T input

if(T<170 || T>CritTemp)

disp('ERROR: INVALID TEMPERATURE');

VaporPressure = NaN;

%CASE 2: From 170K to BP

elseif(T>=170 && T<=266.65)

Y=9.745-2336.5203/(T-26.614);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure);

%CASE 3: BP to CP

elseif(T>266.65 && T<=CritTemp)

Y=9.6916-2407.7014/(T-18.0434);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure)

end

end

%Psat3

%Mickey Huang

%Find Vapor Pressure of n-C4H10 at different temperatures

%Based on Antoine fit of tabulated data

function [VaporPressure] = Psat3(T)

format long

%Critical Temperature

CritTemp = 425.2;

%CASE 1: INVALID T

%Throw errors and complain a lot if you get a bad (high/low) T input

if(T<170 || T>CritTemp)

disp('ERROR: INVALID TEMPERATURE');

VaporPressure = NaN;

%CASE 2: From 170K to BP

elseif(T>=170 && T<=272.65)

Y=9.3851-2292.2293/(T-27.9602);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure);

%CASE 3: BP to CP

elseif(T>272.65 && T<=CritTemp)

Y=10.2311-2861.4834/(T+8.2712);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure)

end

end

%Psat4

%Mickey Huang

%Find Vapor Pressure of cis-2-C4H8 at different temperatures

%Based on Antoine fit of tabulated data

function [VaporPressure] = Psat4(T)

format long

%Critical Temperature

CritTemp = 435.5;

%CASE 1: INVALID T

%Throw errors and complain a lot if you get a bad (high/low) T input

if(T<170 || T>CritTemp)

disp('ERROR: INVALID TEMPERATURE');

VaporPressure = NaN;

%CASE 2: From 170K to BP

elseif(T>=170 && T<=276.8)

Y=9.4383-2310.2283/(T-31.5591);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure);

%CASE 3: BP to CP

elseif(T>272.65 && T<=CritTemp)

Y=10.0635-2711.8215/(T-6.9010);

VaporPressure=exp(Y);

fprintf('%f bars.\n',VaporPressure)

end

end

flash1.m Description

flash1 is a subroutine that determines the properties of the outlet streams of a flash reactor using feed component molar flow rates (zF), the temperature in the flash (T), and the pressure in the flash (P) to return the mole fractions of the four chemical species in the vapor outlet stream (y) liquid outlet stream (x), flow rate of vapor outlet stream (V) and flow rate of liquid outlet stream (L). The subroutine called Psat, another subroutine, to calculate the saturated pressure. And the 4 saturated pressure are compressed into a 4x1 matrices that correspond with the four species in order of increasing boiling point. The saturated pressure and mole fractions was used to calculate bubble and dew point pressure and check if the input pressure is outside of the temperature range. This subroutine will error if the input conditions do not allow the flash to operate between the mixture dew and bubble point. This subroutine uses a root-finding algorithm on a nonlinear equation Rachford-Rice equation to determine V and then back-calculate all other relevant variables.

Yes

Input conditions check

Inputs

zF,T,P

V

Solve Rachford-Rice equation using root-finding algorithm

Outputs

x, y, V, L

Find x, y

Find L

Error message:

Output: NaN

No

function [x y V L] = flash1(zF, T, P)

%% input check

disp('Temperature in Kelvin and pressure in bar');

disp('zF must be entered in the order of Stream B, E, F, G to get the right answer');

%Calculate Vapor Pressures at the input Temperature and pressure

m=iscolumn(zF);

if m ~= 1

disp('zF must be a column vector');

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

n=size(zF);

if n ~= 4

disp('4 input stream required');

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

if size(P) ~= 1

disp('P dimension incorrect')

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

if P < 0

disp('negative input pressure detected');

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

if size(T) ~= 1

disp('T dimension incorrect')

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

Psat = [Psat1(T) Psat2(T) Psat3(T) Psat4(T)]';

