Scilab Textbook Companion for Principles And Modern Applications Of Mass Transfer Operations by J. Benitez[1]

Created by
Ashwani Kumar
B-tech Part III
Chemical Engineering
IIT-BHU
College Teacher
NA
Cross-Checked by

August 10, 2013

[1] Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

Book Description

Title: Principles And Modern Applications Of Mass Transfer Operations

Author: J. Benitez

Publisher: John Wiley & Sons Inc., New Jersey

Edition: 2

Year: 2009

ISBN: 978-0-470-18178-2

Scilab numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

AP Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes

Chapter 1

Fundamentals of Mass transfer

Scilab code Exa 1.1 MOLECULAR MASS TRANSFER

```
clear;
clc;

// Illustration 1.1
// Page: 6

printf('Illustration 1.1 - Page: 6\n\n');

//*****Data*****
T = 300; // [K]
P = 500; // [kPa]
R = 8.314; // [J/mole.K]
//*****//
printf('Illustration 1.1 (a) - Page: 6\n\n');
// Solution (a)
// Using equation 1.7
C = P/(R*T); // [Total molar concentration, kmole/cubic m]
printf("Total molar concentration in the gas feed is %f kmole/cubic m\n\n",C);

printf('Illustration 1.1 (b) - Page: 7\n\n');
// Solution (b)

// Mixture of gases
// Components a-CH4 , b-C2H6 , c-nC3H8 , d-nC4H10
// Basis: 100 kmole of gas mixture
n_a = 88; // [kmole]
n_b = 4; // [kmole]
n_c = 5; // [kmole]
n_d = 3; // [kmole]
M_a = 16.04; // [gram/mole]
M_b = 30.07; // [gram/mole]
M_c = 44.09; // [gram/mole]
M_d = 58.12; // [gram/mole]
m_a = n_a*M_a; // [kg]
m_b = n_b*M_b; // [kg]
m_c = n_c*M_c; // [kg]
m_d = n_d*M_d; // [kg]
n_total = n_a+n_b+n_c+n_d; // [kmole]
m_total = m_a+m_b+m_c+m_d; // [kg]
M_avg = m_total/n_total; // [kg/kmole]
row = C*M_avg; // [mass density, kg/cubic m]
printf("Average molecular weight of gas feed is %f kg/kmole\n",M_avg);
printf("Density of gas feed is %f kg/cubic m\n\n",row);

printf('Illustration 1.1 (c) - Page: 7\n\n');
// Solution (c)

// Mass fraction of each component
x_a = m_a/m_total;
x_b = m_b/m_total;
x_c = m_c/m_total;
x_d = m_d/m_total;
printf("Mass fraction of CH4, C2H6, nC3H8, nC4H10 are %f, %f, %f, %f respectively",x_a,x_b,x_c,x_d);
```

Scilab code Exa 1.2 Concentration of a Potassium Nitrate Wash Solution

```
clear;
clc;

// Illustration 1.2
// Page: 7

printf('Illustration 1.2 - Page: 7\n\n');

//*****Data*****
// Component  a-KNO3   b-H20
T = 293; // [K]
s_eqm = 24; // [percent by weight, %]
row = 1162; // [density of saturated solution, kg/cubic m]
//*****//

printf('Illustration 1.2 (a)- Page: 7\n\n');
// Solution (a)

// Basis: 100 kg of fresh wash solution
m_a = (s_eqm/100)*100; // [kg]
m_b = 100 - m_a; // [kg]
M_a = 101.10; // [gram/mole]
M_b = 18.02; // [gram.mole]
// Therefore moles of component 'a' and 'b' are
n_a = m_a/M_a; // [kmole]
n_b = m_b/M_b; // [kmole]

m_total = 100; // [basis, kg]
n_total = n_a+n_b; // [kmole]
// Average molecular weight
M_avg = m_total/n_total; // [kg/kmole]
```

```
// Total molar density of fresh solution
C = row/M_avg; // [kmole/cubic m]
printf("Total molar density of fresh solution is %f kmole/cubic m\n\n",C);

printf('Illustration 1.2 (b)- Page: 8\n\n');
// Solution (b)

// mole fractions of components 'a' and 'b'
x_a = n_a/n_total;
x_b = n_b/n_total;
printf("Mole fraction of KNO3 and H2O is %f %f",x_a,x_b);
```

Scilab code Exa 1.3 Material Balances on a Bio Artificial Kidney

```
clear;
clc;

// Illustration 1.3
// Page: 9

printf('Illustration 1.3 - Page:9 \n\n');

//*****Data*****//
// Blood contains two parts a-blood cells  b-plasma
f_a = 45; // [percent of blood cells by volume]
f_b = 55; // [percent of plasma by volume]
r = 1200; // [Rate of blood which is pumped through artificial kidney, mL/minute]
m_urine = 1540; // [mass of urine collected, g]
x_u = 1.3; // [urea concentration, percent by weight]
// Data for sample of blood plasma
c_urea = 155.3; // [mg/dL]
```

```
d = 1.0245; // [specfic gravity of plasma]
//*****//

printf('Illustration 1.3 (a) - Page:9 \n\n');
// Solution (a)

// Basis: 4 hours
// Assuming that the rate of formation and decomposition of urea during the procedure is negligible and that no urea is removed by the patient s kidneys
// Therefore urea in  clean  blood = urea in  dirty  blood - urea in urine

m_u = m_urine*(x_u/100); // [mass of urea in urine, g]
// total volume of plasma that flows through the artificial kidney in 4 hours
V_b = r*60*(f_b/100)*(1/100)*4; // [dL]
// urea in dirty blood from given plasma concentration
m_ud = c_urea*(1/1000)*V_b; // [g]
// urea removal efficiency
n = (m_u/m_ud)*100;
printf("Urea removal efficiency is %f\n\n",n);

printf('Illustration 1.3 (b) - Page:10 \n\n');
// Solution (b)

m_uc = m_ud-m_u; // [mass of urea on clean blood, g]
m_p = d*100*V_b; // [Mass of plasma entering, g]
m_rem = m_p-m_urine; // [Mass of plasma remaining, g]
V_brem = m_rem/(d*100); // [Volume of plasma remaining, dL]
c_y = (m_uc*1000)/V_brem; // [urea concentration in remaining plasma, mg/dL]
printf("urea concentration in the plasma of the
    cleansed blood is %f mg/dL",c_y);
```

Scilab code Exa 1.6 Calculation of Diffusivity by the Wilke Lee Equation with Known Values of the Lennard Jones Parameters

```
clear;
clc;

// Illustration 1.6
// Page: 21

printf('Illustration 1.6 - Page:21 \n\n');
// Solution

//*****Data*****//
//  a-CS2   b-air
T = 273; // [K]
P = 1; // [bar]
// 1 bar = 10^5 Pa
// Values of the Lennard-Jones parameters (sigma and E/K) are obtained from Appendix B:
sigma_a = 4.483; // [1st Lennard-Jones parameter, Angstrom]
sigma_b = 3.620; // [Angstrom]
d_a = 467; // [d = E/K 2nd Lennard-Jones parameter, K]
d_b = 97; // [K]
M_a = 76; // [gram/mole]
M_b = 29; // [gram/mole]
sigma_ab = (sigma_a+sigma_b)/2; // [Angstrom]
d_ab = sqrt(d_a*d_b); // [K]
M_ab = 2/((1/M_a)+(1/M_b)); // [gram/mole]

T_star = T/d_ab;
a = 1.06036; b = 0.15610; c = 0.19300; d = 0.47635;
```

```
e = 1.03587; f = 1.52996; g = 1.76474; h = 3.89411;
ohm = ((a/T_star^b)+(c/exp(d*T_star))+(e/exp(f*T_star))+(g/exp(h*T_star)));

// Substituting these values into the Wilke-Lee equation yields (equation 1.49)
D_ab = ((10^-3*(3.03-(.98/sqrt(M_ab)))*T^1.5)/(P*(sqrt(M_ab))*(sigma_ab^2)*ohm)); // [square cm/s]
printf("The diffusivity of carbon disulfide vapor in air at 273 K and 1 bar is %e square cm/s\n",D_ab);

// The experimental value of D_ab obtained from Appendix A:
D_abexp = (.894/(P*10^5))*10^4; // [square cm/s]
percent_error = ((D_ab-D_abexp)/D_abexp)*100; // [%]
printf("The percent error of the estimate, compared to the experimental value is %f ",percent_error);
```

Scilab code Exa 1.7 Calculation of Diffusivity by the Wilke Lee Equation with Estimated Values of the Lennard Jones Parameters

```
clear;
clc;

// Illustration 1.7
// Page: 22

printf('Illustration 1.7 - Page:22 \n\n');
// Solution

//*****Data*****//
//    A-C3H5Cl     B-air
T = 298; // [K]
```

```
P = 1; // [bar]
//*****//

// Values of the Lennard-Jones parameters for allyl chloride must be estimated from equations (1.46) and (1.47).
// From Table 1.2
V_bA = 3*14.8+5*3.7+24.6; // [cubic cm/mole]
// From equation 1.46
sigma_A = 1.18*(V_bA)^(1/3); // [1st Lennard-Jones parameter, Angstrom]
// Normal boiling-point temperature for allyl chloride is Tb = 318.3 K
// From equation 1.47, E/K = 1.15*Tb
T_b = 318.3; // [K]
d_A = 1.15*T_b; // [2nd Lennard-Jones parameter for C3H5Cl  E/K, K]
M_A = 76.5; // [gram/mole]

// Lennard-Jones parameters for air
sigma_B = 3.62; // [Angstrom]
d_B = 97; // [2nd Lennard-Jones parameter for air E/K, K]

M_B = 29; // [gram/mole]

sigma_AB = (sigma_A+sigma_B)/2; // [Angstrom]
d_AB = sqrt(d_A*d_B); // [K]
M_AB = 2/((1/M_A)+(1/M_B)); // [gram/mole]

T_star = T/d_AB;
a = 1.06036; b = 0.15610; c = 0.19300; d = 0.47635; e = 1.03587; f = 1.52996; g = 1.76474; h = 3.89411;
ohm = ((a/T_star^b)+(c/exp(d*T_star))+(e/exp(f*T_star))+(g/exp(h*T_star)));

// Substituting these values into the Wilke-Lee
```

```
equation yields (equation 1.49)
D_AB = ((10^-3*(3.03-(.98/sqrt(M_AB)))*T^1.5)/(P*(sqrt(M_AB))*(sigma_AB^2)*ohm)); // [square cm/s]
printf("The diffusivity of allyl chloride in air at 298 K and 1 bar is %e square cm/s\n",D_AB);

// The experimental value of D_AB reported by Lugg (1968) is 0.098 square cm/s
D_ABexp = .098; // [square cm/s]
percent_error = ((D_AB-D_ABexp)/D_ABexp)*100; // [%]
printf("The percent error of the estimate , compared to the experimental value is %f ",percent_error);
```

Scilab code Exa 1.8 Calculation of Liquid Diffusivity in Aqueous Solution

```
clear;
clc;

// Illustration 1.8
// Page: 26

printf('Illustration 1.8 - Page:26 \n\n');
// Solution

//*****Data*****//
// solute A-C2H60   solvent B-water
T = 288; // [K]
//*****//
// Critical volume of solute
V_c = 167.1; // [cubic cm/mole]
// Calculating molar volume using equation 1.48
V_ba = 0.285*(V_c)^1.048; // [cubic cm/mole]
u_b = 1.153; // [Viscosity of liquid water at 288 K, cP]
M_solvent = 18; // [gram/mole]
```

```
phi_b = 2.26; // [association factor of solvent B]

printf('Illustration 1.8 (a) - Page:26 \n\n');
// Solution (a)

// Using the Wilke-Chang correlation , equation 1.52
D_abo1 = (7.4*10^-8)*(sqrt(phi_b*M_solvent))*T/(u_b*(V_ba)^.6); // [diffusivity of solute A in very dilute solution in solvent B, square cm/s]
printf("Diffusivity of C2H60 in a dilute solution in water at 288 K is %e square cm/s\n",D_abo1);
// The experimental value of D_abo reported in Appendix A is 1.0 x 10^-5 square cm/s
D_aboexp = 1*10^-5; // [square cm/s]
percent_error1 = ((D_abo1-D_aboexp)/D_aboexp)*100; // [%]
printf("The percent error of the estimate , compared to the experimental value is %f\n\n ",percent_error1);

printf('Illustration 1.8 (b) - Page:27 \n\n');
// Solution (b)

// Using the Hayduk and Minhas correlation for aqueous solutions equation 1.53
E = (9.58/V_ba)-1.12;
D_abo2 = (1.25*10^-8)*(((V_ba)^-.19)-0.292)*(T^1.52)*(u_b^E); // [square cm/s]
printf("Diffusivity of C2H60 in a dilute solution in water at 288 K is %e square cm/s\n",D_abo2);
percent_error2 = ((D_abo2-D_aboexp)/D_aboexp)*100; // [%]
printf("The percent error of the estimate , compared to the experimental value is %f ",percent_error2);
```

Scilab code Exa 1.9 Calculation of Liquid Diffusivity in Dilute Nonaqueous Solution

```
clear;
clc;

// Illustration 1.9
// Page: 27

printf('Illustration 1.9 - Page:27 \n\n');
// Solution

//*****Data*****//
//    A-acetic acid(solute)      B-acetone(solvent)
T = 313; // [K]
// The following data are available (Reid, et al., 1987):

// Data for acetic acid
T_bA = 390.4; // [K]
T_cA = 594.8; // [K]
P_cA = 57.9; // [bar]
V_cA = 171; // [cubic cm/mole]
M_A = 60; // [gram/mole]

// Data for acetone
T_bB = 329.2; // [K]
T_cB = 508; // [K]
P_cB = 47; // [bar]
V_cB = 209; // [cubic cm/mole]
u_bB = 0.264; // [cP]
M_B = 58; // [gram/mole]
phi = 1;

```

```
printf('Illustration 1.9 (a) - Page:27 \n\n');
// Solution (a)
// Using equation 1.48
V_bA = 0.285*(V_cA)^1.048; // [cubic cm/mole]

// Using the Wilke-Chang correlation , equation 1.52
D_abo1 = (7.4*10^-8)*(sqrt(phi*M_B))*T/(u_bB*(V_bA)^.6);
printf("Diffusivity of acetic acid in a dilute solution in acetone  at 313 K using the Wilke-Chang correlation is %e square cm/s\n",D_abo1);
// From Appendix A, the experimental value is 4.04*10^-5 square cm/s
D_aboexp = 4.04*10^-5; // [square cm/s]
percent_error1 = ((D_abo1-D_aboexp)/D_aboexp)*100; // [%]
printf("The percent error of the estimate, compared to the experimental value is %f\n\n ",percent_error1);

printf('Illustration 1.9 (b) - Page:28 \n\n');
// Solution (b)

// Using the Hayduk and Minhas correlation for nonaqueous solutions

V_bA = V_bA*2; // [cubic cm/mole]
V_bB = 0.285*(V_cB)^1.048; // [cubic cm/mole]

// For acetic acid (A)
T_brA = T_bA/T_cA; // [K]
// Using equation 1.55
alpha_cA =  0.9076*(1+((T_brA)*log(P_cA/1.013))/(1-T_brA));
sigma_cA = (P_cA^(2/3))*(T_cA^(1/3))*(0.132*alpha_cA-0.278)*(1-T_brA)^(11/9); // [dyn/cm]

// For acetone (B)
```

```
T_brB = T_bB/T_cB; // [K]
// Using equation 1.55
alpha_cB =  0.9076*(1+((T_brB*log(P_cB/1.013))/(1-
    T_brB)));
sigma_cB = (P_cB^(2/3))*(T_cB^(1/3))*(0.132*alpha_cB
    -0.278)*(1-T_brB)^(11/9); // [dyn/cm]

// Substituting in equation 1.54
D_abo2 = (1.55*10^-8)*(V_bB^0.27)*(T^1.29)*(sigma_cB
    ^0.125)/((V_bA^0.42)*(u_bB^0.92)*(sigma_cA^0.105)
    );

printf("Diffusivity of acetic acid in a dilute
    solution in acetone  at 313 K using the Hayduk
    and Minhas correlation is %e square cm/s\n",
    D_abo2);

percent_error2 = ((D_abo2-D_aboexp)/D_aboexp)*100;
    // [%]
printf("The percent error of the estimate, compared
    to the experimental value is %f\n\n ",
    percent_error2);
```

Scilab code Exa 1.10 Diffusion Coefficients in the System Acetone Benzene

```
clear;
clc;

// Illustration 1.10
// Page: 30

printf('Illustration 1.10 - Page:30 \n\n');
// Solution

```

```
//*****Data*****//
// acetone-1   benzene-2
T = 298; // [K]
x_1 = 0.7808;
x_2 = 1-x_1;
// The infinite dilution diffusivities are
D_12o = 2.75*10^-9; // [square m/s]
D_21o = 4.15*10^-9; // [square m/s]
// From the NRTL equation, for this system at the given temperature and concentration the thermodynamic correction factor r = 0.871.
r = 0.871;
D_12exp = 3.35*10^-9; // [square m/s]
//*****//

// Using equation 1.56
D_12 = (D_12o^x_2)*(D_21o^x_1);
D_12 = D_12*r;
printf("The theoritical value of Fick diffusivity is %e square m/s",D_12);
// The predicted value of the Fick diffusivity is in excellent agreement with the experimental result.
```

Scilab code Exa 1.11 Calculation of Effective Diffusivity in a Multicomponent Gas Mixture

```
clear;
clc;

// Illustration 1.11
// Page: 33

printf('Illustration 1.11 - Page:33 \n\n');
// Solution
```

```

//*****Data*****//
// ammonia-1 nitrogen-2 hydrogen-3
T = 300; // [K]
P = 1; // [bar]
y_1 = .40;
y_2 = .20;
y_3 = .40;
//*****//

// Lennard-Jones parameter for ammonia
sigma_1 = 2.9; // [Angstrom]
d_1 = 558.3; // [E/K, K]
M_1 = 17; // [gram/mole]

// Lennard-Jones parameter for nitrogen
sigma_2 = 3.798; // [Angstrom]
d_2 = 71.4; // [E/K, K]
M_2 = 28; // [gram/mole]

// Lennard-Jones parameter for hydrogen
sigma_3 = 2.827; // [Angstrom]
d_3 = 59.7; // [E/K, K]
M_3 = 2; // [gram/mole]

// Binary diffusivitiy of ammonia in nitrogen (D_12)

sigma_12 = (sigma_1+sigma_2)/2; // [Angstrom]
d_12 = sqrt(d_1*d_2); // [K]
M_12 = 2/((1/M_1)+(1/M_2)); // [gram/mole]

T_star12 = T/d_12;
a = 1.06036; b = 0.15610; c = 0.19300; d = 0.47635; e = 1.03587; f = 1.52996; g = 1.76474; h = 3.89411;
ohm12 = ((a/T_star12^b)+(c/exp(d*T_star12))+(e/exp(f*T_star12))+(g/exp(h*T_star12)));

```

```
// Substituting these values into the Wilke-Lee equation yields (equation 1.49)
D_12 = ((10^-3*(3.03-(.98/sqrt(M_12)))*T^1.5)/(P*(sqrt(M_12))*(sigma_12^2)*ohm12)); // [square cm/s]
printf("The diffusivitiy of ammonia in nitrogen %e square cm/s\n",D_12);

// Binary diffusivitiy of ammonia in hydrogen (D_13)

sigma_13 = (sigma_1+sigma_3)/2; // [Angstrom]
d_13 = sqrt(d_1*d_3); // [K]
M_13 = 2/((1/M_1)+(1/M_3)); // [gram/mole]

T_star13 = T/d_13;
a = 1.06036; b = 0.15610; c = 0.19300; d = 0.47635; e = 1.03587; f = 1.52996; g = 1.76474; h = 3.89411;
ohm13 = ((a/T_star13^b)+(c/exp(d*T_star13))+(e/exp(f*T_star13))+(g/exp(h*T_star13)));

// Substituting these values into the Wilke-Lee equation yields (equation 1.49)
D_13 = ((10^-3*(3.03-(.98/sqrt(M_13)))*T^1.5)/(P*(sqrt(M_13))*(sigma_13^2)*ohm13)); // [square cm/s]
printf("The diffusivitiy of ammonia in hydrogen %e square cm/s\n",D_13);

// Figure 1.5 shows the flux of ammonia (N_1) toward the catalyst surface, where
// it is consumed by chemical reaction, and the fluxes of nitrogen (N_2) and hydrogen (N_3)
// produced by the reaction migrating away from the same surface.

// Therefore N_1 = N_2+N_3
// From equation 1.59
// N_2 = -(0.5)*N_1     and     N_3 = -(1.5)*N_1

// Substituting in equation (1.58) we obtain
D_1eff = (1+y_1)/((y_2+0.5*y_1)/D_12 + (y_3+1.5*y_1)/D_13); // [square cm/s]
printf("The effective diffusivity of ammonia in the gaseous mixture is %e square cm/s",D_1eff);
```

Scilab code Exa 1.12 Calculation of Effective Diffusivity in a Multicomponent Stagnant Gas Mixture

```
clear;
clc;

// Illustration 1.12
// Page: 34

printf('Illustration 1.12 - Page:34 \n\n');
// Solution

//*****Data*****//
//  ammonia-1  nitrogen-2   hydrogen-3
T = 300; // [K]
P = 1; // [bar]
y_1 = .40;
y_2 = .20;
y_3 = .40;
//*****//

// The binary diffusivities are the same as for Example 1.11.
D_12 = 0.237; // [square cm/s]
D_13 = 0.728; // [square cm/s]

// mole fractions of nitrogen (2) and hydrogen (3)
```

```
   on an ammonia (1)-free base from equation (1.61)
y_21 = y_2/(1-y_1);
y_31 = y_3/(1-y_1);
// Substituting in equation (1.60) gives us
D_1eff = 1/((y_21/D_12)+(y_31/D_13));
printf("The effective diffusivity of ammonia in the
   gaseous mixture is  %e square cm/s",D_1eff);
```

Scilab code Exa 1.13 Calculation of Effective Diffusivity of a Dilute Solute in a Homogeneous Mixture of Solvents

```
clear;
clc;

// Illustration 1.13
// Page: 36

printf('Illustration 1.13 - Page:36 \n\n');
// Solution

//*****Data*****
// acetic acid-1   water-2   ethyl alcohol-3
T = 298; // [K]
// The data required data for water at 298 K
u_2 = 0.894; // [cP]
V_c1 = 171; // [cubic cm/mole]
// From equation 1.48
V_b1 = 62.4; // [cubic cm/mole]
// Substituting in equation (1.53)
// the infinite dilution diffusion coefficient of
   acetic acid in water at 298 K
E = (9.58/V_b1)-1.12;
D_abo12 = (1.25*10^-8)*(((V_b1)^-.19)-0.292)*(T
   ^1.52)*(u_2^E); // [square cm/s]

```

```

// Data for acetic acid
T_b1 = 390.4; // [K]
T_c1 = 594.8; // [K]
P_c1 = 57.9; // [bar]
V_c1 = 171; // [cubic cm/mole]
M_1 = 60; // [gram/mole]

// Data for ethanol
T_b3 = 351.4; // [K]
T_c3 = 513.9; // [K]
P_c3 = 61.4; // [bar]
V_c3 = 167; // [cubic cm/mole]
M_3 = 46; // [gram/mole]
u_3 = 1.043; // [cP]

// Using the Hayduk and Minhas correlation for nonaqueous solutions

// According to restriction 3 mentioned above, the molar volume of the acetic acid to be used in equation (1.54) should be
V_b1 = V_b1*2; // [cubic cm/mole]
// The molar volume of ethanol is calculated from equation (1.48)
V_b3 = 60.9; // [cubic cm/mole]


// For acetic acid (1)
T_br1 = T_b1/T_c1; // [K]
// Using equation 1.55
alpha_c1 =  0.9076*(1+((T_br1)*log(P_c1/1.013))/(1-T_br1));
sigma_c1 = (P_c1^(2/3))*(T_c1^(1/3))*(0.132*alpha_c1-0.278)*(1-T_br1)^(11/9); // [dyn/cm]

// For ethanol (3)
T_br3 = T_b3/T_c3; // [K]
```

```
// Using equation 1.55
alpha_c3 =  0.9076*(1+((T_br3*log(P_c3/1.013))/(1-T_br3)));
sigma_c3 = (P_c3^(2/3))*(T_c3^(1/3))*(0.132*alpha_c3-0.278)*(1-T_br3)^(11/9); // [dyn/cm]

// Substituting in equation 1.54
D_abo13 = (1.55*10^-8)*(V_b3^0.27)*(T^1.29)*(sigma_c3^0.125)/((V_b1^0.42)*(u_3^0.92)*(sigma_c1^0.105));

// The viscosity of a 40 wt% aqueous ethanol solution at 298 K is u_mix = 2.35 cP
u_mix = 2.35; // [cP]
// The solution composition must be changed from mass to molar fractions following a procedure similar to that illustrated in Example 1.2
// Accordingly, a 40 wt% aqueous ethanol solution converts to 20.7 mol%.
// Therefore mole fraction of ethanol (x_3) and water (x_2)

x_3 = 0.207;
x_2 = 1-x_3;
// Using equation 1.62
D_1eff = ((x_2*D_abo12*(u_2^0.8))+(x_3*D_abo13*(u_3^0.8)))/(u_mix^0.8);
printf("The diffusion coefficient of acetic acid at very low concentrations diffusing into a mixed solvent containing 40.0 wt percent ethyl alcohol in water at a temperature of 298 K is %e square cm/s\n\n",D_1eff);

// The experimental value reported by Perkins and Geankoplis (1969) is
D_1exp = 5.71*10^-6; // [square cm/s]
percent_error = ((D_1eff-D_1exp)/D_1exp)*100; // [%]
printf("The error of the estimate is %f\n",
   percent_error);
```

Scilab code Exa 1.14 Steady State Equimolar Counterdiffusion

```
clear;
clc;

// Illustration 1.14
// Page: 39

printf('Illustration 1.14 - Page:39 \n\n');
// Solution

//*****Data*****
//  Binary gaseous mixture of components A and B
P = 1; // [bar]
T = 300; // [K]
R = 8.314; // [cubic m.Pa/mole.K]
delta = 1; // [mm]
y_A1 = 0.7;
y_A2 = 0.2;
D_AB = 0.1; // [square cm/s]
//*****//

// Using equation 1.72
N_A = (D_AB*10^-4)*(P*10^5)*(y_A1-y_A2)/(R*T*delta
   *10^-3);
printf("The molar flux of component A is %f mole/
   square m.s",N_A);
```

Scilab code Exa 1.15 Steady State Diffusion of A Through Stagnant B

```
clear;
```

```
clc;

// Illustration 1.15
// Page: 43

printf('Illustration 1.15 - Page:43 \n\n');
// Solution

//*****Data*****//
// Diffusion of A through stagnant B
P_total = 1.0; // [bar]
P_B1 = 0.8; // [bar]
P_B2 = 0.3; // [bar]
//*****//

// Using equation 1.83
P_BM = (P_B2-P_B1)/(log(P_B2/P_B1)); // [bar]
// using the result of Example 1.14, we have
N_A = 0.2; // [mole/square m.s]
N_A = N_A*P_total/P_BM; // [moloe/square m.s]
printf("The molar flux of component A is %f mole/square m.s",N_A);
```

Scilab code Exa 1.16 Production of Nickel Carbonyl Steady State One Dimensional Binary Flux Calculation

```
clear;
clc;

// Illustration 1.16
// Page: 44

printf('Illustration 1.16 - Page:44 \n\n');
// Solution

```

```
//*****Data*****//
// Nickel Carbonyl-A     carbon monoxide-B
T = 323; // [K]
P = 1; // [atm]
R = 8.314; // [cubic m.Pa/mole.K]
y_A1 = 1.0;
y_A2 = 0.5;
delta = 0.625; // [mm]
D_AB = 20; // [square mm/s]
//*****//

// The stoichiometry of the reaction determines the relation between the fluxes: from equation (1-59), N_B = - 4N_A and N_A + N_B = -3NA
// Molar flux fraction si_A = N_A/(N_A+N_B) = N_A/(-3*N_A) = -1/3
si_A = -1/3;
// Using equation 1.78
N_A = si_A*(D_AB*10^-6*P*10^5*log((si_A-y_A2)/(si_A-y_A1))/(R*T*delta*10^-3));
printf("The molar flux of component A is %f mole/square m.s",N_A);
```

