MULTICOMPONENT BATCH DISTILLATION COLUMN SIMULATION AND
STATE OBSERVER DESIGN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

UĞUR YİLDİZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

IN

THE DEPARTMENT OF CHEMICAL ENGINEERING

DECEMBER 2002
Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Timur Doğu  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Canan Özgen  
Supervisor

Exminating Committee Members

Prof. Dr. Türker Gürkan

Prof. Dr. Canan Özgen

Prof. Dr. Kemal Leblebicioğlu

Prof. Dr. Güniz Gürüz

Prof. Dr. Bilgin Kısıkürek
ABSTRACT

MULTICOMPONENT BATCH DISTILLATION COLUMN SIMULATION AND STATE OBSERVER DESIGN

Yıldız, Uğur

M.S., Department of Chemical Engineering

Supervisor: Prof. Dr. Canan Özgen

December 2002, 180 pages

In the control of batch and continuous distillation columns, one of the most challenging problem is the difficulty in measuring compositions. This problem can be handled by estimating the compositions from readily available online temperature measurements using a state observer. The aim of this study is to design a state observer that estimates the product composition in a multicomponent batch distillation column (MBDC) from the temperature measurements and to test this observer using a batch column simulation. To
achieve this, first a model for MBDC is prepared and compared with the data from literature where a case column is utilized. After checking the validity of the simulation package, it is used as a fictitious process for the performance evaluations.

In the second phase of the study, an extended Kalman Filter (EKF) is designed by utilizing a simplified model of MBDC and it is implemented for performance investigation on the case column with 8 trays separating the mixture of cyclohexane, n-heptane and toluene. The simplified model utilized in EKF results in response, which have some deviation with rigorous model, mainly due to the simplification of vapor-liquid equilibrium relationship. In the performance evaluation, the tuning parameters of EKF; the diagonal terms of process noise covariance matrix and the diagonal terms of measurement model noise covariance matrix are changed in the range of $50 - 1 \times 10^{-7}$ and $0.5 - 5 \times 10^{8}$ and the optimum values are found as $0.00001$ and $5000$, respectively. The effect of number of measurement points is also investigated with a result of number of component measurements. The effect of measurement period value is also studied and found that it has a major effect on the performance which has to be determined by the available computational facilities. The control of the column is done by utilizing the designed EKF estimator and the estimator is successfully used in controlling the product purities in MBDC under variable reflux-ratio operation.

Keywords: Batch Distillation, Simulation, State Observer, Kalman Filter.
ÖZ

ÇOK BİLEŞENLİ KESİKLİ DAMİTMA

KOLONU BENZETİMİ VE DURUM

GÖZLEMLEYİCİ TASARIMI

Yıldız, Üğur

Yüksek Lisans, Kimya Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Canan Özgen

Arahlık 2002, 180 sayfa

Kesikli ve sürekli damıtma işlemlerinin denetimindeki en önemli sorunlardan biri, derişim ölçümlerindeki zorluklardır. Bu problem, durum gözlemleyicisi (state observer) kullanılarak, kolaylıkla ölçülebilen gerçek zamanlı sıcaklık verilerinden derişim tahminleri yapılarak çözülebilmektedir. Bu çalışmada amaç, çok bölünenli bir kesikli damıtma kolonundaki ürün derişimlerini sıcaklık ölçümlerinden tahmin edecek bir durum gözlemleyicisi tasarlamak ve geliştirilen bir kolon benzetimi kullanarak, bu gözlemleyiciyi sınamaktır. Bunu gerçekleştirebilmek için ilk
olarak çok bileşenli bir kesikli damıtma kolon modeli hazırlanmış ve örnek bir kolonun kullanılıldığını literatür verileriyle karşılaştırılmıştır. Benzetim programının geçerliliğinin sınanmasından sonra, başarım değerlendirme işlemlerinde bu program kurmaca bir süreç olarak kullanılmıştır.

Çalışmanın ikinci aşamasında, Geliştirilmiş Kalman Filitre (GKF), kolonun basitleştirilmiş bir modeli kullanılarak tasarlanmış ve performansının incelenmesi için cyclo-hexane, n-heptane ve toluene karışımının ayrıştırılması için kullanılan 8 tepsili bir kesikli damıtma kolonunda kullanılmıştır. GKF’de kullanılan basitleştirilmiş model, başlıca nedeni buhar - sıvı denge ilişkisinin basitleştirilmesinden doğan, detaylı modelden bazı sapmalar içeren bir sonuç vermiştir. Performans değerlendirme işlemlerinde, GKF ayar parametreleri olan, süreç gürültüsü kovarians matriksi köşegen elemanları ve süreç ölçüm model gürültüsü kovarians matriksi köşegen elemanları, \(50 - 1 \times 10^{-7}\) ve \(0.5 - 5 \times 10^{8}\) değerleri arasında değiştirilerek, en iyi değerler 0.00001 ve 5000 olarak bulunmuştur. Ayrıca, ölçüm sayısının etkisi incelenerek bileşen sayısı kadar ölçüm sonucuna varılmıştır. Ölçüm aralığının etkisi de incelenerek performans üzerinde önemli bir etkiye sahip olduğu gözlemlemiştir ve mevcut hesaplama imkanları doğrultusunda belirlenmesi gerektiği tespit edilmiştir. Kolon denetimi tasarlanan GKF gözlemleyici kullanılarak yapılmış ve gözlemleyici, değişken geri döngü işletimindeki çok bileşenli damıtma kolonu ürün safliğını denetiminde başarı ile kullanılmıştır.

Anahtar Kelimeler: Kesikli Damıtma, Benzetim, Durum Gözlemleyici, Kalman Filitre.
This thesis is dedicated to my family.
I would like to thank my supervisor Prof. Dr. Canan Özgen for counselling me in all grades of my thesis and for her constant encouragement motivating me, especially at those times, when I sank into despair. I would like to thank Prof. Dr. Kemal Leblebioğlu for his valuable suggestions and stimulating discussions.

I would like to thank; my best friend Umut Oran, who was with me all the way and helped me in every possible way; Mustafa Tekin Dokucu, who never hesitated to share his innovative solutions; Engin Aytürk for sharing his precious gaiety; IPC Team members, Mesut Kaya, Üster Akbay, Almıla Bahar, Evren Güner, Necati Doğru, and Işık Aşar for making the jobs simpler.

Last but certainly not least, I would like to thank my family for supporting me and always making me feel comfortable at all times.
# TABLE OF CONTENTS

ABSTRACT ................................................................. iii
ÖZ ................................................................. v
DEDICATION ............................................................ vii
ACKNOWLEDGMENTS ................................................... viii
TABLE OF CONTENTS ...................................................... ix
LIST OF TABLES ........................................................ xiii
LIST OF FIGURES ........................................................ xiv
LIST OF SYMBOLS ........................................................ xvi

CHAPTER

1  INTRODUCTION ......................................................... 1

2  LITERATURE SURVEY .................................................. 7
   2.1 Studies on Batch Distillation Modelling and Simulation ........ 7
   2.2 Studies on Estimation of Compositions in Distillation Columns .................. 13

3  BATCH DISTILLATION COLUMN MODELLING ...................... 22
   3.1 Operation of a MBDC .............................................. 23
   3.2 Modelling Assumptions ........................................... 26
   3.3 MBDC Model Derivation ........................................... 28
      3.3.1 Overhead Flowrates for Finite Reflux Ratio ............ 31
      3.3.2 Overhead Flowrates for Total Reflux .................. 33
      3.3.3 Molar Holdup Calculations ............................... 33
      3.3.4 Algebraic Equations and Calculation of Physical Parameters ..................... 34
   3.4 Storage Tanks Model ............................................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Initial Conditions</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary of Model Equations</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>KALMAN FILTERING</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>Basics of Kalman Filtering</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Kalman Filtering Theory with a Simple Example</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Linear Dynamic Models</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Kalman Filter Problem Formulation</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Kalman Filter Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Extended Kalman Filter</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>EKF STATE OBSERVER FOR BATCH DISTILLATION COLUMNS</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Observability Criteria for a MBDC</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>EKF State Observer for MBDC</td>
<td>61</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Model Simplification</td>
<td>63</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Linearization of the Simplified Model</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>SIMULATION CODE AND CASE COLUMN</td>
<td>71</td>
</tr>
<tr>
<td>6.1</td>
<td>Simulation Code</td>
<td>71</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Thermodynamic Library</td>
<td>74</td>
</tr>
<tr>
<td>6.1.2</td>
<td>M-files for Nonlinear Model Used in EKF</td>
<td>75</td>
</tr>
<tr>
<td>6.1.3</td>
<td>M-files for Linearized Model Used in EKF</td>
<td>76</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Main Simulation Program</td>
<td>76</td>
</tr>
<tr>
<td>6.2</td>
<td>Description of Case Column</td>
<td>79</td>
</tr>
<tr>
<td>7</td>
<td>RESULTS AND DISCUSSION</td>
<td>82</td>
</tr>
<tr>
<td>7.1</td>
<td>Rigorous Simulation Model for MBDC</td>
<td>82</td>
</tr>
<tr>
<td>7.2</td>
<td>The Test of the Model Used in EKF</td>
<td>83</td>
</tr>
<tr>
<td>7.3</td>
<td>Implementation of EKF</td>
<td>86</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Effect of Tuning Parameters of EKF</td>
<td>88</td>
</tr>
<tr>
<td>7.3.1.1</td>
<td>Effect of the Diagonal Terms of Process Noise Covariance Matrix, ( q )</td>
<td>88</td>
</tr>
<tr>
<td>7.3.1.2</td>
<td>Effect of the Diagonal Terms of Measurement Model Noise Covariance Matrix, ( r )</td>
<td>94</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Effect of Number of Measurement Points</td>
<td>98</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Effect of Measurement Period, ( \Delta t_m )</td>
<td>99</td>
</tr>
</tbody>
</table>
D.4.1 KF_NonlinearModel_Sys.m .......................... 169
D.4.2 KF_NonlinearModel_Out.m .......................... 170
D.4.3 KF_NonlinearModel_Eqilib.m ......................... 172
D.4.4 KF_NonlinearModel_Press.m ......................... 173

D.5  EKF Linearized Model Code .............................. 173
D.5.1 KF_LinearizedModel_SysF.m .......................... 173
D.5.2 KF_LinearizedModel_SysH.m .......................... 176
D.5.3 KF_LinearizedModel_Eqilib.m ......................... 177
D.5.4 KF_LinearizedModel_Press.m .......................... 178
D.5.5 KF_LinearizedModel_CalcDyDx.m ...................... 178
D.5.6 KF_LinearizedModel_CalcTXD.m ...................... 179
LIST OF TABLES

3.1 Assumptions made in the model development 28
3.2 Summary of MBDC Rigorous Model Equations 38
3.3 Summary of MBDC Rigorous Model Equations (cont’d) 39
4.1 Equations of Continuous-Discrete Extended Kalman Filter 58
5.1 Summary of MBDC Simplified Model Equations 65
6.1 The overall structure of the simulation code. 73
6.2 Design parameters for the case column 79
6.3 Parameters for the optimal reflux ratio policy. 80
6.4 Additional parameters for the rigorous simulation of the case column 81
6.5 Additional parameters for the simplified model used in EKF 81
7.1 Change of IAE scores with respect to $q$. 88
7.2 Change of IAE scores with respect to $r$. 94
7.3 Change of IAE scores with respect to number of measurements. 99
7.4 Change of IAE scores with respect to location of three measurements. 99
7.5 Change of IAE scores with respect to $\Delta t_m$. 100
LIST OF FIGURES

1.1 Commonly used column configurations. .............................................. 3

3.1 The schematic of a Multicomponent Batch Distillation system. . . 24

4.1 Typical application of Kalman Filter estimator. ............................... 42
4.2 Conditional density of position based on the measurement \( z_1 \) . . . 45
4.3 Conditional density of position based on the measurement \( z_2 \). . . 46
4.4 Conditional density of position based on measurement \( z_1 \) and \( z_2 \). 47
4.5 Propagation of conditional probability density. ............................... 49
4.6 Recursive nature of Kalman Filtering estimation. ........................... 52
4.7 Timing diagram of Kalman Filter algorithm. ................................. 55

6.1 Flow diagram of the main program algorithm. ............................... 77
6.2 Flow diagram of the main program algorithm (cont’d). .................... 78

7.1 Comparison of the reflux-drum compositions from the developed model with the literature ................................................................. 84
7.2 Mismatch between the process and the simplified model used in EKF. ........................................................ .............................................. 85
7.3 Mismatch between the modified process model and the EKF model. 86
7.4 Change of IAE sum with respect to \( q \). ........................................... 89
7.5 Effect of the diagonal terms of process noise covariance matrix for \( q = 1 \times 10^{-7} \). (a) Reflux-drum compositions, (b) Reboiler compositions ........................................................ .............................................. 90
7.6 Effect of the diagonal terms of process noise covariance matrix for \( q = 0.00001 \). (a) Reflux-drum compositions, (b) Reboiler compositions ........................................................ .............................................. 91
7.7 Effect of the diagonal terms of process noise covariance matrix for \( q = 0.1 \). (a) Reflux-drum compositions, (b) Reboiler compositions 92
7.8 Effect of the diagonal terms of process noise covariance matrix for \( q = 50 \). (a) Reflux-drum compositions, (b) Reboiler compositions 93
7.9 Change of IAE sum with respect to \( r \). ........................................... 95
7.10 Effect of the diagonal terms of measurement model noise covariance matrix for \( r = 0.5 \). (a) Reflux-drum compositions, (b) Reboiler compositions ........................................................ .............................................. 96
7.11 Effect of the diagonal terms of measurement model noise covariance matrix for \( r = 5 \times 10^8 \). (a) Reflux-drum compositions, (b) Reboiler compositions ........................................................ .............................................. 97
7.12 Effect of the period of temperature measurements for $\Delta t_m = 1\text{ min.}$ (a) Reflux-drum compositions, (b) Reboiler compositions 101
7.13 Effect of the period of temperature measurements for $\Delta t_m = 20\text{ min.}$ (a) Reflux-drum compositions, (b) Reboiler compositions 102
7.14 Effect of the initial conditions for $X_0 = [1/3, 1/3, 1/3]$ and $P_0 = 0.00001$. (a) Reflux-drum compositions, (b) Reboiler compositions 104
7.15 The closed-loop responses of the MBDC under the scheduling controller with actual composition feedback. (a) Reflux-drum compositions, (b) Reboiler compositions 106
7.16 The closed-loop responses of the MBDC under the scheduling controller with estimated composition feedback. (a) Reflux-drum compositions, (b) Reboiler compositions 108

B.1 Continuous-time system—discrete-time measurement model 118
B.2 Discrete-time system—discrete-time measurement model 119
B.3 Continuous-time system—continuous-time measurement model 120
LIST OF SYMBOLS

\[\begin{align*}
NT & \quad \text{Number of tarys} \\
NC & \quad \text{Number of components} \\
x & \quad \text{State; Liquid fraction, (}mol/mol\text{)} \\
y & \quad \text{Vapor fraction, (}mol/mol\text{)} \\
M & \quad \text{Molar liquid holdup, (}mol\text{)} \\
L & \quad \text{Liquid molar flow rate, (}mol/h\text{)} \\
V & \quad \text{Vapor molar flow rate, (}mol/h\text{)} \\
h & \quad \text{Liquid mixture enthalpy, (}J/mol\text{)} \\
H & \quad \text{Vapor mixture enthalpy, (}J/mol\text{)} \\
Q & \quad \text{Heat load, (}J/mol \cdot h\text{)} \\
D & \quad \text{Distillate flow rate, (}mol/h\text{)} \\
K & \quad \text{Filter gain} \\
P & \quad \text{Pressure, (}Pa\text{); Error Covariance matrix} \\
R & \quad \text{Reflux Ratio} \\
t & \quad \text{Time, (}hr\text{)} \\
T & \quad \text{Temperature, (}K\text{)} \\
u & \quad \text{Process input; Velocity} \\
w & \quad \text{Process noise; Acentric factor} \\
Mw & \quad \text{Molecular weight, (}kg/mol\text{)} \\
X & \quad \text{Estimator State} \\
S & \quad \text{Molar liquid holdup of storage tank, (}mol\text{)} \\
EFF & \quad \text{Murphree tray efficiency} \\
\hat{Cp} & \quad \text{Heat capacity coefficient} \\
\end{align*}\]

\[\begin{align*}
\Delta & \quad \text{Increment} \\
\sigma & \quad \text{Standard deviation} \\
\delta & \quad \text{Numerical approximation for time derivatives} \\
\rho & \quad \text{Liquid density, (}kg/m^3\text{)} \\
u & \quad \text{Volumetric holdup, (}m^3\text{)} \\
\end{align*}\]

Subscripts

\begin{align*}
\_ & \quad \text{Specific property} \\
i & \quad \text{Stage index} \\
j & \quad \text{Component index} \\
f & \quad \text{Feed property} \\
\end{align*}

Superscripts

\begin{align*}
\hat{} & \quad \text{Estimated value} \\
^{-1} & \quad \text{Inverse} \\
\text{T} & \quad \text{Transpose} \\
\_ & \quad \text{Estimated value before measurement} \\
\_ & \quad \text{Estimated value after measurement} \\
guess & \quad \text{Guess value} \\
c & \quad \text{Critical value} \\
boil & \quad \text{Boiling value} \\
\end{align*}

Abbreviations

\begin{align*}
\text{DAE} & \quad \text{Differential Algebraic Equation} \\
\text{PID} & \quad \text{Proportional Integral Derivative} \\
\text{CPU} & \quad \text{Central Processor Unit} \\
\end{align*}
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIMC</td>
<td>Nonlinear Internal Model Control</td>
</tr>
<tr>
<td>r.h.s.</td>
<td>Right Hand Side</td>
</tr>
<tr>
<td>l.h.s.</td>
<td>Left Hand Side</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>NRTL</td>
<td>Nonrandom Two Liquids</td>
</tr>
<tr>
<td>VLE</td>
<td>VaporLiquid Equilibrium</td>
</tr>
<tr>
<td>MBDC</td>
<td>Multicomponent Batch Distillation Column</td>
</tr>
<tr>
<td>ELO</td>
<td>Extended Luenberger Observer</td>
</tr>
<tr>
<td>ISE</td>
<td>Integral Square Error</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

“Batch distillation process is an important separation technique especially used in the fine speciality, pharmaceutical, biochemical and food industries. Because the demand and the uncertainty in specifications for high-value-added and low-volume speciality chemicals has increased recently, the use of batch distillation is becoming increasingly popular” (Barolo et al., 2001 and Kim et al., 1999). Instead of using many continuous columns in series, multiple products can be obtained from a single batch distillation column during a single batch run. Moreover, batch distillation processes can easily handle variations both in the product specifications and in the feed composition. This flexibility of batch distillation processes provides the ability to cope with a market characterized by short product life times and strict specification requirements.

Although the fixed cost required for constructing a batch column is lower than that for a series of continuous column, the operating costs of batch distillation
are higher due to more energy consumption. Since, energy costs are not too influential in production of fine chemicals, “batch distillation is often attractive for this class of products” (Luyben, 1988).

The flexibility of batch distillation allow for configuring the column in a number of different ways. Lotter et al. (1997) gives four commonly used column configurations as conventional column (used in this study), inverted column, middle vessel column and multi-vessel column, which are given in Figure 1.1. In all of these configurations, the main challenging problem is the optimization of design and operating parameters which are given by Luyben (1988) as:

- the number of trays,
- the size of initial charge,
- the reflux ratio as a function of time,
- the switching times between slop-cut distillation and product-cut distillation,
- the method of recycling slop-cut products.

As stated by Diwekar (1996), the first step in the above optimization work is to gain the knowledge of the system which is related to the modelling of the process. After obtaining the optimized parameters, the designed system and its operation strategy need to be examined for its feasibility and performance. In this respect, the simulation of the process permits to realize this test more efficiently than the experimental work does. Therefore, the modelling again becomes inevitable.

Another issue in batch distillation is the operation of the column at the conditions that are obtained from the optimization work. In other words, the
Figure 1.1: Commonly used column configurations.
operation scenario; including reflux ratio policy, switching times, and method of recycling, is required to be realized in a convenient control system. Here, the problem of designing a control system is out of the scope. However, in order to employ the operation scenario; the designed controller will require continuous information flow from the column, including the compositions throughout the column or temperatures reflecting the composition knowledge. The reason for this requirement is that, the value of reflux ratio and switching between product and slop cut distillations are optimized which are subject to the composition profile along the column and obtained as a function of it. Therefore, the need for knowledge of current composition in the column becomes obvious.

This composition knowledge can be generated by means of direct composition analyzers. Although there is a great development in the technology of online composition analyzers, such as gas chromatography, they bring large measurement delays and high investment and maintenance costs (Mejdell et al., 1991, Oisiovici et al., 2000 and Venkateswarlu et al., 2001). The most popular alternative to the composition controllers utilizing analyzers is standard temperature feedback controllers. Although, temperature measurements are inexpensive and have negligible measurement delays, they are not accurate indicators of composition (Mejdell et al., 1991). Another alternative is inferential control systems incorporating state estimators which uses secondary temperature measurements.

State estimation can be defined as the process of extracting information from data which contain valuable information about a system and state estimator is the tool responsible for gathering valuable measurements to infer the desired
information. Modern estimators also use known relationships in computing the desired information; taking into account the measurement errors, the effects of disturbances and control actions on the system, and prior knowledge about the system and measuring devices. While gathering these elements, they make use of some error criteria and try to minimize errors in some respect. The criteria and the method of minimization characterize the method of estimation and the use of minimization makes the estimate (extracted information) “optimal”. If this optimality is realized statistically, the estimator type becomes stochastic; if deterministically it becomes deterministic. The estimator used in this work falls in the stochastic category and it is named as Kalman Filter.

The Kalman Filter is generally defined as “optimal recursive data processing algorithm” (Maybeck, 1979). A more specific definition can be stated as “an estimator for what is called the linear-quadratic problem, which is the problem of estimating the instantaneous “state” of a linear dynamic system perturbed by white noise—by using measurements linearly related to the state but corrupted by white noise” (Grewal and Andrews, 2001). A more descriptive definition is given as “Kalman Filter is a computational algorithm that processes measurements to deduce a minimum error estimate of the state of a system by utilizing: knowledge of system and measurement dynamics, assumed statics of system noises and measurement errors, and initial condition information” (Gelb, 1974).

In this study, the aim is to design a state estimator that infers the component concentrations of the multicomponent batch distillation column from the measured tray temperatures. The designed estimator must be further tested using a rigorous column simulation to find its performance. This necessitates
the derivation of rigorous simulation model of multicomponent batch distillation column from first principle equations. Also, the model is to be verified with the previous studies in order to check its reliability.

The extended Kalman Filter (EKF) is selected as the state estimator. In the literature, EKF has shown to provide good results in the chemical industry that includes model uncertainties, unmeasured process disturbances and noisy measurements. Because it is based on the linear dynamic model of the process, the rigorous model used in the simulation is adapted to the estimator algorithm mainly by simplifying the equilibrium model and by means of linearization. The performance of the developed estimator is tested by using the rigorous column simulation and discrete measurements of the top product compositions.
CHAPTER 2

LITERATURE SURVEY

2.1 Studies on Batch Distillation Modelling and Simulation

Huckaba et al. (1960) presented the first significant simulation of a batch distillation column for separation of a binary mixture. The model used in the simulation code employed material and heat balances around each tray and was limited by the assumptions of constant-weight tray holdups, linear enthalpy relationships (i.e. relative volatility) and adiabatic operation. However, the model was able to incorporate nonlinear equilibrium relationships and plate efficiency correlations. The validity of the model was also tested in comparison to experimental data.

Meadows (1963) developed the first comprehensive model of a multicomponent batch distillation including the equations for material, energy and volume
balances around theoretical trays. The assumptions made in the model formulation included ideal trays, constant-volume tray holdups, negligible vapor holdup, perfect mixing on all trays and adiabatic operation. However, no experimental verification of the model was presented.

Distefano (1968) derived a mathematical model multicomponent batch distillation and gave a step-by-step calculation algorithm of the developed model. The study was mainly concerned with analysis of the most widely used numerical integration routines used in simulation of the model. The mathematical model was formulated from the material and heat balances around different stages of the column and incorporated the same assumptions made by Meadows (1963). Distefano (1968) stated that due to the radical changes in compositions on each tray, the assumption of constant-mass tray holdup was invalid. For this reason, the assumption of constant-volume tray holdup was employed and the molar tray holdup was allowed to vary by changes in the liquid density which is a function of composition. For the solution of model’s DAE, the initial condition were chosen as the state of the column at total-reflux steady-state, which requires the additional calculation of total-reflux steady-state equations. Therefore, this choose for initial condition causes the lack of simulating dynamic behavior through the total-reflux operation mode.

In the article mainly concerned with the effect of certain design variables on the degree of separation obtainable in a multicomponent batch distillation, Stewart et al. (1973) used the model of Distefano (1968) with the extensions including non-ideal trays and different holdup assumptions. Although the modified model supported both the simulations of constant-distillate-rate and
constant-heat-input policies, it simulated the batch column only during the operation of finite reflux-ratio mode. The validity of the mathematical model used to simulate the dynamic behavior of the multicomponent column operation had been proved through good agreement between the theoretical and experimental results.

Young et al. (1987) presented the validity of the commonly used simplifying assumptions; negligible vapor holdup and constant pressure for low and high pressure columns. They proved that the assumption of negligible vapor holdup in columns operating at high pressures (> 10atm) results in erroneous predictions of dynamic response. As the pressure in the column increases, vapor density of the compounds (especially ones having low critical-pressure value) also increases, which results in comparable holdup values of vapor phase with that of liquid phase. Therefore, it can be concluded that the holdup of the material in vapor phase should be included in dynamic models of high pressure columns. Moreover, in their work, it was shown that the effects of vapor hydraulics have to be incorporated in the models of vacuum columns where changes in pressure are significant.

Reuter et al. (1989) developed a model of multicomponent batch distillation column with chemical reaction and their control systems by taking as a base, a commercial software package for dynamic simulation of continuous distillation processes. The computer algorithm for simulation enabled (i) optimizing batch processes with respect to time and energy, (ii) investigating control configurations, and (iii) developing reduced-order process models to be used in online estimation and control. They also realized that the method of variable-step-time integration
shortens the calculation time without impairing the accuracy of the simulation result. Furthermore, it was shown that despite the use of constant liquid holdup assumption in the column and in the condenser, the dynamic responses of the closed-loop simulation agree with that of the experimental study, even for the processes with a large difference in holdup between reactor and column.