%Check that the input temperature is able to return a vapor pressure,

%otherwise the program will throw an error.

if(isnan(Psat)==1)

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

%stream properties calculation

%Approximating equilibrium constant as the ratio of Vapor Pressure to Pressure

F = sum(zF);

z = zF/F;

K = Psat/P;

%Check that flash is operating within bubble dew point

Pbubble = (z(1)\*Psat1(T)+z(2)\*Psat2(T)+z(3)\*Psat3(T)+z(4)\*Psat4(T));

Pdew = (z(1)/Psat1(T)+z(2)/Psat2(T)+z(3)/Psat3(T)+z(4)/Psat4(T))^-1;

if P > Pbubble

disp('Error: Pressure above bubble-point')

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

if P < Pdew

disp('Error: Pressure below dew-point')

x = NaN;

y = NaN;

V = NaN;

L = NaN;

return

end

%Check to see if z adds up to 1, if negative component in zF is

%detected, then sum(z) can't be equal to 1

if sum(z)~= 1

disp('mole fractions of the stream does not add up to 1');

return

end

%% Solving using Rachford-Rice equation

%Defining Rachford-Rice equation as a function of V, all other variables are known.

fun= @(V) sum(zF.\*(1-K)./(F+V\*(K-1)));

%Solve the equation using root finding algorithm

V = fzero(fun, sum(zF));

%Calculate other properties of the outlet streams

L = F - V;

x = zF./((K-1)\*V+F);

y = K.\*x;

L

V

xB=x(1)

xE=x(2)

xF=x(3)

xG=x(4)

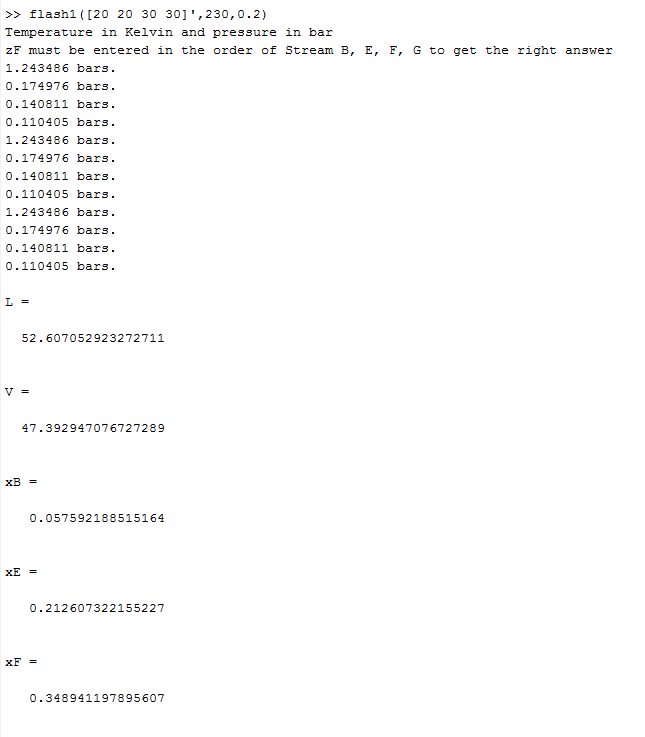
yB=y(1)

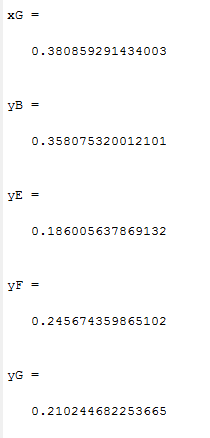
yE=y(2)

yF=y(3)

yG=y(4)