Scilab code Exa 1.19 Steady State Molecular Diffusion in Liquids

```
clear;
clc;

// Illustration 1.19
// Page: 54

printf('Illustration 1.19 - Page:54 \n\n');
// Solution

//*****Data*****//
```

```
// a-CuS04     b-H2O
T = 273; // [K]
delta = 0.01; // [mm]
sol_ab = 24.3; // [gram/100 gram water]
den_ab = 1140; // [kg/cubic m]
D_ab = 3.6*10^-10; // [square m/s]
den_b = 999.8; // [kg/cubic m]
//*****//

// both fluxes are in the same direction; therefore, they are both positive and relation is N_b = 5 N_a (where N_b and N_a are molar fluxes of component 'a' and 'b')
// From equation (1.76), si_a = 1/6 = 0.167
si_a = 0.167;
// Calculation of mole fraction of component 'a'
// Basis: 100 gram H2O (b)
M_a = 159.63; // [gram/mole]
M_b = 18; // [gram/mole]
M_c =249.71; // [here M_c is molecular mass of hydrated CuSO4, gram/mole]
m_a = 24.3; // [gram]
m_c = m_a*(M_a/M_c); // [here m_c is the mass of CuSO4 in 24.3 gram of crystal, gram]
m_d = m_a-m_c; // [here m_d is mass of hydration of water in the crystal, gram]
m_b_total = 100+m_d; // [total mass of water, gram]

x_a1 = (m_c/M_a)/((m_c/M_a)+(m_b_total/M_b));
x_a2 = 0;

// At point 1, the average molecular weight is
M_1 = x_a1*M_a+(1-x_a1)*M_b; // [gram/mole]
// At point 2, the average molecular weight is
M_2 = x_a2*M_a+(1-x_a2)*M_b
// Molar density at point 1 and 2
row_1 = den_ab/M_1; // [kmole/cubic m]
row_2 = den_b/M_2
```

```
row_avg = (row_1+row_2)/2; // [kmole/cubic m]

// Using equation 1.96

N_a = si_a*D_ab*row_avg*log((si_a-x_a2)/(si_a-x_a1))/(delta*10^-3); // [kmole/square m.s]
rate = N_a*M_c*3600; // [kg/square m of crystal surface area per hour]
printf("the rate at which the crystal dissolves in solution is %f kg/square m of crystal surface area per hour",rate);
```

Scilab code Exa 1.20 Steady State Molecular Diffusion in Porous Solid

```
clear;
clc;

// Illustration 1.20
// Page: 58

printf('Illustration 1.20 - Page:58 \n\n');
// Solution

//*****Data*****//
// A-hydrogen   B-ethane
T = 373; // [K]
P = 10; // [atm]
d = 4000; // [Angstrom]
e = 0.4; // [porosity]
t = 2.5; // [tortuosity]
D_AB = 0.86/P; // [square cm/s]
k = 1.3806*10^-23; // [J/K]
//*****//

// Using data from Appendix B for hydrogen and
```

```
    ethane, and equation (1.45)
sigma_A = 2.827; // [Angstrom]
sigma_B = 4.443; // [Angstrom]
sigma_AB = ((sigma_A+sigma_B)/2)*10^-10; // [m]

lambda = k*T/(sqrt(2)*3.14*(sigma_AB^2)*P
    *1.01325*10^5); // [m]
lambda = lambda*10^10; // [Angstrom]
// From equation 1.101
K_n = lambda/d;
printf("The value of a dimensionless ratio, Knudsen
    number is %f\n\n",K_n);
// If K_n is less than 0.05 then diffusion inside
    the pores occurs only by ordinary molecular
    diffusion and equation 1.100 can be used to
    calculate D_ABeff
D_ABeff = D_AB*e/t;
printf("The effective diffusivity of hydrogen in
    ethane is %f square cm/s",D_ABeff);
```

Scilab code Exa 1.21 Knudsen Diffusion in Porous Solid

```
clear;
clc;

// Illustration 1.21
// Page: 60

printf('Illustration 1.21 - Page:60 \n\n');
// Solution

//*****Data*****//
//   a-oxygen   b-nitrogen
T = 293; // [K]
P = 0.1; // [atm]
```

```
d = 0.1*10^-6; // [m]
e = 0.305; // [porosity]
t = 4.39; // [tortuosity]
k = 1.3806*10^-23; // [J/K]
l = 2*10^-3; // [m]
R = 8.314; // [cubic m.Pa/mole.K]
x_a1 = 0.8;
x_a2 = 0.2;
M_a = 32; // [gram/mole]
M_b = 28; // [gram/mole]
//*****//

// Using data from Appendix B for oxygen and nitrogen, and equation (1.45)
sigma_a = 3.467; // [Angstrom]
sigma_b = 3.798; // [Angstrom]
sigma_AB = ((sigma_a+sigma_b)/2)*10^-10; // [m]

lambda = k*T/(sqrt(2)*3.14*(sigma_AB^2)*P*1.01325*10^5); // [m]
// From equation 1.101
K_n = lambda/d;
printf("The value of a dimensionless ratio, Knudsen number is %f\n\n",K_n);
// If K_n is greater than 0.05 then transport inside the pores is mainly by Knudsen diffusion
// Using equation 1.103
D_Ka = (d/3)*(sqrt(8*R*T)/sqrt(3.14*M_a*10^-3)); // [square m/s]

// Using equation 1.107
D_Kaeff = D_Ka*e/t; // [square m/s]

p_a1 = (x_a1*P)*1.01325*10^5; // [Pa]
p_a2 = (x_a2*P)*1.01325*10^5; // [Pa]

// Using equation 1.108
N_a = D_Kaeff*(p_a1-p_a2)/(R*T*l); // [mole/square m
.s]
// Now using the Graham s law of effusion for Knudsen diffusion
// N_b/N_a = -sqrt(M_a/M_b) ,therefore
N_b = -N_a*sqrt(M_a/M_b); // [mole/square m.s]

printf("The diffusion fluxes of both components oxygen and nitrogen are %e mole/square m.s and %e mole/square m.s respectively\n",N_a,N_b);
```

Scilab code Exa 1.22 Combined Molecular and Knudsen Diffusion in a Porous Solid

```
clear;
clc;

// Illustration 1.22
// Page: 61

printf('Illustration 1.22 - Page:61 \n\n');
// Solution

//*****Data*****//
//   a-oxygen   b-nitrogen
T = 293; // [K]
P = 0.1; // [atm]
d = 0.3*10^-6; // [m]
e = 0.305; // [porosity]
t = 4.39; // [tortuosity]
k = 1.3806*10^-23; // [J/K]
R = 8.314; // [cubic m.Pa/mole.K]
l = 2*10^-3; // [m]
D_ab = 2.01*10^-4; // [square m/s]
y_a1 = 0.8;
y_a2 = 0.2;
```

```
//*****//

// Using data from Appendix B for oxygen and nitrogen , and equation (1.45)
sigma_a = 3.467; // [Angstrom]
sigma_b = 3.798; // [Angstrom]
sigma_AB = ((sigma_a+sigma_b)/2)*10^-10; // [m]

lambda = k*T/(sqrt(2)*3.14*(sigma_AB^2)*P*1.01325*10^5); // [m]
// From equation 1.101
K_n = lambda/d;
printf("The value of a dimensionless ratio , Knudsen number is %f\n\n",K_n);

// It means that both molecular and Knudsen diffusion are important and equation (1.109) must be used to calculate N_a
// From example 1.21     N_b/N_a = -1.069
// Therefore si_a = 1/(1+(N_b/N_a))
si_a = 1/(1+(-1.069));

// From equation 1.100
D_abeff = D_ab*e/t; // [square m/s]

// From equation 1.103
D_Ka = (d/3)*(sqrt(8*R*T)/sqrt(3.14*M_a*10^-3)); // [square m/s]

// Using equation 1.107
D_Kaeff = D_Ka*e/t; // [square m/s]

Y_a = 1+(D_abeff/D_Kaeff);

// Using equation 1.109 to calculate N_a
N_a = (si_a*P*1.01325*10^5*D_abeff*log((si_a*Y_a-y_a2)/(si_a*Y_a-y_a1)))/(R*T*l);
N_b = -1.069*N_a;
```

```
printf(”The diffusion fluxes of both components oxygen and nitrogen are %e mole/square m.s and %e mole/square m.s respectively\n”,N_a,N_b);
```

Scilab code Exa 1.23 Dextrin Diffusion in a Porous Membrane

```
clear;
clc;

// Illustration 1.23
// Page: 62

printf(’Illustration 1.23 – Page:62 \n\n’);
// Solution

//*****Data*****//
// A–beta dextrin   B–water
T = 293; // [K]
d = 88.8; // [Average pore diameter, Angstrom]
d_mol = 17.96; // [Molecular diameter, Angstrom]
e = 0.0233; // [porosity]
t = 1.1; // [tortuosity]
D_AB = 3.22*10^-6; // [square cm/s]
//*****//

// Using equation 1.111 to calculate restrictive factor
K_r = (1-(d_mol/d))^4

// Using equation 1.110 to calculate effective diffusivity
D_ABeff = e*D_AB*K_r/t; // [square cm/s]
printf(”The effective diffusivity of beta–dextrin at 298 K is %e square cm/s”,D_ABeff);
```

Scilab code Exa 1.24 Hydrodynamic Flow in a Porous Diaphragm

```
clear;
clc;

// Illustration 1.24
// Page: 63

printf('Illustration 1.24 - Page:63 \n\n');
// Solution

//*****Data*****//
//   a-nitrogen
P_atm = 1.01325*10^5; // [Pa]
T = 300; // [K]
P_2 = 10130; // [Pa]
P_1 = 500+P_2; // [Pa]
d = 0.01*10^-2; // [average pore diameter, m]
u = 180; // [micro Poise]
u = 180*10^-6*10^-1; // [Pa.s]
l = 25.4*10^-3; // [m]
v = 0.05; // [volumetric flow rate, cubic m/square m.s]
R = 8.314; // [cubic m.Pa/mole.K]
//*****//

printf('Illustration 1.24 (a) - Page:63 \n\n');
// Solution (a)

P_avg = (P_1+P_2)/2; // [Pa]
// The mean free path for nitrogen is from equation (1.102)
lambda = 0.622*10^-6; // [m]
K_n = lambda/d;
```

```
// Therefore, Knudsen diffusion will not occur and all the flow observed is of a hydrodynamic nature.

// From the ideal gas law, the nitrogen flux corresponding to the volumetric flow rate of 0.05 m3/m2-s at 300 K and 1 atm

N_a = P_atm*v/(R*T); // [mole/square m.s]
// Using equation 1.113
B_o = u*R*T*N_a*l/(P_avg*(P_1-P_2)); // [square m]
printf("The value of the viscous flow parameter is %e square m\n\n",B_o);

printf('Illustration 1.24 (b) - Page:64 \n\n');
// Solution (b)

T1 = 393; // [K]
u = 220; // [micro Poise]
u = 220*10^-6*10^-1; // [Pa.s]
// Substituting in equation (1.113) the new values of temperature and viscosity and the value of B_o, obtained in part (a) while maintaining the pressure conditi// ons unchanged, we get N_a
N_a1 = B_o*P_avg*(P_1-P_2)/(l*T*u*R); // [mole/square m.s]
v1 = N_a1*R*T/P_atm; // [cubic m(measured at 300 K and 1 atm)/ square m.s]
printf("The nitrogen flow to be expected at 393 K with the same pressure difference is %e cubic m(measured at 300 K and 1 atm)/ square m.s\n",v1);
```

Chapter 2

Convective Mass Transfer

Scilab code Exa 2.1 Mass Transfer Coefficients in a Blood Oxygenator

```
clear;
clc;

// Illustration 2.1
// Page: 94

printf('Illustration 2.1 -  Page: 94\n\n');

// solution

//*****Data*****//
// a-oxygen   b-stagnant water
T = 310; // [K]
// Since the solubility of oxygen in water at 310 K is extremely low, we are dealing with dilute solutions
k_L = 3.3*10^-5; // [coefficient based on the oxygen concentration difference in the water, m/s]
row = 993; // [kg/cubic m]
M_b = 18; // [gram/mole]
//*****//
```

```

// Since we are dealing with very dilute solutions
// Therefore, c = (row/M_avg) = row/M_b
c = row/M_b; // [kmole/cubic m]
// Using equation 2.10
k_x = k_L*c; // [kmole/square m.s]
printf("The mass-transfer coefficient based on the mole fraction of oxygen in the liquid is %e kmole/square m.s\n\n",k_x);
```

Scilab code Exa 2.2 Mass Transfer Coefficient in a Gas Absorber

```
clear;
clc;

// Illustration 2.2
// Page: 95

printf('Illustration 2.2 -  Page: 95\n\n');

// solution

//*****Data*****//
//  a-ammonia   b-air
T = 300; // [K]
P = 1; // [atm]
y_a1 = 0.8; // [ammonia mole fraction in the bulk of the gas phase]
y_a2 = 0.732; // [ammonia gas-phase mole fraction on interface]
N_a = 4.3*10^-4; // [ammonia flux,  kmole/square m.s]
//*****//

// Using equation 2.2
```

```
F_g = N_a/log((1-y_a2)/(1-y_a1)); // [kmole/square m.s]
printf("The mass-transfer coefficient in the gas phase at that point where flux is 4.3*10^-4 is %e kmole/square m.s\n\n",F_g);
```

Scilab code Exa 2.3 Mass Transfer Coefficient in a Packed Bed Distillation Column

```
clear;
clc;

// Illustration 2.3
// Page: 96

printf('Illustration 2.3 -  Page: 96\n\n');

// solution

//*****Data*****//
// a-methanol   b-water
P = 101.3; // [kPa]
y_a1 = 0.707; // [mole fraction at interface]
y_a2 = 0.656; // [mole fraction at bulk of the gas]
k_g = 1.62*10^-5; // [mass-transfer coefficient at a point in the column, kmole/square m.s.kPa]
//*****//

// Using equation 2.14
k_y = k_g*P; // [kmole/square m.s]
// Using equation 2.12
N_a = k_y*(y_a1-y_a2); // [kmole/square m.s]
printf("The methanol flux at the point of given mass transfer coefficient is %e kmole/square m.s\n\n",N_a);
```

Scilab code Exa 2.4 Mass Transfer into a Dilute Stream Flowing Under Forced Convection in a Circular Conduit

```
clear;
clc;

// Illustration 2.4
// Page: 99

printf('Illustration 2.4 -  Page: 99\n\n');

// solution
// Mass Transfer into a Dilute Stream Flowing Under Forced Convection in a Circular Conduit

n = 6; // [number of variables]
//   Variables                       Symbols              Dimensions
// Tube diameter                       D                  L
// Fluid density                       row                M/L^3
// Fluid viscosity                     u                  M/(Lt)
// Fluid velocity                      v                  L/t
// Mass diffusivity                    D_AB               L^2/t
// Mass-transfer coefficient           kc                 L/t

// To determine the number of dimensionless parameters to be formed, we must know the rank, r, of the dimensional matrix.
// The dimensional matrix is
DM = [0,0,1,1,0,0;1,1,-3,-1,2,1;-1,-1,0,0,-1,-1];
[E,Q,Z ,stair ,rk]=ereduc(DM,1.d-15);
printf("Rank of matrix is %f\n\n",rk);

```

```
//The numbers in the table represent the exponent of M, L, and t in the dimensional expression of each of the six variables involved. For example, the dimensional expression of p is M/Lt; hence the exponents are 1, -1, and -1

// From equation 2.16
i = n-rk; // [number of dimensional groups]
// Let the dimensional groups are pi1, pi2 and pi3
// Therefore pi1 = (D_AB)^a*(row)^b*(D)^c*kc
//           pi2 = (D_AB)^d*(row)^e*(D)^f*v
//           pi3 = (D_AB)^g*(row)^h*(D)^i*u

// Solving for pi1
// M^0*L^0*t^0 = 1 = (L^2/t)^a*(M/L^3)^b*(L)^c*(L/t)

// Solution of simultaneous equation
function[f]=F(e)
    f(1) = 2*e(1)-3*e(2)+e(3)+1;
    f(2) = -e(1)-1;
    f(3) = -e(2);
    funcprot(0);
endfunction

// Initial guess:
e = [0.1 0.8 0.5];
y = fsolve(e,F);
a = y(1);
b = y(2);
c = y(3);
printf("The coefficients of pi1 are %f %f %f\n\n",a,b,c);
// Similarly the coefficients of pi2 and pi3 are calculated
// Therefore we get pi1 = kc*D/D_AB = Sh i.e. Sherwood Number
//                  pi2 = v*D/D_AB = P_ed i.e. Peclet Number
//                    pi3 = u/(row*D_AB) = Sc i.e. Schmidt Number

// Dividing pi2 by pi3 gives
//        pi2/pi3 = D*v*row/u = Re i.e. Renoylds number

printf('The result of the dimensional analysis of forced-convection mass transfer in a circular conduit indicates that a correlating relation could be of the form\n Sh = function(Re,Sc)\n which is analogous to the heat transfer correlation \n Nu = function(Re,Pr)');
```

Scilab code Exa 2.6 Mass Transfer to Fluid Flow Normal to a Cylinder

```
clear;
clc;

// Illustration 2.6
// Page: 111

printf('Illustration 2.6 -  Page: 111\n\n');

// solution
//*****Data*****//
// a-UF6    b-air
M_a = 352; // [molecular weight of UF6, gram/mole]
M_b = 29; // [gram/mole]
d = 0.01; // [diameter , m]
x = 0.1; // [length exposed to air stream , m]
v = 1; // [m/s]
Ts = 303; // [surface temperature of solid , K]
P_a = 27; // [vapor pressure of UF6, kPa]
Tb = 325; // [bulk temperature of solid ,K]
```

```
P = 101.3; // [kPa]
R = 8.314; // [cubic m.Pa/mole.K]
//*****//

y_a1 = P_a/P; // [mole fraction at point 1]
y_a2 = 0; // [mole fraction at point 2]

// Along the mass-transfer path-cylinder surface (point 1) to bulk air (point 2)
Tavg = (Ts+Tb)/2; // [K]

// At point 1, the gas is saturated with UF6 vapor, while at point 2 the gas is virtually free of UF6
// Therefore
Pavg = (P_a+0)/2; // [average partial pressure, kPa]
y_a = Pavg/P; // [mole fraction of UF6]

Mavg = M_a*y_a+M_b*(1-y_a); // [gram/mole]
row_avg = P*Mavg/(R*Tavg); // [kg/cubic m]

// Parameter for c-O2, d-N2 and a-UF6
yi_c = 0.182;   yi_d = 0.685;  yi_a = 0.133;
Tc_c = 154.6;   Tc_d = 126.2;  Tc_a = 505.8; // [K]
Pc_c = 50.4;    Pc_d = 33.9;   Pc_a = 46.6;  // [bar]
M_c = 32;       M_d  = 28;     M_a  = 352;  // [gram/mole]
V_c = 73.4;     V_d  = 89.8;  V_a  = 250; // [cubic cm/mole]
Z_c = 0.288;    Z_d  = 0.290;  Z_a  = 0.277;

// From equation 2.52 and 2.53
Tcm = yi_c*Tc_c+yi_d*Tc_d+yi_a*Tc_a; // [K]
Pcm = 10^6*R*Tcm*(yi_c*Z_c+yi_d*Z_d+yi_a*Z_a)/((yi_c*V_c+yi_d*V_d+yi_a*V_a)*100000); // [bar]
M_avg = yi_c*M_c+yi_d*M_d+yi_a*M_a; // [gram/mole]

// From equation 2.50
```

```
Em =  0.176*(Tcm/(M_avg^3*Pcm^4))^(1/6); // [uP]^-1

// From equation 2.51
Trm = Tavg/Tcm;
f_Trm = (0.807*Trm^0.618)-(0.357*exp(-0.449*Trm))+(0.340*exp(-4.058*Trm))+0.018;
// From equation 2.49
u = f_Trm/Em; // [uP]
u = u*10^-7; // [viscosity , kg/m.s]

Re = d*v*row_avg/u; // [Renoylds number]

// Diffusivity of UF6 vapors in air at 314 K and 1 atm from equation 1.49
D_ab = 0.0904; // [square cm/s]

Sc = u/(row_avg*D_ab*10^-4); // [Schmidt number]

Sh_avg = 0.43 + 0.532*Re^0.5*Sc^0.31; // [Sherwood number]
// From equation 1.7
c = P/(R*Tavg); // [kmole/cubic m]
// From Table 2.1
F_av = Sh_avg*D_ab*c*10^-4/d; // [kmole/square m.s]

// From equation 2.2
N_avg = F_av*log((1-y_a2)/(1-y_a1)); // [kmole/square m.s]
S = 2*%pi*d^2/4 +%pi*d*x; // [total surface area of the cylinder , square m]

w_a = N_avg*S*M_a; // [rate of sublimation of the solid , kg/s]
printf("Rate of sublimation of a cylinder of UF6 is %e kg/s\n\n",w_a);
```

Scilab code Exa 2.7 The Chilton Colburn Analogy

```
clear;
clc;

// Illustration 2.7
// Page: 116

printf('Illustration 2.7 -  Page: 116\n\n');

// solution
//*****Data*****//
// a-benzene       b-nitrogen
T = 300; // [K]
P = 101.3; // [kPa]
v =10; // [m/s]
R = 8.314; // [cubic m.Pa/mole.K]
//*****//

// Combining the given correlation with the definitions of j-H, and St_H, from Table 2.1 yields
//              j_H = h*Pr^(2/3)/(Cp*row*v) = h*Pr^(2/3)/(Cp*Gy) = f(Re)
// Therefore
//              h = Cp*Gy*f(Re)/(Pr)^(2/3) = 20*(Gy)^0.5     for carbon dioxide

// Since Re = row*v*l/u = Gy*l/u, where 'l' is a characteristic length, the function f(Re) must be compatible with 20*Gy^0.5 .Therefore, let f(Re) = bRe^n, where 'b'  and 'n' are constants to be evaluated. Then,

//              h =  (Cp*Gy*b/Pr^(2/3))*(l*Gy/u)^n = 20*Gy^0.5
// Comparing both sides of equation, we get
//              n+1 =0.5
// Therefore
n = -0.5;
// Data on the properties of C02 at 300 K and 1 bar
u = 1.5*10^-5; // [viscosity, P]
Pr = 0.77; // [Prandtl number]
Cp = 853; // [J/kg.K]
// Therefore
//             b = 5.086*l^0.5
//             j_D = j_H = f(Re) = 5.086*(l^0.5)*Re^-0.5
// From Table 2.1
//             F = j_D*c*v/Sc^(2/3) = 5.086*(l^0.5)*c*v/(Re^0.5*Sc^(2/3)) = 5.086*(row*v*u)^0.5/(Mavg*Sc^(2/3))

// Vapor pressure of benzene
P_a = exp(15.9008-(2788.51/(T-52.36))); // [mm of Hg]
P_a = P_a*101.3/760; // [kPa]

// Parameter for a-benzene, b-nitrogen
yi_a = 0.07;      yi_b = 0.93;
Tc_a = 562.2;     Tc_b = 126.2;   // [K]
Pc_a = 48.9;      Pc_b = 33.9;   // [bar]
M_a = 78.1;       M_b  = 28;    // [gram/mole]
V_a = 259;        V_b  = 89.8; // [cubic cm/mole]
Z_a = 0.271;      Z_b  = 0.290;
sigma_a = 5.349; sigma_b = 3.798; // [Angstrom]
ek_a = 412.3;     ek_b = 71.4; // [E/k, K]


// From equation 2.52 and 2.53
Tcm = yi_b*Tc_b+yi_a*Tc_a; // [K]
Pcm = 10^6*R*Tcm*(yi_b*Z_b+yi_a*Z_a)/((yi_b*V_b+yi_a
*V_a)*100000); // [bar]
M_avg = yi_b*M_b+yi_a*M_a; // [kg/kmole]
printf("Average molecular weight is %f kg/kmole\n\n",M_avg);
row = P*M_avg/(R*T); // [kg/cubic m]
printf("Density of mixture is %f kg/cubic m\n\n",row);
// From equation 2.50
Em =  0.176*(Tcm/(M_avg^3*Pcm^4))^(1/6); // [uP]^-1

// From equation 2.51
Trm = T/Tcm;
f_Trm = (0.807*Trm^0.618)-(0.357*exp(-0.449*Trm))+(0.340*exp(-4.058*Trm))+0.018;
// From equation 2.49
u = f_Trm/Em; // [uP]
u = u*10^-7; // [viscosity, kg/m.s]
printf("Average viscosity of mixture is %e kg/m.s\n\n",u);

// Calculating diffusivity of benzene using equation 1.49
D_ab = 0.0986; // [square cm/s]
Sc = u/(row*D_ab*10^-4); // [Schmidt number]

F = 5.086*(row*v*u)^0.5/(M_avg*Sc^(2/3)); // [kmole/square m.s]
printf("The required mass transfer coefficient is %e kmole/square m.s\n\n",F);
```

Scilab code Exa 2.8 Benzene Evaporation Along a Vertical Flat Plate

```
clear;
clc;

```

```
// Illustration 2.8
// Page: 120

printf('Illustration 2.8 -  Page: 120\n\n');

// solution
//*****Data*****//
//   a-liquid benzene     b-nitrogen
T = 300; // [K]
l = 3; // [length of vertical plate, m]
b = 1.5; // [width of vertical plate, m]
P = 101.3; // [kPa]
v = 5; // [velocity across the width of plate, m/s]
row_a = 0.88; // [gram/cubic cm]
//*****//

y_a1 = 0.139; // [mole fraction of benzene at inner edge]
y_a2 = 0;

// The film conditions, and average properties, are identical to those in Example 2.7, only the geometry is different
// Therefore
M_avg = 31.4; // [kg/kmole]
row = 1.2; // [kg/cubic m]
u = 161*10^-7; // [kg/m.s]
D_ab = 0.0986; // [square cm/s]
Sc = 1.3; // [Schmidt Number]
Re = row*v*b/u; // [Renoylds Number]

if(Re>4000)
    printf('The flow across the plate is turbulent\n\n');
    else(Re<2000)
    printf('The flow across the plate is laminar\n\n');
    end
```

```

// Using equation 2.57
Sh_l = 0.036*Re^0.8*Sc^(1/3);

// Nitrogen (component B) does not react with benzene (component A), neither dissolves in the liquid; therefore, NB = 0 and siA = 1. The F-form of the mass-transfer coefficient should be used
F = Sh_l*1.26*D_ab*10^-4/(M_avg*b); // [kmole/square m.s]
N_a = F*log((1-y_a2)/(1-y_a1)); // [kmole/square m.s]

// The total mass rate of evaporation over the surface of the plate
S = 1.5*3; // [square m]
M_a = 78.1; // [gram/mole]
wa = N_a*S*M_a*60*1000; // [gram/min]

V = wa/row_a; // [volumetric flow rate, ml/min]

printf("Liquid benzene should be supplied at the top of the plate at the rate %f ml/min so that evaporation will just prevent it from reaching the bottom of the plate.\n\n",V);
```