Because of large computational time and huge memory requirement, the rigorous models used in the simulation of batch distillation processes are too complex to be used in optimal design calculations. Diwekar et al. (1991) introduced a short-cut method for optimal design of multicomponent batch distillation column based on the modified version of the Fenske-Underwood-Gilliland method which was used in the design of continuous columns. The developed technique was applied to both binary and multicomponent systems operating under variable and constant reflux conditions and the composition profiles predicted by their method matched closely to the profiles obtained from the rigorous plate-to-plate calculation. In addition, their short-cut method was proven to handle well both non-ideal systems and the dynamic effect of holdup. An other advantage of the technique is that the computation time and memory requirements is not affected by the increase in the number of plates.

In another paper, Diwekar et al. (1991b) proposed a comprehensive package for simulation, design, optimization and optimal control of multicomponent batch distillation column and named BATCH-DIST which utilized the previously obtained models and techniques. This general purpose simulation package was developed in Fortran-77 platform and it had a user friendly interface allowing to choose the task, the operating policy and the model. The included tasks were
simulation, design, optimization and optimal control, the supported policies were constant reflux, variable reflux and optimal control laws and the models available in the package were short-cut, modified short-cut, semi rigorous, reduced-order dynamic and rigorous dynamic. Also in this study, the validity of the models was tested using several experimental cases.

Mori *et al.* (1995) investigated the improvements which could be applied to the solution of multicomponent multistage batch distillation problems as a means to get more precise and flexible results from rigorous simulations. Mori *et al.* (1995) took the models developed by Distefano (1968) and Guy (1983) as a base and obtained a new simulator. Then they employed the simulated responses to verify the numerical solution algorithm, by comparing both (i) with the simulation using Runge-Kutta method and (ii) with the experiment. Their algorithm incorporated a generalized Implicit Euler integration supported with an overall normalized Θ method accelerating convergence. It should be noted that, due to assumptions in the base models, the simulation starts from the total-reflux steady-state, which prevents the simulation from giving the dynamic behavior during the start-up period. The algorithm resulted in a higher accuracy and lesser CPU-time than Runge-Kutta integration.

Winkel *et al.* (1995) introduced the evaluation of an equation oriented modelling tool *gPROMS* which was suitable for the modelling and simulation of batch plant operations. *gPROMS* was tested using a realistic industrial case plant comprising three batch reactors, one buffer tank and two batch distillation units. The authors showed that *gPROMS* was able to simulate the plants where both continuous discrete changes take place and interact to a significant extent.
Mehlhorn et al. (1996) investigated mass transfer and tray hydraulics issues in batch distillation modelling. They have integrated non-equilibrium and equilibrium models to get more accurate model for batch distillation columns with perforated plates. They handled the uncertainties in the mass transfer which was described within the equilibrium-tray model by means of tray efficiency and by using the non-equilibrium-tray model supported by the mass transfer coefficient and the mass transfer area. The developed model was experimentally verified both for binary and multicomponent systems.

Mujtaba et al. (1997) used a detailed dynamic model of batch distillation with chemical reaction to develop efficient techniques for solving optimal operation and scheduling problems. Because the solution of dynamic optimization problems based on detailed dynamic models is very expensive for online applications necessitating a quick and reliable solution, they built a new framework incorporating polynomial based dynamic optimization. In the first principle dynamic model of batch distillation with chemical reaction, they employed common simplifying assumptions including negligible vapor holdup, adiabatic trays, constant molar holdup on trays, and in the condenser, perfect mixing in trays, fast energy dynamics, constant operating pressure and total condensation with no sub-cooling. The authors also stated that the developed framework can be easily applied to nonreactive and/or nonconventional batch distillation with or without chemical reaction.
2.2 Studies on Estimation of Compositions in Distillation Columns

One of the first detailed studies on inferential control and estimation of chemical and petroleum processes, especially including distillation units, was presented by the three-part study of (Joseph et al., 1978a). In the first part, they gave the design methods for a static estimator inferring unmeasurable product compositions from direct temperature measurements in a distillation column. Employing the developed estimator, they formed the inferential control system which adjusts the control effort to counteract the effect of the measurable disturbances, using the estimated values of the unmeasured process outputs. Moreover, the selection criteria for secondary measurements was investigated to design more accurate and sensitive estimator and it was found that the number of measurements should be (i) as high as not to exceed a desired relative error limit reflecting the accuracy and also (ii) as low as not to exceed a desired condition number inversely reflecting the sensitivity. The inferential control system including a static estimator was applied to an industrial distillation column in order to maintain the bottom product composition and a linear input-output model of the process was used in the design. It was seen that the applied inferential control systems were more effective than the temperature-feedback control system to manage the process against the unmeasured disturbances.

In the second part of the study (Brosilow et al., 1978), the static estimator designed in Part I (Joseph et al., 1978a) was combined into a dynamic control system in order to gain ability of managing the inferential control system in
the limits of stability and robustness. The developed dynamic control system was applied to a simulated multicomponent distillation described by a linearized model, in order to keep top and bottom compositions about the desired set points. Then it was concluded that, in rejecting the effects of disturbances in the feed composition, the inferential control system was performed significantly better than the temperature-feedback control and it showed less oscillatory behavior.

The last part of the study (Joseph et al., 1978) presented the construction of optimal and suboptimal dynamic estimators using Kalman Filtering technique of stochastic estimation. The design method achieved linear dynamic estimators giving optimal estimates at the steady-state which was established after input disturbances. It was pointed out that the simpler method of designing an inferential control system was to execute the two parts, estimator and controller, separately. Then, these estimators were incorporated to an inferential control system in order to verify the efficiency and the performance of whole system within the control of an simulated multicomponent distillation column. The simulation results showed that both optimal and suboptimal estimators were good in inferring the output composition values and that when the deviation of the output from the steady-state is large, the model mismatch becomes major source of the estimation error. In addition, comparison with temperature feedback controller indicated better performance of inferential control system providing that best tuning parameters were established.

First considerable study on state estimation of multicomponent batch distillation was introduced by Quintero-Marmol et al. (1991). They used an extended Luenberger Observer (ELO) to infer compositions throughout
the column by means of temperature measurements and by assuming availability of flowrates and pressures. Luenberger Observer is a linear state estimation algorithm, which ground on the linear system model and secondary measurements. Its extended version uses the nonlinear model of the process and its linearized form. Moreover, the results of the study showed that a distillation column is observable if the number of measurements is at least one less than number of components in the system. Their investigations to find measurement locations concluded that placing at least one measurement in the reboiler always improved the results because the reboiler was the unit of the column having largest inertia. Because of lack of initial conditions knowledge, the number of measurements was found to be larger so that the estimator was less sensitive to the deviation between guessed and actual initial conditions. It was also showed that the number of measurements needed was independent of the number of trays in the column. Besides the designed estimator incorporating these outcomes was examined for a simulated batch distillation column and found to be suitable for composition monitoring even if the mismatches between the model and the process existed.

In another research, Quintero-Marmol et al. (1992) applied a modified-$\theta$ method which utilized the temperature measurements, distillate flow rate and heat input, in order to develop a model-based inferential control for multicomponent batch distillation systems. To infer the distillate compositions, they explored two approaches; (i) an estimator using a rigorous steady-state model of the process and (ii) an estimator using a quasi-dynamic nonlinear model. The former ignored the holdups in the column and in the reflux-drum and
calculated the distillate compositions from measured flow rates and temperatures by means of material balances. The latter approach approximated the mixture in the upper part of the column as binary and determined the composition on a tray using the known pressure. However, both approaches give only the composition at the distillate stream.

Mejdell et al. (1991) aimed to develop a static estimator for inferring the product compositions of a continuous distillation column using the temperatures on all trays. For this purpose, they employed principal-component-regression and partial-least-squares estimators supported with weighting functions and logarithmic temperature transformations. In the study, it was also emphasized that control systems using single temperature measurement do not indicate the product compositions accurately and it is not reliable due to the following reasons:

- the relationship between a temperature and a product composition is also a function of the feed composition as well as the other product composition,
- flow pulses and improper mixing on the trays cause temperature variations, which cannot be eliminated by single temperature measurement,
- pressure changes caused by varying liquid holdups and tray performance result in temperature variations.

Because of these dependencies, keeping a temperature at the desired set-point is not sufficient to get constant product composition. The authors also stated that, as well as yielding better estimates, the developed estimator with two PID controllers was less sensitive both to measurement location and to measurement noise than single-temperature control system.
Another important study on composition estimation of distillation column was presented by Baratti et al. (1995). They applied an estimation technique of extended Kalman Filtering to binary distillation columns and checked its performance through a pilot distillation column used in separation of ethanol-water mixture. The paper is mainly interested in obtaining a process model which is sufficiently accurate to guarantee robustness and good performance of the estimator. They employed some simplifying assumptions in the modelling in order to make the design of estimator easy and cheap. The implemented assumptions were mainly; (i) equi-molal overflow, (ii) linear pressure drop profile, (iii) Murphree tray efficiency and (iv) vapor-liquid equilibrium (VLE) described by Antoine equation and NRTL-based activity coefficients. The authors used an heuristic criterion in locating the temperature measurements, resulting in one thermometer in the bottom and one close enough to the top. It was also pointed out that the numerical integration method was not needed to be very accurate but sufficient to be simple and fast because the numerical integration errors could be treated as the model error as long as they were comparable. For testing the robustness of the estimator, the vapor-liquid equilibrium (VLE) model was changed by assuming ideal behavior for the liquid phase. This virtually created mismatch declined the performance and it was seen that the estimator was unable to tolerate the model errors caused by the vapor-liquid equilibria description. However, using erroneous values for tray holdup and tray efficiency was handled by the estimator leading to satisfactory estimates. Therefore, the authors concluded that, “a major effort in development of the model to be used in the estimation algorithm, has to be devoted to the accurate description of
vapor-liquid equilibria” rather than dynamic effects.

In another study of Baratti et al. (1998), the previously developed state estimator (Baratti et al., 1995) was further extended to infer the top composition of multicomponent distillation. They obtained good agreement between the estimation and the off-line composition measurements and it was indicated that moving from binary to multicomponent systems needs more accurate models of the vapor-liquid equilibria.

For monitoring a continuous distillation column, Yang et al. (1997) introduced a reduced-order model which was developed using the collocation method and cubic spline functions. To infer the product compositions and the feed condition, they incorporated an extended Kalman Filter algorithm. This algorithm was responsible to identify the parameters of the reduced-order model such as equilibrium constants and column holdup. In the study, Kalman Filter was not directly used for estimating the compositions but for identifying the model parameters.

Barolo et al. (1998) proposed an inferential control method for maintaining a constant-distillate-composition operation of batch distillation unit. The developed inferential controller bases on Nonlinear Internal Model Control (NIMC) framework suggested by Henson et al. (1991) and the state estimation algorithm of ELO which was previously applied to batch distillation by Quintero-Marmol et al. (1991). To verify the controller and to test its robustness, it was applied to a simulated batch distillation column for non-ideal systems and multicomponent mixtures. The model used in the simulation of the column utilized the following assumptions; constant vapor boilup rate, constant reflux
drum level, ideal trays, boiling feed, total condensation with no sub-cooling, negligible heat losses and atmospheric operation. In addition, the Non-random Two Liquids (NRTL) model was used to calculate vapor-liquid equilibria of binary mixtures while the Clausius-Clapeyron was employed for multicomponent mixtures. However, in order to provide more realistic simulation, the authors included several simplifications leading to process/model mismatch in the model used by the estimator. Mainly, they neglected the tray hydraulics and used an empirical VLE model estimated from the plant simulation. The comparison between the inferential control loop and the PI controller with temperature feedback, showed that the performance of the developed controller is good in maintaining the product composition constant. In addition, the state estimator inferred the product composition only by a slight error. This estimation error is mainly due to the difference between VLE description in the process and in the model. It was suggested that although the ELO is enough to compensate for measurement noise, under high noisy conditions the use of a stochastic estimator such as an extended Kalman Filter (EKF) is to be used. They realized that as the number of measurement increases, the ELO becomes harder to be tuned.

In the field of batch distillation estimation one of the most recent studies was presented by Oisiovici et al. (2000). They designed a discrete extended Kalman Filter to infer from temperature measurements, the composition profile of multicomponent batch distillation column (MBDC). It was stated that the need for using EKF instead of ELO results from the opportunity to work under a stochastic framework. This advanced feature of EKF offers the ability of dealing with time-varying process including plant/model mismatch and process and/or
measurement noise. In developing the model for EKF, they employed some simplifying assumptions to reduce the computational burden. The assumptions made in the model of the batch distillation system are, equimolal overflow, theoretical trays, negligible vapor holdup, constant liquid holdup, constant pressure, negligible reflux drum holdup and total condenser. They obtained the sensor model by solving the bubble-point temperature problem for each measurement tray. the developed EKF estimator was tested by employing a rigorous batch distillation simulator. Investigating the mismatch between the process and the model had showed that, the error between them can be handled by means of a noise adaptive Kalman filter when the difference is expected to be high. However, the authors pointed out that even if the process is exactly modelled, a composition estimator is still required for accurate predictions when the initial condition could not be known exactly. Disturbing the temperature measurements by a Gauss distributed noise does not significantly effect the estimation performance. In addition, exploring sampling period and number of measurements indicated that tuning the EKF heavily depends on dynamics of the system studied and is to be determined through a trial-and-error procedure.

Another research on optimal state estimation of MBDC was carried out by Venkateswarlu et al. (2001). They employed the estimation methods of extended Kalman filter, adaptive fading Kalman filter and steady state Kalman filter in order to infer compositions in all trays, reboiler, condenser and final products from temperature measurements. The first principal dynamic model used in all the estimators was derived by assuming constant tray and reflux drum holdups, constant vapor boilup rate and constant flowrates in the column. The relationship
between temperature and compositions required by the measurement model was obtained from Antoine equation based VLE. They implemented each estimator to a simulated MBDC in order to check their performances using the performance indices, named capacity factor, integral square error (ISE) and operation time of a batch. However, they didn’t present any investigation on effects of mismatch of VLE description on robustness. As mentioned previously, Oisiovici et al. (2000) and Baratti et al. (1995) have pointed out that the most significant part of developing the estimator model is to attain an accurate but a simple VLE model in order to effectively utilize temperature measurement in estimation. Venkateswarlu et al. (2001) showed that even in the presence of measurement noise and wide changes in the design parameters, EKF performs well in the state estimation.
CHAPTER 3

BATCH DISTILLATION COLUMN

MODELLING

In this chapter, a rigorous model of a Multicomponent Batch Distillation Column (MBDC) will be derived. Section 3.1 is devoted to its conventional operation strategy. Then in the following Section 3.2, the assumptions made in the model development will be discussed. The model derivation is to be introduced in Section 3.3 including the subsections for modelling of the production stage, for the total reflux stage, for molar holdup calculations and for the algebraic equations including the calculation of thermodynamical and physical properties used in the model. The model providing the compositions and holdups in the storage tanks is given in Section 3.4. Finally, the selection of initial conditions required for the solution of ODEs of the model will be discussed in Section 3.5.
3.1 Operation of a MBDC

In a batch distillation operation two types of products are handled (Luyben, 1988). The one named as *slop-cut* which is the byproduct of off-specification material and the other named as *product-cut* which is the product satisfying the specified purities. The operation of a batch column is divided into a number of stages as in the order of realization; start-up period, distillation at total-reflux, withdrawal of the lightest product, removal of a slop-cut, withdrawal of the next heaviest product, removal of a second slop-cut and, so on. If the aim of production is to separate each compound from the feed mixture at the specified purity levels, the number of product-cuts is $NC$ in the feed (*i.e.* $NC$:number of compounds) and the number of the maximum possible slop-cuts is $NC - 1$. Figure 3.1 shows the basic elements of a MBDC system and their configuration. It consists of reboiler, trays assembled in the column, coupled unit of condenser and reflux-drum, product and slop-cut storage tanks.

In the conventional operation of a batch column, firstly the feed mixture to be distilled is charged to the reboiler to which heat is then supplied. Nextly the vapor leaving the reboiler starts to condensate throughout the trays in the column and simultaneously the condensed liquid establishes the initial tray holdups. Then the vapor reaching the top of the column is condensed by means of the condenser and then the condensate fills the reflux-drum. However, due to the difficulties which arise in the simulation of this highly transient process, a hypothetical initialization procedure is used.
Figure 3.1: The schematic of a Multicomponent Batch Distillation system.
The column operation is initiated by charging the feed mixture to the column, from its top, resulting in establishment of initial holdups in the order of condenser, reflux-drum, trays and reboiler. During this initialization period, no distillate is withdrawn from the column but instead the column is operated at total-reflux condition or at high reflux ratios to establish the desired purity level of the lightest compound in the reflux-drum. Then the first product-cut is started to distill by setting the reflux-ratio to a pre-specified value and in the same time the distillate stream is transferred to first product-cut storage tank. Due to the decreasing amount of the lightest compound in the column, after some time its composition level in the first product-cut tank begins to decrease. At this point, the distillate stream is diverted to the first slop-cut tank, if the composition of the next heaviest compound in the reflux-drum is below its specified purity. If however, it is not so low, or, during the slop-cut distillation, it starts to pass the specification level, then again the distillate is diverted to the second product-cut tank and the reflux-ratio is set to its new value. This cyclic operation between the product-cut and the slop-cut distillation continues until all the intermediate compounds is separated. Finally, the content of the reboiler is taken as the final product-cut which is rich in the heaviest compound.

Up to here the procedure of a single batch operation has been outlined. To be able to make use of obtained slop-cuts, they may be re-separated by the next batch operation. There are many strategies of utilizing the slop-cuts in the literature (Sorensen et al., 1999). However, in this study only single batch operation is considered and no details are given about long-time scheduling and strategies in batch columns.
3.2 Modelling Assumptions

The accuracy requirements of any model depend on the place where the model will be used. For example, model based optimization problems do not require very accurate models but they ask for fast models. In many studies on batch distillation columns, highly accurate simulation models are not need, since the developments in the batch distillation is not as fast as that of the continuous columns which requires accurate models to analyze and test the designed systems at strict and hard conditions.

As given in Chapter 2, there are many studies on the rigorous modelling of distillation columns. They use the same basic strategy in the simulation model development which was used initially by the first studies on rigorous modelling of distillation columns. In batch column modelling, this common strategy was initiated by Meadows (1963) and Distefano (1968) which were followed by Stewart et al. (1973). In addition, the recent studies of Venkateswarlu et al. (2001), Furlonge et al. (1999) and Perry et al. (1999) can be given as the examples using this common strategy. Therefore, in this study this strategy is used and the modelling is grounded on the study of Distefano (1968) where the model employed heat and material balances and used the assumptions of

• negligible vapor holdup,

• constant volume of tray liquid holdup,

• negligible fluid dynamics lags,

• adiabatic operation,

• ideal trays.
The assumption of negligible vapor holdup have been discussed by Young et al. (1987) and they stated, “in columns operating at moderate pressures (less than 10atm), this assumption is usually a good one”. The validity of the assumption can be illustrated by a simple example which was also given by Young et al., (1987). The vapor holdup is only 2.3% of the total holdup on a tray and can be neglected (See Appendix A). If the column were operated at 10atm, the vapor holdup becomes 19.2% of the total holdup. As a result assumption of negligible vapor holdup is always reasonable.

In addition, in the study of Distefano (1968) on which the current model is based, the assumption of constant volume tray liquid holdup was discussed and it was realized that because of the severe variations in the tray compositions, assumption of constant molar or weight holdup is invalid in batch distillation calculations. Therefore using the constant—volume—holdup assumption will be employed in the model derivation and variations in the molal tray holdup will be provided by variations in the liquid density that is a function of composition, temperature and pressure.

Instead of using ideal tray assumption, to represent the non-ideality in the phase equilibrium, Murphree tray efficiency formulation is employed assuming the temperature equilibria between the vapor and the liquid phases. Consequently, the basic assumptions made in the model development are summarized in Table 3.1.
Table 3.1: Assumptions made in the model development

<table>
<thead>
<tr>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Negligible vapor holdup</td>
</tr>
<tr>
<td>2. Constant volume of tray liquid holdup</td>
</tr>
<tr>
<td>3. Constant liquid molar holdup in the reflux-drum</td>
</tr>
<tr>
<td>4. Total condenser</td>
</tr>
<tr>
<td>5. Negligible fluid dynamic lags</td>
</tr>
<tr>
<td>6. Linear pressure drop profile</td>
</tr>
<tr>
<td>7. Murphree tray efficiency</td>
</tr>
<tr>
<td>8. Approximated enthalpy derivatives</td>
</tr>
<tr>
<td>9. Adiabatic operation</td>
</tr>
</tbody>
</table>

3.3 MBDC Model Derivation

As stated previously, the model formulated in this study is based on the model of Distefano (1968). However because this model takes the total-reflux steady-state as the initial condition it cannot represent the behavior of batch column during the course of total-reflux operation. The developed model consists of two different phases of operation. To represent the column during the finite reflux-ratio operation (i.e. distillate withdrawal phase), the model of Distefano (1968) is to be used by means of some modifications. In the second part, the total-reflux operation is to be derived by the model simulating separately.

Nevertheless, both parts of the model results from heat and energy balances around the different elements of the column shown in Figure 3.1. Applying only the assumption of negligible vapor holdup, the total molar mass balance, the component balance and the energy balance for each element of the column are written as follows,

for reboiler: $j = 1 \ldots NC$
\[
\frac{dM_1}{dt} = L_2 - V_1 \tag{3.1}
\]
\[
\frac{dM_{1x_{1j}}}{dt} = L_2x_{2j} - V_1y_{1j} \tag{3.2}
\]
\[
\frac{d(M_1h_1)}{dt} = L_2h_2 - V_1H_1 + Q_1 \tag{3.3}
\]

For trays: \( i = 2 \ldots NT + 1; j = 1 \ldots NC \)

\[
\frac{dM_i}{dt} = L_{i+1} + V_{i-1} - L_i - V_i \tag{3.4}
\]
\[
\frac{d(M_ix_{ij})}{dt} = L_{i+1}x_{i+1,j} + V_{i-1}y_{i-1,j} - L_i x_{ij} - V_i y_{ij} \tag{3.5}
\]
\[
\frac{d(M_ih_i)}{dt} = L_{i+1}h_{i+1} + V_{i-1}H_{i-1} - L_i h_i - V_i H_i \tag{3.6}
\]

For reflux-drum-condenser system: \( j = 1 \ldots NC \)

\[
\frac{dM_{NT+2}}{dt} = V_{NT+1} - L_{NT+2} - D \tag{3.7}
\]
\[
\frac{d(M_{NT+2x_{NT+2,j}})}{dt} = V_{NT+1}y_{NT+2,j} - L_{NT+2}x_{NT+2,j} - Dx_{NT+2,j} \tag{3.8}
\]
\[
\frac{d(M_{NT+2h_{NT+2}})}{dt} = V_{NT+1}H_{NT+2} - L_{NT+2}h_{NT+2} - Dh_{NT+2} - Q_{NT+2} \tag{3.9}
\]

Where \( x \) and \( y \) are liquid and vapor fractions \( (mol/mol) \), \( M \), molar liquid holdup \( (mol) \), \( L \) and \( V \), liquid and vapor molar flow rates \( (mol/h) \), \( h \) and \( H \), liquid and vapor mixture enthalpy \( (J/mol) \), \( Q_1 \) and \( Q_{NT+2} \), reboiler and condenser heat loads \( (J/mol \cdot h) \), \( D \), distillate flow rate \( (mol/h) \) and the subscripts \( i \) and \( j \) are for stage and component numbers, \( NT \) and \( NC \) are number of trays and number of components, respectively (i.e. \( i = 1 \) for reboiler, \( i = 2 \ldots NT + 1 \) for trays and \( i = NT + 2 \) for reflux-drum-condenser unit).
Extracting left-hand-side(l.h.s.) of Equation 3.2

\[ \frac{dM_1 x_{1j}}{dt} = M_1 \frac{dx_{1j}}{dt} + x_{1j} \frac{dM_1}{dt} \]  \hspace{1cm} (3.10)

and inserting Equation 3.1

\[ \frac{dM_1 x_{1j}}{dt} = M_1 \frac{dx_{1j}}{dt} + x_{1j}(L_2 - V_1) \]  \hspace{1cm} (3.11)

gives the following equation

\[ M_1 \frac{dx_{1j}}{dt} = L_2 x_{2j} - V_1 y_{1j} - L_2 x_{1j} + V_1 x_{1j} \]  \hspace{1cm} (3.12)

and then rearranging Equation 3.12 gives the time derivative of the compositions in the reboiler as

\[ \frac{dx_{1j}}{dt} = \frac{[L_2(x_{2j} - x_{1j}) - V_1(y_{1j} - x_{1j})]}{M_1} \]  \hspace{1cm} (3.13)

Similarly, combining Equation 3.4 and Equation 3.5 gives the time derivative of the compositions in the trays as

\[ \frac{dx_{ij}}{dt} = [V_i(y_i - x_{ij}) + L_{i+1}(x_{i+1,j} - x_{ij}) - V_i y_{ij} - x_{ij})] / M_i \]  \hspace{1cm} (3.14)

If the assumption of constant molar liquid holdup in the reflux-drum, is employed, Equation 3.7 becomes

\[ V_{NT+1} = L_{NT+2} + D \]  \hspace{1cm} (3.15)

and inserting Equation 3.15 to Equation 3.8 gives

\[ \frac{dx_{NT+2,j}}{dt} = [V_{NT+1}(y_{NT+1,j} - x_{NT+2,j})] / M_{NT+2} \]  \hspace{1cm} (3.16)

Therefore the time derivative of the compositions throughout the column, which are the state equations of the column, are obtained and given by Equations 3.13, 3.14 and 3.16.
In order to solve those equations, the vapor and liquid flowrates $V_i$ and $L_i$ are required. Extracting l.h.s. of Equation 3.6 and inserting $\frac{dM_i}{dt}$ from Equation 3.4 as

$$\frac{d(M_i h_i)}{dt} = M_i \frac{dh_i}{dt} + h_i \frac{dM_i}{dt}$$

(3.17)
and equating l.h.s. of Equation 3.6 and Equation 3.17 yield the vapor flow entering the $i^{th}$ tray as

$$V_{i-1} = [V_i(H_i - h_i) + L_{i+1}(h_i - h_{i+1}) + M_i \delta_i(h_i)] / (H_{i-1} - h_i)$$

(3.18)
where $\delta_i(\cdot)$ operation is the numerical differentiation approximation for $\frac{d(\cdot)}{dt}$ given by

$$\delta_i(\beta) = \frac{\beta(t_k) - \beta(t_{k-1})}{t_k - t_{k-1}}$$

(3.19)
where $\beta$ is a dummy function. $\delta_i(h_i)$ is the approximation for $\frac{dh_i}{dt}$.