end





flash2.m Description

flash2 is a subroutine that determines the properties of the outlet streams using feed component molar flow rates (zF), the temperature in the flash (T), and the flow rate of the liquid outlet stream (L) to return the mole fractions of the four chemical species in the vapor outlet stream (y) and liquid outlet stream (x), flow rate of vapor outlet stream (V) and pressure in the flash (P). The subroutine called Psat, another subroutine, to calculate the saturated pressure; the four pressures are packaged into a 4x1 matrix that correspond with the four species in order of increasing boiling point. This subroutine will return one error if the input conditions do not allow the flash to operate between the mixture dew and bubble point. This subroutine uses Newton’s method for nonlinear systems of equations to solve a system of 10 nonlinear equations of 10 variables. The program was hardcoded a Jacobian matrix (J) and use that matrix to take a newton step and constantly update the Jacobian. This subroutine will continue until the result reaches convergence or maximum number of iterations are reached. After the calculations has completed, the ratio of V/F will be evaluated to check if the result is outside of bubble-dew range.

Outputs

x,y,V,P

No

Yes

Bubble dew point check

Max iteration reached

No

Check Convergence

Newton Step

No

Y

Error

Output:NaN

Calculate Jacobian

Inputs Check

Inputs

zF,T,L

%Credit for the seniors for telling me to use for loop and vertcat to do

%this Newton-Raphson's problem instead of using while loop

function [x y V P] = flash2(zF, T, L)

disp('Temperature in Kelvin');

disp('zF must be entered in the order of Stream B, E, F, G to get the right answer');

%% input check

%set maximum number of iterations for the Newton's method algorithm

kmax = 10000;

epsilon = 1e-6;

%Calculate Vapor Pressures at the input Temperature and pressure

m=iscolumn(zF);

if m ~= 1

disp('zF must be a column vector');

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

n=size(zF);

if n ~= 4

disp('4 input stream required');

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

%Calculate Vapor Pressures at the input Temperature

Psat = [Psat1(T) Psat2(T) Psat3(T) Psat4(T)]';

%Check that the input temperature is able to return a vapor pressure,

%otherwise throw an error.

if(isnan(Psat)==1)

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

if size(L)~=1

disp('L dimension incorrect')

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

if L <= 0

disp('No L detected');

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

if size(T) ~= 1

disp('T dimension incorrect')

x = NaN;

y = NaN;

V = NaN;

P = NaN;

return

end

F = sum(zF);

z = zF/F;

if sum(z) ~= 1

disp('mole fraction does not add up')

return

end

%% Initialization

%Initial Guesses for Newton's method

x{1} = z;

y{1} = z;

V{1} = F-L;

P{1} = 1;

%all variables packaged into an array with vertcat function

A{1} = vertcat(x{1},y{1},V{1},P{1});

%% Newton Raphson

%Begin iteration

for i=1:kmax

%Jacobian Matrix

%x1...x4 y1...y4 V P

J = [1 1 1 1 0 0 0 0 0 0; %sumx

0 0 0 0 1 1 1 1 0 0; %sum y

-Psat(1) 0 0 0 P{i} 0 0 0 0 y{i}(1); %Py=Psatx

0 -Psat(2) 0 0 0 P{i} 0 0 0 y{i}(2);

0 0 -Psat(3) 0 0 0 P{i} 0 0 y{i}(3);

0 0 0 -Psat(4) 0 0 0 P{i} 0 y{i}(4);

L 0 0 0 V{i} 0 0 0 y{i}(1) 0; %zF=yV+xL

0 L 0 0 0 V{i} 0 0 y{i}(2) 0;

0 0 L 0 0 0 V{i} 0 y{i}(3) 0;

0 0 0 L 0 0 0 V{i} y{i}(4) 0];

%residuals

r(1) = sum(x{i}) - 1;

r(2) = sum(y{i}) - 1;

r(3:6) = P{i}\*y{i}-Psat.\*x{i};

r(7:10) = y{i}\*V{i}+x{i}\*L-zF;

del=-J\r';

%Take a newton step

A{i+1} = A{i}+del;

%repackage variables into vars array

x{i+1} = A{i+1}(1:4);

y{i+1} = A{i+1}(5:8);

V{i+1} = A{i+1}(9);

P{i+1} = A{i+1}(10);