Scilab code Exa 2.9 Evaporation of a Drop of Water Falling in Air

```
clear;
clc;

// Illustration 2.9
// Page: 123

printf('Illustration 2.9 - Page: 123\n\n');
```

```

// solution
//*****Data*****//
// a-water    b-air
dp1 = 10^-3; // [diameter of spherical drop of water , m]
Tair = 323; // [K]
P = 101.3; // [kPa]
Twater = 293; // [K]
R = 8.314; // [cubic m.Pa/mole.K]
M_a = 18; // [gram/mole]
M_b = 29; // [gram/mole]
//*****//

dp2 = (1/2)^(1/3)*dp1; // [m]
dp = (dp1+dp2)/2; // [m]

row_p = 995; // [density of water, kg/cubic m]
row1b = 1.094; // [density of air, kg/cubic m]
u = 1.95*10^-5; // [kg/m.s]
row_pr = row_p-row1b; // [kg/cubic m]
g = 9.8; // [accleration due to gravity, square m/s]
// Combining equation 2.68 and 2.69
Ga = 4*dp^3*row1b*row_pr*g/(3*u^2); // [Galileo Number]

// Relationship between Re and Cd
// Re/Cd = Re^3/Ga = 3*row^2*vt^3/(4*g*u*row_pr)

// The following correlation is used to relate Re/Cd , to Ga
// ln(Re/Cd)^(1/3) = -3.194 + 2.153*ln(Ga)^(1/3) - 0.238*(ln(Ga)^(1/3))^2 + 0.01068*(ln(Ga)^(1/3))^3
// Therefore let A = (Re/Cd)
A = exp(-3.194 + 2.153*log(Ga^(1/3)) - 0.238*(log(Ga^(1/3)))^2 + 0.01068*(log(Ga^(1/3)))^3);

// Therefore 'vt' will be
```

```
vt = A*(4*g*row_pr*u/(3*row1b^2))^(1/3); // [Terminal velocity of particle , m/s]
printf("Terminal velocity of particle is %f m/s\n\n",vt);

P_w = 2.34; // [vapor pressure of water , kPa]
y_w = P_w/P; // [mole fraction of water at the inner edge of the gas film]
M_avg = 18*y_w+29*(1-y_w); // [gram/mole]

row2b = P*M_avg/(R*Twater); // [kg/cubic.m]
delta_row = row2b - row1b; // [kg/cubic.m]

Tavg = (Tair+Twater)/2; // [K]
// At Temperature equal to Tavg density and viscosity are
row3 = 1.14; // [kg/cubic.m]
u1 = 1.92*10^-5; // [kg/m.s]

Grd = g*row3*delta_row*(dp^3)/(u1^2);

// Diffusivity of water at Tavg and 1 atm is
D_ab = 0.242*10^-4; // [square m/s]
Sc = u1/(row3*D_ab); // [Schmidt Number]
Re = dp*row3*vt/u1; // [Renoylds Number]

// From equation 2.65 Re is greater than 0.4*Grd^0.5*Sc^(-1/6)
// Therfore equation 2.64 can be used to calculate mass transfer coefficient

Sh = 2+0.552*(Re^0.5)*Sc^(1/3); // [Sherwood Number]
// From Table 2.1
// Sh = kc*P_bm*dp/(P*D_ab), since P_bm is almost equal to P
// Therefore
// Sh = kc*dp/D_ab;
kc = Sh*D_ab/dp; // [m/s]
```

```

ca2 = 0; // [dry air concentration]
ca1 = P_w/(R*Twater); // [interface concentration, kmole/cubic.m]
// Average rate of evaporation
wa = %pi*dp^2*M_a*kc*(ca1-ca2)*1000; // [g/s]

// Amount of water evaporated
m = row_p*%pi*dp1^3/12*1000; // [g]
// Time necessary to reduce the volume by 50%
t = m/wa; // [s]

D = t*vt; // [distance of fall, m]
printf("The distance of fall is %f m\n\n",D);
```

Scilab code Exa 2.10 Mass Transfer for a Single Cylinder

```
clear;
clc;

// Illustration 2.10
// Page: 127

printf('Illustration 2.10 - Page: 127\n\n');

// solution

// Example 2.6 using equation 2.73
// Values of the dimensionless parameters calculated in Example 2.6
Re = 1223; // [Renoylds Number]
Sc = 0.905; // [Schmidt Number]
c = 0.039; // [molar density, kg/cubic m]
v = 1; // [gas velocity, m/s]
// Therefore
```

```
Gm = c*v; // [kmole/square m.s]
// From equation 2.9
// Kg*P = ky
// Therefore substituting in equation 2.73 we obtain
ky = 0.281*Gm/(Re^0.4*Sc^0.56); // [kmole/square m.s]
// Now equation 2.73 were obtained under very dilute concentrations
// Therefore
y_bm = 1;
// From equation 2.6
F = ky*y_bm; // [kmole/square m.s]
printf("Mass transfer coefficient is %e kmole/square m.s\n\n",F);
```

Scilab code Exa 2.11 Simultaneous Heat and Mass Transfer in Pipe

```
clear;
clc;

// Illustration 2.11
// Page: 129

printf('Illustration 2.11 -  Page: 129\n\n');

// solution
//*****Data*****//
// a-water   b-air
D = 25.4*10^-3; // [diameter of wetted wall tower, m]
Gy = 10; // [mass velocity, kg/square m.s]
T1 = 308; // [K]
P = 101.3; // [kPa]
p_a1 = 1.95; // [partial pressure of water vapor, kPa]
```

```
R = 8.314; // [cubic m.Pa/mole.K]
M_a = 18; // [gram/mole]
Cpa = 1.88; // [kJ/kg.K]
//*****//

// Properties of dry air at 308 K and 1 atm pressure
    are
u = 1.92*10^-5; // [kg/m.s]
row = 1.14; // [kg/cubic m]
D_ab = 0.242*10^-4; // [square m/s]
Sc = 0.696; // [Schmidt number]
Cp = 1.007; // [kJ/kg.K]
k = 0.027; // [W/m.K]
Pr = 0.7; // [Prandtl number]

Re = D*Gy/u; // [Renoylds number]
// From equation 2,74
Sh = 0.023*Re^0.83*Sc^0.44; // [Sherwood number]
// From Table 2.1
kg = Sh*D_ab/(R*T1*D)*1000; // [mole/square m.s.kPa]
printf(”kg is %e\n”,kg);
// To estimate the heat-transfer coefficient, we use
    the Dittus-Boelter equation for cooling,
   equation 2.80
Nu = 0.023*Re^0.8*Pr^0.3; // [Nusselt number]
// From Table 2.1
h = Nu*k/D; // [W/square m.K]
printf(”h is %f\n”,h);
T =373.15; // [K]
lambda_a = 40.63; // [kJ/mole]
Tc = 647.1; // [K]

// Solution of simultaneous equation 2.78 and 2.79
function[f]=F(e)
    f(1) = kg*(p_a1 - exp(16.3872-(3885.7/(e(1)
       -42.98))))-e(2);
    f(2) = e(2)*M_a*Cpa*(T1-e(1))/(1-exp(-e(2)*M_a*
       Cpa/h)) + 1000*e(2)*lambda_a*((1-(e(1)/Tc))
/(1-(T/Tc)))^0.38;
    funcprot(0);
endfunction

// Initial guess
e = [292 -0.003];
y = fsolve(e,F);
Ti = y(1);
Na = y(2);

printf("The temperature of the liquid water and the rate of water evaporation is %f K and %e mole/square m.s respectively",Ti,Na);
```

Scilab code Exa 2.12 Air Humidification in Wetted Wall Column

```
clear;
clc;

// Illustration 2.12
// Page: 131

printf('Illustration 2.12 -  Page: 131\n\n');

// solution
//*****Data*****//
// a-water    b-dry air
D = 25.4*10^-3; // [Internal diameter of tower, m]
Z = 1.5; // [length of the wetted section, m]
Gy = 10; // [mass velocity of air, kg/square m.s]
Tair = 308; // [K]
Twater = 295; // [K]
P = 101.3; // [kPa]
M_a = 18; // [gram/mole]
M_b = 29; // [gram/mole]
```

```
R = 8.314; // [cubic m.Pa/mole.K]
//*****//

// The water vapor partial pressure at the interface remains constant at the vapor pressure of liquid water at 295 K, which is pa1 = Pa = 2.64 kPa
// The water vapor partial pressure at the bulk of the gas phase increases from pA2 = pAin = 0 for the dry inlet air to pa2= pAout for the air leaving the tower
Pa = 2.64; // [kPa]

Gm = Gy/M_b; // [Assuming that gas phase is basically dry air , kmole/square m.s]
// The properties of dry air at 308 K and 1 atm are (from example 2.9)
row = 1.14; // [kg/cubic m]
u = 1.92*10^-5; // [kg/m.s]
D_ab = 0.242*10^-4; // [square m/s]
Sc = 0.692; // [Schmidt number]

Re = Gy*D/u; // [Renoylds number]

if(Re<35000 & Re>2000)
// From equation 2.74
Sh = 0.023*Re^0.83*Sc^0.44; // [Sherwood number]

printf(”Sherwood number is %f\n\n”,Sh);
else()
    printf(’We cannot use equation 2.74’)
end

c = P/(R*Tair); // [kmole/cubic m]
// Now using equation 2.89
Pa_out = Pa*(1-exp((-4*Sh*Z*c*D_ab)/(Gm*D^2))); // [kPa]
printf(”The partial pressure of water in the air leaving the tower is %e kPa\n\n”,Pa_out);
```

Scilab code Exa 2.13 Air Humidification in a Packed Bed

```
clear;
clc;

// Illustration 2.13
// Page: 134

printf('Illustration 2.13 - Page: 134\n\n');

// solution
//*****Data*****//
// a-water    b-dry air
Gy = 10; // [kg/square m.s]
dp = 3.5*10^-3; // [diameter of spherical glass beads, m]
D = 25.4*10^-3; // [Internal diameter of tower, m]
Tair = 308; // [K]
Twater = 295; // [K]
P = 101.3; // [kPa]
M_a = 18; // [gram/mole]
M_b = 29; // [gram/mole]
R = 8.314; // [cubic m.Pa/mole.K]

// The properties of dry air at 308 K and 1 atm are (from example 2.12)
row = 1.14; // [kg/cubic m]
u = 1.92*10^-5; // [kg/m.s]
D_ab = 0.242*10^-4; // [square m/s]
Sc = 0.692; // [Schmidt number]
c = 0.04; // [mole/cubic m]
Gm = 0.345; // [kmole/square m.s]

Re = Gy*dp/u; // [Renoylds number]
```

```
if(Re<2500 & Re>10)

// Subsituting in equation 2.90
jd = 1.17*Re^-0.415;
printf("Renoylds number is %f\n\n",Re);
else()
end

Std = 0.052/(Sc^(2/3));
// From Table 2.1
Sh = Std*Re*Sc; // [Sherwood number]
// From equation 2.94
e = 0.406+0.571*(dp/D); // [bed porosity]

printf('Illustration 2.13(a) - Page: 134\n\n');
// Solution(a)
// Now Paout = 0.99*Pa
// Using equation 2.93 to calculate 'Z'
deff('[y] = f12(Z)','y = 0.99 - 1 + exp(-6*(1-e)*Sh*c*Z*D_ab/(Gm*dp^2))');
Z = fsolve(0.06,f12);
printf("The depth of packing required is %e m\n\n",Z);

printf('Illustration 2.13(b) - Page: 136\n\n');
// Solution(b)
// From equation 2.95
deltaP = (150*(1-e)/Re + 1.75)*(1-e)*Gy^2*Z/(dp*row*e^3); // [Pa]
printf("The gas pressure drop through the bed is %f Pa\n\n",deltaP);
```

Scilab code Exa 2.14 Design of a Hollow Fiber Boiler Feed Water Deaerator

```
clear;
clc;

// Illustration 2.14
// Page: 138

printf('Illustration 2.14 -  Page: 138\n\n');

// solution
// a-oxygen     b-water
// To design the deaerator, We will use commercially available microporous polypropylene hollow fibers in a module
// Given data:
m = 40000; // [kg/hr]
Twater = 298; // [K]
v = 0.1; // [superficial velocity, m/s]
P = 101.3; // [kPa]
V = 40*10^-3; // [Flow rate of nitrogen, cubic m/min]
d = 2.90*10^-4; // [Outside diameter of fibres, m]
pf = 0.4; // [Packing factor]
a = 46.84*100; // [surface area per unit volume, m^-1]
R = 8.314; // [cubic m.Pa/mole.K]
// *****//

dw = 1000; // [density of water, kg/cubic m]
Ql = m/(3600*1000); // [volumetric water flow rate, cubic m/s]
// Shell diameter
D = (4*Ql/(%pi*v))^0.5; // [Shell diameter, m]

// the properties of dilute mixtures of oxygen in water at 298 K
u = 0.9; // [cP]
// Diffusivity from equation 1.53
D_ab = 1.93*10^-9; // [square m/s]
```

```
Sc = 467; // [Schmidt number]

Re = d*v*dw/(u*10^-3); // [Renoylds number]

// Substituting in equation (2−97) gives
Sh = 0.53*(1-1.1*pf)*((1-pf)/pf)^-0.47*(Re^0.53*Sc^0.33);

kl = Sh*D_ab/d; // [mass−transfer coefficient on the shell side, m/s]

// From the specified BFW flow rate
L = m/(3600*18); // [kmole/s]
// From ideal gas law
V1 = V*P/(Twater*R*60); // [kmole/s]
// From the solubility of oxygen in water at 298 K,
M = 4.5*10^4;
A = L/(M*V1); // [Absorption factor]
printf(”Absorption factor is %f\n”,A);

// For 99% removal of the dissolved oxygen
// x_in/x_out = b = 100
b = 100;
c = 55.5 // [molar density, kmole/cubic m]
// Substituting in equation 2.99 yields
V_T = (L*log(b*(1-A)+A))/(kl*a*c*(1-A)); // [cubic m]

// The module length, Z is
Z = V_T/(%pi*D^2/4);
printf(”The shell diameter and module length is %f m and %f m respectively\n\n”,D,Z);
```

Chapter 3

Interphase Mass Transfer

Scilab code Exa 3.1 Application of Raoults Law to a Binary System

```
clear;
clc;

// Illustration 3.1
// Page: 161

printf('Illustration 3.1 -  Page: 161\n\n');

// solution

//*****Data*****//
// a-benzene  b-toluene
T = 300; // [K]
x_a = 0.4; // [mole fraction in liquid phase]
// Antoine constants for benzene and toluene are given
// For benzene
A_a = 15.9008;
B_a = 2788.51;
C_a = -52.36;
// For toluene
```

```
A_b = 16.0137;
B_b = 3096.52;
C_b = -53.67;
//*****//

// Using equation 3.5 vapor pressure of component 'a' and 'b'
P_a = exp(A_a-(B_a/(T+C_a))); // [mm of Hg]
P_b = exp(A_b-(B_b/(T+C_b))); // [mm of Hg]

P_a = P_a*101.3/760; // [kPa]
P_b = P_b*101.3/760; // [kPa]
// Partial pressure of component 'a' and 'b'
p_a = x_a*P_a; // [kPa]
p_b = (1-x_a)*P_b; // [kPa]
P_total = p_a+p_b; // [kPa]

printf("The total equilibrium pressure of the binary system of benzene and toluene is %f kPa\n\n",P_total);

y_a = p_a/P_total; // [mole fraction in vapor phase]
printf("The composition of the vapor in equilibrium is %f\n\n",y_a);
```

Scilab code Exa 3.2 Henrys Law Saturation of Water with Oxygen

```
clear;
clc;

// Illustration 3.2
// Page: 162

printf('Illustration 3.2 - Page: 162\n\n');

```

```
// solution
//*****Data*****//
// A-oxygen   B-water
T = 298; // [K]
H = 4.5*10^4; // [atm/mole fraction]
P = 1; // [atm]
row_B = 1000; // [density of water, kg/cubic m]
M_B = 18; // [Molecular mass of water, gram/mole]
M_A = 32; // [,Molecular mass of oxygen, gram/mole]
//*****//

// Dry air contains 21% oxygen; then p_A = y*P = 0.21 atm
// Therefore using Henry's Law
p_A = 0.21; // [atm]
x_A = p_A/H; // [mole fraction in liquid phase]

// Basis: 1L of saturated solution
// For 1 L of very dilute solution of oxygen in water, the total moles of solution, n_t, will be approximately equal to the moles of water
n_t = row_B/M_B
// Moles of oxygen in 1L saturated solution is
n_o = n_t*x_A; // [mole]
// Saturation concentration
c_A = n_o*M_A*1000; // [mg/L]
printf("The saturation concentration of oxygen in water exposed to dry air at 298 K and 1 atm is %f mg/L\n\n",c_A);
```

Scilab code Exa 3.3 Material Balances Combined with Equilibrium Relations Algebraic Solution

```
clear;
clc;
```

```

// Illustration 3.3
// Page: 162

printf('Illustration 3.3 -  Page: 162\n\n');

// solution
//*****Data*****//
// a-ammonia   b-air   c-water
T = 300; // [K]
P = 101.3; // [kPa]
R = 8.314; // [cubic m.Pa/mole.K]
V_b = 15; // [cubic m]
m_a = 10; // [kg]
m_c = 45; // [kg]
M_a = 17; // [molecular mass of ammonia, gram/mole]
M_c = 18; // [molecular mass of water, gram/mole]
//*****//

n_b = V_b*P/(R*T); // [kmole]
n_a = m_a/M_a; // [kmole]
n_c = m_c/M_c; // [kmole]

// L_a as the number of kmol of ammonia in the liquid phase when equilibrium is achieved
// And n_a-L_a kmol of ammonia will remain in the gas phase
// x_a = L_a/(n_c+L_a)                          (1)
// y_a = (n_a-L_a)/(n_b+n_a-L_a)                (2)
// gamma = 0.156+0.622*x_a*(5.765*x_a-1)        (3)   for x_a <= 0.3
// y_a = 10.51*gamma*x_a;                       (4)
// Equations (1),(2),(3), and (4) are solved simultaneously
deff('[y] = f12(L_a)','y = ((n_a-L_a)/(n_b+n_a-L_a))-(10.51*(0.156+(0.622*(L_a/(n_c+L_a))*(5.765*(L_a/(n_c+L_a))-1)))*(L_a/(n_c+L_a)))');
L_a = fsolve(0.3,f12); // [kmole]
```

```

x_a = L_a/(n_c+L_a);
y_a = (n_a-L_a)/(n_b+n_a-L_a);
gammma_a = 0.156+0.622*x_a*(5.765*x_a-1);

printf("At equilibrium the ammonia content of the liquid phase will be %f\n\n",x_a);
printf("At equilibrium the ammonia content of the gas phase will be %f\n\n",y_a);
printf("The amount of ammonia absorbed by the water will be %f kmole\n\n",L_a);
```

Scilab code Exa 3.4 Mass Transfer Resistances During Absorption of Ammonia by Water

```
clear;
clc;

// Illustration 3.4
// Page: 169

printf('Illustration 3.4 -  Page: 169\n\n');

// solution
//*****Data*****//
//  a-ammonia
T  = 300; // [K]
P = 101.3; // [kPa]
Kg = 2.75*10^-6; // [kmole/square m.s.kPa]
m = 1.64;
res = 0.85; // [gas phase resistance]
xa_g = 0.115/100; // [mole fraction of NH3 in liquid phase at a point]]
ya_g = 8/100; // [mole fraction of NH3 in gas phase at a point]
```

```
//*****//

Ky = Kg*P; // [kmole/square m.s]
// Using equation 3.24
ky = Ky/res; // [kmole/square m.s]
// Using equation 3.21
deff('[y] = f12(kx)','y = (m/kx)-(1/Ky)+(1/ky)');
kx = fsolve(0.0029,f12); // [kmole/square m.s]

// Interfacial concentrations at this particular point in the column, using equation (3.15)
ystar_a = m*xa_g;
// Using equation 3.12
N_a = Ky*(ya_g-ystar_a); // [kmole/square m.s]
// Gas-phase interfacial concentration from equation (3.9)
ya_i = ya_g-(N_a/ky);
// Since the interfacial concentrations lie on the equilibrium line, therefore
xa_i = ya_i/m;
// Cross checking the value of N_a
N_a = kx*(xa_i-xa_g); // [kmole/square m.s]

printf("The individual liquid film coefficient and gas film coefficient are %e kmole/square m.s %e kmole/square m.s respectively\n\n",kx,ky);
printf("The gas phase and liquid phase interfacial concentrations are %f and %f respectively\n\n",ya_i,xa_i);
```

Scilab code Exa 3.5 Absorption of Ammonia by Water Use of F Type Mass Transfer Coefficients

```
clear;
clc;
```

```

// Illustration 3.5
// Page: 171

printf('Illustration 3.5 -  Page: 171\n\n');

// solution
//*****Data*****//
//  a-ammonia
T = 300; // [K]
P = 101.3; // [kPa]
ya_g = 0.6; // [ammonia concentration in bulk gas]
xa_l = 0.12; // [ammonia concentration in bulk liquid]
Fl = 3.5*10^-3; // [kmole/square m.s]
Fg = 2*10^-3; // [kmole/square m.s]
//*****//

// Algebraic solution (a)

// In gas phase substance 'A' is ammonia and 'B' is air
// Assuming N_BG = 0 and sia_AG = 1  and
// In liquid phase substance 'B' is water
// Assuming N_BL = 0 and sia_AL = 1
// Then equation 3.29 reduces to 3.30

// Using equation 3.30, 3.8(a),3.6(a)
// ya_i = 1-(1-ya_g)*((1-xa_l)/(1-xa_i))^(Fl/Fg)      3.30
// ya_i = 10.51*gamma*xa_i        3.8(a)
// gamma = 0.156+0.622*xa_i*(5.765*xa_i-1)       3.6(a)

deff('[y] = f12(xa_i)','y = 1-(1-ya_g)*((1-xa_l)/(1-xa_i))^(Fl/Fg) - 10.51*(0.156+0.622*xa_i*(5.765*xa_i-1))*xa_i');
```

```
xa_i = fsolve(0.2,f12);

ya_i = 1-(1-ya_g)*((1-xa_l)/(1-xa_i))^(Fl/Fg);
printf("The local gas and liquid interfacial concentrations are %f and %f respectively\n\n",ya_i,xa_i);
// Using equation 3.28
N_a = Fg*log((1-ya_i)/(1-ya_g));
printf("The local ammonia mass-transfer flux is %e kmole/square m.s\n\n",N_a);
```

Scilab code Exa 3.6 Distillation of a Mixture of Methanol and Water in a Packed Tower Use of F Type Mass Transfer Coefficients

```
clear;
clc;

// Illustration 3.6
// Page: 175

printf('Illustration 3.6 - Page: 175\n\n');

// solution
//*****Data*****//
// a-methanol b-water
T = 360; // [K]
P = 101.3; // [kPa]
lambda_a = 33.3; // [MJ/kmole]
lambda_b = 41.3; // [MJ/kmole]
Fg = 0.0017; // [kmole/square m.s]
Fl = 0.0149; // [kmole/square m.s]
yag = 0.36; // [bulk gas phase concentration]
xag = 0.20; // [bulk liquid phase concentration]
R = 1.987;
//*****//
```

```

// From energy balance
// Nb = -(lambda_a/lambda_b)*Na
// and    sia_ag = sia_al = 1/(1-(lambda_a/lambda_b))
sia_ag =5.155;
sia_al = sia_ag;
// Therefore equation 3.29 becomes
// yai = 5.155-4.795(4.955/(5.155-xai))^8.765

// Using equation 3.33, 3.34, 3.35
V2 = 18.07; // [cubic cm/mole]
V1 = 40.73; // [cubic cm/mole]
a12 = 107.38; // [cal/mole]
a21 = 469.5; // [cal/mole]

// Solution of simultaneous equation
function[f]=F(e)
    f(1) = e(1)+e(2)-1;
    f(2) = e(3)+e(4)-1;
    f(3) = e(3)-5.155+4.795*(4.955/(5.155-e(1)))^(Fl/Fg);
    f(4) = e(3)-((e(1)*exp(16.5938-(3644.3/(e(5)-33))))*(exp(-log(e(1)+e(2)*(V2/V1*exp(-a12/(R*e(5)))))+e(2)*(((V2/V1*exp(-a12/(R*e(5))))/(e(1)+e(2)*(V2/V1*exp(-a12/(R*e(5))))))-((V1/V2*exp(-a21/(R*e(5))))/(e(2)+e(1)*(V1/V2*exp(-a21/(R*e(5)))))))))/P;
    f(5) = e(4)-((e(2)*exp(16.2620-(3800/(e(5)-47))))*(exp(-log(e(2)+e(1)*(V1/V2*exp(-a21/(R*e(5)))))-e(1)*(((V2/V1*exp(-a12/(R*e(5))))/(e(1)+e(2)*(V2/V1*exp(-a12/(R*e(5))))))-((V1/V2*exp(-a21/(R*e(5))))/(e(2)+e(1)*(V1/V2*exp(-a21/(R*e(5)))))))))/P;
    funcprot(0);
endfunction

// Initial guess
```

```
e =[0.1 0.9 0.2 0.8 300];
y = fsolve(e,F);
xai = y(1);
xbi = y(2);
yai = y(3);
ybi = y(4);
T = y(5); // [K]

printf("yai is %f\n",yai);
printf("ybi is %f\n",ybi);
printf("xai is %f\n",xai);
printf("xbi is %f\n",xbi);
printf("Temperature is %f\n",T);
// Local Methanol flux , using equation 3.28
Na = sia_ag*Fg*log((sia_ag-yai)/(sia_ag-yag)); // [kmole/square m.s]
printf("Local Methanol flux is %e kmole/square m.s\n\n",Na);
```

Scilab code Exa 3.7 Recovery of Benzene Vapors from a Mixture with Air

```
clear;
clc;

// Illustration 3.7
// Page: 183

printf('Illustration 3.7 – Page: 183\n\n');

// solution
//*****Data*****//
// 1–benzene a–absorber s–steams
T = 300; // [K]
P = 101.3; // [kPa]
R = 8.314; // [gas constant]
```

```
v = 1; // [cubic m/s]
// Gas in
y1a = 0.074;
// Liquid in
x2a = 0.0476
// Recovery is 85 %
// Calculations for absorber section

V1a = P*v/(R*T); // [kmole/s]
// Inert gas molar velocity
Vsa = V1a*(1-y1a); // [kmole/s]
Y1a = y1a/(1-y1a); // [kmole of benzene/kmole of dry gas]

X2a = x2a/(1-x2a); // [kmole of benzene/kmole of oil]
// Since the absorber will recover 85% of the benzene in the entering gas, the concentration of the gas leaving it will be
r = 0.85;
Y2a = (1-r)*Y1a; // [kmole of benzene/kmole of dry gas]
// The benzene-wash oil solutions are ideal, and the pressure is low; therefore, Raoult s law applies. From equations 3.1, 3.44, and 3.45
//      yia = 0.136*xia
// or   Yia/(1+Yia) = 0.136*Xia/(1+Xia)

// Data_eqm = [Xia Yia]
Data_eqm = [0 0;0.1 0.013;0.2 0.023;0.3 0.032;0.4 0.04;0.6 0.054;0.8 0.064;1 0.073;1.2 0.080;1.4 0.086];

// Here because of the shape of equilibrium curve, the operating line for minimum oil rate must be tangent to curve
// Therefore
// From the curve X1a_max = 0.91
```

```
X1a_max = 0.91; // [kmol benzene/kmol oil]

// For minimum operating line slope is
S = (Y1a-Y2a)/(X1a_max-X2a); // [kmol oil/kmol air]
// Therfore
Lsa_min = S*Vsa; // [kmole oil/s]
Data_minSlope1 = [X2a Y2a;X1a_max Y1a];

// For Actual operating line, oil flow rate is twice the minimum
Lsa = 2*Lsa_min; // [kmole oil/s]
M_oil = 198; // [molecular weight of oil, gram/mole]

Wsa = Lsa*M_oil; // [mass flow rate of oil, kg/s]
// Using equation 3.47 to calculate the actual concentration of the liquid phase leaving the absorber
X1a = X2a + Vsa*(Y1a-Y2a)/Lsa; // [kmol benzene/kmol oil]
Data_opline1 = [X2a Y2a;X1a Y1a];

scf(1);
plot(Data_eqm(:,1),Data_eqm(:,2),Data_minSlope1(:,1),Data_minSlope1(:,2),Data_opline1(:,1),Data_opline1(:,2));
xgrid();
legend('Equilibrium line for absorber','Minimum Flow Rate Line for absorber','Operating Line for absorber');
xlabel("Xa, mole benzene/mole oil");
ylabel("Ya, mole benzene/mole air");

// Calculations for stripping section
Lss = Lsa;
X2s = X1a;
X1s = X2a;
Y1s = 0;
T = 373; // [K]
```

```
// Applying Raoult s law at this temperature gives us
// yis = 1.77*xis
// Yis/(1+Yis) = 1.77*Xis/(1+Xis)

// Equilibrium data
// Data_equm = [Xis Yis]
Data_equm = [0 0;0.05 0.092;0.1 0.192;0.15 0.3;0.2 0.418;0.25 0.548;0.3 0.691;0.35 0.848;0.4 1.023;0.45 1.219;0.5 1.439];

// Similar procedure as above is followed
// The operating line for minimum oil rate must be tangent to curve
// Therefore from the curve
Y2s_max = 1.175; // [kmol benzene/kmol steam]
S = (Y2s_max-Y1s)/(X2s-X1s); // [kmole oil/kmole steam]
Vss_min = Lss/S; // [kmole/s]
Vss = 1.5*Vss_min; // [kmole/s]
Mss = 18; // [molecular weight of steam, gram/mole]
Wss = Vss*Mss; // [kg steam/s]

Data_minSlope2 = [X1s Y1s;X2s Y2s_max];

Y2s_act = Y1s + Lss*(X2s-X1s)/Vss; // [kmol benzene/kmol steam]

Data_opline2 = [X1s Y1s;X2s Y2s_act];


scf(2);
plot(Data_equm(:,1),Data_equm(:,2),Data_minSlope2(:,1),Data_minSlope2(:,2),Data_opline2(:,1),Data_opline2(:,2));
xgrid();
legend('Equilibrium line for stripping','Minimum Flow Rate for stripping Line','Operating Line for
    stripping');
xlabel("Xa, mole benzene/mole oil");
ylabel("Ya, mole benzene/mole air");

printf("The oil circulation rate and steam rate
   required for the operation is %f kg/s %f kg steam
   /s respectively\n\n",Wsa,Wss);
```