Moreover, solving Equation 3.4 for $L_i$ gives

$$L_i = V_{i-1} + L_{i+1} - V_i - \delta_i(M_i)$$

(3.20)
where $\delta_i(M_i)$ is the approximation for $\frac{dM_i}{dt}$. Starting from $i = NT + 1$ and solving Equation 3.18 and Equation 3.20 yield all the flowrates but firstly require the vapor flow entering the condenser, $V_{NT+1}$ and the reflux flow entering the top tray, $L_{NT+2}$.

### 3.3.1 Overhead Flowrates for Finite Reflux Ratio

If the reflux ratio is defined as

$$R = \frac{L_{NT+2}}{D}$$

(3.21)
or

\[ L_{NT+2} = RD \]  

(3.22)

Equation 3.15 becomes

\[ V_{NT+1} = D(R + 1) \]  

(3.23)

Employing the assumptions of constant holdup and total condenser and expanding l.h.s. of Equation 3.9 give

\[ M_{NT+2} \frac{dh_{NT+2}}{dt} = V_{NT+1}(H_{NT+1} - h_{NT+2}) - Q_{NT+2} \]  

(3.24)

Rearranging and applying \( \delta_t(\cdot) \) approximation to Equation 3.24 yield

\[ Q_{NT+2} = V_{NT+1}(H_{NT+1} - h_{NT+2}) - M_{NT+2}\delta_t(h_{NT+2}) \]  

(3.25)

Inserting Equation 3.23 in Equation 3.25 to eliminate \( V_{NT+1} \) results in Equation 3.26.

\[ Q_{NT+2} = D(R + 1)(H_{NT+1} - h_{NT+2}) - M_{NT+2}\delta_t(h_{NT+2}) \]  

(3.26)

When the energy balance is applied around the overall column, Equation 3.27 is obtained.

\[ Q_1 - Q_{NT+2} - Dh_{NT+2} = \sum_{n=1}^{NT+2} \frac{d(M_nh_n)}{dt} \]  

(3.27)

Using Equation 3.26, \( Q_{NT+2} \) can be eliminated from Equation 3.27 to find the distillate rate, \( D \) in the form of

\[ D = \frac{Q_1 - \sum_{n=1}^{NT+1} \delta_t(M_nh_n)}{(R + 1)H_{NT+1} - Rh_{NT+2}} \]  

(3.28)

As a result, by using the distillate rate, \( D \) from Equation 3.28, the required flowrates, \( V_{NT+1} \) and \( L_{NT+2} \) can be calculated from Equation 3.23 and 3.22, respectively.
3.3.2 Overhead Flowrates for Total Reflux

For total reflux operation, the overhead flowrates, $V_{NT+1}$ and $L_{NT+2}$ cannot be obtained from the equations derived in Section 3.3.1 because they become undefined for infinite reflux ratio. Therefore, it is needed to find the flowrates from another formulation.

At total reflux condition, no product is withdrawn from the column such that

$$D = 0$$

(3.29)

and

$$V_{NT+1} = L_{NT+2}$$

(3.30)

Therefore, if the value of the reflux flow, $L_{NT+2}$ is found, the vapor flowrate of the stream entering the condenser is obtained.

If the energy balance is applied to the joined system of all the trays and the reboiler, the following equation is obtained by use of Equation 3.30

$$\sum_{n=2}^{NT+1} \frac{d(M_n h_n)}{dt} + \frac{d(M_1 h_1)}{dt} = Q_1 + L_{NT+2}(h_{NT+2} - H_{NT+1})$$

(3.31)

and arranging Equation 3.31 gives $L_{NT+2}$ as

$$L_{NT+2} = \frac{Q_1 - \frac{d(M_1 h_1)}{dt} - \sum_{n=2}^{NT+1} \frac{d(M_n h_n)}{dt}}{(H_{NT+1} - h_{NT+2})}$$

(3.32)

3.3.3 Molar Holdup Calculations

The calculation of molar holdups is another important step in the development of the model in the reboiler, on trays and in the condenser-reflux-drum system. By utilizing constant volume of liquid holdup, the molar holdups on trays can be
calculated as

\[ M_i = \frac{\rho_i^{\text{avg}}}{M_{w_i}^{\text{avg}}} v_i \]  

(3.33)

where \( \rho_i^{\text{avg}} \) is the average density of the mixture on the \( i^{\text{th}} \) tray, \( M_{w_i}^{\text{avg}} \), the average molecular weight of the mixture on the \( i^{\text{th}} \) tray, \( v_i \) is the volume of the liquid tray holdup.

In addition, the reboiler holdup at any time, \( t \), is calculated from an algebraic equation given as

\[ M_1 = M_0^f - \sum_{n=2}^{NT+2} M_n - \int_0^t D(\tau) d\tau \]  

(3.34)

where \( M_0^f \) is the molar amount of feed initially charged to the column.

### 3.3.4 Algebraic Equations and Calculation of Physical Parameters

The mole fraction sum for liquid and vapor phases are stated as

\[ \sum_{n=1}^{NC} x_n = 1; \sum_{n=1}^{NC} y_n = 1 \]  

(3.35)

The linear pressure drop profile are given as

\[ P_i = P_1 - i \cdot (P_1 - P_{NT+2})/NT \]  

(3.36)

where \( P_i \) the pressure in \( i^{\text{th}} \) tray, \( P_1 \), the pressure in the reboiler and \( P_{NT+2} \), the pressure in the reflux drum.

The effects of non-equilibrium between the liquid and the vapor phases on a tray are incorporated to the model by Murphree tray efficiency formulation specified as

\[ y_{ij} = y_{i-1,j} + Eff_{\text{Murphree}}(y^{*}_{ij} - y_{i-1,j}) \]  

(3.37)
where \( y_{ij}^* \) is the composition of vapor in phase equilibrium with liquid on \( i^{th} \) tray with composition \( x_{ij} \); \( y_{ij} \), the actual composition of vapor leaving \( i^{th} \) tray; \( y_{i-1,j} \), the actual composition of vapor entering \( i^{th} \) tray.

Unlike to Distefano (1968) where relative volatility approach was used for the calculation of the equilibrium temperature and the composition of vapor phase at equilibrium with the liquid phase, a thermodynamic model using Peng-Robinson EOS is used and represented by

\[
[T_i, y_{ij}^*] = f [(x_{ik}, k = 1 \ldots NC), T_i^{\text{guess}}, P_i]
\]

(3.38)

where \( T_i^{\text{guess}} \) is initial temperature guess for trial-error calculation of equilibrium temperature, \( T_i \).

Also, for the calculation of mixture enthalpy, average mixture densities and average molecular weights of a mixture, again Peng-Robinson equation of state is used. The detailed calculation procedure and its derivation are given in the recent work of Dokucu (2002). The representative functions for the physical properties of a mixture are expressed as a function of composition, temperature and pressure, as follows,

The enthalpy of liquid mixture on \( i^{th} \) stage of the column:

\[
h_i = f [(x_{ik}, k = 1 \ldots NC), T_i, P_i]
\]

(3.39)

The enthalpy of vapor mixture on \( i^{th} \) stage of the column:

\[
H_i = f [(y_{ik}, k = 1 \ldots NC), T_i, P_i]
\]

(3.40)

The average density of liquid mixture on \( i^{th} \) stage of the column:

\[
\rho_i^{\text{avg}} = f [(x_{ik}, k = 1 \ldots NC), T_i, P_i]
\]

(3.41)
The average molecular weight of liquid mixture on $i^{th}$ stage of the column:

$$M_{w_{avg}}^i = f \left[ (x_{ik}, k = 1 \ldots N_C), T_i, P_i \right] \quad (3.42)$$

Moreover, Equations 3.38-3.42 require the parameters of critical temperature, $T^c_j$, critical pressure, $P^c_j$, boiling temperature, $T^\text{boil}_j$, molecular weight, $Mw_j$, acentric factor, $w_j$ and heat capacity coefficients, $Cp^a_j$, $Cp^b_j$, $Cp^c_j$, $Cp^d_j$ for each component, $j = 1 \ldots N_C$.

### 3.4 Storage Tanks Model

To monitor the holdups and the compositions of product-cut and slop-cut tanks, the tanks are modelled as an integrator. Then, the instantaneous molar holdup in $i^{th}$ tank is calculated as

$$S_i = \int_{S_{T_i}}^{E_{T_i}} D(\tau) d\tau$$

where $S_i$ is the molar holdup in $i^{th}$ tank, $S_{T_i}$ and $E_{T_i}$ the time of start and end of storing the product in $i^{th}$ tank.

Also, the molar liquid fraction of each component, $j$ in $i^{th}$ tank is obtained through Equation 3.44

$$s_{ij} = \frac{\int_{S_{T_i}}^{E_{T_i}} x_{NT+2,j}(\tau) D(\tau) d\tau}{\int_{S_{T_i}}^{E_{T_i}} D(\tau) d\tau}$$

where $s_{ij}$ is the molar fraction of component $j$ in $i^{th}$ tank and $x_{NT+2,j}$, the molar fraction of component $j$ in the distillate.
3.5 Initial Conditions

The model ODE require initial conditions for the solution. Unlikely to the study of Distefano (1968), in this study, the initial condition is selected as the state of the column at which the fresh feed at its boiling point is charged to the column from the top. Therefore, all the compositions throughout the column are equal to that of fresh feed and expressed as

\[ x_{ij} = x_{j}^{\text{feed}} \quad \text{for} \quad i = 1 \ldots NT + 2 \quad j = 1 \ldots NC \]  

(3.45)

The initial flowrates are taken as

\[ V_i = \frac{Q_1}{(H_1 - h_2)} \quad \text{for} \quad i = 2 \ldots NT + 2 \]  

(3.46)

\[ L_i = V_{NT+2} \quad \text{for} \quad i = 1 \ldots NT + 1 \]  

(3.47)

3.6 Summary of Model Equations

The model equations given in the previous sections are summarized in Table 3.2 including the rate of change of each component’s composition throughout the column, tray vapor and liquid flowrates, overhead flowrates both for total-reflux and finite reflux operations, molar holdup amounts in the reboiler and on the trays, and the algebraic equations for mole fraction sums together with Murphree tray efficiency formulation and physical properties.
Table 3.2: Summary of MBDC Rigorous Model Equations

<table>
<thead>
<tr>
<th>Compositions and Holdups</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reboiler Dynamics</strong></td>
</tr>
<tr>
<td>( M_1 = M_0^f - \sum_{n=2}^{NT+2} M_n - \int_0^t D(\tau) d\tau )</td>
</tr>
<tr>
<td>( \frac{dx_{1j}}{dt} = \frac{[L_2(x_{2j} - x_{1j}) - V_1(y_{1j} - x_{1j})]}{M_1} )</td>
</tr>
<tr>
<td>( j = 1 \ldots NC )</td>
</tr>
<tr>
<td><strong>Tray Dynamics</strong></td>
</tr>
<tr>
<td>( M_i = \frac{\rho_{\text{avg}}}{M_{w_i}} \bar v_i )</td>
</tr>
<tr>
<td>( \frac{dx_{ij}}{dt} = \frac{[V_{i-1}(y_{i-1,j} - x_{ij}) + L_i(x_{i+1,j} - x_{ij}) - V_i(y_{ij} - x_{ij})]}{M_i} )</td>
</tr>
<tr>
<td>( i = 2 \ldots NT + 1; j = 1 \ldots NC )</td>
</tr>
<tr>
<td><strong>Reflux-Drum Dynamics</strong></td>
</tr>
<tr>
<td>( \frac{dx_{NT+2,j}}{dt} = \frac{[V_{NT+1}(y_{NT+1,j} - x_{NT+2,j})]}{M_{NT+2}} )</td>
</tr>
<tr>
<td>( j = 1 \ldots NC )</td>
</tr>
</tbody>
</table>

| Composition Sums |
| \( \sum_{n=1}^{NC} x_n = 1; \sum_{n=1}^{NC} y_n = 1 \) |

| Flowrates for given \( R \) and \( Q_1 \) |
| **Trays** |
| \( V_{i-1} = [V_i(H_i - h_i) + L_i(h_i - h_{i+1}) + M_i\delta_i(h_i)] / (H_{i-1} - h_i) \) |
| \( L_i = V_{i-1} + L_{i+1} - V_i - \delta_i(M_i) \) |
| \( i = NT + 1 \ldots 2 \) |
| **Overhead Flowrates for Total Reflux** |
| \( D = 0 \) |
| \( V_{NT+1} = L_{NT+2} \) |
| \( L_{NT+2} = \frac{Q_1 - \delta_i(M_i h_i) - \sum_{n=2}^{NT+1} \delta_i(M_n h_n)}{(H_{NT+1} - h_{NT+2})} \) |
| **Overhead Flowrates for Finite Reflux Ratio** |
| \( D = \frac{Q_1 - \sum_{n=1}^{NT+1} \delta_i(M_n h_n)}{(R+1)H_{NT+1} - H_{NT+2}} \) |
| \( V_{NT+1} = D(R + 1) \) |
| \( L_{NT+2} = RD \) |
Table 3.3: Summary of MBDC Rigorous Model Equations (cont’d)

<table>
<thead>
<tr>
<th>Pressure Drop Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_i = P_1 - i \cdot (P_1 - P_{NT+2})/NT )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thermodynamic Models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VLE Calculation</strong></td>
</tr>
<tr>
<td>([T_i, y_{i,j}^*] = f [(x_{ik}, k = 1 \ldots NC), T_i^{guess}, P_i] )</td>
</tr>
<tr>
<td><strong>Murphree Tray Efficiency</strong></td>
</tr>
<tr>
<td>( y_{ij} = y_{i-1,j} + E f f_{Murphree}(y_{i,j}^* - y_{i-1,j}) )</td>
</tr>
<tr>
<td><strong>Enthalpy Calculations</strong></td>
</tr>
<tr>
<td>( h_i = f [(x_{ik}, k = 1 \ldots NC), T_i, P_i] )</td>
</tr>
<tr>
<td>( H_i = f [(y_{ik}, k = 1 \ldots NC), T_i, P_i] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_{avg_i} = f [(x_{ik}, k = 1 \ldots NC), T_i, P_i] )</td>
</tr>
<tr>
<td>( Mw_{avg_i} = f [(x_{ik}, k = 1 \ldots NC), T_i, P_i] )</td>
</tr>
</tbody>
</table>
4.1 Basics of Kalman Filtering

A Kalman Filter is simply defined as “optimal recursive data processing algorithm” (Maybeck, 1979) and it works over a stochastic framework. By describing each word one-by-one, the basic elements of a Kalman Filter can be covered easily.

The definition starts with “optimal” which is a subjective term which can be defined in many different ways and it is dependent of the criteria chosen to evaluate performance. It has been shown that a Kalman Filter is optimal, subject to almost any criterion (Gelb, 1974). This optimality is realized by the fact that Kalman Filter utilizes all information that can be provided to it. It gathers all available measurements, regardless of their contribution, by means of the following:

- dynamic models of the system and measurement devices,
• statistical characterization of the process noises, measurement errors and uncertainty in the dynamics models,

• initial conditions of the variables to be estimated and their statistical descriptions.

The word “recursive” in the above definition means that Kalman Filter does not need to store the whole history of data and to reevaluate every time when a new measurement is captured. Therefore, this basic element makes the implementation of Kalman Filter practical. That is a vital advantage over other data processing frameworks. Lastly, “data processing algorithm” term in the definition specifies that Kalman Filter is a digital computer program of a filter to incorporate discrete-time measurement samples.

Another feature of Kalman Filter is being in the class of stochastic framework where there are many superior attributes not available in the deterministic workspace. In addition, deterministic system theories do not provide a totally sufficient tool of performing the analysis and the design of the systems because;

(i) There are always some uncertainty in any mathematical model of a system.

(ii) Dynamic systems are driven not only by controller actions but also by disturbances which cannot be neither controlled nor modelled deterministically.

(iii) Sensors do not provide perfect and complete data about a system. The unavailability or the cost of the sensor makes one unable to obtain all the information we would like to know. Also, functionally related measurements
taken from different sensors, make difficult to estimate the variables of interest due to lack of handling partially redundant data. Lastly, sensors do not yield exact measurements but append their own system dynamics and distortions as well.

Figure 4.1: Typical application of Kalman Filter estimator.

Figure 4.1 shows the schematic of common application of Kalman Filter to estimate the system states. A generalized system is driven by control inputs and the measuring devices supply some information about the system states to be estimated. These system inputs and outputs are all the information which is explicitly available from the physical system (bordered by dashed line) for estimation purposes. Here the question is “How this explicit information is to be used to estimate the system states?” Because the desired states often cannot be measured directly, the answer obviously appeared as “by a state estimator” which is responsible for inferring the states from the available information.

As an example problem, a temperature measurement system in a batch distillation column directly produces tray temperatures from which product composition is to be inferred. Here, the inference is not simple but complicated as stated below;
(i) A batch column is typically driven by unknown inputs as well as known control efforts (e.g. reflux-ratio, boiluprate). Some examples to unknown inputs can be given as fluctuations in the steam and the cooling water, heat interactions of the column with surrounding and uncertainties in the feed condition.

(ii) The relationships along the compositions and measured temperatures are known only with some degree of uncertainty. It is mainly because of insufficient description of vapor-liquid equilibrium model, inaccurate modelling of tray hydraulics and neglected dynamics of temperature sensors.

(iii) Any measurement may be corrupted by noise biases and device inaccuracies.

These complications can be handled in a systematic and optimal manner introduced by Kalman Filtering. A Kalman Filter combines all available measurement data, plus prior knowledge about the system and measuring devices, to produce the best estimate of the desired variables in such a manner that the error is minimized statistically.

4.2 Kalman Filtering Theory with a Simple Example

It is better to describe the structure and the algorithm of Kalman Filtering by means of a simple example which was previously introduced by Maybeck (1979). Due to the simplicity of estimating one’s exact location from a single measurement device, it is chosen as a descriptive example.

Imagine that one is lost at sea during the night and do not know anything
about his location. Then, he takes a star to determine his position in a single dimension by some means. At some time, $t_1$, he detects his position to be $z_1$. Nevertheless, due to human error and device inaccuracies, his measurement always contains some uncertainty and the level of this uncertainty is specified either by standard deviation, or by variance. Here, he takes the standard deviation, $\sigma_{z_1}$ or equivalently, the variance, $\sigma^2_{z_1}$ as the precision of the measurement, $z_1$. Assuming a Gaussian density, now he can set up the conditional probability of $x(t_1)$, his position at time $t_1$, conditioned on the measurement, obtained as $z_1$. Figure 4.2 shows the pilot of this variable, $f_{x(t_1)|z(t_1)}(x|z_1)$ as a function of the position $x$. It expresses the probability of being in any one position, based upon the measurement he took. In other words, the larger $\sigma_{z_1}$ is, the broader the probability peak is, so the higher the uncertainty is.

Based on this conditional probability density, the best estimate of his location is

$$\hat{x}(t_1) = z_1$$

(4.1)

with the variance of the error in the estimate as

$$\sigma^2_x(t_1) = \sigma^2_{z_1}$$

(4.2)

Consider, another measurement is recorded by a trained navigator friend just after him, at time $t_2 \cong t_1$ (hence he has not moved at all) and obtained as $z_2$ with a variance $\sigma^2_{z_2}$. Assuming that his friend has a higher skill, the variance of his measurement is smaller than his. Figure 4.3 depicts the conditional density of the position at time $t_2$, based only on the measurement $z_2$. The narrower peak indicates that his friend’s measurement makes him rather certain about the
position.

Now he has two measurements to estimate the position. The question is, “How are these data used to obtain new estimate?” It has been proved that, the conditional density of the position at time $t_2 \cong t_1$, $x(t_2)$, based on both $z_1$ and $z_2$, is a Gaussian density with mean $\mu$ and variance $\sigma^2$ given as

$$
\mu = \left[ \frac{\sigma^2_{z_2}}{\left(\sigma^2_{z_1} + \sigma^2_{z_2}\right)} \right] z_1 + \left[ \frac{\sigma^2_{z_1}}{\left(\sigma^2_{z_1} + \sigma^2_{z_2}\right)} \right] z_2 \quad (4.3)
$$

$$
1/\sigma^2 = \left(1/\sigma^2_{z_1}\right) + \left(1/\sigma^2_{z_2}\right) \quad (4.4)
$$

and Figure 4.4 shows the curve of this new conditional density. Equation 4.4 implies that due to lower value of $\sigma$ than either $\sigma_{z_1}$ or $\sigma_{z_2}$, the precision of the new position estimate has been increased by joining the two measurements.
Based on this density, the best estimate is

\[ \hat{x}(t_2) = \mu \]  \hspace{1cm} (4.5)

with an error variance, \( \sigma^2 \).

By analyzing Equation 4.3, one can get more understanding of underlying theory. If \( \sigma_{z_1} \) were equal to \( \sigma_{z_2} \) (that is to say the measurements were equal in precision), from the equation, the best estimate would be simply the average of two measurements. In another point of view, if \( \sigma_{z_1} \) were larger than \( \sigma_{z_2} \) (that is to say the precision of the measurement \( z_2 \) were higher than that of \( z_1 \)), from the equation, the measurement \( z_2 \) would be “weighted” more heavily than \( z_1 \). Lastly, even for very large value of \( \sigma_{z_2} \), the variance of the estimate is less than \( \sigma_{z_1} \), that is, even poor quality measurement yields some information and enriches the filter.

Figure 4.3: Conditional density of position based on the measurement \( z_2 \) alone.
output in precision.

Rearranging Equation 4.3 gives

\[
\hat{x}(t_2) = \left[\frac{\sigma^2_{z_2}}{\sigma^2_{z_1} + \sigma^2_{z_2}}\right] z_1 + \left[\frac{\sigma^2_{z_1}}{\sigma^2_{z_1} + \sigma^2_{z_2}}\right] z_2
\]

\[= z_1 + \left[\frac{\sigma^2_{z_1}}{\sigma^2_{z_1} + \sigma^2_{z_2}}\right] [z_2 - z_1] \tag{4.6}\]

and the form used in Kalman Filter algorithm can be obtained with further arrangements as

\[
\hat{x}(t_2) = \hat{x}(t_1) + K(t_2)[z_2 - \hat{x}(t_1)] \tag{4.7}\]

where

\[
K(t_2) = \frac{\sigma^2_{z_1}}{\sigma^2_{z_1} + \sigma^2_{z_2}} \tag{4.8}\]

Equations 4.7 and 4.8 provide the optimal estimate at time \(t_2\), \(\hat{x}(t_2)\), as the best prediction of its value before \(z_2\) is taken, \(\hat{x}(t_1)\), plus a correction term of an optimal
weighting value times the difference between $z_2$ and the previous best prediction, $\hat{x}(t_1)$. This “predictor-corrector” structure makes the filter more efficient and feasible due to the need for fast and little algorithm consuming less CPU-time and memory. If this structure is applied to this example, he becomes the predictor and his friend becomes the corrector.

From Equations 4.4 and 4.8, the Kalman Filter-algorithmic form of the correction equation for the error variance can be obtained as

$$\sigma^2_x(t_2) = \sigma^2_x(t_1) - K(t_2)\sigma^2_x(t_1) \quad (4.9)$$

Up to here, the static estimation problem is handled. Presume that he moves before the next measurement is taken. Assume that the dynamic model of his motion is simply as

$$\frac{dx}{dt} = u + w \quad (4.10)$$

where $u$ is his velocity and $w$ is the noise term denoting his uncertainty in the knowledge of the actual velocity due to disturbances, effects not included in the first order model and the like. The model of the “noise” $w$ is approximated as a white Gaussian with zero mean and variance $\sigma^2_w$.

When the behavior of the conditional density of the position is analyzed, given $z_1$ and $z_2$, during the motion (see Figure 4.5), it is seen that, beginning with the best estimated position, $\hat{x}(t_2)$, and as time goes, the density moves along the $x$-axis at the velocity $u$, while spreading out about its mean according to the model of motion. It spreads out; because, as he moves, his knowledge of exact position becomes less certain due to the continuous addition of “noise” in the model. At time $t_3^-$, just before the measurement at time $t_3$, the density $f_{x(t_3)|z(t_1),z(t_2)}(x|z_1, z_2)$
shown in Figure 4.5 is a Gaussian density with mean and variance given by

\[ \hat{x}(t_3^-) = \hat{x}(t_2) + u [t_3 - t_2] \]  \hspace{1cm} (4.11) \\

\[ \sigma^2_x(t_3^-) = \sigma^2_x(t_2) - \sigma^2_w [t_3 - t_2] \]  \hspace{1cm} (4.12)

where \( \hat{x}(t_3^-) \) is the optimal prediction of the position at time \( t_3^- \) and \( \sigma^2_x(t_3^-) \) is the variance of error in the prediction.

After a measurement is taken at time \( t_3 \) as \( z_3 \) with an assumed variance \( \sigma^2_{z_3} \), as before, now there are two Gaussian densities available; one incorporating all the information before measurement and the other provided by the new measurement. When these densities are combined, a Gaussian density containing the information about the new best estimate of the position, is obtained, and
expressed by the mean
\[ \hat{x}(t_3) = \hat{x}(t_3^-) + K(t_3) \left[ z_3 - \hat{x}(t_3^-) \right] \] (4.13)
and variance
\[ \sigma^2_x(t_3) = \sigma^2_x(t_3^-) - K(t_3)\sigma^2_x(t_3^-) \] (4.14)
where the optimal gain \( K(t_3) \) is given by
\[ K(t_3) = \sigma^2_x(t_3^-)/ \left[ \sigma^2_x(t_3^-) + \sigma^2_{z_3} \right] \] (4.15)

Note the form of the equation for \( K(t_3) \), Equation 4.15. If the measurement noise variance, \( \sigma^2_{z_3} \) is large, then the optimal weighting factor, \( K(t_3) \) becomes small. That means; the measurement with a larger noise should get less importance and weighted lightly. In the limit as \( \sigma^2_{z_3} \) goes to infinity, \( K(t_3) \) becomes zero and \( \hat{x}(t_3) \) equals \( \hat{x}(t_3^-) \), i.e. a measurement with infinite noise is entirely rejected. If the variance of the dynamic system noise, \( \sigma^2_w \) is large then, the variance of the prediction, \( \sigma^2_x(t_3^-) \) becomes large (see Equation 4.12), resulting in large \( K(t_3) \). That means; the measurement should be weighted more heavily than the prediction done by means of the system model. Furthermore, as \( \sigma^2_w \) goes to infinity, so does \( \sigma^2_x(t_3^-) \) and \( K(t_3) \) becomes unity, so Equation 4.13 gives
\[ \hat{x}(t_3) = \hat{x}(t_3^-) + 1 \cdot \left[ z_3 - \hat{x}(t_3^-) \right] = z_3 \] (4.16)

Therefore, the filter completely ignores the information obtained from the system dynamic model and takes the measurement as the optimal estimate. However, if the model and the optimal estimate at time \( t_2 \), \( \hat{x}(t_2) \) are perfect (i.e., \( \sigma^2_w = 0 \) and \( \sigma^2_x(t_2) = 0 \)), then the new measurement, \( z_3 \) is rejected and the prediction before
measurement, \( \hat{x}(t^-) \) becomes the best estimate. In other words, \( K(t_3) \) becomes zero and \( \hat{x}(t_3) \) is equal to \( \hat{x}(t^-) \).