%convergence

if(abs(norm(A{i+1})-norm(A{i}))/norm(A{i})<epsilon)

x=x{end}

y=y{end}

V=V{end}

P=P{end}

%Check condition

if(V/F<=0 || V/F>=1)

x = NaN;

y = NaN;

V = NaN;

P = NaN;

disp('tempearture-Pressure condtion outside of bubble-dew range');

return

end

return;

end

end

disp('Max iteration reached, did not reach convergence')

x = NaN;

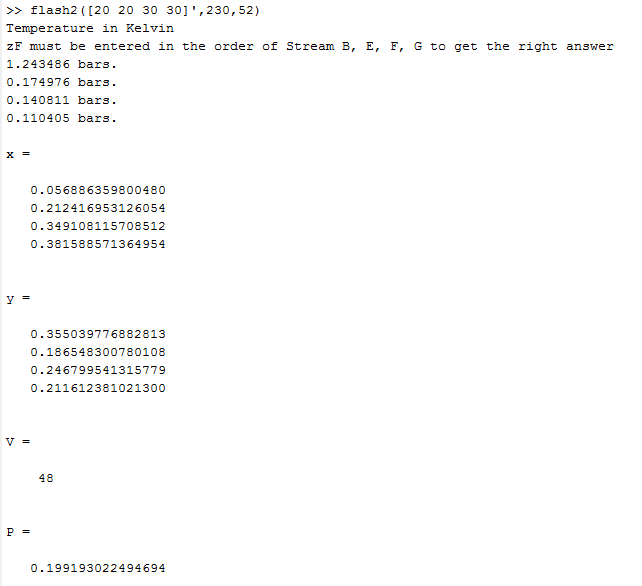
y = NaN;

V = NaN;

P = NaN;

return

end



Note: Same set up as flash1 ex. But use the L=52 calculated by flash1 as input for flash2. Results matches

flash3.m Description

flash3 is a subroutine that determines the properties of the outlet streams of a flash reactor using feed component molar flow rates (zF), the temperature in the flash (T), and the desired mole fraction of the lightest boiling species in the vapor outlet (y1) to return the mole fractions of the remaining three chemical species in the vapor outlet stream (y2, y3, y4), the mole fractions of the liquid outlet stream (x), flow rate of vapor outlet stream (V), flow rate of liquid outlet stream (L) and the pressure in the flash (P). An input check was first implemented to check valid input. Then the a system of 5 equations was set up to solve for y2,y3,y4,V, and P. An initial guess was set up and this initial guess was input into a Newton algorithm to calculate a better initial guess.

This subroutine uses the method of steepest descent for nonlinear systems to compute a gradient (D) and take a step of .5. If the new result was greater than the old result, the step is decreased using Armijo’s rule, and the iteration restarts using the new step size. If the new result is less than the old result, the initial guess vector will be updated, and the new initial guess will be used in computation. The procedure will continue until a tolerance is reached or maximum number of iteration reached. After the calculations has completed (if converge), bubble-dew point condition will be checked, and back calculation will be used to find other variables.

Backward calculation

Output:x,z2,z3,z4,V,P

Y

No

N

Bubble-dew point check

N

Check Convergence

Yes, update step size

Newfun>old fun

Take a steep step

Calculate gradient

Better initial guess

Newton’s method

Initial guess

Error

Output:NaN

Input check

Input

zF,T,y1

%%Steepest descent is very senstitive to initial guesses. This algorithm

%%attempts to improve the previous steepest descent algorithm by

%%incorporating Newton's Method to find a better initial guess.