Scilab code Exa 3.8 Adsorption of Nitrogen Dioxide on Silica Gel

```
clear;
clc;

// Illustration 3.8
// Page: 190

printf('Illustration 3.8 -  Page: 190\n\n');

// solution
//*****Data*****//
// 1-Nitrogen dioxide  2-air
T = 298; // [K]
P = 101.3; // [kPa]
y1 = 0.015;
V1 = 0.5; // [mass flow rate of the gas entering the
    adsorber, kg/s]
M1 = 46; // [gram/mole]
M2 = 29; // [gram/mole]
// Data_eqm1 = [P1 m] (where 'P1' is Partial
   pressure of NO2 in mm of Hg, 'm' is solid
   concentration in kg NO2/kg gel)
Data_eqm1 = [0 0;2 0.4;4 0.9;6 1.65;8 2.60;10
   3.65;12 4.85];
//*****//

```

```
Y1 = y1*M1/((1-y1)*M2); // [kg NO2/kg air]
// For 85% removal of the NO2,
Y2 = 0.15*Y1; // [kg NO2/kg air]
// Since the entering gel is free of NO2,
X2 = 0;
// The equilibrium data are converted to mass ratios
    as follows:
// Yi = P1/(760-P1)*46/29 (kg NO2/kg air)  Xi = m
   /100 (kg NO2/kg gel)
// Equilibrium data
// Data_eqm = [Xi*100 Yi*100]
for i = 1:7;
    Data_eqm(i,2) = Data_eqm1(i,1)*M1*100/((760-
       Data_eqm1(i,1))*M2);
    Data_eqm(i,1) = Data_eqm1(i,2);
end

//Data_eqm = [0 0;0.4 0.42;0.9 0.83;1.65 1.26;2.6
   1.69;3.65 2.11;4.85 2.54];

// The operating line for minimum slope is tangent
   to curve, from which we get
X1_max = 0.0375; // [kg NO2/kg gel]

wb1 = 1/(1+Y1);
Vs = V1*wb1; // [mass velocity of the air, kg/s]
Ls_min = Vs*(Y1-Y2)/(X1_max-X2); // [kg gel/s]
Data_minSlope = [X2 Y2;X1_max Y1]*100;
// Operating line
Ls = 2*Ls_min; // [kg gel/s]

X1 = X2 + Vs*(Y1-Y2)/Ls; // [kg NO2/kg gel]

scf(4);
plot(Data_eqm(:,1),Data_eqm(:,2),Data_minSlope(:,1),
   Data_minSlope(:,2));
xgrid();
legend('Equilibrium line ','Minimum Flow Rate Line')
```

```
;
xlabel("Xa*100, kg NO2/kg gel ");
ylabel("Ya*100, kh NO2/kg air");

printf("Mass flow rate of the and the composition of the gel leaving the absorber are %f kg/s and %f\n\n",Ls,X1);
```

Scilab code Exa 3.9 Cocurrent Adsorption of NO2 on Silica Gel

```
clear;
clc;

// Illustration 3.9
// Page: 194

printf('Illustration 3.9 -  Page: 194\n\n');

// solution
//*****Data*****//
// 1-Nitrogen dioxide  2-air
// From Example 3.8
Y1 = 0.0242; // [kg NO2/kg air]
Y2 = 0.0036; // [kg NO2/kg air]
Vs = 0.488; // [kg air/s]
M1 = 46; // [gram/mole]
M2 = 29; // [gram/mole]
// However here
X1 = 0;
// Data_eqm1 = [P1 m] (where 'P1' is Partial pressure of NO2 in mm of Hg, 'm' is solid concentration in kg NO2/kg gel)
Data_eqm1 = [0 0;2 0.4;4 0.9;6 1.65;8 2.60;10 3.65;12 4.85];

```

```
// The equilibrium data are converted to mass ratios as follows:
// Yi = P1/(760-P1)*46/29 (kg NO2/kg air)   Xi = m/100 (kg NO2/kg gel)
// Equilibrium data
// Data_eqm = [Xi*100 Yi*100]
for i = 1:7;
    Data_eqm(i,2) = Data_eqm1(i,1)*M1*100/((760-Data_eqm1(i,1))*M2);
    Data_eqm(i,1) = Data_eqm1(i,2);
end

// From the intersection of the minimum operating line and equilibrium curve
X2_max = 0.0034; // [kg NO2/kg gel]
S = (Y1-Y2)/(X1-X2_max); // [kg gel/kg air]
Ls_min = -S*Vs; // [kg/s]

Ls = 2*Ls_min; // [kg/s]
Data_minSlope = [X1 Y1;X2_max Y2]*100;


scf(4);
plot(Data_eqm(:,1),Data_eqm(:,2),Data_minSlope(:,1),Data_minSlope(:,2));
xgrid();
legend('Equilibrium line ','Minimum Flow Rate Line');
xlabel("Xa*100, kg NO2/kg gel");
ylabel("Ya*100, kh NO2/kg air");

printf("The mass velocity of the silica gel required for cocurrent operation is %f kg/s which is 11 times that required for countercurrent operation\n\n",Ls);
```

Scilab code Exa 3.10 Benzene Recovery System Number of Ideal Stages

```
clear;
clc;

// Illustration 3.10
// Page: 199

printf('Illustration 3.10 -  Page: 199\n\n');

// solution
//*****Data*****//
// From Example 3.7
X2a = 0.05; X0 = X2a; // [kmole benzene/kmole oil]
Y2a = 0.012; Y1 = Y2a; // [kmole benzene/kmole dry gas]
X1a = 0.480; Xn = X1a; // [kmole benzene/kmole oil]
Y1a = 0.080; Yn1 = Y1a; // [kmole benzene/kmole dry gas]
// Ideal stages for absorber section

m = 0.097; // [mole of oil/mole of dry gas]
Lsa = 0.006; // [kmole/s]
Vsa = 0.038; // [kmole/s]
A = Lsa/(m*Vsa); // [Absorption factor]

// From equation 3.54 by Kremser equation
Nk = log((((Yn1-m*X0)*(1-1/A))/(Y1-m*X0))+1/A)/(log(A));
printf("Number of ideal stages from Kremser equation in the absorber is %f\n\n",Nk);

// Ideal stages from graph
// Stair case construction is being made between
```

```
equilibrium curve and operating line from piont X2a,Y2a to X1a,Y1a
// A more precise estimate of stages
// From figure 3.25 or from graph made for absorber in Example 3.7
Xa = 0.283;
Xb = 0.480;
Xc = 0.530;
Na = 3+(Xb -Xa)/(Xc-Xa);
printf("The number of ideal stages from graph in the absorber is %f\n\n",Na);

// Ideal satges for stripping section
X2s = 0.480; X0 = X2s; // [kmol benzene/kmol oil]
Y2s = 0.784; Y1 = Y2s; // [kmol benzene/kmol steam]
X1s = 0.05; Xn = X1s; // [kmol benzene/kmol oil]
Y1s = 0; Yn1 = Y1s; // [kmol benzene/kmol steam]

// Similarly here also stair case construction is being made between equilibrium curve and operating line from piont X0,Y1 to Xn,Yn1
// A more precise estimate of stages
// From figure 3.26 or from graph made for stripping section in Example 3.7
Ns = 5+(0.070-0.050)/(0.070-0.028);

printf("The number of ideal stages from graph in the stripping section is %f\n\n",Ns);
```

Chapter 4

Equipment for Gas Liquid Mass Transfer Operations

Scilab code Exa 4.2 Specific Liquid Holdup and Void Fraction in Second and Third Generation Random Packings

```
clear;
clc;

// Illustration 4.2
// Page: 227

printf('Illustration 4.2 -  Page: 227\n\n');

// solution
//*****Data*****//
u = 3*10^-6; // [Kinematic viscosity, square m/s]
v = 0.01; // [Superficial liquid velocity, m/s]
g = 9.8; // [square m/s]
//*****//
// From table 4.1
// For metal pall rings
a_pr = 112.6; // [ square m/cubic m]
e_pr = 0.951;
```

```
Ch_pr = 0.784;
// For Hiflow rings
a_hr = 92.3; // [square m/cubic m]
e_hr = 0.977;
Ch_hr = 0.876;

// Renoylds and Froude's number for metal pall rings
Rel_pr = v/(u*a_pr);
Frl_pr = v^2*a_pr/g;
// From equation 4.5 since Rel is greater than 5, for pall rings
// ah/a = x_pr
x_pr = 0.85*Ch_pr*Rel_pr^0.25*Frl_pr^0.1;
// From equation 4.3
hl_pr = (12*Frl_pr/Rel_pr)^(1/3)*(x_pr)^(2/3);


// Renoylds and Froude's number for Hiflow rings
Rel_hr = v/(u*a_hr);
Frl_hr = v^2*a_hr/g;
// From equation 4.5 since Rel is greater than 5, for pall rings
// ah/a = x_pr
x_hr = 0.85*Ch_hr*Rel_hr^0.25*Frl_hr^0.1;
// From equation 4.3
hl_hr = (12*Frl_hr/Rel_hr)^(1/3)*(x_hr)^(2/3);

printf("The specific liquid holdup for Metal pall ring and Hiflow ring are %f cubic m holdup/cubic m packed bed and %f cubic m holdup/cubic m packed bed respectively\n\n",hl_pr,hl_hr);
```

Scilab code Exa 4.3 Pressure Drop in Beds Packed with First and Third Generation Random Packings

```
clear;
clc;

// Illustration 4.3
// Page: 233

printf('Illustration 4.3 -  Page: 233\n\n');

// solution
//*****Data*****//
// a-ammonia   b-air   c-water
P = 101.3; // [kPa]
T = 293; // [K]
R = 8.314;
Vb = 20; // [kmole/h]
xab = 0.05;
Vc = 1500; // [kg/h]
d = 0.9; // [ammonia absorbed]
Ma = 17; // [gram/mole]
Mb = 29; // [gram/mole]
Mc = 18; // [gram/mole]
g = 9.8; // [square m/s]
//*****//

// For Inlet gas
Mg = (1-xab)*Mb+xab*Ma; // [gram/mole]
V = Vb*Mg/3600; // [kg/h]
rowg = P*Mg/(R*T); // [kg/cubic m]
Qg = V/rowg; // [cubic m/s]

// For exiting liquid
b = Vb*xab*Ma*d; // [ammonia absorbed in kg/h]
L = (Vc+b)/3600; // [kg/s]
rowl = 1000; // [kg/cubic m]

X = (L/V)*(sqrt(rowg/rowl));
// From equation 4.8
Yflood = exp(-(3.5021+1.028*log(X)+0.11093*(log(X))
   ^2));


printf('Illustration 4.3(a) - Page: 233\n\n');
// Solution(a)
// For 25-mm ceramic Raschig rings
Fp = 179; // [square ft/cubic ft]
ul = 0.001; // [Pa.s]
// From equation 4.6
Csflood = sqrt(Yflood/(ul^0.1*Fp)); // [m/s]
// From equation 4.7
vgf = Csflood/(sqrt(rowg/(rowl-rowg))); // [m/s]
// From equation 4.9
deltaPf = 93.9*(Fp)^0.7; // [Pa/m of packing]

// For operation at 70% of the flooding velocity
f = 0.7;
// From equation 4.10
vg = f*vgf; // [m/s]
D = sqrt(4*Qg/(vg*%pi));

// From Table 4.1, for 25 mm ceramic Raschig rings
a_c = 190; // [square m/cubic m]
Ch_c = 0.577;
e_c = 0.68;
Cp_c = 1.329;

// From equation 4.13
dp = 6*(1-e_c)/a_c; // [m]
// From equation 4.12
Kw = 1/(1+(2*dp/(3*D*(1-e_c))));

// The viscosity of the gas phase is basically that of air at 293 K and 1 atm
ug = 1.84*10^-5; // [kg/m.s]
// From equation 4.15
Reg = vg*rowg*dp*Kw/(ug*(1-e_c));
// From equation 4.14
```

```
sia_o = Cp_c*((64/Reg)+(1.8/(Reg^0.08)));

// From equation 4.11
// deltaP_o/z = T
T = sia_o*a_c*rowg*vg^2/(2*Kw*e_c^3); // [Pa/m]

// Now
Gx = L/(%pi*D^2/4); // [kg/square m.s]
Rel = Gx/(a_c*ul);
Frl = Gx^2*a_c/(rowl^2*g);

// From equation 4.5
// ah/a = x_pr
x = 0.85*Ch_c*Rel^0.25*Frl^0.1;
// From equation 4.3
hl = (12*Frl/Rel)^(1/3)*(x)^(2/3);

// From equation 4.16
// daltaP/deltaP_o = Y
Y = (e_c/(e_c-hl))^1.5*exp(Rel/200);
// Therefore
// deltaP/z = H
H = Y*T; // [Pa/m]

printf("The superficial velocity is %f m/s\n",vgf);
printf("The pressure drop at flooding is %f Pa/m\n",deltaPf);
printf("The superficial velocity at 70 percent of flooding is %f m/s\n",vg);
printf("The column inside diameter at 70 percent of flooding is %f m\n",D);
printf("The pressure drop for operation at 70 percent of flooding is %f Pa/m\n\n",H);


printf('Illustration 4.3(b) - Page: 236\n\n');
// Solution (b)
// Similarly for 25 mm metal Hiflow rings above
```

```
quantities are determined
Fp1 = 42; // [square ft/cubic ft]
Csflood1 = sqrt(Yflood/(ul^0.1*Fp1)); // [m/s]
vgf1 = Csflood1/(sqrt(rowg/(rowl-rowg))); // [m/s]
// From equation 4.9
deltaPf1 = 93.9*(Fp1)^0.7; // [Pa/m of packing]

// For operation at 70% of the flooding velocity
f = 0.7;
// From equation 4.10
vg1 = f*vgf1; // [m/s]
D1 = sqrt(4*Qg/(vg1*%pi));

// For Hiflow rings
a_h = 202.9; // [square m/cubic m]
e_h = 0.961;
Ch_h = 0.799;
Cp_h = 0.689;

// From equation 4.13
dp1 = 6*(1-e_h)/a_h; // [m]
// From equation 4.12
Kw1 = 1/(1+(2*dp1/(3*D1*(1-e_h))));

// The viscosity of the gas phase is basically that of air at 293 K and 1 atm
ug = 1.84*10^-5; // [kg/m.s]
// From equation 4.15
Reg1 = vg1*rowg*dp1*Kw1/(ug*(1-e_h));
// From equation 4.14
sia_o1 = Cp_h*((64/Reg1)+(1.8/(Reg1^0.08)));

// From equation 4.11
// deltaP_o/z = T
T1 = sia_o1*a_h*rowg*vg1^2/(2*Kw1*e_h^3); // [Pa/m]

// Now
Gx1 = L/(%pi*D1^2/4); // [kg/square m.s]
```

```
Rel1 = Gx1/(a_h*ul);
Frl1 = Gx1^2*a_h/(rowl^2*g);

// From equation 4.5
// ah/a = x_pr
x1 = 0.85*Ch_h*Rel1^0.25*Frl1^0.1;
// From equation 4.3
hl1 = (12*Frl1/Rel1)^(1/3)*(x1)^(2/3);

// From equation 4.16
// daltaP/deltaP_o = Y
Y1 = (e_h/(e_h-hl1))^1.5*exp(Rel1/200);
// Therefore
// deltaP/z = H
H1 = Y1*T1; // [Pa/m]


printf("The superficial velocity is %f m/s\n",vgf1);
printf("The pressure drop at flooding is %f Pa/m\n",deltaPf1);
printf("The superficial velocity at 70 percent of flooding is %f m/s\n",vg1);
printf("The column inside diameter at 70 percent of flooding is %f m\n",D1);
printf("The pressure drop for operation at 70 percent of flooding is %f Pa/m\n\n",H1);
```

Scilab code Exa 4.4 Design of a Packed Bed Ethanol Absorber

```
clear;
clc;

// Illustration 4.4
// Page: 237

```

```
printf('Illustration 4.4 - Page: 237\n\n');

// solution
//*****Data*****//
// a-ethanol   b- gas(CO2 rich vapor)   c-liquid water
P = 110; // [kPa]
T = 303; // [K]
R = 8.314;
Vb = 180; // [kmole/h]
xab = 0.02; // [molar composition of ethanol in gas]
Vc = 151.5; // [kmole/h]
d = 0.97; // [ethanol absorbed]
Ma = 46; // [gram/mole]
Mb = 44; // [gram/mole]
Mc = 18; // [gram/mole]
g = 9.8; // [square m/s]
//*****//

// For Inlet gas
Mg = (1-xab)*Mb+xab*Ma; // [gram/mole]
V = Vb*Mg/3600; // [kg/h]
rowg = P*Mg/(R*T); // [kg/cubic m]
Qg = V/rowg; // [cubic m/s]

// For exiting liquid
b = Vb*xab*Ma*d; // [ethanol absorbed in kg/h]
L = (Vc*Mc+b)/3600; // [kg/s]
rowl = 986; // [kg/cubic m]

X = (L/V)*(sqrt(rowg/rowl));
// From equation 4.8
Yflood = exp(-(3.5021+1.028*log(X)+0.11093*(log(X))^2));

printf('Illustration 4.4(a) - Page: 237\n\n');
// Solution(a)

// For 50 mm metal Hiflow rings
Fp = 16; // [square ft/cubic ft]
ul = 6.31*10^-4; // [Pa.s]
// From equation 4.6
Csflood = sqrt(Yflood/(ul^0.1*Fp)); // [m/s]
// From equation 4.7
vgf = Csflood/(sqrt(rowg/(rowl-rowg))); // [m/s]
// From equation 4.9
deltaPf = 93.9*(Fp)^0.7; // [Pa/m of packing]

// For operation at 70% of the flooding velocity
f = 0.7;
// From equation 4.10
vg = f*vgf; // [m/s]
D = sqrt(4*Qg/(vg*%pi));

// From Table 4.1, for 50 mm metal Hiflow rings
a = 92.3; // [square m/cubic m]
Ch = 0.876;
e = 0.977;
Cp = 0.421;

// From equation 4.13
dp = 6*(1-e)/a; // [m]

// From equation 4.12
Kw = 1/(1+(2*dp/(3*D*(1-e))));

// The viscosity of the gas phase is basically that
   of air at 303 K and 110 kPa
ug = 1.45*10^-5; // [kg/m.s]
// From equation 4.15
Reg = vg*rowg*dp*Kw/(ug*(1-e));
// From equation 4.14
sia_o = Cp*((64/Reg)+(1.8/(Reg^0.08)));

// From equation 4.11
// deltaP_o/z = I
```

```
I = sia_o*a*rowg*vg^2/(2*Kw*e^3); // [Pa/m]

// Now
Gx = L/(%pi*D^2/4); // [kg/square m.s]
Rel = Gx/(a*ul);
Frl = Gx^2*a/(rowl^2*g);

// From equation 4.5
// ah/a = x
x = 0.85*Ch*Rel^0.25*Frl^0.1;
// From equation 4.3
hl = (12*Frl/Rel)^(1/3)*(x)^(2/3);

// From equation 4.16
// daltaP/deltaP_o = Y
Y = (e/(e-hl))^1.5*exp(Rel/200);
// Therefore
// deltaP/z = H
H = Y*I; // [Pa/m]

printf('Since the pressure drop is too high, we must increase the tower diameter to reduce the pressure drop.\n');
// The resulting pressure drop is too high; therefore, we must increase the tower diameter to reduce the pressure drop. Appendix D presents a Mathcad computer
// program designed to iterate automatically until the pressure drop criterion is satisfied.
// From the Mathcad program we get
D1 = 0.738; // [m]
printf("The tower diameter for pressure drop of 300 Pa/m of packed height is %f m\n\n",D1);

printf('Illustration 4.4(b) - Page: 241\n\n');
// Solution(b)

// For the tower diameter of D = 0.738 m, the
```

```
   following intermediate results were obtained from
    the computer program in Appendix D:
vg1 = 2.68; // [m/s]
vl1 = 0.00193; // [m/s]
hl1 = 0.017;
ah1 = 58.8; // [square m/cubic m]
Reg1 = 21890;
Rel1 = 32.6;
Kw1 = 1/(1+(2*dp/(3*D1*(1-e))));


f1 = vg1/vgf;
printf(”The fractional approach to flooding
   conditions is %f\n\n”,f1);

printf('Illustration 4.4(c) – Page: 242\n\n');
// Solution(c)
// For ethanol
Vc_a = 167.1; // [cubic cm/mole]
sigma_a = 4.53*10^-10; // [m]
// E/k = M
M_a = 362.6; // [K]

// For carbon dioxide
sigma_b = 3.94*10^-10; // [m]
M_b = 195.2; // [K]

// From equation 1.48
Vb_a = 0.285*Vc_a^1.048; // [cubic cm/mole]

e1 = (9.58/(Vb_a)-1.12);
// From equation 1.53
Dl = 1.25*10^-8*((Vb_a)^-0.19 - 0.292)*T^1.52*(ul
   *10^3)^e1; // [square cm/s]

// From equation 1.49
Dg = 0.085; // [square cm/s]

```

```
// From Table 4.2, for 50 mm metal Hiflow rings
Cl = 1.168
Cv = 0.408;
// From equation 4.17
kl = 0.757*Cl*sqrt(Dl*a*vl1*10^-4/(e*hl1)); // [m/s]
mtcl = kl*ah1; // [s^-1]

Sc = ug/(rowg*Dg*10^-4);
// From equation 4.18
ky = 0.1304*Cv*(Dg*10^-4*P*1000/(R*T))*(Reg1/Kw1)^(3/4)*Sc^(2/3)*(a/(sqrt(e*(e-hl1)))); // [mole/square m.s]
mtcg = ky*ah1*10^-3; // [kmole/cubic m.s]
printf("The gas and liquid volumetric mass transfer coefficients are %e kmole/cubic m.s and %e s^-1 respectively.\n\n",mtcg,mtcl);
```

Scilab code Exa 4.5 Stripping Chloroform from Water by Sparging with Air

```
clear;
clc;

// Illustration 4.5
// Page: 245

printf('Illustration 4.5 -  Page: 245\n\n');

// solution
//*****Data*****//
//  a-chloroform     b-water     c-air
T = 298; // [K]
Dv = 1; // [vessel diameter, m]
Vb = 10; // [kg/s]
ca = 240*10^-6; // [gram/l]
```

```
xr = 0.9; // [chloroform which is to be removed]
m = 220;
Ds = 0.5; // [diameter of sparger , m]
no = 90; // [number of orifices]
Do = 3*10^-3; // [diameter of orifice , m]
nm = 0.6; // [mechanical efficiency]
rowb = 1000; // [kg/cubic m]
R = 8.314;
Mc = 29; // [gram/mole]
Mb = 18; // [gram/mole]
g = 9.8; // [square m/s]
//*****//

Vair = 0.1; // [kg/s as calculated in chapter 3]
mg = Vair/no; // [mass flow rate through each orifice , kg/s]
ug = 1.8*10^-5; // [kg/m.s]
Reo = 25940; // [Renoylds number]
// From equ. 4.20
dp = 0.0071*Reo^-0.05; // [m]

// Since the water column height is not known , therefore an iterative procedure must be implemented.
// Assuming column height , Z = 0.5 m
Z = 0.5; // [m]
// For Z = 0.5 m
rowl = 1000; // [kg/cubic m]
Ps = 101.3; // [kPa]
Po = Ps +  (1000*9.8*0.5/1000); // [kPa]
Pavg = (Po+Ps)/2; // [kPa]
rowg = Pavg*Mc/(R*T); // [kg/cubic m]

area = %pi*Dv^2/4; // [square m]
vg = Vair/(rowg*area); // [m/s]
// In this case rowl = rowg and sigma = sigmaAW
// From equation 4.22
// Vg = vg
// vg/vs = 0.182
vs = vg/0.182; // [m/s]
vl = -Vb/(rowl*area); // [negative because water flows downward, m/s]
// From equ 4.21

deff('[y] = f12(phig)','y = vs - (vg/phig)-(-vl/(1-phig))');
phig = fsolve(0.1,f12);
// Now in this case
S = vl/(1-phig);
// Value of 'S' comes out to be less than 0.15 m/s
// Therefore
dp = (dp^3*Po/Pavg)^(1/3); // [m]
// From equ 4.23
a = 6*phig/dp; // [m^-1]
// Now we calculate diffusivity of chloroform
Vba = 88.6; // [cubic cm/mole]
u = 0.9*10^-3; // [Pa-s]
e = (9.58/(Vba)-1.12);
// From equation 1.53
Dl = 1.25*10^-8*((Vba)^-0.19 - 0.292)*T^1.52*(u*10^3)^e; // [square cm/s]

// And Schmidt number is
Scl = 833; // [Schmidt Number]

// Now we calculate  dp*g^(1/3)/Dl^(2/3) = J
J = dp*g^(1/3)/(Dl*10^-4)^(2/3)
Reg = dp*vs*rowl/u; // [Gas bubble Renoylds number]
// From equ 4.25
Shl = 2 + 0.0187*Reg^0.779*Scl^0.546*J^0.116;

// For dilute solution xbm = 1 or c = 55.5 kmole/cubic m
// Then for Nb = 0
c = 55.5; // [kmole/cubic m]
kx = Shl*c*Dl*10^-4/dp; // [kmole/square m.s]
```

```
mtc = kx*a; // [kmole/cubic m.s]

L = Vb/Mb; // [kmole/s]
Gmx = L/area; // [kmole/square m.s]
V = Vair/Mc; // [kmole/s]
A = L/(m*V); // [absorption factor]

// From equ 4.28
 // For, xin/xout = x = 10
 x = 10;
Z = (Gmx/(kx*a*(1-A)))*log(x*(1-A)+A);

// With this new estimated Z ,we again calculate average pressure in the    // column of water
Po1 = 110.1; // [kPa]
Pavg1 = 105.7; // [kPa]
rowg1 = Pavg1*Mc/(R*T);
// Now value of rowg1 obtained is very close to value used in the first    // iteration. Therefore on three iteractions we achieve a value of 'Z'
Z1 = 0.904; // [m]

rowgo = Po1*Mc/(R*T); // [kg/cubic m]
vo1 = 4*mg/(%pi*Do^2*rowgo); // [m/s]
// Therefore,    vo1^2/(2*gc) = F
gc = 1;
F = vo1^2/(2*gc); // [J/kg]
// And            R*T*log(Po/Ps)/Mc = G
G = R*T*1000*log(Po1/Ps)/Mc; // [J/kg]
Zs = 0
// And   (Z1-Zs)*g/gc = H
H = (Z1-Zs)*g/gc; // [J/kg]
// From equ 4.27
W = F+G+H; // [J/kg]
// Now the air compressor power is
W1 = W*Vair*10^-3/nm; // [kW]

printf("The depth of the water column required to
   achieve the specified 90percent removal
   efficiency is %f m\n\n",Z1);
printf("The power required to operate the air
   compressor is %f kW\n\n",W1);
```