The structural algorithm of the filter can be summarized as follows. Firstly, a measurement with its variance is provided to the estimation problem. In conventional Kalman Filtering estimation, it corresponds to the initial condition knowledge needed by the estimation algorithm to start. Secondly, by some means (in the example, it was the capability of the friend), a measurement for inferring the position is obtained. In Kalman filter algorithm, this measurement is utilized in the update stage to generate the best estimation of the state and its variance. Thirdly, the statistical model of the movement is used to predict the position and its variance after some time passed. This process matches the propagation stage which is executed in the Kalman Filtering to achieve the best prediction of the state and its variance. Even though, not realized in the example, the next step is to process the update stage again for yielding the best estimate using the best prediction and the new measurement. Then, again the term is for the prediction stage and so on. Thus, the recursive property of the Kalman Filtering algorithm becomes obvious and it is graphically shown in Figure 4.6.

4.3 Linear Dynamic Models

A physical system that may be driven by known control efforts and by unknown inputs can be considered for which the characteristics of some system outputs by specifying their mean and covariance is wanted to be determined. This characterization may be compulsory to begin designs of estimators or controllers for a process and to analyze the performance of such devices. Therefore, a
Figure 4.6: Graphical presentation of recursive nature of Kalman Filtering estimation.

A mathematical model of the process is a common alternative to produce a sufficient representation of the behavior of the actual process outputs. This sufficiency depends on the purpose of using this representation.

The Kalman Filter requires a linear state space model describing the system dynamics and the relationships between the system states and the measurements. The Kalman Filter is named according to the type of the model selected. Conventionally, three types of model are used in Kalman Filter implementations, resulting in three different types of the filter so, models are given as follows, (i) continuous-time system model with discrete-time measurements, (ii) discrete-time system model with discrete-time measurements, (iii) continuous-time system model with continuous-time measurements (See Appendix B).

### 4.4 Kalman Filter Problem Formulation

Consider a system, adequately described in the form of a linear stochastic differential equation for yielding the state propagation, using discrete-time noise-corrupted linear measurements given as the system outputs. Therefore, the
The system model derived in Appendix B.1 is given by

\[ \dot{x}(t) = F(t)x(t) + B(t)u(t) + G(t)w(t) \]  

where the noise process, \( w(t) \) is modelled as white Gaussian noise with statistics

\[ E\{w(t)\} = 0 \]  

\[ E\{w(t)w^T(t')\} = \begin{cases} Q(t) & , t = t' \\ 0 & , t \neq t' \end{cases} \]  

Measurements are available at discrete-time points \( t_0, t_1, t_2, ..., t_k, ... \) and are described by the measurement model

\[ z(t_k) = H(t_k)x(t_k) + v(t_k) \]  

where the noise \( v(t_k) \) is modelled as a discrete-time white Gaussian noise with statistics

\[ E\{v(t_k)\} = 0 \]  

\[ E\{v(t_k)v^T(t_l)\} = \begin{cases} R(t_k) & , t_k = t_l \\ 0 & , t_k \neq t_l \end{cases} \]  

Then, the problem is to combine measurement data taken from the actual system with the information, provided by the system model and the measurement model with statistical descriptions of their uncertainties, in order to produce an optimal estimate of the system state. This problem has been solved by means of Kalman Filtering technique and the detailed derivation can be found in literature (Gelb, 1974 and Maybeck, 1979). Therefore, the mathematical and algorithmic
description of the Kalman Filtering solution to the estimation problem will be introduced in the following section.

4.5 Kalman Filter Algorithm

The Kalman Filter has a two-step recursive calculation algorithm whose timing diagram is given in Figure 4.7. The first named as *the propagation stage* is responsible to calculate the prediction of the state at the current time using the best state estimate at the previous time step. The second is named as *the update stage* and updates the prediction found in the first stage using the measurements taken from the actual process and calculates the best state estimate. The propagation stage integrates Equation 4.23 and Equation 4.24 from the previous time step $t_{k-1}$ to the current time $t_k$ and uses the best state estimate $\hat{x}^+(t_{k-1})$ and its error covariance $P^+(t_{k-1})$ at the previous time step $t_{k-1}$, in order to calculate the prediction of the state, $\hat{x}^-(t_k)$ and its error covariance $P^-(t_k)$ at the current time step $t_k$.

\[
\dot{x}(t) = F(t)\tilde{x}(t) + B(t)u(t) \tag{4.23}
\]

\[
\dot{P}(t) = F(t)P(t) + P(t)F^T(t) + G(t)Q(t)G^T(t) \tag{4.24}
\]

The update stage utilizes Equation 4.25 and Equation 4.26 and updates the prediction of the state $\hat{x}^-(t_k)$ and its error covariance $P^-(t_k)$ at the current time step $t_k$.

\[
\hat{x}^+(t_k) = \hat{x}^-(t_k) + K(t_k)\left[z(t_k) - H(t_k)\hat{x}^-(t_k)\right] \tag{4.25}
\]

\[
P^+(t_k) = [I - K(t_k)H(t_k)]P^-(t_k) \tag{4.26}
\]
where
\[ K(t_k) = P^-(t_k)H^T(t_k) \left[ H(t_k)P^-(t_k)H^T(t_k) + R(t_k) \right]^{-1} \] (4.27)

In order to initiate the Kalman Filter algorithm, the initial conditions incorporating the initial state, \( \hat{x}(t_0) = \hat{x}_0 \) and its error covariance, \( P(t_0) = P_0 \) are required. At the initialization time step, when the first measurement is taken, requirements of the best state estimate, \( \hat{x}^+(t_{k-1}) = \hat{x}^+(t_{-1}) \) and it error covariance, \( P^+(t_{k-1}) = P^+(t_{-1}) \) at the time step \( t_{-1} \) are supplied by replacing with the initial state \( \hat{x}_0 \) and its error covariance \( P_0 \). Moreover, the quantities used in the Kalman Filter algorithm can be traced through Figure 4.7.

### 4.6 Extended Kalman Filter

As the name implies, Extended Kalman Filter (EKF) is the extended version of the Kalman Filter handling the estimation issues in the nonlinear system theory. EKF uses the nonlinear model of the system given by Equation 4.28

\[ \dot{x}(t) = f(x(t), u(t), t) + G(t)w(t) \] (4.28)
where \( f \) is the vector of the nonlinear system functions and the noise process, \( w(t) \) is modelled as white Gaussian noise with statistics

\[
E\{w(t)\} = 0 \quad (4.29)
\]

\[
E\{w(t)w(t')^T\} = \begin{cases} 
Q(t), & t = t' \\
0, & t \neq t'
\end{cases} \quad (4.30)
\]

and the nonlinear measurement model written as

\[
z(t_k) = h(x(t_k), t_k) + v(t_k) \quad (4.31)
\]

where \( h \) is the vector of the nonlinear measurement functions and noise process, \( v(t_k) \) is modelled as white Gaussian noise with statistics

\[
E\{v(t_k)\} = 0 \quad (4.32)
\]

\[
E\{v(t_k)v(t_l)^T\} = \begin{cases} 
R(t_k), & t_k = t_l \\
0, & t_k \neq t_l
\end{cases} \quad (4.33)
\]

The only difference from their linear counterparts is the inability to write the coefficient matrices separately from the state and input vectors. Although, this form of the models cannot be used in the Kalman Filter algorithm, they could be converted to their linearized forms by using Taylor series expansion at the expense of some approximation errors. In linearization, the model can be linearized each time a new measurement is taken from the system, about an active origin specified by an updated state estimate obtained from a new measurement. If \( f \) is expanded about the current estimate of the state vector, then

\[
f(x(t), u(t), t) = f(\hat{x}(t), u(t), t) + \left. \frac{df(x(t), u(t), t)}{dx(t)} \right|_{x=\hat{x}} (x - \hat{x}) + \cdots \quad (4.34)
\]
where it is assumed the required partial derivatives exist. If the variation \((x - \hat{x})\) is small the first-order approximation to \(f(x(t), u(t), t)\) is obtained by neglecting all higher order terms but the first order. Then Equation 4.34 may be written as

\[ f(x(t), u(t), t) - f(\hat{x}(t), u(t), t) = F(\hat{x}(t), u(t), t)(x - \hat{x}) \]  

(4.35)

where

\[ F(\hat{x}(t), u(t), t) = \left. \frac{df(x(t), u(t), t)}{dx(t)} \right|_{x = \hat{x}} \]  

(4.36)

and its \(ij\)th element is given by

\[ f_{ij}(\hat{x}(t), u(t), t) = \left. \frac{df_i(x(t), u(t), t)}{dx_j(t)} \right|_{x(t) = \hat{x}(t)} \]  

(4.37)

Equation 4.35 indicates that \(f(x(t), u(t), t) - f(\hat{x}(t), u(t), t)\) is linearly proportional to \((x - \hat{x})\) by a coefficient matrix \(F(\hat{x}(t), u(t), t)\).

In the same way of obtaining the linear system model, the measurement model can be linearized about \(\hat{x}^-(t_k)\) and written as

\[ h(x(t_k), t_k) - h(\hat{x}^-(t_k), t_k) = H(\hat{x}^-(t_k), t_k)(x(t_k) - \hat{x}^-(t_k)) \]  

(4.38)

where

\[ H(\hat{x}^-(t_k), t_k) = \left. \frac{dh(x(t_k), t_k)}{dx(t_k)} \right|_{x(t_k) = \hat{x}^-(t_k)} \]  

(4.39)

and its \(ij\)th element is given by

\[ h_{ij}(\hat{x}^-(t_k), t_k) = \left. \frac{dh_i(x(t_k), t_k)}{dx_j(t_k)} \right|_{x(t_k) = \hat{x}^-(t_k)} \]  

(4.40)

When these linearized models described by \(F(\hat{x}(t), u(t), t)\) and \(H(\hat{x}^-(t_k), t_k)\) are implemented in the Kalman Filter equations given in Section 4.5, the extended Kalman Filter are obtained as shown in Table 4.1. Moreover, the algorithm of EKF is the same as that of linear counterpart introduced in Section 4.5.
Table 4.1: Equations of Continuous-Discrete Extended Kalman Filter

<table>
<thead>
<tr>
<th>Model</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>( \dot{x}(t) = f(x(t), u(t), t) + G(t)w(t); w(t) \sim N(0, Q(t)) )</td>
</tr>
<tr>
<td>Measurement</td>
<td>( \bar{z}(t_k) = h(x(t_k), t_k) + v(t_k); k = 0, 1, 2, \ldots; v(t_k) \sim N(0, R(t_k)) )</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>( x(0) = \hat{x}(0) = \hat{x}_0 )</td>
</tr>
<tr>
<td>Error Covariance</td>
<td>( P(0) = P_0 )</td>
</tr>
<tr>
<td>Propagation</td>
<td></td>
</tr>
<tr>
<td>State Estimate</td>
<td>( \hat{x}(t) = f(\hat{x}(t), u(t), t) )</td>
</tr>
<tr>
<td>Error Covariance</td>
<td>( \dot{P}(t) = F(\hat{x}(t), u(t), t)P(t) + P(t)F^T(\hat{x}(t), u(t), t) + Q(t) )</td>
</tr>
<tr>
<td>Update</td>
<td></td>
</tr>
<tr>
<td>State Estimate</td>
<td>( \hat{x}^+(t_k) = \hat{x}^-(t_k) + K(t_k) \left[ \bar{z}(t_k) - h(\hat{x}^-(t_k), t_k) \right] )</td>
</tr>
<tr>
<td>Error Covariance</td>
<td>( P^+(t_k) = \left[ I - K(t_k)H(\hat{x}^-(t_k), t_k) \right] P^-(t_k) )</td>
</tr>
<tr>
<td>Gain Matrix</td>
<td>( K(t_k) = \frac{P^-(t_k)H^T(\hat{x}^-(t_k), t_k)}{\left[ H(\hat{x}^-(t_k), t_k)P^-(t_k)H^T(\hat{x}^-(t_k), t_k) + R(t_k) \right]}^{-1} )</td>
</tr>
<tr>
<td>Definitions</td>
<td></td>
</tr>
<tr>
<td>( N(\mu, \sigma) ): Gaussian distribution with mean, ( \mu ) and covariance ( \sigma )</td>
<td></td>
</tr>
<tr>
<td>( F(\hat{x}(t), u(t), t) = \frac{df(x(t), u(t), t)}{dx(t)}</td>
<td>_{x=\hat{x}} )</td>
</tr>
<tr>
<td>( H(\hat{x}^-(t_k), t_k) = \frac{dh(x(t_k), t_k)}{dx(t_k)}</td>
<td>_{x(t_k)=\hat{x}^-(t_k)} )</td>
</tr>
</tbody>
</table>
In this chapter, the technique of extended Kalman Filter estimation will be applied to MBDC in order to infer the column compositions from the temperature measurements. Firstly, the observability of a multicomponent batch distillation column, which is a must to be able to estimate the system states, will be analyzed. Secondly, due to computational and algorithmic requirements given in Chapter 4, the simplified models of the system and the measurement devices will be developed by taking the rigorous simulation model as base. Then in the next section, the need for Jacobian matrices both for the system and the measurement models will be achieved through analytical linearization.
5.1 Observability Criteria for a MBDC

Most control laws used in the batch distillation are feedback laws and the state-space description of dynamics realizes that the information required for feedback control is the state of the system (Jacobs, 1974). As in the case of batch distillation, in most real controlled processes, the system state (i.e. compositions in the batch distillation) is not identical with the observable outputs (i.e. temperatures in the distillation) but there is a time-varying relationship between the states, \( X \) and the outputs, \( z \) as given in Equation 5.1.

\[
z(t_k) = h(X(t_k), t_k)
\]  

(5.1)

Therefore, the question arises whether or not it is possible to evaluate the state form observations of the output (i.e. measurements). The observability criteria is to be satisfied for solving the problem of inferring unmeasurable state variables from measurements in the minimum possible length of time. So the system with the measurement model (Equation 5.1) is said to be observable if the output, \( z \) embodies sufficient information to infer the state, \( X \) in a finite time.

Here it should be noted that since the observability criteria is evaluated from the linear dynamic measurement model, although the system is observable in reality, corresponding mathematical model may not possess the property of observability.

Employing a degree-of-freedom concept, Yu et al. (1987) found that a distillation column is observable if the number of measurements is at least \((NC - 1)\). The study of Quintero-Marmol et al. (1991), dealing with the design of an extended Luenberger Observer for MBDC, concluded that even though the linear observer in theory needs only \((NC - 1)\) temperature measurements
To be observable, the nonlinear observer needed at least \((NC)\) thermocouples to be effective. In addition, to improve the convergence without effecting the robustness, the authors recommended the use of \((NC + 2)\) measurements.

### 5.2 EKF State Observer for MBDC

As described previously in Chapter 4, EKF algorithm requires the stochastic models of the system (Equation 5.2) and the measurement processes (Equation 5.3)\(^1\)

\[
\dot{X}(t) = f(X(t), u(t), t) + G(t)w(t) \tag{5.2}
\]

where \(f\) is the vector of the nonlinear system functions and the noise process, \(w(t)\) is modelled as white Gaussian noise with statistics

\[
E\{w(t)\} = 0
\]

\[
E\{w(t)w(t')^T\} = \begin{cases} Q(t) & , t = t' \\ 0 & , t \not= t' \end{cases}
\]

\[
z(t_k) = h(X(t_k), t_k) + v(t_k) \tag{5.3}
\]

where \(h\) is the vector of the nonlinear measurement functions and the noise process, \(v(t_k)\) is modelled as white Gaussian noise with statistics

\[
E\{v(t_k)\} = 0
\]

\(^1\)Up to here the states of the EKF estimator have been denoted by \(x\) which may cause confusion with the composition in the column model. So from now on, the state of the estimator will be denoted by capital, \(X\)
\[
E \{ v(t_k)v(t_l)^T \} = \begin{cases} 
R(t_k), & t_k = t_l \\
0, & t_k \neq t_l 
\end{cases}
\]

In Equation 5.2, \( G(t) \) is assumed as unity.

In addition, EKF algorithm needs the linearized versions of these two models, specified by the Jacobian matrices and for the system it is given by Equation 5.4 and for the measurement process by Equation 5.5.

\[
F(\hat{X}(t), u(t), t) = \left. \frac{df(X(t), u(t), t)}{dX(t)} \right|_{X=\hat{X}} 
\]

\[
H(\hat{X}^-(t_k), t_k) = \left. \frac{dh(X(t_k), t_k)}{dX(t_k)} \right|_{X(t_k)=\hat{X}^-(t_k)}
\]

Lastly, in order to initiate the EKF algorithm, the information of initial conditions is required and stated by \( \hat{X}_0 \) for the states and by \( P_0 \) for the error covariances.

As a result, in the following sections, firstly the nonlinear models for the system and for the temperature measurements will be developed in the form required for EKF algorithm. However, the previously developed model for rigorous simulation of the batch column is not suitable for realistic situation in order to be implemented in EKF algorithm. Since it is difficult to obtain the required values of vapor and liquid flowrates and tray holdups with time. In addition, the complexity of the simulation model require high computational time and memory. Therefore, the rigorous column model for simulation will be simplified and the obtained nonlinear models will be linearized to achieve the corresponding Jacobian matrices.
5.2.1 Model Simplification

To simplify the rigorous simulation model of MBDC, which was derived in Chapter 3 and summarized in Table 3.2, some additional assumptions is to be employed. To get rid of the calculations of variable molar holdups on trays, the variations in the liquid holdup can be neglected by adding the assumption of constant molar holdup on trays. Another important simplification is made by neglecting the energy dynamics in the column. Also the assumption of ideal trays allow not to deal with the tray efficiencies. Lastly, instead of using Peng-Robinson EOS for VLE description, more simple correlation is used. The alternative selected in this study is to use Rault’s Law with Antoine’s vapor pressure correlation.

As a result of the assumption of constant tray holdup and neglected energy dynamics, the vapor flowrates throughout the column become equal and so the liquid flowrates do. Therefore the flowrates is written as

\[ V_1 = V_2 = \cdots = V_{NT+1} = V \]  \hspace{1cm} (5.6)

\[ L_2 = L_3 = \cdots = L_{NT+2} = L \]  \hspace{1cm} (5.7)

Also the use of ideal tray assumption yields

\[ y_{ij} = y_{ij}^* \text{ for } i = 1 \ldots NT + 2 \]
\[ j = 1 \ldots NC \]  \hspace{1cm} (5.8)

Equation 5.8 means that the actual composition of the vapor phase is the same as the composition of the vapor phase at equilibrium with the liquid of composition, \( x_{ij} \). Lastly, the phase equilibrium model is described by means of Rault’s Law
stated as

\[ y_{ij}^* = x_{ij} \left( \frac{P_v^j}{P_i} \right) \text{ for } i = 1 \ldots NT + 2 \]

\[ j = 1 \ldots NC \tag{5.9} \]

where \( P_v^j \) is the vapor pressure of \( j^{th} \) component and specified by Antoine’s vapor pressure correlation defined as

\[ \log (P_v^j) = a_j - \frac{b_j}{c_j - T_i} \text{ for } i = 1 \ldots NT + 2 \]

\[ j = 1 \ldots NC \tag{5.10} \]

where \( a_j \), \( b_j \) and \( c_j \) are Antoine’s constants for \( j^{th} \) component. Also the equilibrium temperature on \( i^{th} \) tray, \( T_i \) is calculated from Equation 5.11 by an trial-and-error method and the vapor phase fractions, \( y_{ij}^* \) are then obtained from Equation 5.9 with Equation 5.10.

\[ T_i = \sum_{l=1}^{NC} x_{il} \cdot 10^{\left[ a_i - \frac{b_i}{c_i - T_i} \right]} = P_i \tag{5.11} \]

Consequently, applying Equation 5.6-5.10 to the rigorous model equations given in Table 3.2 produces the simplified model equations which are listed in Table 5.1.
Table 5.1: Summary of MBDC Simplified Model Equations

<table>
<thead>
<tr>
<th>Compositions and Holdups</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reboiler Dynamics</strong></td>
<td></td>
</tr>
<tr>
<td>$\frac{dM_1}{dt} = L - V$</td>
<td></td>
</tr>
<tr>
<td>$\frac{dx_{1j}}{dt} = [L(x_{2j} - x_{1j}) - V(y_{1j} - x_{1j})] / M_1$</td>
<td></td>
</tr>
<tr>
<td>$j = 1 \ldots NC$</td>
<td></td>
</tr>
<tr>
<td><strong>Tray Dynamics</strong></td>
<td></td>
</tr>
<tr>
<td>$\frac{dx_{ij}}{dt} = [V(y_{i-1,j} - y_{ij}) + L(x_{i+1,j} - x_{ij})] / M_i$</td>
<td></td>
</tr>
<tr>
<td>$i = 2 \ldots NT + 1; j = 1 \ldots NC$</td>
<td></td>
</tr>
<tr>
<td><strong>Reflux-Drum Dynamics</strong></td>
<td></td>
</tr>
<tr>
<td>$\frac{dx_{NT+2,j}}{dt} = [V(y_{NT+1,j} - x_{NT+2,j})] / M_{NT+2}$</td>
<td></td>
</tr>
<tr>
<td>$j = 1 \ldots NC$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flowrates for given R and V₀</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vapor</strong></td>
<td></td>
</tr>
<tr>
<td>$V = V₀$</td>
<td></td>
</tr>
<tr>
<td><strong>Liquid</strong></td>
<td></td>
</tr>
<tr>
<td>$L = RD$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Composition Sums</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{n=1}^{NC} x_n = 1; \sum_{n=1}^{NC} y_n = 1$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pressure Drop Profile</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_i = P₁ - i \cdot (P₁ - P_{NT+2}) / NT$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thermodynamic Models</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rault’s Law</strong></td>
<td></td>
</tr>
<tr>
<td>$y_{ij} = x_{ij} \left( \frac{P_v^{n}}{T_i} \right)$</td>
<td></td>
</tr>
<tr>
<td>$i = 1 \ldots NT + 2; j = 1 \ldots NC$</td>
<td></td>
</tr>
<tr>
<td><strong>Antoine’s Vapor Pressure Correlation</strong></td>
<td></td>
</tr>
<tr>
<td>$\log \left( P_v^{n} \right) = a_j - \frac{b_j}{c_j - T_i}$</td>
<td></td>
</tr>
<tr>
<td>$i = 1 \ldots NT + 2; j = 1 \ldots NC$</td>
<td></td>
</tr>
<tr>
<td><strong>Equilibrium Temperature</strong></td>
<td></td>
</tr>
<tr>
<td>$T_i : \sum_{l=1}^{NC} x_{il} \cdot 10^{\left[ a_{il} - \frac{b_i}{T_i - r_l} \right]} = P_i$</td>
<td></td>
</tr>
<tr>
<td>$i = 1 \ldots NT + 2$</td>
<td></td>
</tr>
</tbody>
</table>
Nextly, the nonlinear models in EKF given by Equations 5.2 and 5.3 are defined in terms of states, inputs and outputs of the column simplified model by Equations 5.12 and 5.13

\[
\dot{X}(t) = f(X(t), u(t), t) + w(t) \quad (5.12)
\]

where

\[
X = [x_{11} \ldots x_{1NC}, \ldots, x_{NT+2,1} \ldots x_{NT+2,NC}]^T
\]

\[
f = [\dot{x}_{11} \ldots \dot{x}_{1NC}, \ldots, \dot{x}_{NT+2,1} \ldots \dot{x}_{NT+2,NC}]^T
\]

\[
v = [R, Q_1]^T
\]

\[
z(t_k) = h(X(t_k), t_k) + v(t_k) \quad (5.13)
\]

where

\[
z = [T_{M(1)} \ldots T_{M(NM)}]^T
\]

\[
h = [T_{M(1)}(x_{M(1),1} \ldots x_{M(1),NC}) \ldots T_{M(NM)}(x_{M(NM),1} \ldots x_{M(NM),NC})]^T
\]

and \(NM\) is the number of measurements, \(M(n)\) is indexing function returning the tray number of given measurement \(n\). More generally, the elements of \(f\) and \(X\) vectors are given as

\[
f_i = f_{(k-1)\cdot NC+n} = \dot{x}_{k,n} \quad \text{for} \quad k = 1 \ldots NT + 2 \quad \text{for} \quad n = 1 \ldots NC
\]

\[
X_j = X_{(p-1)\cdot NC+r} = x_{p,r} \quad \text{for} \quad p = 1 \ldots NT + 2 \quad \text{for} \quad r = 1 \ldots NC
\]

(5.14)
5.2.2 Linearization of the Simplified Model

The subject of this section is to find the general forms of the linear system matrix, $F$ (see Table 4.1) given as

$$F'(X(t), u(t), t) = \frac{df(X(t), u(t), t)}{dX(t)}$$  

(5.15)

and the linear measurement matrix, $H$ (see Table 4.1) defined as

$$H'(X(t_k), t_k) = \frac{dh(X(t_k), t_k)}{dX(t_k)}$$  

(5.16)

Evaluation of $F'$:

The elements of $F'$ is written as

$$F'_{ij} = \left[ \frac{df(X(t), u(t), t)}{dX(t)} \right]_{ij} = \frac{df_i(X(t), u(t), t)}{dX_j(t)}$$

(5.17)

for $j = 1 \ldots NC(NT + 2)$

$$i = 1 \ldots NC(NT + 2)$$

or

$$F'_{ij} = \frac{df_i}{dX_j} = \frac{df_{(k-1)NC+n}}{dX_{(p-1)NC+r}} = \frac{d(\dot{x}_{k,n})}{dx_{p,r}}$$

(5.18)

for $k = 1 \ldots (NT + 2)$

$$n = 1 \ldots NC$$

$$p = 1 \ldots (NT + 2)$$

$$r = 1 \ldots NC$$

The problem is to find each $\frac{df_{(k-1)NC+n}}{dX_{(p-1)NC+r}}$ for any combination of the indices $k$, $n$, $p$ and $r$. Due to lack of place, only the evaluation of a single row of $F'$ is given.