%%This method requires high computational cost, to minimize the cost, the

%%10 equations are reduced to 5 equations and different naming was used

function [x y2 y3 y4 V L P] = flash3(zF,T,y1)

Go = 1;

%% input check

m=iscolumn(zF);

if m ~= 1

disp('zF must be a column vector');

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

n=size(zF);

if n ~= 4

disp('4 input stream required');

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

%Calculate Vapor Pressures at the input Temperature

P1=Psat1(T);

P2=Psat2(T);

P3=Psat3(T);

P4=Psat4(T);

Psat = [P1 P2 P3 P4]';

%Check that the input temperature is able to return a vapor pressure,

%otherwise throw an error.

if(isnan(Psat)==1)

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

if size(y1)~=1

disp('y1 dimension incorrect')

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

if y1 <= 0 || y1>1

disp('No y1 detected');

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

if size(T) ~= 1

disp('T dimension incorrect')

x = NaN;

y2 = NaN;

y3 = NaN;

y4 = NaN;

V = NaN;

P = NaN;

L = NaN;

return

end

F = sum(zF);

z = zF/F;

if sum(z) ~= 1

disp('mole fraction does not add up')

return

end

%% Initialization

% x(1) = L

% x(2) = y2

% x(3) = y3

% x(4) = y4

% x(5) = P

f1 = @(x) zF(1)-y1\*(F-x(1))-y1\*x(5)\*x(1)/P1;% f1 = z1\*F - y1\*(F-L) - y1\*P\*L/P1

f2 = @(x) zF(2)-x(2)\*(F-x(1))-x(2)\*x(5)\*x(1)/P2;% f2 = z2\*F - y2\*(F-L) - y2\*P\*L/P2

f3 = @(x) zF(3)-x(3)\*(F-x(1))-x(3)\*x(5)\*x(1)/P3;% f3 = z3\*F - y3\*(F-L) - y3\*P\*L/P3

f4 = @(x) zF(4)-x(4)\*(F-x(1))-x(4)\*x(5)\*x(1)/P4; % f4 = z4\*F - y4\*(F-L) - y4\*P\*L/P4

f5 = @(x) y1+x(2)+x(3)+x(4)-1; % f5 = y1 + y2 + y3 + y4 - 1

%residual

r = @(x) [f1(x); f2(x); f3(x); f4(x); f5(x)];

%Jacobian

jac = @(x) [y1-y1\*x(5)/P1 0 0 0 -y1\*x(1)/P1;

x(2)-x(2)\*x(5)/P2 x(5)-x(1)\*x(5)/P2 0 0 -x(2)\*x(1)/P2;

x(3)-x(3)\*x(5)/P3 0 x(5)-x(1)\*x(5)/P3 0 -x(3)\*x(1)/P3;

x(4)-x(4)\*x(5)/P4 0 0 x(5)-x(1)\*x(5)/P4 -x(4)\*x(1)/P4;

0 1 1 1 0];

%Initial guess

X0 = [F/2 z(2) z(3) z(4) z(1)\*P1/y1]';

%Newton's method to improve initial guess

for k = 1:3 % 3 iteratives step recommended by the project guideline. Anything more than that will lead to bad result

r0 = r(X0);

jac0 = jac(X0);

del = -jac0\r0;

X = X0+del;

X0 = X;

end

%% Steepest Descent

% Setting up g(x) the square of residual to be used in steepest descent

g = @(x) f1(x)^2+f2(x)^2+f3(x)^2+f4(x)^2+f5(x)^2;

%gradient

delG=@(x) [2\*f1(x)\*(y1-y1\*x(5)/P1)+ 2\*f2(x)\*(x(2)-x(2)\*x(5)/P2)+2\*f3(x)\*(x(3)-x(3)\*x(5)/P3)+2\*f4(x)\*(x(4)-x(4)\*x(5)/P4);

2\*f2(x)\*(-(F-x(1))-x(1)\*x(5)/P2)+2\*f5(x);

2\*f3(x)\*(-(F-x(1))-x(1)\*x(5)/P3)+2\*f5(x);

2\*f4(x)\*(-(F-x(1))-x(1)\*x(5)/P4)+2\*f5(x);

2\*f1(x)\*(-y1\*x(1)/P1)+2\*f2(x)\*(-x(2)\*x(1)/P2)+2\*f3(x)\*(-x(3)\*x(1)/P3)+2\*f4(x)\*(-x(4)\*x(1)/P4)];