Scilab code Exa 4.6 Design of a Sieve Tray Column for Ethanol Absorption

```
clear;
clc;

// Illustration 4.6
// Page: 255

printf('Illustration 4.6 -  Page: 255\n\n');

// solution
//*****Data*****//
Ff = 0.9; // [foaming factor]
sigma = 70; // [liquid surface tension, dyn/cm]
Do = 5; // [mm]
//From Example 4.4
// X = 0.016;
p = 15 // [pitch, mm]
// From equ 4.35
// Ah/Aa = A
A = 0.907*(Do/p)^2; // [ratio of vapor hole area to
   tray active area]

// Assume
t = 0.5; // [m]
// From equ 4.32
alpha = 0.0744*t+0.01173;
beeta = 0.0304*t+0.015;

```

```
// Since X<0.1, therefore
X = 0.1;
// From equ 4.31
Cf = alpha*log10(1/X) + beeta;
// Since Ah/Aa > 0.1, therefore
Fha = 1;
Fst = (sigma/20)^0.2; // [surface tension factor]
// From equ 4.30
C = Fst*Ff*Fha*Cf;

// From Example 4.4
rowg = 1.923; // [kg/cubic m]
rowl = 986; // [kg/cubic m]
Qg = 1.145; // [cubic m/s]
// From equation 4.29
vgf = C*(sqrt((rowl-rowg)/rowg)); // [m/s]
// Since X<0.1
// Equ 4.34 recommends Ad/At = B = 0.1
B = 0.1;
// For an 80% approach to flooding, equation 4.33 yields
f = 0.8;
D = sqrt((4*Qg)/(f*vgf*%pi*(1-B))); // [m]
// At this point, the assumed value of tray spacing ( t = 0.5 m) must be   // checked against the recommended values of Table 4.3. Since the calculated
// value of D < 1.0 m, t = 0.5 m is the recommended tray spacing, and no
// further iteration is needed.

deff('[y] = f14(Q)','y = B-((Q-sin(Q))/(2*%pi))');
Q = fsolve(1.5,f14);
Lw = D*sin(Q/2); // [m]
rw = D/2*cos(Q/2); // [m]

At = %pi/4*D^2; // [total cross sectional area, square m]
```

```
Ad = B*At; // [Downcomer area, square m]
Aa = At-2*Ad; // [ Active area over the tray, square m]
Ah = 0.101*Aa; // [Total hole area, square m]

printf('Summarizing, the details of the sieve-tray design are as follows:\n\n');
printf(" Diameter = %f m\n Tray spacing = %f m\n Total cross-sectional area  = %f square m\n Downcomer area = %f square m\n Active area over the tray = %f square m\n Weir length = %f m\n Distance from tray center to weir = %f m\n Total hole area = %f square m\n Hole arrangement: 5 mm diameter on an equilateral-triangular pitch 15 mm between hole centers, punched in stainless steel sheet metal 2 mm thick\n\n",D,t,At,Ad,Aa,Lw,rw,Ah);
```

Scilab code Exa 4.7 Gas Pressure Drop in a Sieve Tray Ethanol Absorber

```
clear;
clc;

// Illustration 4.7
// Page: 257

printf('Illustration 4.7 -  Page: 257\n\n');

// solution//
Do = 5; // [mm]
g = 9.8; // [square m/s]
hw = 50; // [mm]
// From example 4.4
Qg = 1.145; // [cubic m/s]
// From example 4.6
```

```
Ah = 0.062; // [square m]
// Do/l = t = 5/2 = 2.5
t = 2.5;
// Ah/Aa = A = 0.101
A = 0.101;
rowg = 1.923; // [kg/cubic m]
rowl = 986; // [kg/cubic m]
roww = 995; // [kg/cubic m]

vo = Qg/Ah; // [m/s]
// From equation 4.39
Co = 0.85032 - 0.04231*t + 0.0017954*t^2; // [for t >=1]
// From equation 4.38
hd = 0.0051*(vo/Co)^2*rowg*(roww/rowl)*(1-A^2); // [cm]

// From example 4.6
Aa = 0.615; // [square m]
va = Qg/Aa; // [m/s]

// From equation 4.41
Ks = va*sqrt(rowg/(rowl-rowg)); // [m/s]
phie = 0.274;

// From equation 4.4
ql = 0.000815; // [cubic m/s]

// From example 4.6
Lw = 0.719; // [m]
Cl = 50.12 + 43.89*exp(-1.378*hw);
sigma = 0.07; // [N/m]
// From eqution 4.40
hl = phie*(hw*10^-1+Cl*(ql/(Lw*phie))^(2/3));

// From equation 4.42
ho = 6*sigma/(g*rowl*Do*10^-3)*10^2; // [cm]
// From equation 4.37
```

```
ht = hd+hl+ho; // [cm of clear liquid/tray]
deltaPg = ht*g*rowl*10^-2; // [Pa/tray]
printf("The tray gas−pressure drop for the ethanol
    is %f Pa/tray\n\n",deltaPg);
```

Scilab code Exa 4.8 Weeping and Entrainment in a Sieve Tray Ethanol Absorber

```
clear;
clc;

// Illustration 4.8
// Page: 259

printf('Illustration 4.8 −  Page: 259\n\n');

// solution//
// From Example 4.4, 4.6 and 4.7

Do = 5*10^-3; // [m]
rowg = 1.923; // [kg/cubic m]
rowl = 986; // [kg/cubic m]
g = 9.8; // [square m/s]
hl = 0.0173; // [m]
vo = 18.48; // [m/s]
phie = 0.274;
Ks = 0.082; // [m]
A = 0.101; // [Ah/Aa]
t = 0.5; // [m]

Fr = sqrt(rowg*vo^2/(rowl*g*hl)); // [Froude number]
if(Fr>=0.5)
    printf('Weeping is not significant\n\n');
else()
    printf('Significant weeping occurs\n\n');
    end
// From above weeping is not a problem under this circumstances
// From equation 4.47
k = 0.5*(1-tanh(1.3*log(hl/Do)-0.15));

// From equation 4.46
h2q = (hl/phie) + 7.79*(1+6.9*(Do/hl)^1.85)*(Ks^2/(phie*g*A)); // [m]
// From equation 4.45
E = 0.00335*(h2q/t)^1.1*(rowl/rowg)^0.5*(hl/h2q)^k;
// From Example 4.4, the gas mass flow rate is V = 2.202 kg/s
V = 2.202; // [kg/s]
Le = E*V; // [kg/s]
printf("The entrainment flow rate for the ethanol absorber is %f m/s\n\n",Le);
```

Scilab code Exa 4.9 Murphree Efficiency of a Sieve Tray Ethanol Absorber

```
clear;
clc;

// Illustration 4.9
// Page: 264

printf('Illustration 4.9 - Page: 264\n\n');

// solution//
// From examples 4.4, 4.6 and 4.7

Do = 5*10^-3; // [m]
Ml = 18.63; // [molecular weight of water, gram/mole]
```

```
Mg = 44.04; // [molecular weight of carbon dioxide, gram/mole]
rowg = 1.923; // [kg/cubic m]
rowl = 986; // [kg/cubic m]
vo = 18.48; // [m/s]
hl = 0.0173; // [m]
ug = 1.45*10^-5; // [kg/m.s]
phie = 0.274;
A = 0.101; // [Ah/Aa]
Dg = 0.085; // [square cm/s]
Dl = 1.91*10^-5; // [square cm/s]
Aa = 0.614; // [square m]
Qg = 1.145; // [cubic m/s]
t = 0.5; // [m]
h2q = 0.391; // [m]
rw = 0.34; // [m]
ql = 0.000815; // [cubic m/s]
g = 9.8; // [square m/s]
G = 2.202/44.04; // [kg/s]
L = 0.804/18.63; // [kg/s]

Refe = rowg*vo*hl/(ug*phie);

cg =rowg/Mg; // [kmole/cubic m]
cl = rowl/Ml; // [kmole/cubic m]

// For the low concentrations prevailing in the liquid phase, the ethanol-    // water solution at 303 K obeys Henry's law, and the slope of the equilibriu// m curve is m = 0.57
m = 0.57;
// From equation 4.53
a1 = 0.4136;
a2 = 0.6074;
a3 = -0.3195;
Eog = 1-exp(-0.0029*Refe^a1*(hl/Do)^a2*A^a3/((sqrt(Dg*(1-phie)/(Dl*A)))*m*cg/cl+1));
// From equation 4.62
```

```
Deg = 0.01; // [square m/s]
Peg = 4*Qg*rw^2/(Aa*Deg*(t-h2q)); // [Peclet number]
// Since Peclet number is greater than 50, therefore vapor is unmixed
// From equation 4.60
Del = 0.1*sqrt(g*h2q^3); // [square m/s]
// From equation 4.59
Pel = 4*ql*rw^2/(Aa*hl*Del);
N = (Pel+2)/2;
lambda = m*G/L;
// From equation 4.58
Emg = ((1+lambda*Eog/N)^N -1)/lambda*(1-0.0335*lambda^1.073*Eog^2.518*Pel^0.175);
// From example 4.8
E = 0.05;
// Substituting in equation 4.63
Emge = Emg*(1-0.8*Eog*lambda^1.543*E/m);
printf("The entrainment corrected Murphree tray efficiency for the ethanol is %f.\n\n",Emge);
```

Chapter 5

Absorption and Stripping

Scilab code Exa 5.1 Number of Real Sieve Trays in an Absorber

```
clear;
clc;

// Illustration 5.1
// Page: 287

printf('Illustration 5.1 -  Page: 287\n\n');

// solution

//*****Data*****//
// Component 'A' is to be absorbed //
y_N1 = 0.018; // [mole fraction 'A' of in entering gas]
y_1 = 0.001; // [mole fractio of 'A'in leaving gas]
x_0 = 0.0001; // [mole fraction of 'A' in entering liquid]
m = 1.41; // [m = yi/xi]
n_1 = 2.115; // [molar liquid to gas ratio at bottom, L/V]
n_2 = 2.326; // [molar liquid to gas ratio at top, L
```

```
/V]
E_MGE = 0.65;
//*****//

printf('Illustration 5.1 (a) - Page: 287\n\n');
// Solution (a)

A_1 = n_1/m; // [absorption factor at bottom]
A_2 = n_2/m; // [absorption factor at top]

A = sqrt(A_1*A_2);
// Using equation 5.3 to calculate number of ideal stages
N = (log(((y_N1-m*x_0)/(y_1-m*x_0))*(1-1/A) + 1/A))/log(A); // [number of ideal stages]
printf("Number of ideal trays is %f\n",N);
// Using equation 5.5
E_o = log(1+E_MGE*(1/A-1))/log(1/A);
// Therefore number of real trays will be
n = N/E_o;
printf("Number of real trays is %f\n",n);
n = 8;
printf("Since it is not possible to specify a fractional number of trays, therefore number of real trays is %f\n\n",n);

printf('Illustration 5.1 (b) - Page: 287\n\n');

// Solution (b)

// Back checking the answer
printf('Back checking the answer');
N_o = E_o*n;
// Putting N_o in equation 5.3 to calculate y_1
deff('[y] = f16(Z)','y=N_o-(log(((y_N1-m*x_0)/(Z-m*x_0))*(1-1/A) + 1/A))/log(A)');
Z = fsolve(0.001,f16);
printf("Mole fraction of A in leaving gas is %f
   percent which satisfies the requirement that the
   gas exit concentration should not exceed 0.1
   percent.",Z);

// For a tower diameter of 1.5 m, Table 4.3
   recommends a plate spacing of 0.6 m
Z = n*0.6; // [Tower height, m]
printf("The tower height will be %f m",Z);
```

Scilab code Exa 5.3 Packed Tower Absorber for Recovery of Benzene Vapors

```
clear
clc;

// Illustration 5.3
// Page: 295

printf('Illustration 5.3 -  Page: 295\n\n');

// solution
// For tower diameter, packed tower design program
   of Appendix D is run using // the data from
   Example 5.2 and packing parameters from Chapter
   4.

// For a pressure drop of 300 Pa/m, the program
   converges to a tower diameter
Db = 0.641; // [m]
// Results at the bottom of tower
fb= 0.733; // [flooding]
ahb = 73.52; // [m^-1]
Gmyb = 126; // [mol/square m.s]
kyb = 3.417; // [mol/square m.s]
klb = 9.74*10^-5; // [m/s]
```

```

// From equation 2.6 and 2.11
// Fg = ky*(1-y),   Fl = kx*(1-x)
// Assume 1-y = 1-y1    1-x = 1-x1
// let t = 1-y1  u = 1-x1
// Therefore
t = 0.926;
u = 0.676;
Fgb = kyb*t; // [mol/square m.s]
rowlb = 780; // [kg/cubic m]
Mlb = 159.12; // [gram/mole]
c = rowlb/Mlb; // [kmle/cubic m]
Flb = klb*c*u; // [mol/square m.s]
// From equ 5.19
Htgb = Gmyb/(Fgb*ahb); // [m]

// Now, we consider the conditions at the top of the absorber
// For a pressure drop of 228 Pa/m, the program converges to a tower        // diameter
Dt = 0.641; // [m]
// Results at the top of tower
ft = 0.668; // [flooding]
aht = 63.31; // [m^-1]
Gmyt = 118; // [mol/square m.s]
kyt = 3.204; // [mol/square m.s]
klt = 8.72*10^-5; // [m/s]

rowlt = 765; // [kg/cubic m]
Mlt = 192.7; // [gram/mole]
cl = rowlt/Mlt; // [kmole/cubic m]
Fgt = kyt*0.99; // [mole/square m.s]
Flt = klb*cl*0.953; // [mole/square m.s]
// From equ 5.19
Htgt = Gmyt/(Fgt*aht); // [m]
Htg_avg = (Htgb+Htgt)/2; // [m]
Fg_avg = (Fgt+Fgb)/2; // [mole/square m.s]
Fl_avg = (Flb+Flt)*1000/2; // [mole/square m.s]
```

```

// The operating curve equation for this system in terms of mole fractions
// y =

// From Mathcad program figure 5.3
x1 = 0.324;
x2 = 0.0476;
n = 50;
dx = (x1-x2)/n;
me = 0.136;
T = zeros(50,2);
for j=1:50
    x(j) = x2+j*dx;
    y(j) = (0.004+0.154*x(j))/(1.004-0.846*x(j));

    deff('[y] = f12(yint)','y = (1-yint)/(1-y(j)) - ((1-x(j))/(1-yint/me))^(Fl_avg/Fg_avg)');
    yint(j) = fsolve(0.03,f12);
    f(j) = 1/(y(j)-yint(j));
    T(j,1) = y(j);
    T(j,2) = f(j);
end

scf(1);
plot(T(:,1),T(:,2));
xgrid();
xlabel("y");
ylabel("f = 1/(y-yint)");

yo = y(1);
yn = y(50);
// From graph between f vs y
Ntg = 10.612;
// Therefore
Z = Htg_avg*Ntg; // [m]
printf("The total packed height is %f m.\n\n",Z);
deltaPg = 300*Z; // [Pa]
```

```
Em = 0.60; // [mechanical efficiency]
Qg = 1.0;
Wg = (Qg*deltaPg)/Em; // [Power required to force the gas through the tower, W]
L2 = 1.214; // [kg/s]
g = 9.8; // [m/square s]
Wl = L2*g*Z/Em; // [Power required to pump the liquid to the top of the absorber, W]
printf(”The power required to force the gas through the tower is %f W.\n\n”,Wg);
printf(”The power required to pump the liquid to the top of the absorber is %f W.\n\n”,Wl);
```

Scilab code Exa 5.4 Packed Height of an Ethanol Absorber

```
clear
clc;

// Illustration 5.4
// Page: 299

printf('Illustration 5.4 - Page: 299\n\n');

// solution
// Fro example 4.4
m = 0.57;
D = 0.738; // [tower diameter, m]
G = 180; // [rate of gas entering the tower, kmole/h]
L = 151.5; // [rate of liquid leaving the tower, kmole/h]
// Amount of ethanol absorbed
M = G*0.02*0.97; // [kmole/h]
//*****//

```

```
// Inlet gas molar velocity
Gmy1 = G*4/(3600*%pi*D^2); // [kmole/square m.s]
// Outlet gas velocity
Gmy2 = (G-M)*4/(3600*%pi*D^2); // [kmole/square m.s]
// Average molar gas velocity
Gmy = (Gmy1+Gmy2)/2; // [kmole/square m.s]

// Inlet liquid molar velocity
Gmx2 = L*4/(3600*%pi*D^2); // [kmole/square m.s]
// Outlet liquid molar velocity
Gmx1 = (L+M)*4/(3600*%pi*D^2); // [kmole/square m.s]

// Absorption factor at both ends of the column:
A1 = Gmx1/(m*Gmy1);
A2 = Gmx2/(m*Gmy2);
// Geometric average
A = sqrt(A1*A2);

y1 = 0.02;
// For 97% removal of the ethanol
y2 = 0.03*0.02;
// Since pure water is used
x2 = 0;
// From equation 5.24
Ntog = log((y1-m*x2)/(y2-m*x2)*(1-1/A)+1/A)/(1-1/A);

// From example 4.4
// ky*ah = 0.191 kmole/cubic m.s
// kl*ah = 0.00733 s^-1
kyah = 0.191; // [kmole/cubic m.s]
klah = 0.00733; // [s^-1]
rowl = 986; // [kg/cubic m]
Ml = 18; // [gram/mole]
c = rowl/Ml; // [kmole/cubic m]
kxah = klah*c; // [kmole/cubic m.s]

// Overall volumetric mass transfer coefficient
Kyah = (kyah^-1 + m/kxah)^-1; // [kmole/cubic m.s]
```

```

// From equation 5.22
Htog = Gmy/Kyah; // [m]
// The packed height is given by equation 5.21,
Z = Htog*Ntog; // [m]
printf(”The packed height of an ethanol absorber is %f m.\n\n”,Z);
```

Scilab code Exa 5.5 Tray Tower for Adiabatic Pentane Absorption

```
clear
clc;

// Illustration 5.5
// Page: 302

printf('Illustration 5.5 – Page: 302\n\n');

// solution

//*****Data*****//
;// a = CH4 b = C5H12
Tempg = 27;// [OC]
Tempo = 0;// [base temp,OC]
Templ = 35;// [OC]
xa = 0.75;// [mole fraction of CH4 in gas]
xb = 0.25;// [mole fraction of C5H12 in gas]
M_Paraffin = 200;// [kg/kmol]
hb = 1.884;// [kJ/kg K]
//********//

Ha = 35.59;// [kJ/kmol K]
Hbv = 119.75;// [kJ/kmol K]
Hbl = 117.53;// [kJ/kmol K]
Lb = 27820;// [kJ/kmol]
```

```
// M = [Temp (OC) m]
M = [20 0.575;25 0.69;30 0.81;35 0.95;40 1.10;43
   1.25];
// Basis: Unit time
GNpPlus1 = 1;// [kmol]
yNpPlus1 = 0.25;// [kmol]
HgNpPlus1 = ((1-yNpPlus1)*Ha*(Tempg-Tempo))+(
   yNpPlus1*(Hbv*(Tempg-Tempo)+Lb));// [kJ/kmol]
L0 = 2;// [kmol]
x0 = 0;// [kmol]
HL0 = ((1-x0)*hb*M_Paraffin*(Templ-Tempo))+(x0*hb*(
   Templ-Tempo));// [kJ/kmol]
C5H12_absorbed = 0.98*xb;// [kmol]
C5H12_remained = xb-C5H12_absorbed;
G1 = xa+C5H12_remained;// [kmol]
y1 = C5H12_remained/G1;// [kmol]
LNp = L0+C5H12_absorbed;// [kmol]
xNp = C5H12_absorbed/LNp;// [kmol]
// Assume:
Temp1 = 35.6;// [OC]
Hg1 = ((1-y1)*Ha*(Temp1-Tempo))+(y1*(Hbv*(Temp1-
   Tempo)+Lb));// [kJ/kmol]


Qt = 0;
deff('[y] = f30(HlNp)','y = ((L0*HL0)+(GNpPlus1*
   HgNpPlus1))-((LNp*HlNp)+(G1*Hg1)+Qt)');
HlNp = fsolve(2,f30);

deff('[y] = f31(TempNp)','y = HlNp-(((1-x0)*hb*
   M_Paraffin*(TempNp-Tempo))+(x0*hb*(TempNp-Tempo))
   )');
TempNp = fsolve(35.6,f31);
// At Temp = TempNp:
mNp = 1.21;
yNp = mNp*xNp;// [kmol]
GNp = G1/(1-yNp);// [kmol]
HgNp = ((1-yNp)*Ha*(TempNp-Tempo))+(yNp*(Hbv*(TempNp
    -Tempo)+Lb));// [kJ/kmol]
// From equation 5.28 with n = Np-1
deff('[y] = f32(LNpMinus1)','y = LNpMinus1+GNpPlus1
    -(LNp+GNp)');
LNpMinus1 = fsolve(2,f32);// [kmol]

// From equation 5.29 with n = Np-1
deff('[y] = f33(xNpMinus1)','y = ((LNpMinus1*
    xNpMinus1)+(GNpPlus1*yNpPlus1))-((LNp*xNp)+(GNp*
    yNp))');
xNpMinus1 = fsolve(0,f33);// [kmol]

// From equation 5.30 with n = Np-1
deff('[y] = f34(HlNpMinus1)','y = ((LNpMinus1*
    HlNpMinus1)+(GNpPlus1*HgNpPlus1))-((LNp*HlNp)+(
    GNp*HgNp))');
HlNpMinus1 = fsolve(0,f34);// [kJ/kmol]
deff('[y] = f35(TempNpMinus1)','y = HlNpMinus1-(((1-
    xNpMinus1)*hb*M_Paraffin*(TempNpMinus1-Tempo))+(
    xNpMinus1*hb*(TempNpMinus1-Tempo)))');
TempNpMinus1 = fsolve(42,f35);// [OC]

// The computation are continued upward through the
    tower in this manner until the gas composition
    falls atleast to 0.00662.
// Results = [Tray No.(n) Tn(OC) xn yn]
Results = [4.0 42.3 0.1091 0.1320;3 39.0 0.0521
    0.0568;2 36.8 0.0184 0.01875;1 35.5 0.00463
    0.00450];
scf(8);
plot(Results(:,1),Results(:,4));
xgrid();
xlabel('Tray Number');
ylabel('mole fraction of C5H12 in gas');

scf(9);
plot(Results(:,1),Results(:,2));
xgrid();
```

```
xlabel('Tray Number');
ylabel('Temparature(OC)');

// For the cquired y1
Np = 3.75;
printf("The No. of trays will be %f",Np);
```

Chapter 6

Distillation

Scilab code Exa 6.1 Flash Vaporization of a Heptan Octane Mixture

```
clear;
clc;

// Illustration 6.1
// Page: 324

printf('Illustration 6.1 -  Page: 324\n\n');

// solution
//*****Data*****//
//   n-heptane - a     n-octane - b
T1 = 303; // [K]
P = 1; // [bar]
D = 0.6;
W = 0.4;
zf = 0.5;

// Parameters for componenr 'A'
Tc_a = 540.3; // [K]
Pc_a = 27.4; // [bar]
A_a = -7.675;
```

```
B_a = 1.371;
C_a =-3.536;
D_a = -3.202;

// Parameters for component 'B'
Tc_b = 568.8; // [K]
Pc_b = 24.9; // [bar]
A_b = -7.912;
B_b = 1.380;
C_b = -3.804;
D_b = -4.501;

// Using equation 6.5
// x_a = 1-(T/Tc_a);
// P_a = Pc_a*exp((A_a*x_a+B_a*x_a^1.5+C_a*x_a^3+D_a*x_a^6)/(1-x_a)); // [bar]

// x_b = 1-(T/Tc_b);
// P_b = Pc_b*exp((A_b*x_b+B_b*x_b^1.5+C_b*x_b^3+D_b*x_b^6)/(1-x_b)); // [bar]

// m_a = P_a/P;
// m_b = P_b/P;

// Solution of simultaneous equation
function[f]=F(e)
    f(1) = e(2) - (e(3)*Pc_a*exp(((A_a*(1-(e(1)/Tc_a))+B_a*(1-(e(1)/Tc_a))^1.5+C_a*(1-(e(1)/Tc_a))^3+D_a*(1-(e(1)/Tc_a))^6))/(1-(1-(e(1)/Tc_a)))))/P;
    f(2) = 1-e(2) - ((1-e(3))*Pc_b*exp((A_b*(1-(e(1)/Tc_b))+B_b*(1-(e(1)/Tc_b))^1.5+C_b*(1-(e(1)/Tc_b))^3+D_b*(1-(e(1)/Tc_b))^6)/(1-(1-(e(1)/Tc_b)))))/P;
    f(3) = (-W/D) - ((e(2)-zf)/(e(3)-zf));
    funcprot(0);
endfunction

```

```
// Initial guess
e = [400 0.6 0.4];
y = fsolve(e,F);
T = y(1); // [K]
Yd = y(2);
Xw = y(3);

printf("The composition of the vapor and liquid and the temperature in the separator if it behaves as an ideal stage are %f, %f and %f K respectively\n\n",Yd,Xw,T);

// For the capculation of the amount of heat to be added per mole of feed
T0 = 298; // [K]
lambdaA = 36.5; // [Latent heats of vaporization at To = 298 K ,kJ/mole]
lambdaB = 41.4; // [Latent heats of vaporization at To = 298 K ,kJ/mole]
CpA = 0.187; // [kJ/mole.K]
CpB = 0.247; // [kJ/mole.K]
CLA1 = 0.218; // [ 298-303 K,  kJ/mole.K]
CLB1 = 0.253; // [ 298-303 K,  kJ/mole.K]
CLA2 = 0.241; // [ 298-386 K,  kJ/mole.K]
CLB2 = 0.268; // [ 298-386 K,  kJ/mole.K]
// Bubble point calculated when 'D' approaches 0 and Dew point calculated when 'D' approaches 1
Tbp = 382.2 // [Bubble point of the mixture, K]
Tdp = 387.9 // [Dew point of mixture, K]

HF = (T1-T0)*(Xw*CLA1+CLB1*(1-Xw)); // [kJ/mole]
HW = (Tbp-T0)*(Xw*CLA2+CLB2*(1-Xw)); // [kJ/mole]
HG = (Tdp-T0)*(Yd*CpA+(1-Yd)*CpB) + Yd*lambdaA +(1-Yd)*lambdaB; // [kJ/mole]

f =1 // [feed]
// Using equation 6.4
deff('[y] = f14(Q)','y = W/D + (HG-(HF+Q/f))/(HW -(
   HF+Q/f))');
Q = fsolve(40,f14);
printf("The amount of heat to be added per mole of
   feed is %f kJ/mole\n\n",Q);
```

Scilab code Exa 6.2 Flash Vaporization of a Ternary Mixture

```
clear;
clc;

// Illustration 6.2
// Page: 326

printf('Illustration 6.2 - Page: 326\n\n');

// solution
//*****Data*****//
// a-benzene  b-toluene  c-orthoxylene
T = 373; // [K]
P = 101.3; // [kPa]
Pa = 182.7; // [kPa]
Pb = 73.3; // [kPa]
Pc= 26.7; // [kPa]
Zfa = 0.5;
Zfb = 0.25;
Zfc = 0.25;
//*****//
// Therefore
ma = Pa/P;
mb = Pb/P;
mc = Pc/P;
// Let Feed is 1 kmole
// Therefore D+W = 1

// Solution of simultaneous equation
```

```
function[f]=F(e)
    f(1) = e(1)+e(2)-1;
    f(2) = e(2)/e(1) + (e(3)-Zfa)/(e(4)-Zfa);
    f(3) = e(3)-ma*e(4);
    f(4) = e(5)-mb*e(6);
    f(5) = 1-e(3)-e(5) -mc*(1-e(4)-e(6));
    f(6) = e(2)/e(1) + (e(5)-Zfb)/(e(6)-Zfb);
    funcprot(0);
endfunction

// Initial guess
e = [0.326 0.674 0.719 0.408 0.198 0.272];
y = fsolve(e,F);
D = y(1);
W = y(2);
Yad = y(3);
Xaw = y(4);
Ybd = y(5);
Xbw = y(6);
Ycd = 1-Yad-Ybd;
Xcw = 1-Xaw-Xbw;

printf("The amounts of liquid and vapor products are %f and %f respectively\n\n",D,W);
printf("The vapor compositions of components A, B and C are %f, %f and %f respectively\n\n",Yad,Ybd,Ycd);
printf("The liquid composition of components A, B and C are %f, %f and %f respectively\n\n",Xaw,Xbw,Xcw);
```