The solution of whole matrix is given in Appendix C.
If indices in Equation 5.18 are set as $k = 1$ and $n = 1$, the function $f_i$ becomes

$$f_1 = \dot{x}_1 = \dot{x}_{11} = \frac{L(x_{21} - x_{11}) - V(y_{11} - x_{11})}{M_1}$$

which describes the time derivative of component, $n = 1$ in the reboiler, $k = 1$. If the combinations of $p$ and $r$ are realized, corresponding elements of $F'$ is obtained as follows.

for $p = 1$:

$r = 1 \Rightarrow j = 1$

$$F'_{11} = \frac{df_1}{dx_{11}} = \left[ -L - V \left( \frac{dy_{11}}{dx_{11}} - 1 \right) \right] / M_1$$

$r = 2 \Rightarrow j = 2$

$$F'_{12} = \frac{df_1}{dx_{12}} = \left[ -V \frac{dy_{11}}{dx_{12}} \right] / M_1$$

$r = 3 \Rightarrow j = 3$

$$F'_{13} = \frac{df_1}{dx_{13}} = \left[ -V \frac{dy_{11}}{dx_{13}} \right] / M_1$$


for $p = 2$:

$r = 1 \Rightarrow j = 4$

$$F'_{14} = \frac{df_1}{dx_{21}} = \left[ L \right] / M_1$$

$r = 2 \Rightarrow j = 5$

$$F'_{15} = \frac{df_1}{dx_{22}} = 0$$

$r = 3 \Rightarrow j = 6$

$$F'_{16} = \frac{df_1}{dx_{23}} = 0$$
for $p = 3$:

\[ r = 1 \Rightarrow j = 7 \]
\[ F'_{17} = \frac{df}{dx_{31}} = 0 \]
\[ r = 2 \Rightarrow j = 8 \]
\[ F'_{18} = \frac{df}{dx_{32}} = 0 \]
\[ r = 3 \Rightarrow j = 9 \]
\[ F'_{19} = \frac{df}{dx_{33}} = 0 \]

“for $p > 3$” means taking the derivative of Equation 5.19 with respect to the composition of $r^t h$ component on the trays with indices larger than 3. Because there is no composition term with the tray indices larger than 3 in Equation 5.19, the elements of $F'$ becomes zero for $p = 4 \ldots (NT + 2)$ and $r = 1 \ldots NC$. This is mathematically represented by Equation 5.20.

\[ F_{1j} = \frac{df_1}{dX_{(p-1):NC+r}} = \frac{dx_{11}}{dx_{p,r}} = 0 \quad (5.20) \]

for $p = 4 \ldots (NT + 2)$

\[ r = 1 \ldots NC \]

Moreover, there are derivatives in the form of $\frac{dy}{dx}$ which are to be evaluated. For convenience, these calculations are also given in Appendix C.

**Evaluation of $H'$**:

The elements of $H'$ is written as

\[ H'_{ij} = \left[ \frac{dh(X(t_k), t_k)}{dX(t_k)} \right]_{ij} = \frac{dh_i(X(t_k), t_k)}{dX_j(t_k)} \quad (5.21) \]

for $j = 1 \ldots NM$

\[ i = 1 \ldots NC(NT + 2) \]
or

\[ H'_{ij} = \frac{d h_i}{d X_j} = \frac{d h_i}{d X_{(p-1)NC+r}} = \frac{d T_{M(i)}}{dx_{p,r}} \] (5.22)

for \( i = 1 \ldots NM \)

\( p = 1 \ldots (NT + 2) \)

\( r = 1 \ldots NC \)

The measurement model is written as

\[ h_i = T_{M(i)} : \sum_{l=1}^{NC} x_{il} 10^{a_{il}} = P_i \] (5.23)

for \( i = 1 \ldots NM \)

where the definition of \( \alpha \) is given in Appendix C.

Because the temperature of a tray is not a direct function of the compositions on the other trays, the elements of \( H' \) for \( p \neq M(i) \) becomes

\[ H'_{ij} = \frac{d h_i}{dx_{p,r}} = 0 \] (5.24)

for \( r = 1 \ldots NC \)

For \( p = M(i) \) and \( r = 1 \ldots NC \), the elements of \( H' \) is evaluated as follows

\[ H'_{ij} = \frac{d h_i}{dx_{p,r}} = \frac{dT_{M(i)}}{dx_{p,r}} = \frac{dT_p}{dx_{p,r}} = TXD_{pr} \] (5.25)

where the definition of \( TXD_{pr} \) is also given in Appendix C.

Consequently, all the information required for EKF estimator have been obtained. This information incorporates nonlinear and linearized models for the system of MBDC and the measurement process given respectively by \( f^i, h^i, F', H' \).
CHAPTER 6

SIMULATION CODE AND CASE COLUMN

6.1 Simulation Code

The developed rigorous model for MBDC and the designed EKF estimator is simulated on the platform of MATLAB where the written codes are named as “m-file”. For integration of ODE of the model and the estimator, a simple method of Euler’s integration is used with a step-size of $3 \times 10^{-4}$ hours. The written code given in Appendix D consists of three main m-file and uses the library for physical and thermodynamic properties of mixtures which utilizes Peng-Robinson EOS incorporated in the rigorous model of MBDC. The m-files are “Glob_Decls.M”, “Glob_Initial.m” and “Cont_Plant_Mfile.m” and the library is named as “thermo_LIBRARY.dll” which is written in FORTRAN Programming Language. Moreover main simulation code “Cont_Plant_Mfile.m” executes the m-files including the nonlinear and linearized models used in EKF algorithm. “thermo_LIBRARY.dll” library incorporates the functions for phase
equilibrium, enthalpy and density-molecular-weight calculations, respectively named as “thermo_Equilibrium”, “thermo_Enthalpy” an “thermo_Density”. All the m-file codes and the library “thermo_LIBRARY.dll” will be explained in detail and the overall structure of the simulation can be seen in Table 6.1.
Table 6.1: The overall structure of the simulation code.

<table>
<thead>
<tr>
<th>Main Program Codes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Glob_Dec.m</td>
<td></td>
</tr>
<tr>
<td>Glob_Initial.m</td>
<td></td>
</tr>
<tr>
<td>Cont_Plant_Mfile.m</td>
<td></td>
</tr>
<tr>
<td>PressureProfile.m</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thermodynamic Library “thermo_LIBRARY.dll”</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermodynamic Library MATLAB Interface Code</td>
<td></td>
</tr>
<tr>
<td>thermo_Init.m</td>
<td></td>
</tr>
<tr>
<td>thermo_Equilibrium.m</td>
<td></td>
</tr>
<tr>
<td>thermo_Enthalpy.m</td>
<td></td>
</tr>
<tr>
<td>thermo_Density.m</td>
<td></td>
</tr>
<tr>
<td>Thermodynamic Library Fortran DLL Code</td>
<td></td>
</tr>
<tr>
<td>thermo_LIBRARY.f</td>
<td></td>
</tr>
<tr>
<td>thermo_LIBRARY.h</td>
<td></td>
</tr>
<tr>
<td>common_plant.h</td>
<td></td>
</tr>
<tr>
<td>parameter.h</td>
<td></td>
</tr>
<tr>
<td>thermo_data.dat</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EKF Nonlinear Model Codes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KF_NonlinearModel_Sys.m</td>
<td></td>
</tr>
<tr>
<td>KF_NonlinearModel_Out.m</td>
<td></td>
</tr>
<tr>
<td>KF_NonlinearModel_Equilib.m</td>
<td></td>
</tr>
<tr>
<td>KF_NonlinearModel_Press.m</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EKF Linearized Model Codes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KF_LinearizedModel_SysF.m</td>
<td></td>
</tr>
<tr>
<td>KF_LinearizedModel_SysH.m</td>
<td></td>
</tr>
<tr>
<td>KF_LinearizedModel_Equilib.m</td>
<td></td>
</tr>
<tr>
<td>KF_LinearizedModel_Press.m</td>
<td></td>
</tr>
<tr>
<td>KF_LinearizedModel_CalcDyDx.m</td>
<td></td>
</tr>
<tr>
<td>KF_LinearizedModel_CalcTXD.m</td>
<td></td>
</tr>
</tbody>
</table>
6.1.1 Thermodynamic Library

This library has two parts, one part is written in MATLAB, while the other in FORTRAN. The MATLAB part forms only an interface between the MATLAB framework and the library’s FORTRAN code in which all the calculations are done. The MATLAB interface part consists of four m-file functions representing their equivalents in the FORTRAN code which are coupled in a single FORTRAN project. In the MATLAB part, “thermo_Init.m” provides initialization of the library; “thermo_Equilibrium” calls the function from FORTRAN-counterpart which calculates the phase equilibrium temperature and vapor composition for a given liquid composition, initial guess for equilibrium temperature and pressure; “thermo_Enthalpy” calls the function where the liquid and vapor phase enthalpies are evaluated from a given parameters of liquid and vapor composition, temperature and pressure; and finally, “thermo_Density” calls the function calculating average liquid density and molecular weight for the mixture given by its liquid composition, temperature and pressure. The source code for these four interface m-files are given in Appendix D.2.

The FORTRAN part of the library is the actual code where all the calculations are made. This code was written by Dokucu (2002) for the evaluation of physical properties in his study of controlling an industrial distillation column by Model Predictive Controller. In this work, the previously developed code is modified to be used in MATLAB framework. The details of the calculation algorithm can be found from the study of Dokucu (2002). The modified source code contains the main file, “thermo_LIBRARY.f”, and the files “thermo_LIBRARY.h”,

74
“common_plant.h”, “parameter.h” providing the definitions of some algorithmic parameters. The input file “thermo_data.dat” is for supplying the component specifications listed in Table 6.4. The sources for the FORTRAN part of the library “thermo_LIBRARY.dll” can be found in Appendix D.3.

6.1.2 M-files for Nonlinear Model Used in EKF

This part of the simulation program provides the instantaneous states and outputs of the simplified MBDC model at a given time. It has four m-files given in Appendix D.4. The m-file function “KF_NonlinearModel_Sys.m” calculates the time derivatives of the compositions (i.e. the states to be estimated) required for propagation stage of the EKF algorithm in order to integrate the state, up to the time of new measurement and it corresponds to Equation 5.2. “KF_NonlinearModel_Out.m” evaluates the predicted outputs of the system required for update stage of the EKF algorithm in order to obtain the updated state estimates and it is the equivalent of Equation 5.2.

“KF_NonlinearModel_Equilib.m” is used for computing the phase equilibrium temperature and vapor phase composition of a mixture specified by its liquid composition and pressure. This subroutine employs Newton-Raphson Root Finding technique to solve Equation 5.11 which incorporates Rault’s Law and Antoine’s equation. “KF_NonlinearModel_Press.m” evaluates the pressure drop profile in the column for given pressures of reboiler and reflux-drum employing Equation 3.36.
6.1.3 M-files for Linearized Model Used in EKF

As in the case of nonlinear model, this part has also four main m-files which are shown in Appendix D.5. It is responsible for computation of linear system coefficient matrix (Equation 5.4) and linear measurement coefficient matrix (Equation 5.5). “KF_LinearizedModel_SysF.m” and “KF_LinearizedModel_SysH.m” calculate the system matrix and the measurement matrix, respectively. Unlikely the nonlinear model case, these two m-files also employ two more subroutines “KF_LinearizedModel_CalcDyDx.m” and “KF_LinearizedModel_CalcTXD.m” which provide the matrix of derivative for vapor phase composition vector with respect to liquid composition vector and the matrix of derivatives for the temperature vector with respect to the liquid composition vector, respectively. The other two main m-files “KF_LinearizedModel_Equilib.m” and “KF_LinearizedModel_Press.m” are the same as in nonlinear model case.

6.1.4 Main Simulation Program

As previously stated, the main program consists of three m-files and a supporting function “PressureProfile.m” which simply computes the linear pressure profile in the column. Two m-files “Glob_Decss.m” and “Glob_Initial.m” are responsible to define and initialize the operational parameters and the trial-design parameters for EKF estimator (e.g. initial conditions, number and location of measurements, process and measurement covariance matrices). Lastly, but most importantly, the m-file “Cont_Plant_Mfile.m” is the main file where the simulation of whole plant is realized and its overall flow diagram is shown in Figure 6.1.
Declare variables and Call "Glob_Decs.m"

Initialize "thermo_LIBRARY.dll" and Call "Glob_Initial.m"

Initialize Rigorous Simulation Variables

Initialize EKF Estimator Variables

Calculate physical properties from "thermo_LIBRARY.dll"

Calculate molar holdups for trays and reflux-drum

If \( t=0 \)

Yes

Calculate initial flowrates

No

Calculate approx. derivatives

If \( R=\infty \)

Yes

Calculate flowrates for Total Reflux Ratio

No

Calculate flowrates for Finite Reflux Ratio

Evaluate time derivatives of compositions

Column Model Calculations

Figure 6.1: Flow diagram of the main program algorithm.
Take temperature measurements and system inputs

Find predicted measurements, $h$ and linearized measurement matrix, $H$

Find best estimates and error covariances

Set current best estimates and error covariances as initial conditions for propagation

Run controller to obtain the manipulated variables

Set next measurement time as $t_{m} = t_{m} + Dt_{m}$

Evaluate time derivatives of states and error covariances

If $t = t_{stop}$

Write variables to output files and display

Integrate and then normalize the column compositions

Integrate state estimates and error covariances

Increment integration time as $t = t + Dt$

Stop Simulation

Figure 6.2: Flow diagram of the main program algorithm (cont’d).
6.2 Description of Case Column

A column for simulation is selected as a case study in order to examine the reliability and accuracy of the developed rigorous simulation and to test the performance of EKF estimator designed in this work.

The selected column is the one which was simulated by Mujtaba et al. (1993) in their study on the subject of optimal operation of MBDC and it separates the mixture of cyclo-hexane, n-heptane and toluene. The sketch of the column can be seen in Figure 3.1 and the design specifications of the column are listed in Table 6.2.

Table 6.2: Design parameters for the case column

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trays</td>
<td>8</td>
</tr>
<tr>
<td>Condenser-Reflux-Drum Holdup</td>
<td>0.02 kmol</td>
</tr>
<tr>
<td>Trays Holdup</td>
<td>0.01 kmol</td>
</tr>
<tr>
<td>Maximum Boil-up Rate</td>
<td>2.75 kmol/h</td>
</tr>
</tbody>
</table>

The batch distillation column is under the perfect control of reflux-drum level and has two degrees-of-freedom for manipulation which are reboiler heat load, $Q_1$ and reflux-ratio, $R$. In this study, the reboiler heat load, $Q_1$ is kept at its maximum value given by design while the reflux-ratio, $R$ is used as manipulated variable in order to realize the optimal operation policy recommended by Mujtaba et al. (1993) and previously described in Section 3.1. For the column under study, Table 6.3 shows the optimal reflux-ratio profile given by Mujtaba et al. (1993) in order to yield two product-cuts with the desired purity levels of 0.9 and 0.8 from the mixture of cyclo-hexane, n-heptane and toluene with the composition of $(0.407, 0.394, 0.199)$. Therefore, in all of the simulations in this
study, this optimal reflux ratio profile is employed.

Table 6.3: Parameters for the optimal reflux ratio policy.

<table>
<thead>
<tr>
<th>Amount of Fresh Feed</th>
<th>2.93 kmol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed Composition</td>
<td></td>
</tr>
<tr>
<td><em>cyclo-hexane</em></td>
<td>0.407</td>
</tr>
<tr>
<td><em>n-heptane</em></td>
<td>0.394</td>
</tr>
<tr>
<td><em>toluene</em></td>
<td>0.199</td>
</tr>
<tr>
<td>Desired purity of comp. 1 in Product-cut 1</td>
<td>0.9</td>
</tr>
<tr>
<td>Desired purity of comp. 2 in Product-cut 2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Optimum Reflux Profile**

<table>
<thead>
<tr>
<th>Time Interval (hour)</th>
<th>Reflux Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 2.04</td>
<td>0.875</td>
</tr>
<tr>
<td>2.04 – 3.4</td>
<td>0.911</td>
</tr>
<tr>
<td>3.4 – 6.17</td>
<td>0.933</td>
</tr>
<tr>
<td>6.17 – 6.51</td>
<td>0.831</td>
</tr>
<tr>
<td>6.51 – 8.35</td>
<td>0.876</td>
</tr>
</tbody>
</table>

As an addition to Table 6.2, Table 6.4 lists the parameters required for the simulation of the case column developed in Chapter 3. Table 6.5 tabulates the parameters for the simplified model implemented in EKF estimator as an addition to Table 6.2.
Table 6.4: Additional parameters for the rigorous simulation of the case column

<table>
<thead>
<tr>
<th>Column Specifications</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tray Volumetric Holdups</td>
<td>0.00132 $m^3$</td>
</tr>
<tr>
<td>Reboiler Heat Load</td>
<td>8.15x10$^7$ $J/h$</td>
</tr>
<tr>
<td>Reboiler Pressure</td>
<td>101000 $Pa$</td>
</tr>
<tr>
<td>Reflux-Drum Pressure</td>
<td>101600 $Pa$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties of Components</th>
<th>cyclo-hexane</th>
<th>n-heptane</th>
<th>toluene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mol. Weight ($kg/mol$)</td>
<td>84.141x10$^{-3}$</td>
<td>100.204x10$^{-3}$</td>
<td>92.141x10$^{-3}$</td>
</tr>
<tr>
<td>Critical Temp. ($K$)</td>
<td>553.68</td>
<td>540.20</td>
<td>591.80</td>
</tr>
<tr>
<td>Boiling Temp. ($K$)</td>
<td>353.15</td>
<td>371.55</td>
<td>383.95</td>
</tr>
<tr>
<td>Critical Press. ($Pa$)</td>
<td>4.100x10$^6$</td>
<td>2.736x10$^6$</td>
<td>4.113x10$^6$</td>
</tr>
<tr>
<td>Acentric Factor</td>
<td>0.212</td>
<td>0.346</td>
<td>0.262</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heat Capacity Coeff. ($J/mol.K$)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C^a_p$</td>
<td>-66.672040</td>
</tr>
<tr>
<td>$C^b_p$</td>
<td>68.843536x10$^{-2}$</td>
</tr>
<tr>
<td>$C^c_p$</td>
<td>-38.50535x10$^{-5}$</td>
</tr>
<tr>
<td>$C^d_p$</td>
<td>80.62568x10$^{-9}$</td>
</tr>
</tbody>
</table>

Table 6.5: Additional parameters for the simplified model used in EKF

<table>
<thead>
<tr>
<th>Column Specifications</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboiler Pressure</td>
<td>101000 $Pa$</td>
</tr>
<tr>
<td>Reflux-Drum Pressure</td>
<td>101600 $Pa$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties of Components</th>
<th>cyclo-hexane</th>
<th>n-heptane</th>
<th>toluene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antoine’s constants for</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T : ^oC, P : torr$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>6.85146</td>
<td>6.89386</td>
<td>6.95087</td>
</tr>
<tr>
<td>$b$</td>
<td>1206.407</td>
<td>1264.370</td>
<td>1342.310</td>
</tr>
<tr>
<td>$c$</td>
<td>223.136</td>
<td>216.640</td>
<td>219.187</td>
</tr>
</tbody>
</table>
CHAPTER 7

RESULTS AND DISCUSSION

The study is performed in four phases. In the first phase, a simulation model for MBDC is achieved and compared with data from literature. In the second phase, a Kalman Filter for the estimation of product compositions for a Multicomponent Batch Distillation from temperature measurements is designed. In the third phase of the study, the designed EKF is implemented on a case Batch Distillation Column to check the performance of the EKF. The forth phase is aimed to use the designed EKF for control purposes in a MBDC. The results and their discussions of the four phases will be given below.

7.1 Rigorous Simulation Model for MBDC

The model development explained in Chapter 3, which is written in MATLAB, is tested for validity by using the case column (Mujtaba et al., 1993) given in Chapter 6. As stated before the column is initialized by the composition of the
feed charged to the column. The system is worked under reflux ratio policy of Mujtaba et al. (1993). The response of the column in terms of the reflux-drum compositions of the components is given in Figure 7.1 (a) for Mujtaba et al. (1993) and Figure 7.1 (b) for the rigorous model developed in this study. In Figure 7.1 (a) the curve numbered as 4 shows the changes in the reflux ratio. It is seen from the responses that the software prepared in this study have the same results with that of the literature under the same operating conditions for the case column. This enables the use of the rigorous model to test the EKF designed. The rigorous model will be referred as “process” in the following sections.

7.2 The Test of the Model Used in EKF

As given in Chapter 5, a simplified MBDC model must be utilized in the design of EKF. The details of the simplification is given in Chapter 5. A simulation run is done to observe the mismatch between the rigorous and simplified models. This run is performed under the same conditions of the run explained in Section 7.1. The response in terms of reflux-drum compositions are given in Figure 7.2. The trend of the compositions are similar, however, the deviations in compositions increase between the period of 3.5 hours and 6 hours. This period is the period of slop-cut distillation. In the optimum reflux ratio policy, the reflux-drum concentration values in terms of products are important to determine the change of reflux ratios during the operation. Thus, any deviation will result in wrong timing for the change of reflux ratio and also in the switching between product-cut and slop-cut tanks. The sources of this mismatch are due to the simplification assumptions used in the development of the model for EKF algorithm. To
Figure 7.1: Comparison of the reflux-drum compositions from the developed model with the literature

realize this argument, one of the major assumptions, specifying the type of VLE relationship is equalized both in the rigorous model representing the process and
the simple model for EKF. Therefore, instead of Peng-Robinson EOS for VLE calculations, the same formulation of Rault’s Law used in the simple model is utilized to the rigorous model to only see the effect of this simplification. The modified rigorous model is simulated under the same conditions in the previous run. The comparison between the responses of the modified rigorous model and the EKF model are shown in Figure 7.3.

![Process/Model Mismatch](image)

Figure 7.2: Mismatch between the process and the simplified model used in EKF.

It is seen that although there are still discrepancies between the modified process model and the EKF model, the mismatch is highly reduced by equalizing the VLE calculations. It is concluded that the major source of the mismatch between the process and the model is due to the selection of VLE formulation.
Figure 7.3: Mismatch between the modified process model and the EKF model.

utilized in the model for EKF algorithm. As a result in the model development, the most important part is the selection of VLE formulation.

7.3 Implementation of EKF

The simulation test runs for tuning the EKF is done without considering any changes in VLE relationship of the EKF model because VLE relationship does not change the effects of tuning parameters on the performance of EKF. Further, it is aimed to obtain the optimum values for these parameters in the worst case (i.e. process/model mismatch). The tuning parameters for EKF as given in Chapter 5 are the diagonal terms of process noise covariance matrix, $q$, and
the diagonal terms of measurement model noise covariance matrix, \( r \). Also, the
effect of number of measurement points, and measurement period, \( \Delta t_m \) will be
illustrated. It is known that, in initialization of the EKF, initial estimates vector,
\( X_0 \) and its error covariance vector, \( P_0 \) are also important. These will be discussed
also.

In all of the simulation test runs, the integral absolute error (IAE) is chosen as
the performance criteria reflecting the fitness of the EKF design parameters. The
formulation of IAE between the actual and the estimated fractions of a component
is given by Equation 7.1

\[
IAE_i = \int_{0}^{T} |X_i(\tau) - x_i(\tau)| d\tau \tag{7.1}
\]

where \( X_i(\tau) \) is the estimated composition of \( i^{th} \) component, \( x_i(\tau) \), the actual
one and \( T \), the total time of batch. In the performance evaluation, instead of
analyzing the IAE scores of each component separately, the sum of the IAE scores
is selected. Moreover, this total score is calculated both for the reflux-drum and
the reboiler composition estimations as given by Equations 7.2 and 7.3

\[
IAE_{RD} = \sum_{i=1}^{NC} IAE_i \tag{7.2}
\]

\[
IAE_{RB} = \sum_{i=1}^{NC} IAE_i \tag{7.3}
\]

where \( IAE_{RD} \) and \( IAE_{RB} \) are the performance scores in the estimation of the
reflux-drum and the reboiler compositions, respectively. As a result, the optimum
value of the considered design parameter is obtained from the simulation run
giving the lowest sum of \( IAE_{RD} \) and \( IAE_{RB} \) values.
7.3.1 Effect of Tuning Parameters of EKF

The simulation runs for this section are done using the same procedure as explained in Sections 7.1 and 7.2.

7.3.1.1 Effect of the Diagonal Terms of Process Noise Covariance Matrix, $q$

The optimum value of the diagonal terms of process noise covariance matrix, $q$ is searched in the range where the EKF estimator is stable. Performing some trial runs, the stability region of the estimator is found where the value of $q$ is in the range of 50 and $1\times10^{-7}$. This region is searched by changing the value of $q$ in 10 folds. The IAE scores for a given $q$ value are listed in Table 7.1 in the order of increasing IAE sum and the plot of IAE sum with respect to $q$ is given in Figure 7.4.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$IAE_{RD}$</th>
<th>$IAE_{RB}$</th>
<th>IAE sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000001</td>
<td>0.6298</td>
<td>0.1706</td>
<td>0.8004</td>
</tr>
<tr>
<td>0.000001</td>
<td>0.6226</td>
<td>0.1829</td>
<td>0.8055</td>
</tr>
<tr>
<td>0.0000001</td>
<td>0.6234</td>
<td>0.1847</td>
<td>0.8081</td>
</tr>
<tr>
<td>0.0001</td>
<td>0.8644</td>
<td>0.1562</td>
<td>1.0206</td>
</tr>
<tr>
<td>0.001</td>
<td>1.2914</td>
<td>0.3698</td>
<td>1.6612</td>
</tr>
<tr>
<td>0.01</td>
<td>1.5160</td>
<td>0.8969</td>
<td>2.4129</td>
</tr>
<tr>
<td>50</td>
<td>1.2679</td>
<td>1.4496</td>
<td>2.7175</td>
</tr>
<tr>
<td>10</td>
<td>1.2777</td>
<td>1.4778</td>
<td>2.7555</td>
</tr>
<tr>
<td>1</td>
<td>1.3179</td>
<td>1.5965</td>
<td>2.9144</td>
</tr>
<tr>
<td>0.1</td>
<td>1.4308</td>
<td>1.5540</td>
<td>2.9848</td>
</tr>
</tbody>
</table>
In Figure 7.5 - 7.8, the change of reflux-drum and reboiler liquid compositions are shown for the actual and estimated values for some selected values of \( q \). In all these runs the diagonal terms of measurement model noise covariance matrix, \( r \) is selected as 5000. In the figures, top plots show the reflux-drum liquid compositions and bottoms show the reboiler liquid compositions. The lowest IAE sum occurs at \( q = 0.00001 \) which is selected as optimal value for the diagonal terms of process noise covariance matrix.