% Descending %the following algorithm is based on armijo's rule on http://www.mit.edu/~dimitrib/PTseng/516/hmwk1\_add.txt

for k = 1:20000

nf = 0; %nf=number of function evaluation

for i = 1:100 % for loop to find a

a = 0.5^nf;

d = -delG(X0);

objfunc = g(X0);

newobj = g(X0+a\*d);

if newobj > objfunc % algorithm is not descending, decrease a and check again

nf = nf + 1;

else

Xnext = X0 + a\*d; %newobj is descending, break the sub for loop and goes back to main for loop

break

end

end

if a\*norm(d)/norm(Xnext) < 1e-9 % once convergence, method complete

break

else

X0 = Xnext; %update Xnext to be the next X0

end

if i == 100 || k == 20000

disp('Max iteration reached')

Go = 0;

break

end

end

% Output value updated

L = Xnext(1)

y2 = Xnext(2)

y3 = Xnext(3)

y4 = Xnext(4)

P = Xnext(5)

%% Backward calculationn

% Determine x through yi = Kxi = Pi\*xi/P

% This rearranges to xi = P\*yi/Pi

x=[y1\*P/P1;y2\*P/P2;y3\*P/P3;y4\*P/P4]

V=F - L

% Checks to make sure that the calculated pressure and temperature

% combinations are valid

%Very difficult to calculate bubble point temperature

Pbubble = (z(1)\*P1+z(2)\*P2+z(3)\*P3+z(4)\*P4);

Pdew = (z(1)/P1+z(2)/P2+z(3)/P3+z(4)/P4)^-1;

if P > Pbubble

disp('Pressure above bubble-point')

return

end

if P < Pdew

disp('Pressure below dew-point')

return

end

if V/F<=0 || V/F>=1

disp('Outside of bubble-dew condition')

return

end

% When Go Marker is turned off

if Go == 0

x = NaN;

y2 = NaN;

y3 = NaN;

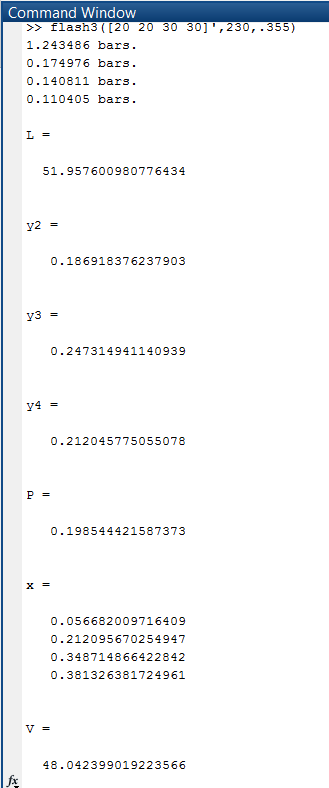
y4 = NaN;

L = NaN;

V = NaN;

P=NaN;

end



function [x y V L P] = MHflash(zF, T ,P, L, y1)

%% Input check

%zF check

m=iscolumn(zF);

if m ~= 1

disp('zF must be a column vector');

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

n=size(zF);

if n ~= 4

disp('4 input stream required');

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

F=sum(zF);

z=zF/F;

%To make sure the mole fraction add up to 1 to prevent negative stream

if sum(z)~=1

disp('invalid stream')

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

%Temperature check

Psat = [Psat1(T) Psat2(T) Psat3(T) Psat4(T)]';

if(sum(isnan(Psat))>0)

disp('ERROR: INVALID TEMPERATURE');

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

if size(T)~=1

disp('invalid temperature dimension')

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

%Valid P --> run flash1.

if(P>0)

%Pass arguments to flash1 subroutine.