Scilab code Exa 6.3 Differential Distillation of a Heptane Octane Mixture

```
clear;
clc;
```

```

// Illustration 6.3
// Page: 328

printf('Illustration 6.3 -  Page: 328\n\n');

// solution
//*****Data*****//
//   n-heptane - a    n-octane - b
P = 1; // [bar]

// Basis:
F = 100; //  [mole]
// Therefore
D = 60; // [mole]
W = 40; // [mole]
xf = 0.5;
// Substituting in equation 6.11 yields
// log(F/W) = Integration of dx/(y_star-x) from xw to 0.50

// The equilibrium-distribution data for this system can be generated by calculating the liquid composition (x = xw) at the dew point (D = l.O). for different feed  // compositions (y_star = z).
y_star = [0.5 0.55 0.60 0.65 0.686 0.70 0.75];
x = [0.317 0.361 0.409 0.460 0.5 0.516 0.577];
for i = 1:7
    f(i) = 1/(y_star(i)-x(i));
end

area = [0.317 5.464;0.361 5.291;0.409 5.236;0.460 5.263;0.5 5.376;0.516 5.435;0.577 7.78];
// LHS of equation 6.11
a = log(F/W);

scf(4);
plot(area(:,1),area(:,2));
```

```
xgrid();
legend('area under curve');
xlabel("x");
ylabel("1/(y_satr-x)");

// When the area becomes equal to 0.916, integration is stopped; this occurs at
xw = 0.33; // [mole fraction of heptane in residue]
yd =( F*xf-W*xw)/D; // [mole fraction of heptane]
printf("The composition of the composited distillate and the residue are %f and %f respectively\n\n",yd,xw);
```

Scilab code Exa 6.4 Rectification of a Benzene Toluene Mixture

```
clear;
clc;

// Illustration 6.4
// Page: 342

printf('Illustration 6.4 - Page: 342\n\n');

// solution
//*****Data*****//
T = 298; // [K]
Fa = 200; // [feed, kmole/hr]
zf = 0.6;
yd = 0.95; xd = yd;
xw = 0.05;
q = 0.5; // [Lf/F]
//*****//

printf('Illustration 6.4(a) - Page: 342\n\n');
// Solution (a)

// Solution of simultaneous equation
function[f]=F(e)
    f(1) = Fa - e(1)-e(2);
    f(2) = zf*Fa - yd*e(1) - xw*e(2);
    funcprot(0);
endfunction

// Initial guess
e = [120 70];
y = fsolve(e,F);
D = y(1);
W = y(2);
printf("Quantity of liquid and vapor products are %f kmole/h and %f kmole/h respectively\n\n",D,W);


printf('Illustration 6.4(b) - Page: 342\n\n');
// Solution(b)
// VLE data is generated in the same manner as generated in Example 6.1 by applying Raoult's law
// VLE_data = [T,x,y]
VLE_data = [379.4 0.1 0.21;375.5 0.2 0.37;371.7 0.3 0.51;368.4 0.4 0.64;365.1 0.5 0.71;362.6 0.6 0.79;359.8 0.7 0.86;357.7 0.8 0.91;355.3 0.9 0.96];
// From figure 6.14
// The minimum number of equilibrium stages is stepped off between the equilibrium curve and the 45 degree line, starting from the top, giving
Nmin = 6.7;
printf("The minimum number of theoretical stages is %f\n\n",Nmin);

printf('Illustration 6.4(c) - Page: 342\n\n');
// Solution(c)
// Slope of q-line = Lf/F/(1-(Lf/F))
s = q/(1-q);
```

```
// For minimum reflux ratio
// From figure 6.12 y-intercept is
i = 0.457;
// Therefore Rmin is
Rmin = xd/i -1;
printf("The minimum reflux ratio is %f mole reflux/mole distillate\n\n",Rmin);

printf('Illustration 6.4(d) - Page: 343\n\n');
// Solution(d)
R = 1.3*Rmin;
// The y-intercept of the rectifying-section operating line is
ia = xd/(R+1);
// The operating line for the stripping section is drawn to pass through the point x = y = xw = 0.05 on the 45" line and the point of intersection of the q-line  // and the rectifying-section operating line.
// Therefore from figure 6.15
Nact = 13;
// But it include boiler
Nact1 = Nact-1;
printf("The number of equilibrium stages for the reflux ratio specified is %f\n",Nact1);
// For the optimal feed-stage location, the transition from one operating line to the other occurs at the first opportunity
// after passing the operating-line intersection
// Therefore from figure 6.15 shows that
printf("The optimal location of the feed stage for the reflux ratio specified is sixth from the top\n\n");

printf('Illustration 6.4(e) - Page: 344\n\n');
// Solution(e)
L = R*D; // [kmole/h]
V = L+D; // [kmole/h]
```

```
// From equation 6.27
Lst = L+q*Fa; // [kmole/h]
// From equation 6.28
Vst = V+(q-1)*Fa; // [kmole/h]

// For 50% vaporization of the feed ( zf = 0.60), from calculations similar to those illustrated in Example 6.1, the separator temperature and the equilibrium    // compositions are
Tf = 365.5; // [K]
yf = 0.707;
xf = 0.493;

// Latent heat vaporisation data at temperature T = 298 K
lambdaA = 33.9; // [kJ/mole]
lambdaB = 38; // [kJ/mole]
// Heat capacities of liquids (298-366 K)
Cla = 0.147; // [kJ/mole.K]
Clb = 0.174; // [kJ/mole.K]
// Heat capacities of gases, average in the range 298 to 366 K
Cpa = 0.094; // [kJ/mole.K]
Cpb = 0.118; // [kJ/mole.K]
// Substituting in equation 6.6 gives
Hf = 0;
Hlf = (Tf-T)*(xf*Cla+(1-xf)*Clb); // [kJ/mole of liquid feed]
// From equation 6.7
Hvf = (Tf-T)*(yf*Cpa+(1-yf)*Cpb) + yf*lambdaA + (1-yf)*lambdaB; // [kJ/mole of vapor feed]

Lf = Fa*q; // [kmole/h]
Vf = Fa*(1-q); // [kmole/h]
// From equation 6.3
Qf = (Hvf*Vf +Hlf*Lf-Fa*Hf)*1000/3600; // [kW]


```

```
Tlo = 354.3; // [Bubble point temperature, K]
T1 = 355.8; // [Dew point temperature, K]
y1 = 0.95; // [composition of saturated vapor at dew point]
x0 = 0.95; // [composition of saturated liquid at bubble point]
Hv1 = (T1-T)*(y1*Cpa+(1-y1)*Cpb) + y1*lambdaA + (1-y1)*lambdaB; // [kJ/mole of vapor feed]
Hlo = (Tlo-T)*(x0*Cla+(1-x0)*Clb); // [kJ/mole of liquid feed]

// An energy balance around condenser
Qc = V*(Hv1-Hlo)*1000/3600; // [kW]

// A flash-vaporization calculation is done in which the fraction vaporized is known (53.8/75.4 = 0.714) and the concentration
// of the liquid residue is fixed at xw = 0.05
// The calculations yield
Tr = 381.6; // [K]
x12 = 0.093;
y13 = 0.111;
T12 = 379.7; // [Bubble point of the liquid entering in the reboiler, K]

Hl12 = (T12-T)*(x12*Cla+(1-x12)*Clb); // [kJ/mole of liquid feed]
Hv13 = (Tr-T)*(y13*Cpa+(1-y13)*Cpb) + y13*lambdaA + (1-y13)*lambdaB; // [kJ/mole of vapor feed]

Hlw = (Tr-T)*(xw*Cla+(1-xw)*Clb); // [kJ/mole of liquid feed]

// An energy balance around the reboiler
Qr = (Vst*Hv13+W*Hlw-Lst*Hl12)*1000/3600; // [kW]
printf("The thermal load of the condenser, reboiler, and feed preheater are %f kW, %f kW and %f kW respectively\n\n",Qc,Qr,Qf);
```

Scilab code Exa 6.7 Overall Efficiency of a Benzene Toluene Fractionator

```
clear;
clc;

// Illustration 6.7
// Page: 358

printf('Illustration 6.7 - Page: 358\n\n');

// solution
//*****Data*****//
// a-benzene   b-toluene
xa = 0.46;
xb = 0.54;
Tb = 395; // [bottom temp., K]
Tt = 360; // [top temp., K]
alphab = 2.26;
alphat = 2.52;
D = 1.53; // [diameter of column, m]
f = 0.81; // [flooding]
deltaP = 700; // [average gas-pressure drop, Pa/tray]
//*****//

Tavg = (Tb+Tt)/2; // [K]
alpha_avg = (alphab+alphat)/2;

printf('Illustration 6.7(a) - Page: 359\n\n');
// Solution(a)

// Constants for components 'a' and 'b'
Aa = 4.612;
Ba = 148.9;
```

```
Ca = -0.0254;
Da = 2.222*10^-5;
ua = exp(Aa+Ba/Tavg+Ca*Tavg+Da*Tavg^2); // [cP]

Ab = -5.878;
Bb = 1287;
Cb = 0.00458;
Db = -0.450*10^-5;

ub = exp(Ab+Bb/Tavg+Cb*Tavg+Db*Tavg^2); // [cP]

// At the average column temperature
ul = ua^xa*ub^xb; // [cP]
K = alpha_avg*ul;
// From the O Connell correlation
Eo = 0.52782-0.27511*log10(K) + 0.044923*(log10(K))^2;
printf("The overall tray efficiency using the O Connell correlation is %f.\n\n",Eo);

printf('Illustration 6.7(b) - Page: 359\n');
// Solution(b)

Nideal = 20; // [number of ideal stages]
Nreal = Nideal/(Eo); // [nnumber of real stages]
disp(Nreal);
// Since real stages cannot be fractional, therefore
Nreal = 34;
// From Table 4.3 tray spacing
t = 0.6; // [m]
// Adding 1 m over the top tray as an entrainment separator and 3 m beneath // the bottom tray for bottoms surge capacity, the total column height is
Z = 4+Nreal*t; // [m]
printf("The number of real trays and the total tower height are %f and %f m respectively.\n\n",Nreal,Z);

printf('Illustration 6.7(c) -  Page: 359\n\n');
// Solution(c)

// Total gas pressure drop
deltaPc = deltaP*Nreal/1000; // [kPa]
printf("The total gas-pressure drop through the column is %f kPa.\n\n",deltaPc);
```

Scilab code Exa 6.10 Use of Fenske Equation for Ternary Distillation

```
clear;
clc;

// Illustration 6.10
// Page: 371

printf('Illustration 6.10 -  Page: 371\n\n');

// solution
//*****Data*****//
// A-toluene   B-1,2,3-trimethyl benzene   C-benzene
// Solution of above three are ideal
// Feed
za = 0.40;
zb = 0.30;
zc = 0.30;
// Bottom
FRAd = 0.95; // [recovery of toluene in distillate]
FRBw = 0.95; // [recovery of 1,2,3-trimethyl benzene in the bottom]
P = 1; // [atm]

// First estimate of distillate composition
xc = 40/70;
```

```
xa = 30/70;
xb = 0;
// The bubble point temperature for this solution is
Tb = 390; // [K]
// The corresponding parameters for benzene, toluene and 1,2,3-trimethyl benzene
// For toluene
Tc_a = 568.8; // [K]
Pc_a = 24.9; // [bar]
A_a = -7.912;
B_a = 1.380;
C_a =-3.804;
D_a = -4.501;
// For 1,2,3-trimethyl benzene
Tc_b = 664.5; // [K]
Pc_b = 34.5; // [bar]
A_b = -8.442;
B_b = 2.922;
C_b =-5.667;
D_b = -2.281;
// For benzene
Tc_c = 540.3; // [K]
Pc_c = 27.4; // [bar]
A_c = -7.675;
B_c = 1.371;
C_c =-3.536;
D_c = -3.202;


// At the estimated reboiler temperature of 449.3 K
Tr = 449.3; // [K]
// P = [Toluene;1,2,3-trimethyl benzene;Benzene]
P1 = zeros(3,6);
// P = [Tc Pc A B C D]
P1 = [568.8 24.9 -7.912 1.380 -3.804 -4.501;664.5 34.5 -8.442 2.922 -5.667 2.281;540.3 27.4 -7.675 1.371 -3.536 -3.202;];

```

```
for i=1:3
    P1(i) = P1(i,2)*exp((P1(i,3)*(1-Tr/P1(i,1))+P1(i,4)*(1-Tr/P1(i,1))^1.5+P1(i,5)*(1-Tr/P1(i,1))^3+P1(i,6)*(1-Tr/P1(i,1))^6)/(1-(1-Tr/P1(i,1))));
end
PA1 = P1(1); // [bar]
PB1 = P1(2); // [bar]
PC1 = P1(3); // [bar]
alphaAB1 = PA1/PB1;
alphaCB1 = PC1/PB1;

// At the estimated distillate temperature of 390 K
Td = 390; // [K]
// P = [Toluene;1,2,3-trimethyl benzene;Benzene]
P2 = zeros(3,6);
// P = [Tc Pc A B C D]
P2 = [568.8 24.9 -7.912 1.380 -3.804 -4.501;664.5 34.5 -8.442 2.922 -5.667 2.281;540.3 27.4 -7.675 1.371 -3.536 -3.202;];

for i=1:3
    P2(i) = P2(i,2)*exp((P2(i,3)*(1-Td/P2(i,1))+P2(i,4)*(1-Td/P2(i,1))^1.5+P2(i,5)*(1-Td/P2(i,1))^3+P2(i,6)*(1-Td/P2(i,1))^6)/(1-(1-Td/P2(i,1))));
end

PA2 = P2(1); // [bar]
PB2 = P2(2); // [bar]
PC2 = P2(3); // [bar]
alphaAB2 = PA2/PB2;
alphaCB2 = PC2/PB2;

// The geometric-average relative volatilities are
alphaAB_avg = sqrt(alphaAB1*alphaAB2);
alphaCB_avg = sqrt(alphaCB1*alphaCB2);

```

```
// From equation 6.66
Nmin = log(FRAd*FRBw/((1-FRAd)*(1-FRBw)))/log(alphaAB_avg);

// From equation 6.67
FRCd = alphaCB_avg^Nmin/((FRBw/(1-FRBw))+alphaCB_avg^Nmin); // [fractional recovery of benzene in the distillate]

printf("The number of equilibrium stages required at total reflux is %f.\n",Nmin);
printf("The recovery fraction of benzene in the distillate is %f.\n\n",FRCd);
printf('Thus, the assumption that virtually all of the LNK will be recovered in the distillate is justified.');
```

Scilab code Exa 6.11 Underwood Equations for Ternary Distillation

```
clear;
clc;

// Illustration 6.11
// Page: 376

printf('Illustration 6.11 - Page: 376\n\n');

// solution
//*****Data*****//
// 1-toluene  2-1,2,3--trimethylbenzene  3-benzene
// Basis: 100 kmol/h of feed
F = 100; // [kmole/h]
// Since feed is saturated, therefore
q = 0;
// From example 6.10
```

```
x1d = 0.3;
x2d = 0.3;
x3d = 0.4;
a12 = 3.91;
a32 = 7.77;
a22 = 1;
// Equ 6.78 gives
deff('[y] = f14(Q)','y = 1- a12*x1d/(a12-Q)-a22*x2d/(a22-Q)-a32*x3d/(a32-Q)');
Q = fsolve(2,f14);

// From the problem statement
// d1 = D*x1d     d2 = D*x2d
d1 = F*x1d*0.95; // [kmol/h]
d2 = F*x2d*0.05; // [kmol/h]
d3 = F*x3d*0.997; // [kmol/h]

// Summing the three distillate , d1,d2 and d3
D = d1+d2+d3; // [kmole/h]

Vmin = a12*d1/(a12-Q)+a22*d2/(a22-Q)+a32*d3/(a32-Q);

// From the mass balance
Lmin = Vmin-D; // [kmol/h]
// Minimum reflux ratio
Rmin = Lmin/D;
printf("The minimum reflux ratio is %f\n\n",Rmin);
```

Scilab code Exa 6.12 Underwood Equations for a Depropanizer

```
clear;
clc;

// Illustration 6.12
// Page: 377
```

```
printf('Illustration 6.12 -  Page: 377\n\n');

// solution
//*****Data*****//
// Componenets  A-propane  B-pentane  C-methane  D-ethane  E-butane  F-hexane
// x-mole fraction  a-relative volatility
xA = 0.25;  aA = 4.08;
xB = 0.11;  aB = 1.00;
xC = 0.26;  aC = 39.47;
xD = 0.09;  aD = 10.00;
xE = 0.17;  aE = 2.11;
xF = 0.12;  aF = 0.50;
// Since propane and pentane are light and heavy key respectively
// Methane and ethane are LNK, hexane is a HNK, while butane is a  sandwich component, meaning that it has a volatility intermediate between the keys.

FRlkd = 0.98;
FRhkd = 0.01;
// For methane
D_CR = (aC-1)/(aA-1)*FRlkd + (aA-aC)/(aA-1)*FRhkd;
// For ethane
D_DR = (aD-1)/(aA-1)*FRlkd + (aA-aD)/(aA-1)*FRhkd;
// For butane
D_ER = (aE-1)/(aA-1)*FRlkd + (aA-aE)/(aA-1)*FRhkd;
// For hexane
D_FR = (aF-1)/(aA-1)*FRlkd + (aA-aF)/(aA-1)*FRhkd;
// Since the feed is 66% vaporized
q = 1-0.66;

// Now equation 6.82 is solved for two values of Q
deff('[y] = f14(Q1)','y = 0.66 - aA*xA/(aA-Q1)-aB*xB/(aB-Q1)-aC*xC/(aC-Q1)-aD*xD/(aD-Q1)-aE*xE/(aE-Q1)-aF*xF/(aF-Q1)');
Q1 = fsolve(1.2,f14);
```

```
deff('[y] = f15(Q2)','y = 0.66 - aA*xA/(aA-Q2)-aB*xB/(aB-Q2)-aC*xC/(aC-Q2)-aD*xD/(aD-Q2)-aE*xE/(aE-Q2)-aF*xF/(aF-Q2)');
Q2 = fsolve(2.5,f15);

// Basis: 100 mole of feed
F = 100; // [mole]
// Let d1 = Dxad, d2 = Dxbd, d3 = Dxcd, and so on
d1 = F*xA*FRlkd; // [moles of propane]
d2 = F*xB*FRhkd; // [moles of pentane]
d3 = F*xC; // [moles of methane]
d4 = F*xD; // [moles of ethane]
d6 = F*xF*0; // [moles of hexane]
// And d5 is unknown
// Applying equation 6,78 for each value of Q

// Solution of simultaneous equation
function[f]=H(e)
    f(1) = e(1) - aA*d1/(aA-Q1)-aB*d2/(aB-Q1)-aC*d3/(aC-Q1)-aD*d4/(aD-Q1)-aE*e(2)/(aE-Q1)-aF*d6/(aF-Q1);
    f(2) = e(1) - aA*d1/(aA-Q2)-aB*d2/(aB-Q2)-aC*d3/(aC-Q2)-aD*d4/(aD-Q2)-aE*e(2)/(aE-Q2)-aF*d6/(aF-Q2);
    funcprot(0);
endfunction

// Initial guess
e = [90 5];
y = fsolve(e,H);
Vmin = y(1); // [mole]
d5 = y(2); // [d5 = Dxed, mole]

// From equ 6.84
D = d1+d2+d3+d4+d5+d6; // [mole]
// From mass balance
Lmin = Vmin-D; // [mole]
```

```
// For minimum reflux ratio
Rmin = Lmin/D;
printf(”The minimum reflux ratio is %f\n\n”,Rmin);
```

Scilab code Exa 6.13 Application of the Gilliland Correlation

```
clear;
clc;

// Illustration 6.13
// Page: 380
printf('Illustration 6.13 –  Page: 380\n\n');

// solution
//*****Data*****//
// A–benzene B–toluene C–1,2,3–trimethylbenzene
// From example 6.10
Nmin = 4.32; // [stages]
// From example 6.11
Rmin = 0.717; // [minimum reflux ratio]
// For R = 1
R = 1;
X = (R-Rmin)/(R+1);
// From equ 6.88
Y = 1-exp((1+54.4*X)/(11+117.2*X)*(X-1)/sqrt(X));
// Fro equ 6.86
N = (Y+Nmin)/(1-Y);
// From example 6.10 99.7% of the LNK (benzene) is recovered in the distillate// , 95% of the light key is in the distillate , and 95% of the heavy key is in// the bottoms

// For a basis of 100 mol of feed , the material balances for three components // are
// For distillate
```

```
nAd = 39.88; // [LNK, moles of benzene]
nBd = 28.5; // [LK, moles of toluene]
nCd = 1.50; // [HK, moles of 1,2,3-trimethylbenzene]
nTd = nAd+nBd+nCd; // [total number of moles]
xAd = nAd/nTd;
xBd = nBd/(nTd);
xCd = nCd/(nTd);

// For bottoms
nAb = 0.12;
nBb = 1.50;
nCb = 28.50;
nTb = nAb+nBb+nCb;
xAb = nAb/nTb;
xBb = nBb/nTb;
xCb = nCb/nTb;

D = nTd;
W = nTb;
// From problem statement
Zlk = 0.3;
Zhk = Zlk;
// Substituting in equation 6.89
// T = Nr/Ns
T = (Zhk/Zlk*W/D*(xBb/xCd)^2)^0.206;

// Solution of simultaneous equation
function[f]=H(e)
    f(1) = e(1)-e(2)*T;
    f(2) = e(1)+e(2)-N;
        funcprot(0);
endfunction

// Initial guess
e = [5 4];
y = fsolve(e,H);
Nr = y(1); // [number of stages in rectifying section]
```

```
Ns = y(2); // [number of stages in stripping section ]
disp(Ns,Nr);
printf('Rounding the estimated equilibrium stage requirement leads to 1 stage as a partial reboiler, 4 stages below the feed, and 5 stages above the feed.');
```

Scilab code Exa 6.14 Rate Based Ternary Distillation Calculations

```
clear;
clc;

// Illustration 6.14
// Page: 387
printf('Illustration 6.14 -  Page: 387\n\n');

// solution
//*****Data*****//
// a-acetone  b-methanol  c-water
yna = 0.2971; yn1a = 0.17; ynIa = 0.3521; mnIa = 2.759; xna = 0.1459;
ynb = 0.4631; yn1b = 0.429; ynIb = 0.4677; mnIb = 1.225; xnb = 0.3865;
ync = 0.2398; yn1c = 0.4010; ynIc = 0.1802; mnIc = 0.3673; xnc = 0.4676;

Fabv = 4.927; // [mol/square m.s]
Facv = 6.066; // [mol/square m.s]
Fbcv = 7.048; // [mol/square m.s]
aI = 50; // [square m]
Vn1 = 188; // [mol/s]
Vn = 194.8; // [mol/s]
//*****//
printf('Illustration 6.14(a) -  Page: 387\n\n');
```

```
// Solution(a)

ya = (yna+ynIa)/2;
yb = (ynb+ynIb)/2;
yc = (ync+ynIc)/2;

Rav = ya/Facv+yb/Fabv+yc/Facv;
Rbv = yb/Fbcv+ya/Fabv+yc/Fbcv;

Rabv = -ya*(1/Fabv-1/Facv);
Rbav = -yb*(1/Fabv-1/Fbcv);
// Thus in matrix form
Rv = [Rav Rabv;Rbav Rbv];
kv = inv(Rv); // [inverse of Rv]
// From equ 6.99
b = [yna-ynIa;ynb-ynIb];
J = kv*b;

// From equ 6.98
Jc = -sum(J); // [mol/square m.s]

printf("The molar diffusional rates of acetone, methanol and water are %f mol/square m.s, %f mol/square m.s and %f mol/square m.s respectively.\n\n",J(1,1),J(2,1),Jc);

printf('Illustration 6.14(b) -  Page: 388\n\n');
// Solution(b)
Ntv = Vn1-Vn; // [mol/s]

// From equation 6.94
Nta = aI*J(1,1)+ya*Ntv;
Ntb = aI*J(2,1)+yb*Ntv;
Ntc = aI*Jc+yc*Ntv;
printf("The mass transfer rates of acetone, methanol and water are %f mol/s ,%f mol/s and %f mol/s respectively.\n\n",Nta,Ntb,Ntc);

printf('Illustration 6.14(c) -  Page: 389\n\n');
// Solution(c)

// Approximate values of Murphree vapor tray efficiency are obtained from   // equation 6.105

EMG_a = (yna-yn1a)/(mnIa*xna-yn1a);
EMG_b = (ynb-yn1b)/(mnIb*xnb-yn1b);
EMG_c = (ync-yn1c)/(mnIc*xnc-yn1c);

printf("The Murphree vapor tray efficiencies for acetone, methanol and water are %f, %f and %f respectively.\n\n",EMG_a,EMG_b,EMG_c);
```

Chapter 7

Liquid Liquid Extraction

Scilab code Exa 7.2 Single Stage Extraction

```
clear;
clc;

// Illustration 7.2
// Page: 433

printf('Illustration 7.2 -  Page: 433\n\n');

// solution
//*****Data*****//
//  'b'-solvent  'f'-feed  'r'-raffinate  'e'-extract  'c'-one of the  // component in  feed
F = 50; // [feed rate, kg/h]
S = 50; // [solvent rate, kg/h]
xcf = 0.6;
xbf = 0;
ycs = 0;
ybs = 1.0;
// The equilibrium data for this system can be obtained from Table 7.1 and  // Figure 7.6
// Plot streams F (xcF = 0.6, xBF = 0.0) and S (yes
   = 0.0, yBs = 1.0). After  // locating streams F
   and S, M is on the line FS; its exact location is
    found // by calculating xcm from

xcm = (F*xcf+S*ycs)/(F+S);

// From figure 7.8
xcr = 0.189;
xbr = 0.013;
yce = 0.334;
ybe = 0.648;
M = F+S; // [kg/h]
// From equation 7.8
E = M*(xcm-xcr)/(yce-xcr); // [kg/h]
R = M-E; // [kg/h]
printf(”The extract and raffinate flow rates are %f
   kg/h and %f kg/h respectively.\n\n”,E,R);
printf(”The compositions when one equilibrium stage
   is used for the separation is %f and %f in
   raffinate phase for component b and c
   respectively and %f and %f in extract phase for
   component b and c respectively.\n\n”,xcr,xbr,yce,
   ybe);
```