![Figure 7.4: Change of IAE sum with respect to \( q \).](image-url)
Figure 7.5: Effect of the diagonal terms of process noise covariance matrix for $q = 1\times 10^{-7}$. (a) Reflux-drum compositions, (b) Reboiler compositions
Figure 7.6: Effect of the diagonal terms of process noise covariance matrix for \( q = 0.00001 \). (a) Reflux-drum compositions, (b) Reboiler compositions
Figure 7.7: Effect of the diagonal terms of process noise covariance matrix for $q = 0.1$. (a) Reflux-drum compositions, (b) Reboiler compositions
Figure 7.8: Effect of the diagonal terms of process noise covariance matrix for $q = 50$. (a) Reflux-drum compositions, (b) Reboiler compositions
7.3.1.2 Effect of the Diagonal Terms of Measurement Model Noise Covariance Matrix, $r$

In these runs, the diagonal terms of measurement model noise covariance matrix, $r$ are changed between $0.5$ and $5 \times 10^8$ increasing in 10 folds and in each run, the diagonal terms of process noise covariance matrix, $q$ is selected as $0.00001$ which was previously determined as optimal in Section 7.3.1.1. As in the case of $q$, the searching region for $r$ is determined by means of the stability concept. Table 7.2 lists the IAE scores for the reflux-drum and the reboiler composition estimations as well as the IAE sum for each value of $r$ and Figure 7.2 presents the relation of IAE sum vs. $r$, graphically. Figures 7.6 and 7.10 and 7.11 show the effect of the diagonal terms of measurement model noise covariance matrix on the EKF performance. The best result (i.e. one having the lowest IAE sum) is obtained for the diagonal terms of process noise covariance matrix, $q = 5000$ as shown in Figure 7.6.

Table 7.2: Change of IAE scores with respect to $r$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$IAE_{RD}$</th>
<th>$IAE_{RB}$</th>
<th>IAE sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>0.6298</td>
<td>0.1706</td>
<td>0.8004</td>
</tr>
<tr>
<td>50000</td>
<td>0.6852</td>
<td>0.2547</td>
<td>0.9399</td>
</tr>
<tr>
<td>500</td>
<td>0.8935</td>
<td>0.1211</td>
<td>1.0146</td>
</tr>
<tr>
<td>500000</td>
<td>0.7690</td>
<td>0.2838</td>
<td>1.0528</td>
</tr>
<tr>
<td>5000000</td>
<td>0.7987</td>
<td>0.2918</td>
<td>1.0905</td>
</tr>
<tr>
<td>50000000</td>
<td>0.8040</td>
<td>0.2931</td>
<td>1.0971</td>
</tr>
<tr>
<td>500000000</td>
<td>0.8046</td>
<td>0.2932</td>
<td>1.0978</td>
</tr>
<tr>
<td>50</td>
<td>1.1310</td>
<td>0.2751</td>
<td>1.4061</td>
</tr>
<tr>
<td>5</td>
<td>1.2630</td>
<td>0.8440</td>
<td>2.1070</td>
</tr>
<tr>
<td>0.5</td>
<td>1.1384</td>
<td>1.5759</td>
<td>2.7143</td>
</tr>
</tbody>
</table>
Figure 7.9: Change of IAE sum with respect to $r$. 
Figure 7.10: Effect of the diagonal terms of measurement model noise covariance matrix for $r = 0.5$. (a) Reflux-drum compositions, (b) Reboiler compositions
Figure 7.11: Effect of the diagonal terms of measurement model noise covariance matrix for $r = 5 \times 10^8$. (a) Reflux-drum compositions, (b) Reboiler compositions.
7.3.2 Effect of Number of Measurement Points

The previous runs were done, utilizing three measurement points for temperatures due to the recommendation (Quintero-Marmol et al., 1991). To see the effects of measurement points in EKF performance several extra runs with one to five measurement points were also conducted in which optimal values for the diagonal terms of process noise covariance matrix, $q$ and the diagonal terms of measurement model noise covariance matrix, $r$ are used. Firstly, to decide on the number of measurement points, homogenously spreading the locations of the measurements throughout the column results in the IAE sums given in Table 7.3. The run having the lowest IAE sum is obtained as the one with three measurement points. In addition, the run with two measurements, which is the minimum number of measurements satisfying the observability criteria, has an IAE score, larger than that of the runs with more measurements and it has an IAE less than that of one-measurement run which is the only run violating the observability requirement. Moreover, as stated previously in Chapter 5.1, using NC measurement points was also recommended by Quintero-Marmol et al. (1991).

To see the effect of measurement locations, more three-measurement runs are simulated using different location alternatives. While the locations of two measurements are changed, the reboiler temperature measurement is kept in all of the runs. The simulation results give the IAE scores for different measurement structures and they are shown in Table 7.4. All the measurement alternatives have larger IAE scores than that of the measurement set including reboiler, $4^{th}$ and top trays and listed in Table 7.3.
Table 7.3: Change of IAE scores with respect to number of measurements.

<table>
<thead>
<tr>
<th>Measurement locations&lt;sup&gt;1&lt;/sup&gt;</th>
<th>$IAE_{RD}$</th>
<th>$IAE_{RB}$</th>
<th>$IAE$ sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rb, 4, NT$</td>
<td>0.6298</td>
<td>0.1706</td>
<td>0.8004</td>
</tr>
<tr>
<td>$rb, 3, 5, NT$</td>
<td>0.6692</td>
<td>0.1447</td>
<td>0.8139</td>
</tr>
<tr>
<td>$rb, 3, 5, 7, NT$</td>
<td>0.6756</td>
<td>0.1425</td>
<td>0.8181</td>
</tr>
<tr>
<td>$rb, NT$</td>
<td>0.6064</td>
<td>0.2219</td>
<td>0.8283</td>
</tr>
<tr>
<td>$NT$</td>
<td>0.6853</td>
<td>0.2744</td>
<td>0.9597</td>
</tr>
</tbody>
</table>

<sup>1</sup>: $rb$ is for reboiler

Table 7.4: Change of IAE scores with respect to location of three measurements.

<table>
<thead>
<tr>
<th>Measurement locations&lt;sup&gt;1&lt;/sup&gt;</th>
<th>$IAE_{RD}$</th>
<th>$IAE_{RB}$</th>
<th>$IAE$ sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rb, 4, 5$</td>
<td>0.6598</td>
<td>0.1575</td>
<td>0.8173</td>
</tr>
<tr>
<td>$rb, 3, 4$</td>
<td>0.6826</td>
<td>0.1413</td>
<td>0.8239</td>
</tr>
<tr>
<td>$rb, 2, 3$</td>
<td>0.7091</td>
<td>0.1267</td>
<td>0.8358</td>
</tr>
<tr>
<td>$rb, 1, 2$</td>
<td>0.7803</td>
<td>0.1186</td>
<td>0.8989</td>
</tr>
</tbody>
</table>

<sup>1</sup>: $rb$ is for reboiler

### 7.3.3 Effect of Measurement Period, $\Delta t_m$.

To see the effect of measurement period, $\Delta t_m$, the previously used value of one integration step time is changed as 5 and 20 times the integration step time of $\Delta t = 3 \times 10^{-4}$ hours and also the values of 1, 3, 5, 10 and 20 min. are tried. The IAE scores for these runs are tabulated in Table 7.5 and the responses in terms of reflux-drum and reboiler compositions for the selected values of 1 min. and 20 min. are shown in Figures 7.12 and 7.13. As expected, the IAE sum of the estimations increases as the measurement period, $\Delta t_m$, increases. Although, the
measurement period selected in this simulation study is the minimum possible value of one times the integration step time, in real-time estimation problems, the value of $\Delta t_m$ is to be chosen considering the limits of the computational power. Because decreasing the measurement period increases the accuracy of estimation at the expense of the computational time required to process single estimation step. The measurement periods of $3\text{min.}$ even $5\text{min.}$ can satisfactorily be used without much change in IAE scores.

Table 7.5: Change of IAE scores with respect to $\Delta t_m$.

<table>
<thead>
<tr>
<th>$\Delta t_m$</th>
<th>$IAE_{RD}$</th>
<th>$IAE_{RB}$</th>
<th>IAE sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 * $\Delta t$</td>
<td>0.6298</td>
<td>0.1706</td>
<td>0.8004</td>
</tr>
<tr>
<td>5 * $\Delta t$</td>
<td>0.6545</td>
<td>0.2364</td>
<td>0.8909</td>
</tr>
<tr>
<td>20 * $\Delta t$</td>
<td>0.7115</td>
<td>0.2639</td>
<td>0.9754</td>
</tr>
<tr>
<td>1$\text{min.}$</td>
<td>0.7443</td>
<td>0.2675</td>
<td>1.0118</td>
</tr>
<tr>
<td>3$\text{min.}$</td>
<td>0.7722</td>
<td>0.2524</td>
<td>1.0246</td>
</tr>
<tr>
<td>5$\text{min.}$</td>
<td>0.7835</td>
<td>0.2422</td>
<td>1.0257</td>
</tr>
<tr>
<td>10$\text{min.}$</td>
<td>0.8263</td>
<td>0.2114</td>
<td>1.0377</td>
</tr>
<tr>
<td>20$\text{min.}$</td>
<td>1.0213</td>
<td>0.1812</td>
<td>1.2025</td>
</tr>
</tbody>
</table>
Figure 7.12: Effect of the period of temperature measurements for $\Delta t_m = 1\text{min}$. 
(a) Reflux-drum compositions, (b) Reboiler compositions
Figure 7.13: Effect of the period of temperature measurements for $\Delta t_m = 20\text{min}$.  
(a) Reflux-drum compositions, (b) Reboiler compositions
7.3.4 Effect of Initial Conditions

This simulation run is done to see the effect of initial state estimate, $X_0$ and its error covariance vector, $P_0$ on the system response. The previous simulations were done using feed composition as the initial estimate throughout the column. In Figure 7.14, the response for initial state estimate, $X_0 = [1/3, 1/3, 1/3]$ and the diagonal terms of its error covariance vector, $P_0 = 0.1$ is shown. This is a fictitious composition for feed when the feed composition is not known. Of course, in this run the deviations in estimation are higher than the previous cases, giving $IAE$ sum of 1.8797. However, they can still be considered agreeable in a case where feed composition is not known. Moreover, the estimations can also be improved with trial-and-error using different tuning parameters for the case of unknown feed composition.
Figure 7.14: Effect of the initial conditions for $X_0 = [1/3, 1/3, 1/3]$ and $P_0 = 0.00001$. (a) Reflux-drum compositions, (b) Reboiler compositions
7.4 Closed-loop performance of EKF

In this phase of the study, it is aimed to analyze the performance of the EKF estimator for a MBDC system in a composition-feedback inferential control structure which realizes an actual scheduling policy explained previously in Section 3.1. In this control law, the compositions in the reflux-drum, the product-cut tanks and the reboiler are the inputs to the controller and the manipulated variable is the reflux-ratio of the column. The pre-specified reflux-ratio values required for the control algorithm is chosen as the optimized ones used in the previous sections. The tank, to which the distillate stream is diverted, and its timing are decided by monitoring the input compositions to the controller and utilizing the actual scheduling policy. In the simulation of this control structure, the monitored compositions can be obtained directly from the process or from the EKF estimator.

Firstly, to create a reference point, a simulation is done, taking the composition knowledge directly from the column as the feedback to the controller. The desired product purities are the set points of the controller which are taken as 0.9, 0.81, 0.69. The response of this reference run in terms of the liquid compositions, both in the reflux-drum and the reboiler are given in Figure 7.15.
Figure 7.15: The closed-loop responses of the MBDC under the scheduling controller with actual composition feedback. (a) Reflux-drum compositions, (b) Reboiler compositions
The Capacity Factor (CAP) (Luyben, 1988) and Batch Time (BT) are selected as the performance criteria to compare the closed-loop responses with the reference run. CAP for a batch operation is defined by Luyben (1988) as the ratio between the total amount of specified products and the total batch time plus feeding duration. If the $P_1$, $P_2$ and $P_3$ are the amounts of products obtained and the feeding duration is taken as 0.5 hour, the Capacity Factor (CAP) is given by

$$\text{CAP} = \frac{P_1 + P_2 + P_3}{BT + 0.5} \quad (7.4)$$

For the previous run, CAP and BT are obtained as 243 mol/hour and 8.22 hour, respectively. The purities of the obtained products are 0.899995, 0.809997 and 0.689988, respectively.

To see the performance of EKF, secondly the same simulation is realized using the estimated compositions as the feedback to the controller. For the design parameters of the EKF estimator, the optimum values obtained in the implementation section (see Section 7.3) are used. The responses of the reflux-drum and the reboiler compositions are shown in Figure 7.16. CAP and BT are found as 308 mol/hour and 7.59 hour, respectively. The errors in estimation of CAP and BT are 21% and 8%, respectively. Although, the error in CAP is high, the actual purities are 0.872628, 0.794142 and 0.686368, having the errors of 3%, 2% and 0.5%, respectively. Thus, a batch distillation column can be controlled for variable reflux ratio policy by the use of EKF estimator utilizing a simplified model satisfactorily.
Figure 7.16: The closed-loop responses of the MBDC under the scheduling controller with estimated composition feedback. (a) Reflux-drum compositions, (b) Reboiler compositions
CHAPTER 8

CONCLUSIONS

The study is aimed to estimate the compositions in the multicomponent batch distillation column from temperature measurements using EKF estimator. As a result, the following conclusions are derived.

1) A rigorous simulation program for MBDC is written satisfactorily with the check for validity with data from the literature.

2) The developed rigorous simulation is capable of representing the behavior of MBDC during the total-reflux operation as well as the operation of distillate withdrawal.

3) In the design of EKF, a simplified and linearized models using a VLE relationship is utilized with success having some deviations. It is found that, the most important part of the modelling effecting the performance of the estimator is the selection of the VLE formulation.

4) EKF parameters of the diagonal terms of process noise covariance matrix
and the diagonal terms of measurement model noise covariance matrix are tuned in the range where the estimator is stable giving the least total IAE scores for the reflux-drum and the reboiler composition estimates.

5) It is concluded that, increasing the number of temperature measurements above the recommended value of NC does not result in better performance. The number of measurements larger than NC decreases the performance of EKF estimator. Although the observability criteria makes NC-1 measurements sufficient, using NC measurements improves the performance of EKF estimator. The measurement locations have significant effects on the performance and it should be spread through out the column homogeneously.

6) It is realized that, decreasing in the measurement period value increases the estimator performance, however, the degree of improvement by decreasing the period is limited by the computational power of the digital computer especially in real-time applications.

7) The designed EKF estimator is successfully used in the composition – feedback inferential control of MBDC operated under variable reflux-ratio policy with an acceptable deviation of $0.5 - 3\%$ from the desired purity level of the products.
REFERENCES


APPENDIX A

VALIDITY OF NEGLIGIBLE VAPOR HOLDUP ASSUMPTION

Consider a column operating at atmospheric pressure. Assuming a liquid density of 800 kg/m$^3$, average temperature of 150°C, a molecular weight of 60 kg/mol, 24 in. tray spacing, 2 in. of liquid holdup and 3 m diameter column, the vapor and liquid holdups are

\[
\text{cross-sectional area} = \pi D^2/4 = 7.07 m^2
\]

\[
\text{liquid - holdup} = [2 \cdot 0.0254 m][7.07 m^2][800 kg/m^3]/[60 kg/mol]
\]

\[
= 4.79 kmol
\]

\[
\text{vapor - density} = P/RT
\]

\[
= [1 atm]/[0.0821 l/t \cdot atm/kmol \cdot K][150 + 273.15 K]
\]

\[
= 0.0288 kmol/m^3
\]

\[
\text{vapor - holdup} = [0.0288 kmol/m^3][7.07 m^2][(24 - 2) \cdot 0.0254 m]
\]

\[
= 0.1138 kmol
\]
So the vapor holdup is only 2.3% of the total holdup on a tray and can be neglected. If the column were operated at 10 atm, the vapor holdup becomes 1.138 kmol and it is 19.2% of the total holdup. As a result assumption of negligible vapor holdup is always reasonable.
APPENDIX B

LINEAR STATE SPACE MODELS

B.1 Continuous-time system—discrete-time measurement model

First type of model can be expressed by Equation B.1 and Equation B.2

\[
\dot{x}(t) = F(t)x(t) + B(t)u(t) + G(t)w(t) \tag{B.1}
\]

\[
z(t_k) = H(t_k)x(t_k) + v(t_k) \tag{B.2}
\]

where the variables are defined as

\(x(t)\), \(n \times 1\) state vector

\(z(t_k)\), \(m \times 1\) discrete-time measurement vector

\(u(t)\), \(r \times 1\) deterministic input vector

\(F(t)\), \(n \times n\) system coefficient matrix

\(B(t)\), \(n \times r\) input coupling matrix

\(H(t_k)\), \(m \times n\) discrete-time measurement matrix

\(G(t)\), \(n \times s\) process noise coupling matrix
\( w(t), s \times 1 \) system noise process

\( v(t_k), m \times 1 \) measurement noise sequence.

While the system model, Equation B.1 represents the dynamic behavior of the system states, the measurement model, Equation B.2 gives the relationship between the system states and the measurable outputs. The block diagram representation of the model equations are given in Figure B.1. The first two terms on r.h.s. of Equation B.1 and the first term on r.h.s. of Equation B.2 are the deterministic parts of the models and the last terms are the stochastic parts.

The use of this model structure in the current study is due to the ability to model the system in continuous-time and the measurements in discrete-time. Because, the column model developed in the previous chapter (3) is in the form of continuous-time and the temperature measurements are available at discrete times on the digital computer where the filter is implemented.


B.2 Discrete-time system—discrete-time measurement model

Second type of the model can be written in the form of Equation B.3 and Equation B.4 where the system dynamics are modelled in discrete-time and it is the only difference from the type of the previous model. The graphical presentation of the model is shown in Figure B.2.

\[
\begin{align*}
\bar{x}(t_{k+1}) &= \Phi(t_k)\bar{x}(t_k) + B_d(t_k)u(t_k) + G_d(t_k)w(t_k) \\
z(t_k) &= H(t_k)x(t_k) + v(t_k)
\end{align*}
\]  

(B.3)  

(B.4)

where the variables not defined previously are given as

\( \Phi(t_k), n \times 1 \) discrete-time system coefficient matrix

\( B_d(t_k), n \times r \) discrete-time input coupling matrix

\( G_d(t_k), n \times s \) discrete-time process noise coupling matrix

\( w(t_k), s \times 1 \) system noise sequence.

Figure B.2: Discrete-time system—discrete-time measurement model.
B.3 Continuous-time system—continuous-time measurement model

A third possible model formulation schematically presented in Figure B.3 consists of a continuous-time system model as given previously

\[ \dot{x}(t) = F(t)x(t) + B(t)u(t) + G(t)w(t) \]  

(B.5)

with continuously available measurements modelled as

\[ z(t) = H_c(t)x(t) + v(t) \]  

(B.6)

where \( z(t) \), \( m \times 1 \) continuous-time measurement vector, \( H_c(t) \), \( m \times n \) continuous-time measurement matrix and \( v(t) \), \( m \times 1 \) measurement noise process.

Figure B.3: Continuous-time system—continuous-time measurement model.
APPENDIX C

EVALUATION OF THE LINEARIZED MODEL MATRICES F AND H

In this appendix, as previously summarized in Section 5.2.2, the details of evaluating the Linearized System Matrix, $F$ and the Linearized Measurement Matrix, $H$ are given.

C.1 Evaluation of The Linearized System Matrix F

This section is devoted to the evaluation of general form of Linearized System Matrix, $F$ previously given by Equation 5.15. Each element of $F'$ is found by setting the indices of $F'_{ij}$ (i.e. $k$, $n$, $p$ and $r$) in Equation C.1

\[
F'_{ij} = \frac{df_i}{dX_j} = \frac{df_{(k-1)\cdot NC+n}}{dX_{(p-1)\cdot NC+r}} = \frac{d(\hat{x}_{k,n})}{dx_{p,r}} \quad \text{(C.1)}
\]
for \( k = 1 \ldots (NT + 2) \)

\[ n = 1 \ldots NC \]

\[ p = 1 \ldots (NT + 2) \]

\[ r = 1 \ldots NC \]

as follows.

(i) for \( k = 1 \) and \( n = 1 \) \( \Rightarrow i = 1 \):

\[ f_1 = \dot{X}_1 = \dot{x}_{11} = \left[ L(x_{21} - x_{11}) - V(y_{11} - x_{11}) \right] / M_1 \tag{C.2} \]

for \( p = 1 \):

\[ r = 1 \Rightarrow j = 1 \]

\[ F'_{11} = \frac{d f_1}{d x_{11}} = \left[ -L - V \left( \frac{dy_{11}}{dx_{11}} - 1 \right) \right] / M_1 \]

\[ r = 2 \Rightarrow j = 2 \]

\[ F'_{12} = \frac{d f_1}{d x_{12}} = \left[ -V \frac{dy_{11}}{dx_{12}} \right] / M_1 \]

\[ r = 3 \Rightarrow j = 3 \]

\[ F'_{13} = \frac{d f_1}{d x_{13}} = \left[ -V \frac{dy_{11}}{dx_{13}} \right] / M_1 \]

for \( p = 2 \):

\[ r = 1 \Rightarrow j = 4 \]

\[ F'_{14} = \frac{d f_1}{d x_{21}} = \left[ L \right] / M_1 \]

\[ r = 2 \Rightarrow j = 5 \]

\[ F'_{15} = \frac{d f_1}{d x_{22}} = 0 \]

\[ r = 3 \Rightarrow j = 6 \]

\[ F'_{16} = \frac{d f_1}{d x_{23}} = 0 \]
for $p = 3$:

\[
\begin{align*}
\text{r} = 1 & \Rightarrow j = 7 \\
F'_{17} = \frac{df_1}{dx_{31}} = 0 \\
\text{r} = 2 & \Rightarrow j = 8 \\
F'_{18} = \frac{df_1}{dx_{32}} = 0 \\
\text{r} = 3 & \Rightarrow j = 9 \\
F'_{19} = \frac{df_1}{dx_{33}} = 0
\end{align*}
\]

for $p = 4 \ldots (NT + 2)$:

\[
F_{1j} = \frac{df_1}{d\dot{x}_{pr}} = 0
\]

\[
\begin{align*}
\text{r} = 1 \ldots NC \\
j = (p - 1)NC + r
\end{align*}
\]

(ii) for $k = 1$ and $n = 2 \Rightarrow i = 2$:

\[
f_2 = \dot{X}_2 = \dot{x}_{12} = \left[ L(x_{22} - x_{12}) - V(y_{12} - x_{12}) \right] / M_1 \tag{C.3}
\]

for $p = 1$:

\[
\begin{align*}
\text{r} = 1 & \Rightarrow j = 1 \\
F'_{21} = \frac{df_2}{dx_{11}} = \left[ -V \frac{dy_{12}}{dx_{11}} \right] / M_1 \\
\text{r} = 2 & \Rightarrow j = 2 \\
F'_{22} = \frac{df_2}{dx_{12}} = \left[ -L - V \left( \frac{dy_{12}}{dx_{12}} - 1 \right) \right] / M_1 \\
\text{r} = 3 & \Rightarrow j = 3 \\
F'_{23} = \frac{df_2}{dx_{13}} = \left[ -V \frac{dy_{12}}{dx_{13}} \right] / M_1
\end{align*}
\]

123
for $p = 2$:

\[ r = 1 \Rightarrow j = 4 \]

\[ F_{24}' = \frac{df_2}{dx_{21}} = 0 \]

\[ r = 2 \Rightarrow j = 5 \]

\[ F_{25}' = \frac{df_2}{dx_{22}} = [L]/M_1 \]

\[ r = 3 \Rightarrow j = 6 \]

\[ F_{26}' = \frac{df_2}{dx_{23}} = 0 \]

for $p = 3$:

\[ r = 1 \Rightarrow j = 7 \]

\[ F_{27}' = \frac{df_2}{dx_{31}} = 0 \]

\[ r = 2 \Rightarrow j = 8 \]

\[ F_{28}' = \frac{df_2}{dx_{32}} = 0 \]

\[ r = 3 \Rightarrow j = 9 \]

\[ F_{29}' = \frac{df_2}{dx_{33}} = 0 \]

for $p = 4 \ldots (NT + 2)$:

\[ F_{2j}' = \frac{df_2}{dx_{pj}} = 0 \]

\[ r = 1 \ldots NC \]

\[ j = (p - 1)NC + r \]

(iii) for $k = 1$ and $n = 3 \Rightarrow i = 3$:

\[ f_3 = \dot{X}_3 = \dot{x}_{13} = [L(x_{23} - x_{13}) - V(y_{13} - x_{13})]/M_1 \quad \text{(C.4)} \]
for $p = 1$: 

\[ r = 1 \Rightarrow j = 1 \]

\[ F'_{31} = \frac{df_3}{dx_{11}} = \left[-V \frac{dy_{13}}{dx_{11}}\right] / M_1 \]

\[ r = 2 \Rightarrow j = 2 \]

\[ F'_{32} = \frac{df_3}{dx_{12}} = \left[-V \frac{dy_{13}}{dx_{12}}\right] / M_1 \]

\[ r = 3 \Rightarrow j = 3 \]

\[ F'_{33} = \frac{df_3}{dx_{13}} = \left[-L - V \frac{dy_{13}}{dx_{13}} - 1\right] / M_1 \]

for $p = 2$: 

\[ r = 1 \Rightarrow j = 4 \]

\[ F'_{34} = \frac{df_3}{dx_{21}} = 0 \]

\[ r = 2 \Rightarrow j = 5 \]

\[ F'_{35} = \frac{df_3}{dx_{22}} = 0 \]

\[ r = 3 \Rightarrow j = 6 \]

\[ F'_{36} = \frac{df_3}{dx_{23}} = [L] / M_1 \]

for $p = 3$: 

\[ r = 1 \Rightarrow j = 7 \]

\[ F'_{37} = \frac{df_3}{dx_{31}} = 0 \]

\[ r = 2 \Rightarrow j = 8 \]

\[ F'_{38} = \frac{df_3}{dx_{32}} = 0 \]

\[ r = 3 \Rightarrow j = 9 \]

\[ F'_{39} = \frac{df_3}{dx_{33}} = 0 \]

for $p = 4 \ldots (NT + 2)$: 

\[ F_{3j} = \frac{df_3}{d\dot{x}_{pr}} = 0 \]
\[ r = 1 \ldots NC \]
\[ j = (p - 1)NC + r \]