[x y V L] = flash1(zF, T, P);

if(sum(isnan(cat(1,x,y,V,L)))<1) %if any input invalid, isnan(invalid input)=1, error

%%Pacakage into an array cat function % this step check error

return

end

end

%valid L --> run flash2.

if(L>0 && L<sum(zF)) %If L is in the domain of (0, sum(zF))

%Pass arguments to flash2 subroutine.

[x y V P] = flash2(zF, T, L);

if(sum(isnan(cat(1,x,y,V,P)))<1) %exit program if no error

return

end

end

%valid y1-->run flash3.

if(y1>0 && y1<1)

%Pass arguments to flash3 subroutine.

[x y2 y3 y4 V L P] = flash3(zF, T, y1);

if(sum(isnan(cat(1,x,y2,y3,y4,V,L,P)))<1)

%exit program if no error

return

end

%If all three flash fails

else

disp('Your conditions are beyond the range of this program, please update');

x=NaN;

y=NaN;

V=NaN;

L=NaN;

P=NaN;

return

end

end

%Implicit Euler method to solve an unsteady state diffusion problem with no

%chemical reaction

%The algorithm used here is derived from the theory lay out

%https://opencommons.uconn.edu/cgi/viewcontent.cgi?referer=https%3A%2F%2Fwww.google.com%2F&httpsredir=1&article=1118&context=srhonors\_theses

%The program will run 1500 iterations

function MHImplicitEuler

%% Initialization

D=3.4e-5; % m^2/2 diffusivity of methane in air at 400K

di=0; %initial distance m

df=1; %final distance m

ti=0; %Time initial from 0 to 10 seconds specifed in the problem

tf=10;

hd=(df-di)/1500;% distance step size

ht=(tf-ti)/1500; %Stepsize for time

x=linspace(di,df,1501);

t=linspace(ti,tf,1501); %creating equally spacing vector

dL=hd; %delx and delt, the smaller the step size, the more accurate the solution is but at a price of higher computational cost

dt=ht;

%% setting up the and backward differencing

% a banded matrix was set up in the source guide on page 9, this matrix is

% used to solve for the implicit solution of this parabolic PDE

a=D\*dt/(dL^2);

A=zeros(1500,1500); %preallocating space

for i = 1:1500

A(i,i) = 1 + 2\*a; %diagonal matrix

end

for i = 1:1499

A(i,i+1)=-a; %banded matrix

A(i+1,i)=-a;

end

%% BC dC/dx@x=L = 0 C(0,t)=18 C(x,0)=0

BC = zeros(1,1500)';%preallocating space

BC(1,1)=18; %concentration of methane if there is a leak

F=zeros(1500, 1501);

for i= 1:1500

F(:,i+1)= A\(BC + F(:,i));

end

for i = 1:1500

if F(i,end) <= 0.8

d = i; %F is keep getting added for 1500 times, so the L the program will give is the sum distance calculated by all iterations, to find the average, divide the sum by 1500

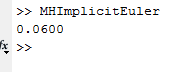
b=d/1500; %divide by number of iteration

fprintf ('%.4f\n', b);

break

end

end



3A. The wall should be closer than 6.0 cm (as shown from previous page) from the suspected leak site to fulfill the design requirement that an alarm alert go off at most 10 seconds after a leak has begun.

3B. This equation is a parabolic partial differential equation that is in the form of the heat equation.

Other PDE we have studied in this class have been elliptic, and elliptic can be solved using method of steepest or gradient descent. Another interesting thing is that this specific problem has a mix boundary conditions of Neumann and Dirichlet.

3C. 3 boundary conditions are needed to solve this problem. They are C(0,t)=18 mol/m^3, C(x,0)=0 and evaluated at L)= 0 since methane does not diffuse through wall.

3D. Diffusion is relatively slow method as seen through the interpretation of the results from this simulation: a 6.0 cm sensor is required to detect dangerous levels of methane buildup after a relative 10 seconds. I can walk at least 10 m per 10s, I move almost 10 times the speed of methane. However, methane is a very small molecule and diffuses relatively fast compared to other gases. So overall, diffusion is a relatively slow process.