Scilab code Exa 7.4 Multistage Countercurrent Extraction

```
clear;
clc;

// Illustration 7.4
// Page: 439

printf('Illustration 7.4 –  Page: 439\n\n');

// solution
```

```
//*****Data*****//
// C-acetic acid    A-water
// f-feed   r-raffinate   s-solvent
f = 1000; // [kg/h]
xCf = 0.35; // [fraction of acid]
xAf = 1-xCf; // [fraction of water]
// Solvent is pure
xAr = 0.02;
yCs = 0;
//*****//

printf('Illustration 7.4(a) -  Page: 440\n\n');
// Solution(a)

// From Figure 7.15
xCMmin = 0.144;
// From equation 7.11
Smin = f*(xCMmin-xCf)/(yCs-xCMmin); // [kg/h]
printf("The minimum amount of solvent which can be
   used is %f kg/h.\n\n",Smin);

printf('Illustration 7.4(b) -  Page: 441\n\n');
// Solution(b)

S = 1.6*Smin; // [kg/h]
// From equation 7.11
xCM = (f*xCf+S*yCs)/(f+S);

// Data for equilibrium line
// Data_eqml = [xCeq yCeq]
Data_eqml = [0.0069 0.0018;0.0141 0.0037;0.0289
   0.0079;0.0642 0.0193;0.1330 0.0482;0.2530
   0.1140;0.3670 0.2160;0.4430 0.3110;0.4640
   0.3620];

// Data for operating line
// Data_opl = [xCop yCop]
Data_opl = [0.02 0;0.05 0.009;0.1 0.023;0.15
```

```
    0.037;0.20 0.054;0.25 0.074;0.30 0.096;0.35
    0.121];


scf(1);
plot(Data_eqml(:,1),Data_eqml(:,2),Data_opl(:,1),
    Data_opl(:,2));
xgrid();
legend(’Equilibrium line,Operating line’);
xlabel(”wt fraction of acetic acid in water
    solutions, xC”);
ylabel(”wt fraction of acetic acid in ether
    solutions, yC”);

// Now number of theoritical stages is determined by
     drawing step by step  // stairs from xC = 0.35
    to xC = 0.02
// From figure 7.16
// Number of theoritical stages ’N’ is
N = 8;
printf(”The number of theoretical stages if the
    solvent rate used is 60 percent above the minimum
     is %f.\n\n”,N);
```

Scilab code Exa 7.5 Multistage Extraction Insoluble Liquids

```
clear;
clc;

// Illustration 7.5
// Page: 444

printf(’Illustration 7.5 -  Page: 444\n\n’);

// solution
```

```
//*****Data*****//
// C-nicotine    A-water    B-kerosene
// F-feed    R-raffinate    S-solvent
F = 1000; // [feed rate, kg/h]
xAF = 0.99; // [fraction of water in feed]
// Because the solutions are dilute therefore
xCF = 0.01; // [fraction of nicotene in feed, kg nicotene/kg water]
xCR = 0.001; // [fraction of nicotene in raffinate, kg nicotene/kg water ]
m = 0.926; // [kg water/kg kerosene]
//*****//

printf('Illustration 7.5(a) - Page: 444\n\n');
// Solution(a)

yCS = 0; // [kg nicotene/kg water]

// Because, in this case, both the equilibrium and operating lines are        // straight,if the minimum solvent flow rate Bmin is used, the concentration // of the exiting extract, yCmax, will be in equilibrium with xCF. Therefore
yCmax = m*xCF; // [kg nicotene/kg kerosene]

A = F*xAF; // [kg water/h]
// From equation 7.17
Bmin = A*(xCF-xCR)/(yCmax-yCS); // [kg kerosene/h]
printf("The minimum amount of solvent which can be used is %f kg kerosene/h.\n\n",Bmin);

printf('Illustration 7.5(b) - Page: 444\n\n');
// Solution(b)

B = 1.2*Bmin; // [kg kerosene/h]
EF = m*B/A;
Nt = log((xCF-yCS/m)/(xCR-yCS/m)*(1-1/EF)+1/EF)/log(EF);
```

```

printf("The number of theoretical stages if the solvent rate used is 20 percent above the minimum is %f .\n\n",Nt);

printf('Illustration 7.5(c) - Page: 444\n');
// Solution(c)

Eme = 0.6; // [Murphree stage efficiency]
// from equation 7.20
Eo = log(1+Eme*(EF-1))/log(EF); // [overall efficiency]
Nr = Nt/Eo; // [number of real stages]
disp(Nr);
// The nearest integer to number of real stages is 11
// Therefore
Nr = 11;
printf("The number of real stages required is %f.\n\n",Nr);
```

Scilab code Exa 7.6 Countercurrent Extraction with Extract Reflux

```
clear;
clc;

// Illustration 7.6
// Page: 449

printf('Illustration 7.6 - Page: 449\n\n');

// solution
//*****Data*****//
// C-styrene    A-ethylbenzene    B-diethylene glycol
```

```
F = 1000; // [kg/h]
XF = 0.6; // [wt fraction of styrene]
XPE = 0.9;
XN = 0.1;
// All above fractions are on solvent basis
// Equilibrium Data for Ethylbenzene (A)-Diethylene
   Glycol (B)-Styrene (C) at 298 K
// Data_eqm = [X Y];
// X - kg C/kg (A+C) in raffinate solution
// Y - kg C/kg (A+C) in extract solution
Data_eqm = [0 0;0.087 0.1429;0.1883 0.273;0.288
   0.386;0.384 0.48;0.458 0.557;0.464 0.565;0.561
   0.655;0.573 0.674;0.781 0.863;0.9 0.95;1 1];
//*****//

printf('Illustration 7.6(a) - Page: 449\n\n');
// Solution(a)

// Minimum theoretical stages are determined on the
   XY equilibrium distribution diagram, stepping
   them off from the diagonal line to the
   equilibrium curve, beginning at XPE = 0.9 and
   ending at XN = 0.1

Data_opl = [0 0;0.09 0.09;0.18 0.18;0.27 0.27;0.36
   0.36;0.45 0.45;0.54 0.54;0.63 0.63;0.72 0.72;0.81
    0.81;0.90 0.90;1 1;];

scf(1);
plot(Data_eqm(:,1),Data_eqm(:,2),Data_opl(:,1),
   Data_opl(:,2));
xgrid();
legend('Equilibrium line','Operating line');
xlabel("X,kg C/kg (A+C) in raffinate solution");
ylabel("Y,kg C/kg (A+C) in extract solution");

// Figure 7.20
Nmin = 9; // [number of ideal stages]
```

```

printf("The minimum number of theoretical stages are %f.\n\n",Nmin);

printf('Illustration 7.6(b) - Page: 450\n\n');
// Solution(b)

// Since the equilibrium-distribution curve is everywhere concave downward// ,the tie line which when extended passes through F provides the minimum
// reflux ratio
// From figure 7.19
NdeltaEm = 11.04;
NE1 = 3.1;
// From equation 7.30
// Y = R_O/P_E, external reflux ratio
Ymin = (NdeltaEm-NE1)/NE1; // [kg reflux/kg extract product]

printf("The minimum extract reflux ratio is %f kg reflux/kg extract product.\n\n",Ymin);

printf('Illustration 7.6(c) - Page: 450\n\n');
// Solution(c)

Y = 1.5*Ymin; // [kg reflux/kg extract product]
// From equation 7.30
NdeltaE = Y*NE1+NE1;
// From figure 7.19
NdeltaR = -24.90;
// From figure 7.21
N = 17.5; // [number of equilibrium stages]

// From figure 7.19
// For XN = 0.1 NRN = 0.0083
NRN = 0.0083;
// Basis: 1 hour
// e = [P_E R_N]
// Solution of simultaneous equation
function[f]=G(e)
    f(1) = F - e(1) - e(2);
    f(2) = F*XF-e(1)*XPE-e(2)*XN;
    funcprot(0);
endfunction
// Initial guess:
e = [600 300];
y = fsolve(e,G);
P_E = y(1); // [kg/h]
R_N = y(2); // [kg/h]

R_O = Y*P_E; // [kg/h]
E_1 = R_O+P_E; // [kg/h]

B_E = E_1*NE1; // [kg/h]
E1 = B_E+E_1; // [kg/h]
RN = R_N*(1+NRN); // [kg/h]
S = B_E+R_N*NRN; // [kg/h]

printf("The number of theoretical stages are %f.\n",N);
printf('The important flow quantities at an extract reflux ratio of 1.5 times the minimum value are\n\n');
printf(" PE = %f kg/h\n RN = %f kg/h\n RO = %f kg/h\n E1 = %f kg/h\n BE = %f kg/h\n E1 = %f kg/h\n RN = %f kg/h\n S = %f kg/h\n",P_E,R_N,R_O,E_1,B_E,E1,RN,S);
```

Scilab code Exa 7.7 Design of a Mixer Settler Extractor

```
clear;
```

```
clc;

// Illustration 7.7
// Page: 454

printf('Illustration 7.7 -  Page: 454\n\n');

// solution
//*****Data*****//
Ff = 1.89; // [cubic m/min]
Fs = 2.84; // [cubic m/min]
t = 2; // [min]
//*****//

printf('Illustration 7.7(a) -  Page: 454\n\n');
// Solution(a)

Q = Ff+Fs; // [total flow rate, cubic m/min]
Vt = Q*t; // [cubic m]
// For a cylindrical vessel H = Dt
Dt = (4*Vt/%pi)^(1/3); // [m]
H = Dt; // [m]
printf("The diameter and height of each mixing vessel is %f m and %f m respectively.\n\n",Dt,H);

printf('Illustration 7.7(b) -  Page: 454\n\n');
// Solution(b)
// Based on a recommendation of Flynn and Treybal (1955),
P = 0.788*Vt; // [mixer power, kW]
printf("The agitator power for each mixer is %f kW.\n\n",P);

printf('Illustration 7.7(c) -  Page: 454\n\n');
// Solution(c)

// Based on the recommendation by Ryan et al. (1959), the disengaging area  // in the settler is
// Dt1*L1 = Q/a = Y
a = 0.2; // [cubic m/min-square m]
Y = Q/a; // [square m]
// For L/Dt = 4
Dt1 = (Y/4)^0.5; // [m]
L1 = 4*Dt1; // [m]
printf("The diameter and length of a settling vessel is %f m and %f m respectively.\n\n",Dt1,L1);

printf('Illustration 7.7(d) - Page: 454\n\n');
// Solution(d)
// Total volume of settler
Vt1 = %pi*Dt1^2*L1/4; // [cubic m]
tres1 = Vt1/Q; // [min]
printf("The residence time in the settling vessel is %f min.\n\n",tres1);
```

Scilab code Exa 7.8 Power Requirements of a Mixer Settler Extractor

```
clear;
clc;

// Illustration 7.8
// Page: 456

printf('Illustration 7.8 - Page: 456\n\n');

// solution
//*****Data*****//
Ff = 1.61; // [flow rate of feed, kg/s]
Fs = 2.24; // [flow rate of solvent, kg/s]
t = 2*60; // [residence time in each mixer, s]
df = 998; // [density of feed, kg/cubic m]
uf = 0.89*10^-3; // [viscosity of feed, kg/m.s]
ds = 868; // [density of solvent, kg/cubic m]
```

```
us = 0.59*10^-3; // [viscosity of solvent, kg/m.s]
sigma = 0.025; // [interfacial tension, N/m]
g = 9.8; // [square m/s]
//*****//

Qf = Ff/df; // [volumetric flow rate of feed, cubic m/s]
Qs = Fs/ds; // [volumetric flow rate of solvent, cubic m/s]
// Volume fractions in the combined feed and solvent entering the mixer
phiE = Qs/(Qs+Qf);
phiR = 1-phiE;

printf('Illustration 7.8(a) - Page: 457\n\n');
// Solution(a)

Q = Qf+Qs; // [total flow rate, cubic m/s]
Vt = Q*t; // [vessel volume, cubic m]
// For a cylindrical vessel, H = Dt
// Therefore,    Vt = %pi*Dt^3/4
Dt = (4*Vt/%pi)^(1/3); // [ diameter, m]
H = Dt; // [height, m]
Di = Dt/3; // [m]
printf("The height and diameter of the mixing vessel are %f m and %f m respectively.\n",Dt,H);
printf("The diameter of the flat-blade impeller is %f m.\n\n",Di);

printf('Illustration 7.8(b) - Page: 457\n\n');
// Solution(b)

// For the raffinate phase dispersed:
phiD = phiR;
phiC = phiE;
deltad = df-ds; // [kg/cubic m]
rowM = phiD*df+phiC*ds; // [kg/cubic m]
uM = us/phiC*(1 + 1.5*uf*phiD/(us+uf)); // [kg/m.s]
```

```
// Substituting in equation 7.34
ohm_min = sqrt(1.03*phiD^0.106*g*deltad*(Dt/Di)^2.76*(uM^2*sigma/(Di^5*rowM*g^2*deltad^2))^0.084/(Di*rowM))*60; // [rpm]
printf("The minimum rate of rotation of the impeller for complete and uniform dispersion.is %f rpm.\n\n",ohm_min);

printf('Illustration 7.8(c) - Page: 457\n\n');
// Solution(c)

ohm = 1.2*ohm_min; // [rpm]

// From equation 7.37
Re = ohm/60*Di^2*rowM/uM; // [Renoylds number]
// Then according to Laity and Treybal (1957), the power number, Po = 5.7
Po = 5.7
// From equation 7.37
P = Po*(ohm/60)^3*Di^5*rowM/1000; // [kW]
// Power density
Pd = P/Vt; // [kW/cubic m]
printf("The power requirement of the agitator at 1.20 times the minimum rotation rate is %f kW.\n\n",P);
```

Scilab code Exa 7.9 Drop Size and Interfacial Area in an Extractor

```
clear;
clc;

// Illustration 7.9
// Page: 460

printf('Illustration 7.9 - Page: 460\n\n');
```

```

// solution
//*****Data*****//
// From example 7.8
Di = 0.288; // [m]
sigma = 0.025; // [N/m]
ohm = 152*1.2/60; // [rps]
ds = 868; // [kg/cubic m]
phiD = 0.385;

// Therefore from equation 7.49
We = Di^3*ohm^2*ds/sigma; // [Weber number]

// From equation 7.50
dvs = Di*0.052*(We)^-0.6*exp(4*phiD); // [m]
disp(dvs);
// Substituting in equation 7.48
a = 6*phiD/dvs; // [square m/cubic m]
printf("The Sauter mean drop diameter and the interfacial area is %e m and %f square m/cubic m respectively.\n\n",dvs,a);
```

Scilab code Exa 7.10 Mass Transfer Coefficients in Agitated Extractor

```
clear;
clc;

// Illustration 7.10
// Page: 461

printf('Illustration 7.10 - Page: 461\n\n');

// solution
//*****Data*****//
Dd = 1.15*10^-9; // [molecular diffusivity of
furfural in water, square m/s]
Dc = 2.15*10^-9; // [molecular diffusivity of furfural in toluene, square m/s]
m = 10.15; // [equilibrium distribution coefficient, cubic m raffinate/cubic m extract]

printf('Illustration 7.10(a) - Page: 461\n\n');
// Solution(a)
// From example 7.8 and 7.9
dvs = 3.26*10^-4; // [m]
Shd = 6.6; // [sherwood number for dispersed phase]
// From equation 7.52
kd = Shd*Dd/dvs; // [dispersed phase mass transfer coefficient, m/s]
printf("The dispersed-phase mass-transfer coefficient is %e m/s.\n\n",kd);

printf('Illustration 7.10(b) - Page: 461\n\n');
// Solution(b)

dd = 998;
dc = 868; // [density of continuous phase, kg/cubic m]
uc = 0.59*10^-3; // [viscosity of continuous phase, kg/m.s]
ohm = 182.2; // [rpm]
g = 9.8; // [square m/s]
Di = 0.288; // [m]
sigma = 0.025; // [N/m]
phiD = 0.385;
Dt = 0.863; // [m]
Scc = uc/(dc*Dc);
Rec = Di^2*ohm/60*dc/uc;
Fr = Di*(ohm/60)^2/g;
Eo = dd*dvs^2*g/sigma;

// From equation 7.53
Shc = 1.237*10^-5*Rec^(2/3)*Scc^(1/3)*Fr^(5/12)*Eo
```

```
^(5/4)*phiD^(-1/2)*(Di/dvs)^2*(dvs/Dt)^(1/2);
// Therefore
kc = Shc*Dc/dvs; // [continuous phase mass transfer coefficient , m/s]
printf("The continuous-phase mass-transfer coefficient is %e m/s.\n\n",kc);

printf('Illustration 7.10(c) - Page: 462\n\n');
// Solution(c)

a = 7065; // [square m/cubic m]
Vt = 0.504; // []
Qd = 0.097/60; // [cubic m/s]
Qc = 0.155/60; // [cubic m/s]

// From equation 7.40
Kod = kd*kc*m/(m*kc+kd); // [m/s]
// From equation 7.45
N_tod = Kod*a*Vt/Qd;
// From equation 7.46
Emd = N_tod/(1+N_tod);
printf("The Murphree dispersed phase efficiency is %f.\n\n",Emd);

printf('Illustration 7.10(d) - Page: 462\n\n');
// Solution(d)
// From equation 7.57
fext = Emd/(1+Emd*Qd/(m*Qc));
printf("The fractional extraction of furfural is %f.\n\n",fext);
```

Scilab code Exa 7.11 Preliminary Design of an RDC

```
clear;
clc;
```

```

// Illustration 7.11
// Page: 466

printf('Illustration 7.11 - Page: 466\n\n');

// solution
//*****Data*****//
// Preliminary Design of an RDC
T = 293; // [K]
F1 = 12250; // [flow rate for dispersed organic phase, kg/h]
F2 = 11340; // [flow rate for continuous aqueous phase, kg/h]
d1 = 858; // [kg/cubic m]
d2 = 998; // [kg/cubic m]
n = 12; // [Equilibrium stages]
//*****//
Qd = F1/d1; // [cubic m/h]
Qc = F2/d2; // [cubic m/h]

// Assume that based on information in Table 7.5
// Vd+Vc = V = 22 m/h
V = 22; // [m/h]
// Therefore column cross sectional area
Ac = (Qd+Qc)/V; // [square m]
// Column diameter
Dt = sqrt(4*Ac/%pi); // [m]

// Assume that based on information in Table 7.5
// 1/HETS = 2.5 to 3.5  m^-1
// Therefore
HETS = 1/3; // [m/theoritical stages]
// Column height
Z = n*HETS; // [m]
printf("The height and diameter of an RDC to extract acetone from a dilute toluene-acetone solution is %f m and %f square m respectively\n\n",Z,Dt);
```

Chapter 8

Humidification Operations

Scilab code Exa 8.1 Humidity of a Saturated Gas Vapor Mixture

```
clear;
clc;

// Illustration 8.1
// Page: 479

printf('Illustration 8.1 - Page: 479\n\n');

// solution
// ****Data****//
P_total = 1; // [bar]
T1 = 320; // [K]
T_c = 562.2; // [K]
P_c = 48.9; // [bar]
A = -6.983;
B = 1.332;
C = -2.629;
D = -3.333;
//*****//

x1 = 1-(T1/T_c);
```

```
deff('[y] = f12(P1)','y = log(P1/P_c)-(A*x1+B*x1^1.5+C*x1^3+D*x1^6)/(1-x1)');
P1 = fsolve(.01,f12);// [bar]
printf("Vapor pressure of benzene at 320 K is %f bar\n\n",P1);

M_benzene = 78 // [gram/mole]
printf('Illustration 8.1 (a)\n');

// Solution (a)
// For nitrogen
M_nitrogen = 28; // [gram/mole]
// From equation 8.2
Y = P1/(P_total - P1); //[mole C6H6/ mole N2]
Y_s1 = Y*(M_benzene/M_nitrogen); // [gram C6H6/gram N2]

printf("Absolute humidity of mixture of benzene and nitrogen is %f gram C6H6/gram N2\n\n",Y_s1);

printf('Illustration 8.1 (b)\n');
// Solution (b)
// For carbon dioxide
M_carbondioxide = 44; // [gram/mole]
// From equation 8.2
Y = P1/(P_total - P1); //[mole C6H6/ mole C02]
Y_s2 = Y*(M_benzene/M_carbondioxide); // [gram C6H6/gram CO2]

printf("Absolute humidity of mixture of benzene and carbon dioxide is %f gram C6H6/gram CO2\n",Y_s2);
```

Scilab code Exa 8.2 Enthalpy of a Saturated Gas Vapor Mixture

```
clear;
```

```
clc;

// Illustration 8.2
// Page: 480

printf('Illustration 8.2 -  Page: 480\n\n');

// solution
// A - water vapor    B - air
// REference state is air

// ****Data****//
T_ref = 273; // [Reference temperature, K]
T = 303; // [K]
P_total = 1; // [atm]
P_A = 4.24; // [Vapor pressure of water at 303K, kPa]
M_A = 18; // [gram/mole]
M_B = 29; // [gram/mole]
C_A = 1.884; // [kJ/kg.K]
C_B = 1.005; // [kJ/kg.K]
lambda = 2502.3; // [Latent heat of Vaporization at 273K, kJ/kg]
//*****//

P_total = P_total*101.325; // [kPa]

// From equation 8.2
Y_s = P_A/(P_total - P_A)*(M_A/M_B); //[kg H2O/ kg dry air]
printf("Absolute humidity of mixture of water vapor and air is %f kg H2O/kg dry air\n\n",Y_s);

// From equation 8.3
H_s = C_B*(T-T_ref) + Y_s*(C_A*(T-T_ref) + lambda); // [kJ/kg dry air]

printf("Enthalpy per unit mass of dry air of a
saturated mixture at 303 K and 1 atm is %f kJ/kg dry air\n",H_s);
```

Scilab code Exa 8.3 Properties of an Unsaturated Gas Vapor Mixture

```
clear;
clc;

// Illustration 8.2
// Page: 482

printf('Illustration 8.3 - Page: 482\n\n');

// solution
// A - water vapor   B - air
//*****Data*****
T = 328; // [dry bulb temperature, K]
P_total = 1; // [atm]
H = 30; // [relative humidity, %]
//*****//
P_vapA = 15.73; // [vapor pressure of water, kPa]
P_total = P_total*101.325; // [kPa]
M_A = 18; // [gram/mole]
M_B = 29; // [gram/mole]

P_A = (H/100)*P_vapA;// [partial pressure of A,kPa]

printf('Illustration 8.3 (a)\n\n');
// At dew point partial pressure is equal to vapor pressure
// Using Antonnie equation we can find dew point temperature

printf("Dew point temperature is 304.5 K\n")

```

```
// From equation 8.1
Y_s = P_A/(P_total-P_A)*(M_A/M_B);
printf("Absolute humidity of air-water mixture at 328 K is %f kg H2O/kg dry air\n\n",Y_s);

printf('Illustration 8.3 (b)\n\n');

//soluton (b)
T_ref = 273; // [K]
C_A = 1.884; // [kJ/kg.K]
C_B = 1.005; // [kJ/kg.K]
lambda = 2502.3; // [Latent heat of Vaporization at 273 K, kJ/kg]

// From equation 8.3
H_s = C_B*(T-T_ref) + Y_s*(C_A*(T-T_ref) + lambda);

printf("Enthalpy per unit mass of dry air of a saturated mixture relative to 273 K is %f kJ/kg dry air\n",H_s);
```

Scilab code Exa 8.4 Adiabatic Saturation Temperature

```
clear;
clc;

// Illustration 8.4
// Page: 484

printf('Illustration 8.4 -  Page: 484\n\n');

// Solution
// a - water vapor    b - air
//*****Data*****
T_G1 = 356; // [K]
```

```
P_total = 101.325; // [kPa]
Y_1 = .03; // [kg water/kg dry air]
//*****//

C_pa = 1.884; // [kJ/kg.K]
C_pb = 1.005; // [kJ/kg.K]

C_s1 = C_pb + Y_1*C_pa;// [kJ/kg.K]

T_1 = 373.15; // [K]
T_c = 647.1; // [K]
M_a = 18.02; // [gram/mole]
M_b = 28.97; // [gram/mole]
lambda_1 = 2256; // [Latent Heat of Vaporizarion at T_1, kJ/kg]

// Using equation 8.10
// T_as = T_G1- (Y_as - Y_1)*lambda_as/C_s1
// where lambda_2 = lambda_1*((1-T_as/T_c)/(1-T_1/T_c))^.38
//        Y_as = P_a/(P_total-P_a)*M_a/M_b
//        and P_a = exp(16.3872-(3885.7/(T_as-42.98))) - Antoine equation for component 'a'

deff('[y] = f12(T_as)',' y = T_as - T_G1 + ((exp(16.3872 - (3885.7/(T_as - 42.98)))/(P_total - (exp(16.3872 - (3885.7/(T_as - 42.98))))))*(M_a/M_b) - Y_1)*(lambda_1*((1-T_as/T_c)/(1-T_1/T_c))^.38/C_s1)');
T_as = fsolve(310,f12); // [K]
printf("Adiabatic Saturation Temperature is %f K\n",T_as);

// Now using equation 8.2

P_a = exp(16.3872-(3885.7/(T_as-42.98))); // [kPa]
Y_as = P_a/(P_total-P_a)*M_a/M_b; // [kg water/kg dry air]
```

```
printf("Absolute humidity is %f kg water/kg dry air\n",Y_as);
```

Scilab code Exa 8.5 Wet Bulb Temperature of an Air Water Mixture

```
clear;
clc;

// Illustration 8.5
// Page: 487

printf('Illustration 8.5 -  Page: 487\n\n');

// Solution
//*****Data*****//
T_w = 320; // [K]
T_g = 340; // [K]
lambda_w = 2413; // [Latent Heat of Vaporization at 320K, kJ/kg]
Y_w1 = 0.073; // [kg water/kg dry air]
//*****//
A = 0.95; // [For air water system,A, kJ/kg.K]

//    here A = hg/ky, psychrometric ratio
//    Air-water mixture is saturated at 320K and 1 atm
//    Using equation 8.15

Y_w2 = Y_w1 - ((T_g-T_w)*A/lambda_w); // [kg water/kg dry air]
printf("Absolute humidity of air-water mixture at 340 K and 1 atm is %f kg water/kg dry air\n ",Y_w2);
```

Scilab code Exa 8.6 Wet Bulb and Adiabatic Saturation Temperatures of an Air Toluene Mixture

```
clear;
clc;

// Illustration 8.6
// Page: 487

printf('Illustration 8.6 - Page: 487\n\n');

// a - toluene    b - air
//*****Data*****
T_G1 = 333; // [K]
P_total = 101.325; // [kPa]
Y_1 = 0.05; // [kg vapor/kg dry air]
//*****//

C_pa = 1.256; // [kJ/kg.K]
C_pb = 1.005; // [kJ/kg.K]

C_s1 = C_pb + Y_1*C_pa

T_1 = 383.8; // [K]
T_c = 591.8; // [K]
M_a = 92; // [gram/mole]
M_b = 28.97; // [gram/mole]
lambda_1 = 33.18*1000/92; // [Latent heat of
    vaporization at T_1, kJ/kg]

// Constants of antoine equation
A = 13.9320;
B = 3057; // [K]
C = -55.52; // [K]
```

```

printf('Illustration 8.6 (a)\n');

// Solution (a)

// Using equation 8.10
//    T_as = T_G1 - (Y_as - Y_1)*lambda_as/C_s1
//    where lambda_2 = lambda_1*((1-T_as/T_c)/(1-T_1/T_c))^.38
//    Y_as = P_a/(P_total-P_a)*M_a/M_b
//    P_a = exp(A-B/(T+c))

deff('[y] = f12(T_as)',' y = T_as - T_G1 + ((exp(13.9320 - (3057/(T_as - 55.52)))/(P_total - (exp(13.9320 - (3057/(T_as - 55.52)))))*(M_a/M_b) - Y_1)*(lambda_1*((1-T_as/T_c)/(1-T_1/T_c))^.38/C_s1)');
T_as = fsolve(273,f12); // [K]
printf("Adiabatic Saturation Temperature is %f K\n",T_as);

// Now using equation 8.2

P_a = exp(13.9320-(3057/(T_as-55.52))); // [kPa]
Y_as = P_a/(P_total-P_a)*M_a/M_b; // [kg vapor/kg dry air]

printf("Absolute humidity is %f kg vapor/kg dry air\n\n",Y_as);

printf('Illustration 8.6 (b)\n');

// Solution (b)

// Thermodynamic properties of mixture of toluene and air
row = 1.06; // [kg/cubic m]
u = 19.5*10^-6; // [P]
```

```
Pr = 0.7;
Dab = 0.1; //[From Wilke-Lee equation, square cm/s]
Sc = u/(row*Dab*10^-4);

// Using equation 8.16

A_1 = C_s1*(Sc/Pr)^0.567; // [kJ/kg.K]
// here A_1 = hg/ky, psychrometric ratio

// Using equation 8.15
//     T_w = T_G1 - (Y_w-Y_1)*lambda_w/(hg/ky)
//    where lambda_w = lambda_1*((1-T_w/T_c)/(1-T_1/T_c))^.38
//    Y_w = P_a/(P_total-P_a)*M_a/M_b
//    P_a = exp(A-B/(T+c))

deff('[z] = f15(T_w)',' z = T_w - T_G1 + ((exp(13.9320 - (3057/(T_w - 55.52)))/(P_total - (exp(13.9320 - (3057/(T_w - 55.52))))))*(M_a/M_b) - Y_1)*(lambda_1*((1-T_w/T_c)/(1-T_1/T_c))^.38/A_1)');
T_w = fsolve(273,f15); // [K]
printf("Wet bulb Temperature is %f K\n",T_w);

// Now using equation 8.2

P_a = exp(13.9320-(3057/(T_w-55.52))); // [kPa]
Y_w = P_a/(P_total-P_a)*M_a/M_b; // [kg vapor/kg dry air]

printf("Absolute humidity is %f kg vapor/kg dry air\n",Y_w);
```