(iii) for \( k = 2 \ldots (NT + 1) \) and \( n = 1 \Rightarrow i = (k - 1)NC + 1 \):

\[ f_i = \dot{X}_i = \dot{x}_{k1} = [V(y_{k-1,1} - y_{k1}) + L(x_{k+1,1} - x_{k1})] / M_k \quad (C.5) \]

for \( p = k - 1 \):

\[ r = 1 \Rightarrow j = (k - 2)NC + 1 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k-1,1}} = \left[ V \frac{dy_{k-1,1}}{dx_{k-1,1}} \right] / M_k \]
\[ r = 2 \Rightarrow j = (k - 2)NC + 2 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k-1,2}} = \left[ V \frac{dy_{k-1,1}}{dx_{k-1,2}} \right] / M_k \]
\[ r = 3 \Rightarrow j = (k - 2)NC + 3 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k-1,3}} = \left[ V \frac{dy_{k-1,1}}{dx_{k-1,3}} \right] / M_k \]

for \( p = k \):

\[ r = 1 \Rightarrow j = (k - 1)NC + 1 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k,1}} = \left[ -V \frac{dy_{k1}}{dx_{k1}} - L \right] / M_k \]
\[ r = 2 \Rightarrow j = (k - 1)NC + 2 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k,2}} = \left[ -V \frac{dy_{k1}}{dx_{k2}} \right] / M_k \]
\[ r = 3 \Rightarrow j = (k - 1)NC + 3 \]
\[ F^r_{ij} = \frac{df_i}{dx_{k,3}} = \left[ -V \frac{dy_{k1}}{dx_{k3}} \right] / M_k \]
for $p = k + 1$:

\[ r = 1 \Rightarrow j = (k)NC + 1 \]
\[ F'_{ij} = \frac{df_i}{dx_{k+1,1}} = [L]/M_k \]
\[ r = 2 \Rightarrow j = (k)NC + 2 \]
\[ F'_{ij} = \frac{df_i}{dx_{k+1,2}} = 0 \]
\[ r = 3 \Rightarrow j = (k)NC + 3 \]
\[ F'_{ij} = \frac{df_i}{dx_{k+1,3}} = 0 \]

for $k = 2$ and $n = 1 \Rightarrow i = (k-1)NC + n$ and

for $p = k + 2 \ldots (NT + 2)$ and $r = 1 \ldots NC \Rightarrow j = (p-1)NC + r$:

\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

for $k = 3 \ldots NT$ and $n = 1 \Rightarrow i = (k-1)NC + n$ and

for $p = 1 \ldots (k-2)\text{and}(k+2) \ldots (NT + 2)$ and $r = 1 \ldots NC \Rightarrow j = (p-1)NC + r$:

\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

for $k = NT + 1$ and $n = 1 \Rightarrow i = (k-1)NC + n$ and

for $p = 1 \ldots (k-2)$ and $r = 1 \ldots NC \Rightarrow j = (p-1)NC + r$:

\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

(v) for $k = 2 \ldots (NT + 1)$ and $n = 2 \Rightarrow i = (k-1)NC + 2$:

\[ f_i = \ddot{X}_i = \ddot{x}_{k2} = [V(y_{k-1,2} - y_{k2}) + L(x_{k+1,2} - x_{k2})]/M_k \]
for $p = k - 1$:

\[ r = 1 \Rightarrow j = (k - 2)NC + 1 \]

\[ F_{ij}' = \frac{df_i}{dx_{k-1,1}} = \left[V \frac{dy_{k-1,1}}{dx_{k-1,1}}\right]/M_k \]

\[ r = 2 \Rightarrow j = (k - 2)NC + 2 \]

\[ F_{ij}' = \frac{df_i}{dx_{k-1,2}} = \left[V \frac{dy_{k-1,2}}{dx_{k-1,2}}\right]/M_k \]

\[ r = 3 \Rightarrow j = (k - 2)NC + 3 \]

\[ F_{ij}' = \frac{df_i}{dx_{k-1,3}} = \left[V \frac{dy_{k-1,3}}{dx_{k-1,3}}\right]/M_k \]

for $p = k$:

\[ r = 1 \Rightarrow j = (k - 1)NC + 1 \]

\[ F_{ij}' = \frac{df_i}{dx_{k,1}} = \left[-V \frac{dy_{k,2}}{dx_{k+1,1}}\right]/M_k \]

\[ r = 2 \Rightarrow j = (k - 1)NC + 2 \]

\[ F_{ij}' = \frac{df_i}{dx_{k,2}} = \left[-V \frac{dy_{k,2}}{dx_{k+1,2}} - L\right]/M_k \]

\[ r = 3 \Rightarrow j = (k - 1)NC + 3 \]

\[ F_{ij}' = \frac{df_i}{dx_{k,3}} = \left[-V \frac{dy_{k,3}}{dx_{k+1,3}}\right]/M_k \]

for $p = k + 1$:

\[ r = 1 \Rightarrow j = (k)NC + 1 \]

\[ F_{ij}' = \frac{df_i}{dx_{k+1,1}} = 0 \]

\[ r = 2 \Rightarrow j = (k)NC + 2 \]

\[ F_{ij}' = \frac{df_i}{dx_{k+1,2}} = \left[L\right]/M_k \]

\[ r = 3 \Rightarrow j = (k)NC + 3 \]

\[ F_{ij}' = \frac{df_i}{dx_{k+1,3}} = 0 \]

for $k = 2$ and $n = 2 \Rightarrow i = (k - 1)NC + n$ and

for $p = k + 2 \ldots (NT + 2)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$: 128
\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

for \( k = 3 \ldots NT \) and \( n = 2 \Rightarrow i = (k - 1)NC + n \) and

for \( p = 1 \ldots (k - 2) \) and \( (k + 2) \ldots (NT + 2) \) and \( r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r: \)

\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

for \( k = NT + 1 \) and \( n = 2 \Rightarrow i = (k - 1)NC + n \) and

for \( p = 1 \ldots (k - 2) \) and \( r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r: \)

\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

(vi) for \( k = 2 \ldots (NT + 1) \) and \( n = 3 \Rightarrow i = (k - 1)NC + 3 \):

\[ f_i = \dot{X}_i = \dot{x}_{k3} = \left[ V(y_{k-1,3} - y_{k3}) + L(x_{k+1,3} - x_{k3}) \right] / M_k \quad \text{(C.7)} \]

for \( p = k - 1: \)

\[ r = 1 \Rightarrow j = (k - 2)NC + 1 \]
\[ F'_{ij} = \frac{df_i}{dx_{k-1,1}} = \left[ V \frac{dy_{k-1,3}}{dx_{k-1,1}} \right] / M_k \]
\[ r = 2 \Rightarrow j = (k - 2)NC + 2 \]
\[ F'_{ij} = \frac{df_i}{dx_{k-1,2}} = \left[ V \frac{dy_{k-1,3}}{dx_{k-1,2}} \right] / M_k \]
\[ r = 3 \Rightarrow j = (k - 2)NC + 3 \]
\[ F'_{ij} = \frac{df_i}{dx_{k-1,3}} = \left[ V \frac{dy_{k-1,3}}{dx_{k-1,3}} \right] / M_k \]
for $p = k$:

\[ r = 1 \Rightarrow j = (k - 1)NC + 1 \]

\[
F'_{ij} = \frac{df_i}{dx_{k,r}} = \left[ -V \frac{du_{k3}}{dx_{k1}} \right] / M_k
\]

\[ r = 2 \Rightarrow j = (k - 1)NC + 2 \]

\[
F'_{ij} = \frac{df_i}{dx_{k,2}} = \left[ -V \frac{du_{k3}}{dx_{k2}} \right] / M_k
\]

\[ r = 3 \Rightarrow j = (k - 1)NC + 3 \]

\[
F'_{ij} = \frac{df_i}{dx_{k,3}} = \left[ -V \frac{du_{k3}}{dx_{k3}} - L \right] / M_k
\]

for $p = k + 1$:

\[ r = 1 \Rightarrow j = (k)NC + 1 \]

\[
F'_{ij} = \frac{df_i}{dx_{k+1,1}} = 0
\]

\[ r = 2 \Rightarrow j = (k)NC + 2 \]

\[
F'_{ij} = \frac{df_i}{dx_{k+1,2}} = 0
\]

\[ r = 3 \Rightarrow j = (k)NC + 3 \]

\[
F'_{ij} = \frac{df_i}{dx_{k+1,3}} = [L] / M_k
\]

for $k = 2$ and $n = 3 \Rightarrow i = (k - 1)NC + n$ and

for $p = k + 2 \ldots (NT + 2)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

\[
F'_{ij} = \frac{df_i}{dx_{p,r}} = 0
\]

for $k = 3 \ldots NT$ and $n = 3 \Rightarrow i = (k - 1)NC + n$ and

for $p = 1 \ldots (k - 2)$ and $(k + 2) \ldots (NT + 2)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

\[
F'_{ij} = \frac{df_i}{dx_{p,r}} = 0
\]

for $k = NT + 1$ and $n = 3 \Rightarrow i = (k - 1)NC + n$ and

for $p = 1 \ldots (k - 2)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

\[
F'_{ij} = \frac{df_i}{dx_{p,r}} = 0
\]
\[ F'_{ij} = \frac{df_i}{dx_{pr}} = 0 \]

(vii) for \( k = NT + 2 \) and \( n = 1 \Rightarrow i = (NT + 1)NC + 1 : \]

\[ f_i = \dot{X}_i = \dot{x}_{NT+2,1} = \left[ V(y_{NT+1,1} - x_{NT+2,1}) \right] / M_{NT+2} \quad (C.8) \]

for \( p = NT : \]

\[ r = 1 \Rightarrow j = (NT - 1)NC + 1 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT,1}} = 0 \]
\[ r = 2 \Rightarrow j = (NT - 1)NC + 2 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT,2}} = 0 \]
\[ r = 3 \Rightarrow j = (NT - 1)NC + 3 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT,3}} = 0 \]

for \( p = NT + 1 : \]

\[ r = 1 \Rightarrow j = (NT)NC + 1 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT+1,1}} = \left[ V \frac{dy_{NT+1,1}}{dx_{NT+1,1}} \right] / M_{NT+2} \]
\[ r = 2 \Rightarrow j = (NT)NC + 2 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT+1,2}} = \left[ V \frac{dy_{NT+1,1}}{dx_{NT+1,2}} \right] / M_{NT+2} \]
\[ r = 3 \Rightarrow j = (NT)NC + 3 \]
\[ F'_{ij} = \frac{df_i}{dx_{NT+1,3}} = \left[ V \frac{dy_{NT+1,1}}{dx_{NT+1,3}} \right] / M_{NT+2} \]
for $p = NT + 2$:

\[
\begin{align*}
  r &= 1 \Rightarrow j = (NT + 1)NC + 1 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,1}} = \frac{[-V]}{M_{NT+2}} \\
  r &= 2 \Rightarrow j = (NT + 1)NC + 2 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,2}} = 0 \\
  r &= 3 \Rightarrow j = (NT + 1)NC + 3 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,3}} = 0
\end{align*}
\]

for $p = 1 \ldots (NT - 1)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

\[
F'_{ij} = \frac{df_i}{dx_{pr}} = 0
\]

(viii) for $k = NT + 2$ and $n = 2 \Rightarrow i = (NT + 1)NC + 2$:

\[
f_i = \dot{X}_i = \dot{x}_{NT+2,2} = \frac{[V(y_{NT+1,2} - x_{NT+2,2})]}{M_{NT+2}} \tag{C.9}
\]

for $p = NT$:

\[
\begin{align*}
  r &= 1 \Rightarrow j = (NT - 1)NC + 1 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,1}} = 0 \\
  r &= 2 \Rightarrow j = (NT - 1)NC + 2 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,2}} = 0 \\
  r &= 3 \Rightarrow j = (NT - 1)NC + 3 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,3}} = 0
\end{align*}
\]
for $p = NT + 1$:

$r = 1 \Rightarrow j = (NT)NC + 1$

$$F'_{ij} = \frac{df_i}{dx_{NT+1,1}} = \left[V \frac{dy_{NT+1,2}}{dx_{NT+1,1}}\right] / M_{NT+2}$$

$r = 2 \Rightarrow j = (NT)NC + 2$

$$F'_{ij} = \frac{df_i}{dx_{NT+1,2}} = \left[V \frac{dy_{NT+1,2}}{dx_{NT+1,2}}\right] / M_{NT+2}$$

$r = 3 \Rightarrow j = (NT)NC + 3$

$$F'_{ij} = \frac{df_i}{dx_{NT+1,3}} = \left[V \frac{dy_{NT+1,2}}{dx_{NT+1,3}}\right] / M_{NT+2}$$

for $p = NT + 2$:

$r = 1 \Rightarrow j = (NT + 1)NC + 1$

$$F'_{ij} = \frac{df_i}{dx_{NT+2,1}} = 0$$

$r = 2 \Rightarrow j = (NT + 1)NC + 2$

$$F'_{ij} = \frac{df_i}{dx_{NT+2,2}} = [-V] / M_{NT+2}$$

$r = 3 \Rightarrow j = (NT + 1)NC + 3$

$$F'_{ij} = \frac{df_i}{dx_{NT+2,3}} = 0$$

for $p = 1 \ldots (NT - 1)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

$$F'_{ij} = \frac{df_i}{dx_{p,r}} = 0$$

(ix) for $k = NT + 2$ and $n = 3 \Rightarrow i = (NT + 1)NC + 3$:

$$f_i = \dot{X}_i = \dot{x}_{NT+2,3} = \left[V(y_{NT+1,3} - x_{NT+2,3})\right] / M_{NT+2}$$  \hspace{1cm} (C.10)
for $p = NT$:

\[
\begin{align*}
  r = 1 & \Rightarrow j = (NT - 1)NC + 1 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,1}} = 0 \\
  r = 2 & \Rightarrow j = (NT - 1)NC + 2 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,2}} = 0 \\
  r = 3 & \Rightarrow j = (NT - 1)NC + 3 \\
  F'_{ij} &= \frac{df_i}{dx_{NT,3}} = 0
\end{align*}
\]

for $p = NT + 1$:

\[
\begin{align*}
  r = 1 & \Rightarrow j = (NT)NC + 1 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+1,1}} = \left[ V \frac{dy_{NT+1,2}}{dx_{NT+1,1}} \right] / M_{NT+2} \\
  r = 2 & \Rightarrow j = (NT)NC + 2 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+1,2}} = \left[ V \frac{dy_{NT+1,3}}{dx_{NT+1,2}} \right] / M_{NT+2} \\
  r = 3 & \Rightarrow j = (NT)NC + 3 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+1,3}} = \left[ V \frac{dy_{NT+1,3}}{dx_{NT+1,3}} \right] / M_{NT+2}
\end{align*}
\]

for $p = NT + 2$:

\[
\begin{align*}
  r = 1 & \Rightarrow j = (NT + 1)NC + 1 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,1}} = 0 \\
  r = 2 & \Rightarrow j = (NT + 1)NC + 2 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,2}} = 0 \\
  r = 3 & \Rightarrow j = (NT + 1)NC + 3 \\
  F'_{ij} &= \frac{df_i}{dx_{NT+2,3}} = [-V] / M_{NT+2}
\end{align*}
\]

for $p = 1 \ldots (NT - 1)$ and $r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$:

\[
F'_{ij} = \frac{df_i}{dx_{pr}} = 0
\]
EVALUATION OF $\frac{dy_{i,k}}{dx_{j,n}}$:

for $i \neq j$:

$$\frac{dy_{i,k}}{dx_{j,n}}$$

for $i = j$:

$$\frac{dy_{i,k}}{dx_{j,n}} = \frac{dy_{i,k}}{dx_{i,n}} = E_{i,kn}$$

where

$$y_{i,k} = x_{i,k}10^{\alpha_{ik}}/P_i$$

$$\alpha_{ik} = a_k - \frac{b_k}{c_k + T_i}$$

and

$$T_i = T_i(x_{i,1}...x_{i,NC}, P_i)$$

Evaluation of $E_{i,kn}$ for $i = 1...NT + 2$, $k = 1...NC$ and $n = 1...NC$:

for $k = n$:

$$E_{i,nn} = \frac{d}{dx_{i,k}} [x_{i,k}10^{\alpha_{ik}}/P_i]$$

or

$$E_{i,nn} = \left(\frac{1}{P_i}\right) \left[10^{\alpha_{ik}} + x_{i,k} \left(\ln(10)10^{\alpha_{ik}} \frac{d\alpha_{ik}}{dx_{i,k}}\right)\right]$$

for $k \neq n$:

$$E_{i,kn} = \frac{d}{dx_{in}} [x_{i,k}10^{\alpha_{ik}}/P_i]$$
\[ E_{i, kn} = \left( 1/P_i \right) \left[ x_{i,k} \left( \ln(10) 10^{\alpha_{ik}} \frac{d\alpha_{ik}}{dx_{in}} \right) \right] \]

**Evaluation of \( \frac{d\alpha_{ik}}{dx_{in}} \) for \( i = 1 \ldots NT + 2, \ k = 1 \ldots NC \) and \( n = 1 \ldots NC \):**

From given
\[ \alpha_{ik} = a_k - \frac{b_k}{c_k + T_i} \]

and
\[ T_i = T_i(x_{i,1}, \ldots x_{i,NC}, P_i) \]

the derivative becomes
\[ \frac{d\alpha_{ik}}{dx_{in}} = \theta_{ik} T X D_{in} \]

where
\[ \theta_{ik} = \frac{b_k}{(c_k + T_i)^2} \]

and
\[ T X D_{in} = \frac{dT_i}{dx_{in}} \]

**Evaluation of \( \frac{dT_i}{dx_{in}} \) for \( i = 1 \ldots NT + 2 \) and \( n = 1 \ldots NC \):**

For given \( T_i \) as
\[ T_i : \sum_{m=1}^{NC} x_{im} \cdot 10^{[\alpha_m - \frac{b_m}{c_m - \tau}]} = P_i \]

the general form of \( T \) \( X \) \( D_{in} = \frac{dT_i}{dx_{in}} \) is obtained as
\[ T X D_{in} = \frac{-10^{\alpha_{ik}}}{\ln(10) \sum_{m=1}^{NC} x_{im} 10^{\alpha_{im}} \theta_{im}} \]

**LAST FORM OF \( E_{i, kn} = \frac{dy_{i,k}}{dx_{i,n}} \):**

for \( k = n \):
\[ E_{i, kk} = \left( 1/P_i \right) \left[ 10^{\alpha_{ik}} + x_{i,k} \ln(10) 10^{\alpha_{ik}} \frac{d\alpha_{ik}}{dx_{ik}} \right] \]
or

\[ E_{i,kk} = \left( \frac{1}{P_i} \right) \left[ 10^{\alpha_{ik}} + \frac{\phi_{ik}}{TOT_i} 10^{\alpha_{ik}} \right] \]

where

\[ \phi_{ik} = \ln(10)x_{i,k}10^{\alpha_{ik}}\theta_{ik} \]

\[ TOT_i = \sum_{m=1}^{NC} \phi_{im}, \theta_{ik} = \frac{b_k}{(c_k + T_i)^2} \]

and

\[ \alpha_{ik} = a_k - \frac{b_k}{c_k + T_i} \]

for \( k \neq n \):

\[ E_{i,kn} = \left( \frac{1}{P_i} \right) \left[ x_{i,k} \ln(10)10^{\alpha_{ik}} \frac{d\alpha_{ik}}{dx_{in}} \right] \]

or

\[ E_{i,kk} = \left( \frac{1}{P_i} \right) \left[ \phi_{ik} - \frac{10^{\alpha_{in}}}{TOT_i} \right] \]

where

\[ \phi_{ik} = \ln(10)x_{i,k}10^{\alpha_{ik}}\theta_{ik} \]

\[ TOT_i = \sum_{m=1}^{NC} \phi_{im}, \theta_{ik} = \frac{b_k}{(c_k + T_i)^2} \]

and

\[ \alpha_{in} = a_n - \frac{b_n}{c_n + T_i} \]

### C.2 Evaluation of The Linearized Measurement Matrix H

This section is devoted to the evaluation of general form of Linearized Measurement Matrix, \( H \) previously given by Equation 5.16. Each element of
$H'$ is found by setting the indices of $H'_{ij}$ (i.e. $i$, $p$ and $r$) in Equation C.11

$$H'_{ij} = \frac{dh_i}{dX_j} = \frac{dh_i}{dX_{(p-1),NC+r}} = \frac{dT_{M(i)}}{dx_{p,r}} \tag{C.11}$$

for $i = 1 \ldots NM$

$p = 1 \ldots (NT + 2)$

$r = 1 \ldots NC$

as follows.

for $i = 1 \ldots NM$:

$$h_i = T_{M(i)}: \sum_{l=1}^{NC} x_{il}10^{\alpha_{il}} = P_i \tag{C.12}$$

for $p \neq M(i)$:

$r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$

$$H'_{ij} = \frac{dh_i}{dx_{p,r}} = 0$$

for $p = M(i)$:

$r = 1 \ldots NC \Rightarrow j = (p - 1)NC + r$

$$H'_{ij} = \frac{dh_i}{dx_{p,r}} = \frac{dT_{M(i)}}{dx_{p,r}} = \frac{dT_p}{dx_{p,r}} = TXD_{pr}$$

or

$$H'_{ij} = TXD_{pr} = \frac{-10^{\alpha_{pr}}}{\ln(10) \sum_{m=1}^{NC} x_{pm}10^{\alpha_{pm}} \theta_{pm}}$$
APPENDIX D

SOURCE CODE AND INPUT FILES

D.1 Main Program Code

D.1.1 Glob_Dec.s

```matlab
% ################################################################################################################## %
% -------------------------------------- Programming Definitions ------------------------------------------- %
% ------------------------------------------------------------------------------------------------------------------ %
% ===========================================================
% Simulation Parameters
% ------------------------------------------------------------------------------------------------------------------
% Dummy variables
% global Dummy1; global Dummy2; global Dummy3; global Dummy4;
% Output Warnings
% global OUT_WARNING;
% % tolerance for the decision to make the component fraction zero
% global zero_tolerance;
% % Set ID file
% % Liquid Profile file
% global FID_lprofile;
% % Vapor Profile file
% global FID_vprofile;
% % Temperature Profile file
% global FID_tprofile;
% % Holdup Profile file
% global FID_holdup;
% % Liquid and Vapor Flowrate Profile file
% global FID_lvflow;
% % Controller Outputs file
% global FID_control;
% % Estimator Outputs file
% global FID_estimator;
% % Tank Outputs file
% global FID_tank;
% % -------------------------------------------------------------- End Programming Definitions -------------------------------------- %
% ################################################################################################################## %
% ################################################################################################################## %
% -------------------------------------- Real Plant Simulation Parameters -------------------------------- %
% ------------------------------------------------------------------------------------------------------------------ %
% % Physical System Definitions
% % Number of Components
% global NC;
% % Number of Trays
% global NT;
% % Murphree Efficiency
% global Eff_Murphree;
% % Tray Efficiencies
% global Eff;
% % Tray(s) Volumetric holdups (m3)
```
global Tray_Vol_Holdup;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Feed Specs.
% Total amount of feed charged to the still pot (moles)
global M_Feed;
% Feed compositions (moles/moles)
global X_Feed;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Column Specs.
% Number of Components - in the order of volatilities :
% 1st is most volatile and last is least volatile
NC = 3;
% Number of Trays
NT = 8;
% Murphree Efficiency
Eff_Murphree = 1.0;
% Tray Efficiencies
Eff = Eff_Murphree * ones(NT,NC);
% Tray(s) Volumetric holdups (m)$
Tray_Vol_Holdup = 0.00132299341903;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Feed Specs.
% Total amount of feed charged to the still pot (moles)
M_Feed = 2930.0;
% Feed compositions (moles/moles)
X_Feed = [0.407; 0.394; 0.199];
D.1.3 Cont_Plant_Mfile.m

function Cont_Plant_Mfile

% Clear command window
clear;
% Include all global variables
Glob_Decs;
% Initialize thermo_LIBRARY.dll
thermo_Init(0);
% Initialize all global variables
display('Global variables are initializing ...');
Glob_Initial;
display('Global variables have been initialized.