Scilab code Exa 8.7 Water Cooling Using Air Graphical Solution

```
clear;
clc;

// Illustration 8.7
// Page: 493

printf('Illustration 8.7 -  Page: 493\n\n');


// solution

//****Data****//
L_min = 2.27;// [kg/square m.s]
G_min = 2;// [kg/square m.s]
L2_prime = 15;// [kg/s]
Templ2 = 318;// [K]
Tempg1 = 303;// [Entering air dry bulb, K]
Tempw1 = 297;// [ Entering air wet bulb, K]
Kya = 0.90;// [kg/cubic m.s]
//*******//

H1_prime = 72.5;// [kJ/kg dry air]
Y1_prime = 0.0190;// [kg water/kg dry air]
Templ1 = 302;// [K]
Cal = 4.187;// [kJ/kg]

// Equilibrium Data:
// Data  = [Temp.(K),H_star(kJ/kg)]
Data_star = [302 100;305.5 114;308 129.8;310.5 147;313 166.8;315.5 191;318 216];

// The operating line for least slope:
H2_star = 210;// [kJ/kg]
Data_minSlope = [Templ1 H1_prime;Templ2 H2_star];
deff('[y] = f14(Gmin)','y = ((L2_prime*Cal)/Gmin)-((H2_star-H1_prime)/(Templ2-Templ1))');
Gmin = fsolve(2,f14);// [kg/s]
Gs = 1.5*Gmin;// [kg/s]
```

```

// For the Operating Line:
y = deff('[y] = f15(H2)','y = ((H2-H1_prime)/(Templ2-Templ1))-((L2_prime*Cal)/Gs)');
H2 = fsolve(2,f15);// [kJ/kg dry air]
Data_opline = [Templ1 H1_prime;Templ2 H2];

scf(4);
plot(Data_star(:,1),Data_star(:,2),Data_minSlope(:,1),Data_minSlope(:,2),Data_opline(:,1),Data_opline(:,2));
xgrid();
legend('Equilibrium line','Minimum Flow Rate Line','Operating Line');
xlabel("Liquid Temperature, K");
ylabel("Enthalphy Of Air Water vapour, kJ / kg dry air");

// Tower cross section Area:
Al = L2_prime/L_min;// [square m]
Ag = Gs/G_min;// [square m]
A = min(Al,Ag);// [square m]
printf("Cross sectional is %f square m\n",A);

// Data from operating line:
// Data1 = [Temp.(K),H_prime(kJ/kg)]
Data1 = [302 72.5;305.5 92;308 106.5;310.5 121;313 135.5;315.5 149.5;318 164.2];

// Driving Force:
Data2 = zeros(7,2);
// Data2 = [Temp[K],driving Force]
for i = 1:7
    Data2(i,1) = Data1(i,1);
    Data2(i,2) = 1/(Data_star(i,2)-Data1(i,2));
end

// The data for operating line as abcissa is plotted
```

```
    against driving force;
Area = 3.28;
// From Eqn. 7.54
deff('[y] = f16(Z)','y = Area-(Kya*Z/G_min)');
Z = fsolve(2,f16);
printf("The height of tower is %f m\n",Z);
NtoG = 3.28;
HtoG = G_min/Kya;// [m]

// Make up water
// Assuming the outlet air is essentially saturated:
Y2_prime = 0.048;// [kg water/kg dry air]
H2 = 164.2; // [kJ/kg dry air]
// This corresponds to an exit-air temperature of 312.8 K

// Approximate rate of evaporation
R = Gs*(Y2_prime-Y1_prime);
printf("Rate of evaporation is %f kg/s\n",R);
```

Scilab code Exa 8.8 Water Cooling Using Air Numerical Solution

```
clear;
clc;

// Illustration 8.8
// Page: 495

printf('Illustration 8.8 -  Page: 495\n\n');

// solution (a)
printf('Illustration 8.8 (a) -  Page: 495\n\n');

// a - water vapor   b - air
//****Data****//
```

```
T_L2 = 314; // [inlet water temperature, K]
T_L1 = 303; // [outlet water temperature, K]
T_d = 306; // [dry bulb temperature ,K]
T_w1 = 298; // [wet bulb temperature, K]
Z = 3; // [packed tower depth, m]
G_x = 3; // [mass velocity, kg/square m.s]
G_s =2.7; // [mass velocity, kg/square m.s]
//*****//

T_o = 273; // [reference temperature, K]
C_al = 4.187; // [kJ/kg.K]
C_pb = 1.005; // [kJ/kg.K]
C_pa = 1.884; // [kJ/kg.K]
P_total = 101.325; // [kPa]
lambda_0 = 2502.3; // [kJ/kg]
M_a = 18.02; // [gram/mole]
M_b = 28.97; // [gram/mole]

// Equilibrium Data:
// Data = [Temp.(K),H_eqm(kJ/kg)],[H_eqm - Equilibrium gas enthalpy]
Data_eqm = [273 9.48;283 29.36;293 57.8;303 99.75;313 166.79;323 275.58;333 461.5];

scf(4);
plot(Data_eqm(:,1),Data_eqm(:,2));
xgrid();
legend('Equilibrium line');
xlabel("Liquid Temperature, K");
ylabel("Enthalphy Of Air Water vapour, kJ / kg dry air");

P_a = exp(16.3872-(3885.7/(T_w1-42.98))); // [kPa]
Y_m1 = P_a/(P_total-P_a)*(M_a/M_b); // [kg water/kg dry air]
H_g1 = C_pb*(T_w1-T_o) + Y_m1*(C_pa*(T_w1-T_o)+lambda_0); // [Enthalpy of saturated mixture, kJ/kg dry air]
```

```

// From overall energy balance
H_g2 = H_g1 + G_x*C_al*(T_L2-T_L1)/G_s; // [Enthalpy of exit air , kJ/kg]

// For calculation of mass transfer unit , Ntog
// Data1 = [T_L1 H_g1 ,..... ,T_L2 H_g2]
Data1 = zeros(10,2);
deltaT = (T_L2-T_L1)/9;
for i = 1:10
    Data1(i,1) = T_L1 + (i-1)*deltaT;
    Data1(i,2) = H_g1 + G_x*C_al*(Data1(i,1)-T_L1)/G_s;
end

// Data for enthalpy of exit air at different temperature varying from T_L1 to T_L2 , operating line
Data1 = [303 76.17;304.22 81.85;305.44 87.53;306.67 93.22;307.89 98.91;309.11 104.59;310.33 110.28;311.56 115.96;312.78 121.65;314 127.35];

// Data of equilibrium gas enthalpy at different temperature varying from T_L1 to T_L2 from the above equilibrium graph
Data2 = [303 100;304.22 107.93;305.44 116.12;306.67 124.35;307.89 132.54;309.11 140.71;310.33 148.89;311.56 157.14;312.78 165.31;314 177.67];

// Driving force
Data3 = zeros(10,2);
// Data3 =[Equilibrium gas enthalpy , driving force]
for i = 1:10
    Data3(i,1) = Data1(i,2);
    Data3(i,2) = 1/(Data2(i,2)-Data1(i,2));
end

// The data for Equilibrium gas enthalpy as abcissa
```

```
is plotted against driving force
Area = 1.642;
N_tog = 1.642;
H_tog = Z/N_tog; // [m]

// Overall volumetric mass-transfer coefficient, K_ya
K_ya = G_s/H_tog;
printf("Overall volumetric mass-transfer coefficient is %f kg/cubic m.s\n\n",K_ya);

// Solution (b)
printf('Illustration 8.8 (b) - Page: 495\n\n');

T_w2 = 288; // [New entering-air wet-bulb temperature, K]
P_a2 = exp(16.3872-(3885.7/(T_w2-42.98))); // [kPa]
Y_m2 = P_a2/(P_total-P_a2)*(M_a/M_b); // [kg water/kg dry air]
H_g11 = C_pb*(T_w2-T_o) + Y_m2*(C_pa*(T_w2-T_o)+lambda_0); // [Enthalpy of saturated mixture, kJ/kg dry air]

// the change in water temperature through the tower must remain the same as in part (a), namely T_L2b-T_L1b = 11K
// Since N_tog is a function of both water temperatures(T_L1',T_L2'), this provides the second relation needed to calculate the values of T_L2b and T_L1b
// The two equations are solved simultaneously by trial and error method, from above we get T_L1' = 297K
T_L1b = 297; // [K]
T_L2b = T_L1b + 11; //[K]
S =  T_L1b - T_w2; // [wet bulb temperature approach, K]
printf("The outlet water temperature and wet bulb
temperature approach is %f K and %f K
respectively ”,T_L1b,S);
```

Chapter 9

Membranes and Other Solid Sorption Agents

Scilab code Exa 9.1 Liquid Flux in Tubular Membrane

```
clear;
clc;

// Illustration 9.1
// Page: 508

printf('Illustration 9.1 - Page: 508\n\n');

// solution
//*****Data*****//
// A-solute   B-solvent
ci_f = 50; // [feed side concentration, mole/cubic m]
ci_p = 15; // [permeate side concentration, mole/cubic m]
t = 2*10^-4; // [membrane thickness, cm]
q_A = 176; // [permeability, barrer]
D = 4*10^-1; // [tube inside diameter, cm]
D_A = 5*10^-5; // [diffusuvity, square cm/s]
```

```
Re = 20000; // [reynolds number]
Sc = 450; // [Schmidt number]
mtc_p = 0.12; // [square cm/s]
//*****//

// From equation 9.6, 1 barrer = 8.3*10^-9 square cm/s
// Therefore
q_A = q_A*8.3*10^-9; // [square cm/s]
Q_A = q_A/t; // [permeance, cm/s]
// The mass-transfer coefficient on the feed side is from equation (2-75) for turbulent flow of a liquid inside a circular pipe:
Sh = 0.023*Re^0.83*Sc^(1/3);
// Now mass transfer coefficient
k_af = Sh*D_A/D; // [cm/s]
// Total resistance to mass transfer
res_total = (1/k_af)+(1/Q_A)+(1/mtc_p); // [s/cm]
// Transmembrane flux of solute A
N_A = (ci_f-ci_p)/(res_total*100); // [mole/square m.s]

printf("The transmembrane flux of solute A is %e mole/square m.s\n\n",N_A);

percent_mem_res = ((1/Q_A)/res_total)*100; // [%]
printf("Membrane resistance is %f percent of the total\n\n",percent_mem_res);
```

Scilab code Exa 9.2 Oxygen Enriched Air by Gas Permeation

```
clear;
clc;

// Illustration 9.2
// Page: 511

printf('Illustration 9.2 -  Page: 511\n\n');

// solution
//*****Data*****//
//  A-oxygen    B-nitrogen
t = 0.2*10^-6; // [m]
qA = 3.97*10^-13; // [mole/m.s.kPa]
qB = 0.76*10^-13; // [mole/m.s.kPa]
v = 1; // [Air flow rate at STP, cubic m/s]
Pp = 0.1*10^6; // [Pa]
R = 8.314 // [cubic m.Pa/mole.K]
T = 298; // [K]
Pf = 1*10^6; // [Pa]
//*****//
// Using equation 9.14
alphaA = qA/qB;
QA = qA/t; // [mole/square m.s.kPa]
// molar flow rate
nf = v*1000/(22.4); // [mole/s]
r = Pp/Pf; // [pressure ratio]
QB = qB/t; // [mole/square m.s.kPa]
alphaid = QA/QB;
xFa = 0.21;
xFb = 0.79;

// For Q = 0.1
Q1 = 0.1
    // Solution of simultaneous equation
function[f]=F(e)
    f(1) = e(1) - (e(3)*(1-e(2)))/((e(2)*(1-e(3))));
    f(2) = e(2) - (xFa - (e(3)*Q1))/(1-Q1);
    f(3) = e(1) - (alphaid*(e(2)*(e(1)-1)+1- (r*e(1)
       )))/(e(2)*(e(1)-1)+1 - r);
    funcprot(0);
endfunction
// Initial guess
```

```
e = [4 0.13 0.4];
y = fsolve(e,F);
alpha1 = y(1);
Xa1 = y(2);
Ya1 = y(3);
Am1 = Ya1*Q1*nf/(QA*(Xa1*Pf-Ya1*Pp))*1000; // [square m]

// For Q = 0.2
Q2 = 0.2
    // Solution of simultaneous equation
function[f]=F(e)
    f(1) = e(1) - (e(3)*(1-e(2)))/((e(2)*(1-e(3))));
    f(2) = e(2) - (xFa - (e(3)*Q2))/(1-Q2);
    f(3) = e(1) - (alphaid*(e(2)*(e(1)-1)+1- (r*e(1))))/(e(2)*(e(1)-1)+1 - r);
    funcprot(0);
endfunction
// Initial guess
e = [4 0.13 0.4];
y = fsolve(e,F);
alpha2 = y(1);
Xa2 = y(2);
Ya2 = y(3);
Am2 = Ya2*Q2*nf/(QA*(Xa2*Pf-Ya2*Pp))*1000; // [square m]

// For Q = 0.9
Q9 = 0.9
    // Solution of simultaneous equation
function[f]=F(e)
    f(1) = e(1) - (e(3)*(1-e(2)))/((e(2)*(1-e(3))));
    f(2) = e(2) - (xFa - (e(3)*Q9))/(1-Q9);
    f(3) = e(1) - (alphaid*(e(2)*(e(1)-1)+1- (r*e(1))))/(e(2)*(e(1)-1)+1 - r);
    funcprot(0);
endfunction
// Initial guess
```

```
e = [4 0.13 0.4];
y = fsolve(e,F);
alpha9 = y(1);
Xa9 = y(2);
Ya9 = y(3);
Am9 = Ya2*Q9*nf/(QA*(Xa9*Pf-Ya9*Pp))*1000; // [square m]

// Similarly for Q =0.3......0.9 , Xa, Ya, alpha and Am are calculated
// Therefore we obtained
// Solution = [Q,alpha ,Xa,Ya]
Solution = zeros(9,4);
Solution = [0.1 4.112 0.181 0.475;0.2 4.062 0.156 0.428;0.3 4.018 0.135 0.385;0.4 3.98 0.118 0.348;0.5 3.949 0.105 0.315;0.6 3.922 0.093 0.288;0.7 3.9 0.084 0.264;0.8 3.881 0.077 0.243;0.9 3.864 0.07 0.226];
Am = [8037;17074;26963;37531;48618;60099;71876;83879;96056;];

disp(Solution);
disp(Am);

printf("The maximum oxygen content of the permeate (%f percent) occurs with the smallest cut (Q = 0.1).\n\n",Ya1*100);
printf("The maximum nitrogen content of the retentate (%f percent) occurs at the largest cut (Q = 0.9).\n\n",(1-Xa9)*100);

printf('The membrane area requirements are very large (e.g, Am = 60,100 square m for Q = 0.6) even though the volumetric flow rate of air is relatively small)');
```

Scilab code Exa 9.4 Freundlich and Langmuir Adsorption Isotherms

```
clear;
clc;

// Illustration 9.4
// Page: 520

printf('Illustration 9.4 - Page: 520\n\n');

// solution
//*****Data*****//

Pexp = [0.276;1.138;2.413;3.758;5.240;6.274;6.688;]; // [MPa]
V = [45.5;91.5;113;121;125;126;126;]; // [cubic cm of CH4/gram carbon]
Ma = 16; // [gram/mole]
Vstp = 22.4; // [L/mole]
q = V*Ma/Vstp; // [mg/g]

// Linearize data for Langmuir isotherm
y = Pexp/q;
y = [0.0030667;0.01264;0.02681;0.0417556;0.0582;0.06971;0.07431;];

W = [Pexp,y];
y = 0:0.001:0.01
scf(1);
plot(W(:,1),W(:,2));
xgrid();
xlabel("Pexp, MPa");
ylabel("y, MPa.mg/g");

```

```
// Now qm = 1/(slope of Pexp v/s y curve)
// From graph of Pexp v/s y, the slope is
s = 0.01022;
// And intercept
i = 5.4865*10^-3;
qm = 1/s; // [mg/g]
K = 1/(qm*i); // [1/MPa]
// Therefore
// qlp = K*qm*p/(1+Kp)
printf("Data for Langmuir isotherm are K = %f MPa^-1 and qm = %f mg/g\n\n",K,qm);

// Linearize data for Freundlich isotherm
// y1 = log(q/(mg/g)) , x1 = log(Pexp/MPa)
y1 = log(q);
x1 = log(Pexp);

X = [x1,y1];
x1 = -2:0.571:1;
y1 = 3:0.285:5;
scf(2);
plot(X(:,1),X(:,2));
xgrid();
xlabel("log(Pexp/(Mpa))");
ylabel("log(q/(mg/g))");

// From graph of log(q) v/s log(Pexp)
// slope = 0.31
s = 0.31;
// and intercept is
i = 4;
// Therefore n = 1/slope
n = 1/s;
k = exp(i); // [(mg CH4/g of carbon.MPa^(-1/n)]
printf("Data for Freundlich isotherm are n = %f and k = %f\n\n",n,k);

// Therefore
// qFp = k*(p/1 Mpa)^(1/n)
printf('Figure 9.6(b) shows a q-p plot of the experimental data and the corresponding predictions of the Langmuir and Freundlich isotherms. It is evident from the plot that in this case, the Langmuir isotherm fits the data significantly better than the Freundlich isotherm.')
```

Scilab code Exa 9.5 Ion Exchange Equilibrium

```
clear;
clc;

// Illustration 9.5
// Page: 526

printf('Illustration 9.5 -  Page: 526\n\n');

// solution
// A-Na+    B-Cu+2
// Using the data from Table 9.1
KA = 1.98;
KB = 3.47;

Q = 2.4; // [eq/L of resin]
// Charge ratio is 'n'
n = 2;
C = 0.05; //[total concentration, eq/L]
// From equ 9.48
KAB = KB/KA;
// From equ 9.47
// ya*(1-xa)^2/(xa*(1-ya)^2) = KAB*Q/C = T
T = KAB*Q/C;
// Substituting values of xA in the range 0.1< xa
```

```
<1.0, we generate the        // distribution curve
for i=1:19
    deff('[y] = f16(ya)','y = ya*(1-i*0.05)^2/(i*0.05*(1-ya)^2) - T');
    ya(i) = fsolve(0.99,f16);
    disp(ya(i));
end

xa = [0.05;0.1;0.15;0.2;0.25;0.3;0.35;0.4;0.45;0.5;0.55;0.6;0.65;0.7;0

A = [xa,ya];

scf(1);
plot(A(:,1),A(:,2));
xgrid();
xlabel("xa,Fraction of Cu+2 in Solution");
ylabel("ya,Fraction of CuR2 in resin");

printf('The curve is similar in shape to an adsorption isotherm of the very favorable type.\n\n');
```

Scilab code Exa 9.8 Fixed Bed Scale Up Using LUB

```
clear;
clc;

// Illustration 9.8
// Page: 535

printf('Illustration 9.8 -  Page: 535\n\n');

// solution
// From example 9.7
```

```
alpha = 0.891;
// For bed length Z = 1.829
Z1 = 1.829; // [m]
LUB = (1-alpha)*Z1; // [length of unused bed, m]
// For this bed length
tb1 = 139.7; // [min]
// If the bed length is increased to Z2 = 3 m
Z2 = 3; // [m]
// New break through time will be given by equation 9.64
tb2 = tb1*(Z2/Z1)*(1-LUB/Z2)/(1-LUB/Z1); // [min]

printf(”The new time of breakthrough assuming constant LUB is %f minute.\n\n”,tb2);
```

Scilab code Exa 9.9 Ion Exchanger Ideal Break Time

```
clear;
clc;

// Illustration 9.9
// Page: 536

printf('Illustration 9.9 – Page: 536\n\n');

// solution
F = 7; // [water flow rate, L/s]
Z = 3; // [m]
d = 2.6; // [m]
A = %pi*d^2/4; // [cross sectional area, square m]
vo = 0.013; // [superficial velocity, m/s]

cf = 7*10^-3; // [Ca2+ ion concentration, eq/L]
qstar_F = 2.9; // [cation capacity, eq/kg]
rowp = 1.34; // [kg/L]
```

```
e = 0.38; // [porosity]
// From equation 9.66
t_star = Z*qstar_F*rowp*(1-e)/(vo*cf*3600); // [hour]

printf("The ideal breakthrough time for the ion exchanger is %f hour.\n\n",t_star);
```

Scilab code Exa 9.11 Dialysis for Sulfuric Acid Purification

```
clear;
clc;

// Illustration 9.11
// Page: 542

printf('Illustration 9.11 - Page: 542\n\n');

// solution
//*****Data*****//
mtc = 0.02; // [mass transfer coefficient, cm/min]
p = 0.03; // [permeance, cm/min]
F = 1; // [cubic m/h]
W = 1000; // [water wash rate, kg/h]
// Density of 25% H2SO4 solution at 298 K is
d1 = 1175; // [kg/cubic m]
x = 0.25; // [fraction of H2SO4 in solution]
cF = 294; // [kg/cubic m]
//*****//

K = (1/p+1/mtc)^-1; // [overall mass transfer coefficient, cm/min]

// Flow of H2SO4 in feed
F_sul = F*d1*x; // [kg/h]
```

```

// For 60% recovery and rest in dialysate
yr = 0.60;
yd = 0.40;
// Transmembrane flow of acid
Ft = F_sul*yr; // [kg/h]
// From the given water transport number, Transmembrane counterflow of water
Fw = Ft*0.8; // [kg/h]

// Now inlet and outlet concentration  from material balances
// Flow of acid in dialysate
Fad = F_sul*yd; // [kg/h]

// Total dialysate flow
D = F*d1-Ft+Fw; // [kg/h]
x_aD = Fad/D; // [mass fraction of acid in dialysate]
disp(x_aD);
// Density of 10.3 wt % aqueous solution of sulfuric acid at 298K is
d2 = 1064; // [kg/cubic m]

cR = x_aD*d2; // [kg/cubic m]
// Flow of acid in diffusate
Fd = Ft; // [kg/h]
// Total Diffusate flow
Di = 1000-Fw+Fd; // [kg/h]
x_aDi = Fd/Di; // [mass fraction acid in diffusate]
disp(x_aDi);
// Density of 17 wt % aqueous solution of sulfuric acid at 298 K is
d3 = 1114; // [kg/cubic m]

cP = x_aDi*d3; // [kg/cubic m]
// At the free end of dialyzer
deltaC1 = cF-cP; // [kg/cubic m]
```

```
// At the dialysate end
deltaC2 = cR-0; // [kg/cubic m]
lmdf = (deltaC2-deltaC1)/(log(deltaC2/deltaC1)); // [Log-mean driving force, kg/cubic m]

// Therefore
Am = Fd*100/(K*lmdf*60);

printf("The membrane area required is %f square m.\n\n",Am);
```

Scilab code Exa 9.12 Water Desalinization by Reverse Osmosis

```
clear;
clc;

// Illustration 9.12
// Page: 545

printf('Illustration 9.12 - Page: 545\n\n');

// solution
//*****Data*****//
// A-NaCl
vo = 0.05; // [superficial velocity of water in the shell, m/s]
T = 298; // [K]
Pf = 70; // [bar]
Pp = 3; // [pressure at permeate side, bar]
p = 1.1*10^-5; // [water permeance, g/square cm.s.bar]
R1 = 0.97; // [salt rejection]
R = 8.314;
xAf = 0.02; // [fraction of NaCl in feed side]
xAp = 0.0005; // [fraction of NaCl in permeate side]
```

```
MA = 58.5; // [gram/mole]
//*****//

printf('Illustration 9.12(a) - Page: 545\n\n');
// Solution(a)

deltaP = Pf-Pp; // [bar]
// Density of both feed and permeate is 1 g/cc
df = 1000; // [kg/cubic m]
dp = df;
// Bulk feed salt concentration
csf = xAf*2*1000/MA; // [kmole/cubic m]
// Bulk permeate salt concentration
csp = xAp*2*1000/MA; // [kmole/cubic m]

// From equation 9.76
pif = R*T*csf/100; // [osmotic pressure at feed side, bar]
pip = R*T*csp/100; // [osmotic pressure at permeate side, bar]
deltapi = pif-pip; // [bar]

Y = deltaP-deltapi; // [bar]
// Transmembrane flux of water
nH2O = p*Y*10^-3/(df*(10^-4*1/(60*60*24))); // [cubic m/square m.day]

printf("The transmembrane flux of water is %f cubic m/square m.day.\n\n",nH2O);

printf('Illustration 9.12(b) - Page: 546\n\n');
// Solution(b)

// Properties of water are
dw = 1000; // [kg/cubic m]
uw = 0.9*10^-3; // [kg/m.s]
DA = 1.6*10^-9; // [Diffusivity of NaCl in water, square m/s]
```

```
d = 290*10^-6; // [outside diameter of fibres, m]
phi = 0.4;
// For a superficial velocity of 5 cm/sec
Re = dw*vo*d/uw; // [Renoylds number]
Sc = uw/(dw*DA); // [Schmidt number]
Sh = 8.63; // [Sherwood number]
// Therefore
ks = Sh*DA/d; // [m/s]
// From equation 9.81
t = nH2O*R1/(ks*24*60*60);
printf(”The concentration polarization factor is %f.\n\n”,t);
```

Scilab code Exa 9.13 Ultrafiltration of Cheese Whey Proteins

```
clear;
clc;

// Illustration 9.13
// Page: 548

printf(’Illustration 9.13 – Page: 548\n\n’);

// solution
//*****Data*****//
// w–water a–proteins
T = 293; // [K]
d = 2; // [diameter of tube, cm]
dw = 1; // [g/cubic cm]
uw = 0.01; // [cP]
Da = 4*10^-7; // [Diffusivity of proteins, square cm/s]
vo = 1.5*100; // [m/s]
Qm = 250*10^-3/3600*100; // [water permeance, cm/s.atm]
```

```
cR = 40; // [g/L]

printf('Illustration 9.13(a) - Page: 549\n\n');
// Solution(a)

v = 25*10^-3/3600*100; // [cm/s]

Re = d*vo*dw/uw; // [Renoylds number]
Sc = uw/(dw*Da); // [Schmidt number]
Sh = 0.0048*Re^0.913*Sc^0.346; // [Sherwood number]
ks = Sh*Da/d;
// From equation 9.87
cS = cR*exp(v/ks); // [g/L]

// From figure 9.12
pi1 = 2; // [osmotic pressure, atm]
// For 100% rejection deltapi = pi1 because pi2 = 0
// Therefore
deltapi = pi1; // [atm]
// From equation 9.83
deltaP = deltapi+(v/Qm);
printf("The required pressure differential to produce a water transmembrane volume flux of 25 L/square m.h when the membrane is clean is %f atm.\n\n",deltaP);


printf('Illustration 9.13(b) - Page: 549\n\n');
// Solution(b)

// Membrane permeance is reduced fivefold by fouling
Qm = Qm/5; // [cm/s.atm]
// Here deltaP remains same
// Equations 9.83 and 9.87, and the osmotic pressure data of Figure 9.12 must be solved simultaneously by trial and error to calculate new values for these three variables.
// The results are
```

```
cS2 = 213; // [g/L]
deltapi2 = 1.63; // [atm]
v2 = 6.53*10^-4; // [cm/s]
printf("The water flux if the applied pressure
    differential remains the same as calculated in
    part (a) is %f L/square m.hr.",v2
    *1000*10^-2*3600);
```