% Step time and Time Span
% Initial Time (hour)
tstart = 0.0;
% Integration time step (hour)
DeltaT = 0.0003;
% Time Span of Simulation (hour)
tfinal = 8.35;
num_step = round((tfinal - tstart)/DeltaT);
% Current Simulation Time (hour)
t = tstart;
% Previuos integration step Time (hour)
t_prv = t;
% Current displaying time step (hour)
disp_t = tstart;

% Time Step to Estimate and Control
% Time Step to Estimate (hour)
estimate_DeltaT = 1*DeltaT;
% Current Estimation time (hour)
estimate_t = tstart;

% Initialize Real Plant
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Initial Operation Params.
% Boiler load (J/hour)
Q_Boiler = 8.151047554853517e+007;
% One Over Reflux ratio (D/L0) (ratio)
R_Ratio_inv = 0.0;
% Distillate Flow Rate (mol/hour)
D_DistillRate = 0.0;
% Amount of product distilled (moles)
M_Distilled = 0.0;
% Initial still pot, tray(s), reflux drum pressure (Pa)
Press(1,1) = 101000.0;
Press(NT+2,1) = 101600.0;
Press(2:NT+1,1) = PressureProfile(Press(1), Press(NT+2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Initialize liquid compositions
% Instantaneous still pot, tray(s), reflux drum liquid compositions (moles/moles)
X_frac = zeros(NT+2,NC);
% Initially still pot, tray(s), reflux drum liquid compositions are equal to that of the feed
for i=1:NT+2; X_frac(i,:) = X_Feed.'; end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define physical properties
% Instantaneous still pot, tray(s), reflux drum vapor compositions (moles/moles)
Y_frac = zeros(NT+2,NC);
% Instantaneous still pot, tray(s), reflux drum temperature (K)
Temp = 360.0 * ones(NT+2,1);
% Instantaneous still pot, tray(s), reflux drum liquid phase enthalpy (J/moles)
H_l_Enthalpy = zeros(NT+2,1);
% Instantaneous still pot, tray(s), reflux drum vapor phase enthalpy (J/moles)
H_v_Enthalpy = zeros(NT+2,1);
% Instantaneous still pot, tray(s), reflux drum liquid phase average desity (kg/m3)
Ro_l_Density = zeros(NT+2,1);
% Instantaneous still pot, tray(s), reflux drum average molecular weight (kg/mol)
Mw_MolWeight = zeros(NT+2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define and Initialize liquid holdups
% Instantaneous still pot, tray(s), reflux drum total holdup amount (moles)
M_Holdup = zeros(NT+2,1);
% Initial reflux drum liquid holdup (moles)
M_Holdup(NT+2,1) = 20.0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define and Initialize liquid and vapor flowrates
% Instantaneous liquid flow rates leaving tray(s) and reflux drum (moles/hour)
L_flow = zeros(NT+2,1);
% Instantaneous vapor flow rates leaving still pot and tray(s) (moles/hour)
V_flow = zeros(NT+2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define outputs
Out_Temp = zeros(NT+2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define and Initialize storage tank compositions
% Instantaneous compositions in storage tanks (moles/moles) (2*(NC-1)xNC)
Tank_X_frac = zeros(2*(NC-1), NC);
% Instantaneous total holdup amount in storage tanks (moles)
Tank_M_Holdup = zeros(2*(NC-1), 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define and Initialize estimated storage tank compositions
% Instantaneous compositions in storage tanks (moles/moles) (2*(NC-1)xNC)
EST_Tank_X_frac = zeros(2*(NC-1), NC);
% Instantaneous total holdup amount in storage tanks (moles)
EST_Tank_M_Holdup = zeros(2*(NC-1), 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define Estimator Variables
% Measurement of Plant Inputs
EST_u = zeros(EST_NumMeasurement, 1);
% Measurement of Plant Outputs
EST_y = zeros(EST_NumOutputs, 1);
% Updated Estimates for States
EST_Xp = zeros(EST_NumStates, 1);
% Updated Estimates for Error Covariances
EST_Pp = zeros(EST_NumStates, EST_NumStates);
% Derivatives of Updated Estimates for States
EST_dXm = zeros(EST_NumStates, 1);
% Derivatives of Updated Estimates for Error Covariances
EST_dPm = zeros(EST_NumStates, EST_NumStates);
% Predicted Derivatives of States for Propagation (from Nonlinear Model)
EST_dXm_prop = zeros(EST_NumStates, 1);
% Predicted System Outputs for Update (from Nonlinear Model)
EST_Yp = zeros(EST_NumOutputStates, 1);
% Linear System Matrix for Propagation (from Linearized Model)
EST_F = zeros(EST_NumStates, EST_NumStates);
% Linear System Output Matrix for Update (from Linearized Model)
EST_H = zeros(EST_NumOutputStates, 1);

% Initialize Estimate
[outputs] = INIT_ESTIMATE;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Initialize Estimator
% Initialize Estimator States and Error Covariances
EST_Xm = EST_X0;
% Updated Estimates for States
EST_Xp = EST_X0;
% Predicted System Outputs for Update (from Nonlinear Model)
EST_Yp = zeros(EST_NumOutputStates, 1);
% Linear System Output Matrix for Update (from Linearized Model)
EST_H = zeros(EST_NumOutputStates, 1);
% % % % % % % Variables being monitored
% % % % % % % Vapor compositions (for monitoring)
DCEKF_LM_Y = zeros(NT+2,NC);
% % % % % % Temperatures (for monitoring)
DCEKF_LM_Temp = zeros(NT+2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define and Initialize estimated storage tank compositions
% Instantaneous compositions in storage tanks (moles/moles) (2*(NC-1)xNC)
EST_Tank_X_frac = zeros(2*(NC-1), NC);
% Instantaneous total holdup amount in storage tanks (moles)
EST_Tank_M_Holdup = zeros(2*(NC-1), 1);

% % % % % % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % % % % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filter Gain
EST_R = zeros(EST_NumStates, EST_NumMeasurement);

% Initialize Controller
% ===========================================================
% CONT_SetPoints : Controller Set Points of Product Specifications
% CONT_Num_Oper_Stage : Number of different operation stage
% CONT_DistillProfile : Distillate Flow rate values of different operation stages (Products and/or Slopcuts)
[CONT_SetPoints, CONT_Num_Oper_Stage, CONT_DistillProfile] = INIT_CONTROL;
% Define Controller Outputs
CONT_QBoiler = zeros(1,1);
CONT_RRatioinv = zeros(1,1);
% Initialize Activated Tank Index
Tank_Activated = 0;
% Initialize Current Stage Number (startup:0, )
CONT_Curr_Stage = 0;
% ===========================================================
% Integration Starts
% ==============================================================
for i=0:num_step;
% Find Current Information of Real Plant Starts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate current trays molar holdups
[M_Holdup] = P_Calc_Mol_Tray_Holdups(t, Ro_l_Density, Mw_MolWeight, M_Holdup);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate Initial Still Pot Holdup
[M_Holdup] = P_Calc_Mol_Still_Holdup(t, M_Holdup);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if t==0; % For initialization of variables depending on physical vars not available in initialization section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate current distillate, vapor and liquid flow rates
% Use initial values
% Calculate initial vapor and liquid flow rates
% Initialize still pot vapor flow rate
V_flow(1) = Q_Boiler/(H_v_Enthalpy(1) - H_l_Enthalpy(2));
% Initialize tray(s) vapor flow rates V_flow(NT+2) : dummy
for i=2:NT+1; V_flow(i) = V_flow(1); end;
% Initialize tray(s) and reflux drum liquid flow rates L_flow(1) : dummy
for i=NT+2:-1:2; L_flow(i) = V_flow(1); end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate Initial Still Pot Holdup
[M_Holdup] = P_Calc_Mol_Still_Holdup(t, M_Holdup, M_Distilled);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate Approximated derivatives
[DX_frac] = P_f(t, X_frac, Y_frac, Temp, M_Holdup, L_flow, V_flow, Q_Boiler, D_DistillRate, R_Ratio_inv);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get Real Plant Outputs
Out_Temp = [Temp];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Keep current parameters for future use
[t_prv, M_Holdup_prv, H_l_Enthalpy_prv] = P_Keep_Current_Vars(t, M_Holdup, H_l_Enthalpy);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find Current Information of Real Plant Ends
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end;
% Take Plant Inputs
EST_u = [V_flow(1,1); D_DistillRate];
% Get Model Inputs
% Vapor boilup rate (moles/h)
EST_model_Vb = EST_u(1,1);
% Distillate Flow Rate (moles/h)
EST_model_D = EST_u(2,1);
if (t <= 1.00001*estimate_t) & (t >= (2-1.00001)*estimate_t);
% Update
% Take measurements @ t
% Take Plant Outputs
EST_z = Out_Temp(EST_MesurementOrder(:),1);
% Update States and Error Covariances
% Find Predicted System Outputs
EST_model_X = reshape(EST_Xm, NC, NT+2)';
[EST_h] = KF_NonlinearModel_Out...
(EST_model_X, EST_model_Hb, EST_model_D, t, NC, NT, EST_model_Vb, EST_model_H,...
EST_model_P, EST_model_Ant, EST_MeasurementOrder);
% Find Linear System Output Matrix
EST_model_X = reshape(EST_Xm, NC, NT+2)';
[EST_H] = KF_LinearizedModel_SysH...
(EST_model_X, EST_model_Hb, EST_model_D, t, NC, NT, EST_model_Vb, EST_model_H,...
EST_model_P, EST_model_Ant, EST_MeasurementOrder);
% Find Filter Gain
EST_K = EST_Pm*EST_H'*inv(EST_H*EST_Pm*EST_H' + EST_R);
% Update States
EST_Xp = EST_Xm + EST_K*(EST_z - EST_h);
% Update Error Covariances
ugur = EST_K*EST_H;
EST_Pp = (eye(size(ugur)) - ugur)*EST_Pm;
% Set current EST_Xp and EST_Pp as initial conditions for Propagation
EST_Xm = EST_Xp;
EST_Pm = EST_Pp;
% Run controller (Find new real plant inputs)
% [CONT_QBoiler, CONT_RRatioinv, Tank_Activated] = CONTROL(t, X_frac, Q_Boiler, R_Ratio_inv, CONT_SetPoints,...
% CONT_Num_Oper_Stage, CONT_DistillProfile);
% [CONT_QBoiler, CONT_RRatioinv, Tank_Activated, CONT_Curr_Stage] = CONTROL_real(t, EST_model_X, Q_Boiler, R_Ratio_inv,...
% CONT_SetPoints, CONT_Num_Oper_Stage, CONT_DistillProfile, Tank_Activated, EST_Tank_X_frac, CONT_Curr_Stage);
% Set next measurement (or update) time
estimate_t = estimate_t + estimate_DeltaT;
end;
% Find Derivatives to Propagate
% propagate from (estimate_t - estimate_DeltaT) to (estimate_t = t)
% to find EST_Xm and EST_Pm
% taking current EST_Xp and EST_Pp (which are updated estimates at t=estimate_t - estimate_DeltaT) as initial conditions
% Find Derivatives of Propagated Estimates for States
% Find Predicted Derivatives of States
EST_model_X = reshape(EST_Xm, NC, NT+2)';
[EST_f, dummy_Y, dummy_Temp] = KF_NonlinearModel_Sys...
(EST_model_X, EST_model_Hb, EST_model_D, t, NC, NT, EST_model_Vb, EST_model_H,...
EST_model_P, EST_model_Ant);  
% Find Derivatives of Propagated Estimates for Error Covariances
% Find Linear System Matrix
EST_model_X = reshape(EST_Xm, NC, NT+2)';
[EST_F] = KF_LinearizedModel_SysF...
(EST_model_X, EST_model_Hb, EST_model_D, t, NC, NT, EST_model_Vb, EST_model_H,...
EST_model_P, EST_model_Ant);  
% Find Derivatives of Still Pot Holdup amount
% Derivative of Still Pot amount
EST_model_dHb = - EST_model_D;
if t >= disp_t;
% Write plant data to Screen
write_plant_to_scr(t, X_frac);
% Write estimator and controller data to Screen
write_estcont_to_scr(t, EST_Xp);
% Write plant data to file
write_plant_to_file(t, X_frac, Y_frac, Temp, M_Holdup, L_flow, V_flow);
% Write estimator and controller data to file
write_estcont_to_file(t, CONT_QBoiler, CONT_Ratioinv, EST_Xp, EST_Pp);
% Write tank compositions and holdups
Tank_X_frac
write_tank_to_file(t, Tank_X_frac, Tank_M_Holdup, EST_Tank_X_frac, EST_Tank_M_Holdup);
disp_t = disp_t + disp_DeltaT;
end;
% Stop simulation when the distillation finishes
if (Tank_Activated == -1)
    break;
end;

% Real Plant Simulation Starts
% Calculate current holdup amount and composition in storage tanks
[Tank_X_frac, Tank_M_Holdup] = P_Calc_Tanks(t, DeltaT, Tank_X_frac, Tank_M_Holdup, X_frac(NT+2,:), D_DistillRate, Tank_Activated);

% Calculate current amount of product distilled
[M_Distilled] = P_Calc_Distilled_Amount(t, DeltaT, D_DistillRate, M_Distilled);

% Calculate current Still Pot Holdup
[M_Holdup] = P_Calc_Mol_Still_Holdup(t, M_Holdup, M_Distilled);

% Take Integration
[t_dummy, X_frac] = P_Int_Euler(t, DeltaT, X_frac, DX_frac);

% Normalize Plant States
[X_frac] = P_Normalize_States(t, X_frac);

% Real Plant Simulation Ends

% Estimator Propagation Integration Starts
% Estimate current holdup amount and composition in storage tanks
EST_model_X = reshape(EST_Xp, NC, NT+2)';
[EST_Tank_X_frac, EST_Tank_M_Holdup] = EST_Tanks(t, DeltaT, EST_Tank_X_frac, EST_Tank_M_Holdup,...
    EST_model_X(NT+2,:), EST_model_D, Tank_Activated);

% Take Integration for Still Pot Holdup Amount
[t_dummy, EST_model_Hb] = EST_Int_Euler(t, DeltaT, EST_model_Hb, EST_model_dHb);

% Take Integration for States
[t_dummy, EST_Xm] = EST_Int_Euler(t, DeltaT, EST_Xm, EST_dXm);

% Normalize States
EST_model_X = reshape(EST_Xm, NC*(NT+2), 1);

% Take Integration for Error Covariances
[t_dummy, EST_Pm] = EST_Int_Euler(t, DeltaT, EST_Pm, EST_dPm);

% Estimator Propagation Integration Ends
% Set Current Time
    t = t + DeltaT;

% Close Output Files
fclose(FID_lprofile);
fclose(FID_vprofile);
fclose(FID_tprofile);
fclose(FID_holdup);
fclose(FID_lvflow);
fclose(FID_control);
fclose(FID_estimator);
fclose(FID_tank);

% Simulation finishes
end;
fprintf('Simulation finished successfully.

');

% End of Main Program function Cont_Plant_Mfile

% P_Calc_Phys_Vars
% Return
% Find current physical variables
% given
% Time, t; Previous system variables (Temp at previous time step is for initial guess)

function [Y_frac, Temp, H_l_Enthalpy, H_v_Enthalpy, Ro_l_Density, Mw_MolWeight] = P_Calc_Phys_Vars(t, X_frac, Press, TempPr)
Glob_Decs;

%% Calculated only by using liquid composition and Pressure
% Set sizes
Y_frac = zeros(size(X_frac)); Temp = zeros(size(TempPr));
H_l_Enthalpy = zeros(size(TempPr)); H_v_Enthalpy = zeros(size(TempPr)); Ro_l_Density = zeros(size(TempPr));
Mw_MolWeight = zeros(size(TempPr));

% Buble Point Calculation
for i=1:NT+2;
    [Temp(i), Dummy3] = thermo_Equilibrium(TempPr(i), Press(i), X_frac(i,:));
    Y_frac(i,:) = Dummy3;
    if (i>1) & (i<NT+2);
        Y_frac(i,:) = Y_frac(i-1,:) + Eff(i-1,:).*(Y_frac(i,:) - Y_frac(i-1,:));
    end;
end;

% Caculate specific phase enthalpies
for i=1:NT+2;
    [H_l_Enthalpy(i), H_v_Enthalpy(i)] = thermo_Enthalpy(Temp(i), Press(i), X_frac(i,:), Y_frac(i,:));
end;

% Caculate liquid phase average desity and average molecular weight
for i=1:NT+2;
    [Mw_MolWeight(i), Ro_l_Density(i)] = thermo_Density(Temp(i), Press(i), X_frac(i,:));
end;

end P_Calc_Phys_Vars

% P_Calc_Mol_Tray_Holdups
% Return
% Calculate current trays molar holdups
% given
% Time, t; Avg. Density, Ro_l_Density; Avg. Molecular Weight, Mw_MolWeight;

function [M_Holdup] = P_Calc_Mol_Tray_Holdups(t, Ro_l_Density, Mw_MolWeight, M_Holdup_pr)
Glob_Decs;

% Set size
M_Holdup = M_Holdup_pr;

% Caculate tray(s) molar holdups (mol)
for i=2:NT+1;
    M_Holdup(i) = (Ro_l_Density(i) / Mw_MolWeight(i)) * Tray_Vol_Holdup;
end;

end P_Calc_Mol_Tray_Holdups

% P_Calc_Mol_Drum_Holdup
% Return
% Calculate current Reflux-Drum molar holdup
% given
% Time, t; Previous Holdups, M_Holdup_pr;

function [M_Holdup] = P_Calc_Mol_Drum_Holdup(t, M_Holdup_pr)
Glob_Decs;

% Reflux Drum Molar holdup is constant
M_Holdup(NT+2) = M_Holdup_pr(NT+2);

end P_Calc_Mol_Drum_Holdup

% P_Calc_Apprx_Deriv
% Return
% Calculate Approximated derivatives
% given
% Time, t;

function [Del_M_Holdup, Del_H_l_Enthalpy, Del_M_Hl] = P_Calc_Apprx_Deriv(t, M_Holdup, H_l_Enthalpy, t_prv, M_Holdup_prv,
H_l_Enthalpy_prv) Glob_Decs;

% Set sizes
Del_M_Holdup = zeros(size(M_Holdup)); Del_H_l_Enthalpy = zeros(size(H_l_Enthalpy)); Del_M_HI = zeros(size(M_Holdup));

% Calculate step size
del_t = t - t_prv;
% d(M_Holdup)/dt approximated by forward differentiation
Del_M_Holdup = (M_Holdup - M_Holdup_prv) / del_t;
% d(H_l_Enthalpy)/dt approximated by forward differentiation
Del_H_l_Enthalpy = (H_l_Enthalpy - H_l_Enthalpy_prv) / del_t;
% d(M_Holdup*H_l_Enthalpy)/dt approximated by forward differentiation
Del_M_Hl = (M_Holdup.*H_l_Enthalpy - M_Holdup_prv.*H_l_Enthalpy_prv) / del_t;

%end P_Calc_Apprx_Deriv

%=============================================================================% P_Calc_LV_for_Total_Reflux
% Return
% Calculates liquid and vapor flow rates for total reflux condition
% or for D=0
% given
% Time, t;
%=============================================================================function [D_DistillRate, L_flow, V_flow] = P_Calc_LV_for_Total_Reflux(t, H_l_Enthalpy, H_v_Enthalpy, M_Holdup, Del_M_Holdup, Del_H_l_Enthalpy, Del_M_Hl, Q_Boiler, R_Ratio_inv) Glob_Decs;
% set sizes
D_DistillRate = zeros(size(1,1)); L_flow = zeros(size(M_Holdup)); V_flow = zeros(size(M_Holdup));
% Calculate distillate flow rate
D_DistillRate = 0.0;
% Calculate liquid flow rate from reflux drum
sum_Del_M_Holdup = sum(Del_M_Holdup(2:NT+1)); sum_Del_M_HI = sum(Del_M_HI(2:NT+1)); L_flow(NT+2) = ( Q_Boiler + H_l_Enthalpy(1)*sum_Del_M_Holdup - M_Holdup(1)*Del_H_l_Enthalpy(1) - sum_Del_M_Hl )... / ( H_v_Enthalpy(NT+1) - H_l_Enthalpy(NT+2));
% fprintf(['Qb=' num2str(Q_Boiler) '
']);
% fprintf(['h(1)= ' num2str(H_l_Enthalpy(1)) ' SumDel_M=' num2str(sum_Del_M_Holdup) '
']);
% fprintf(['M(1)= ' num2str(M_Holdup(1)) ' Del_Hl(1)= ' num2str(Del_H_l_Enthalpy(1)) '
']);
% fprintf(['SumDel_Mh=' num2str(sum_Del_M_Hl) '
']);
% fprintf(['H(NT+1)= ' num2str(H_v_Enthalpy(NT+1)) ' h(NT+2)= ' num2str(H_l_Enthalpy(NT+2)) '
']);
% Calculate vapor flow rate from top tray
V_flow(NT+1) = D_DistillRate * (R+1);
% Calculate liquid flow rate from reflux drum
L_flow(NT+2) = D_DistillRate * R;
% Calculate other vapor and liquid flow rates
for i=NT+1:-1:2;
   V_flow(i-1) = ( V_flow(i)*(H_v_Enthalpy(i) - H_l_Enthalpy(i)) + L_flow(i+1)*(H_l_Enthalpy(i) - H_l_Enthalpy(i+1))... + M_Holdup(i)*Del_H_l_Enthalpy(i) ) / (H_v_Enthalpy(i-1) - H_l_Enthalpy(i));
   L_flow(i) = V_flow(i-1) + L_flow(i+1) - V_flow(i) - Del_M_Holdup(i);
% fprintf(['V' num2str(i-1) '=' num2str(V_flow(i-1)) ' L' num2str(i) '=' num2str(L_flow(i)) '
']);
end;
% pause
%end P_Calc_LV_for_Total_Reflux

%=============================================================================% P_Calc_LV_for_Finite_Reflux
% Return
% Calculates liquid and vapor flow rates for distillate withdrawal
% or for D=0 or D different than zero.
% given
% Time, t;
%=============================================================================function [D_DistillRate, L_flow, V_flow] = P_Calc_LV_for_Finite_Reflux(t, H_l_Enthalpy, H_v_Enthalpy, M_Holdup, Del_M_Holdup, Del_H_l_Enthalpy, Del_M_Hl, Q_Boiler, R_Ratio_inv) Glob_Decs;
% set sizes
D_DistillRate = zeros(size(1,1)); L_flow = zeros(size(M_Holdup)); V_flow = zeros(size(M_Holdup));
% Calculate distillate flow rate
R = 1 / R_Ratio_inv; Dummy1 = sum(Del_M_Hl(1:NT+1));
D_DistillRate = ( Q_Boiler - Dummy1 ) / ( (R+1)*H_v_Enthalpy(NT+1) - R*H_l_Enthalpy(NT+2) );
% Calculate vapor flow rate from top tray
V_flow(NT+1) = D_DistillRate * (R+1);
% Calculate liquid flow rate from reflux drum
L_flow(NT+2) = D_DistillRate * R;
% Calculate other vapor and liquid flow rates for i=NT+1:-1:2;
%fprintf(['V' num2str(i-1) '=' num2str(V_flow(i-1)) ' L' num2str(i) '=' num2str(L_flow(i)) ' 
']);
end;
\[
V_{\text{flow}(i-1)} = (V_{\text{flow}(i)} + (H_v_{\text{Enthalpy}(i)} - H_l_{\text{Enthalpy}(i)}) + L_{\text{flow}(i+1)}(H_l_{\text{Enthalpy}(i)} - H_l_{\text{Enthalpy}(i+1)}) + M_{\text{Holdup}(i)}(H_l_{\text{Enthalpy}(i)} - H_l_{\text{Enthalpy}(i+1)}) / (H_v_{\text{Enthalpy}(i-1)} - H_l_{\text{Enthalpy}(i)});
\]

\[
L_{\text{flow}(i)} = V_{\text{flow}(i-1)} + L_{\text{flow}(i+1)} - V_{\text{flow}(i)} - M_{\text{Holdup}(i)};
\]

end

% P_Call_LV_for_Finite_Reflux

%=============================================================================% P_Call_Distilled_Amount
% Return
% Calculate constant amount of product distilled
% given
% Time, t;
%=============================================================================%function [M_{\text{Distilled}}] = P_Call_Distilled_Amount(t, t_{\text{prev}}, D_{\text{DistillRate}}, M_{\text{Distilled prev}}) Glob_Decs;
% set size
M_{\text{Distilled}} = 0.0;
% Calculate amount of product distilled
M_{\text{Distilled}} = M_{\text{Distilled prev}} + D_{\text{DistillRate}} * t_{\text{delta t}};
%end P_Call_Distilled_Amount

%=============================================================================% P_Call_Mol_Still_Holdup
% Return
% Calculate Still Pot Holdup
% given
% Time, t;
%=============================================================================%function [M_{\text{Holdup}}] = P_Call_Mol_Still_Holdup(t, M_{\text{Holdup prev}}, M_{\text{Distilled}}); Glob_Decs;
% Set sizes
M_{\text{Holdup}} = M_{\text{Holdup prev}};
% Instantaneous still pot total holdup amount (moles/hour)
M_{\text{Holdup}}(i) = M_{\text{Feed}} - sum(M_{\text{Holdup prev}}(2:NT+1)) - M_{\text{Holdup prev}}(NT+2) - M_{\text{Distilled}};
%end P_Call_Mol_Still_Holdup

%=============================================================================% P_f
% Return the derivatives for the continuous states.
%=============================================================================%function [DX_{\text{frac}}] = P_f(t, X_{\text{frac}}, Y_{\text{frac}}, Temp, M_{\text{Holdup}}, L_{\text{flow}}, V_{\text{flow}}, Q_{\text{Boiler}}, D_{\text{DistillRate}}, R_{\text{Ratio inv}}) Glob_Decs;
DX_{\text{frac}} = zeros(size(X_{\text{frac}}));
% Instantaneous still pot, tray(s), reflux drum liquid compositions derivatives (moles/moles/hour)
for j=1:NC;
% still pot
DX_{\text{frac}}(1,j) = ( L_{\text{flow}(2)}*(X_{\text{frac}(2,j)}-X_{\text{frac}(1,j)}) - V_{\text{flow}(1)}*(Y_{\text{frac}(1,j)}-X_{\text{frac}(1,j)})) / M_{\text{Holdup}(1)};
% tray(s)
for i=2:NT+1;
DX_{\text{frac}}(i,j) = ( V_{\text{flow}(i-1)}(Y_{\text{frac}(i-1,j)}-X_{\text{frac}(i,j)}) + L_{\text{flow}(i+1)}(X_{\text{frac}(i+1,j)}-X_{\text{frac}(i,j)}) - V_{\text{flow}(i)}(Y_{\text{frac}(i,j)}-X_{\text{frac}(i,j)}) ) / M_{\text{Holdup}(i)};
end;
% reflux drum
DX_{\text{frac}}(NT+2,j) = ( V_{\text{flow}(NT+1)}(Y_{\text{frac}(NT+1,j)}-X_{\text{frac}(NT+2,j)}) ) / M_{\text{Holdup}(NT+2)};
end;
% end mdlDerivatives

%=============================================================================% P_Keep_Current_Vars
% Return
% Keeps current parameters for future use
% given
% Time: t; Variables at t: ...;
%=============================================================================%function [t_{\text{prev}}, M_{\text{Holdup prev}}, H_{\text{l Enthalpy prev}}] = P_Keep_Current_Vars(t, M_{\text{Holdup}}, H_{\text{l Enthalpy}});
% Previous step time
t_{\text{prev}} = t;
% Previous step still pot, tray(s), reflux drum total holdup amount (moles)
M_{\text{Holdup prev}} = M_{\text{Holdup}};
% Previous step still pot, tray(s), reflux drum liquid phase enthalpy (J/moles)
H_{\text{l Enthalpy prev}} = H_{\text{l Enthalpy}};
%end P_Keep_Current_Vars

%=============================================================================% P_Int_Euler
% Return
% Take integral
% given
% Time: t; Integration Step: delta_t; Previous States: X_{\text{frac}};
% Derivatives: DX_{\text{frac}};
%=============================================================================%function [t_{\text{new}}, X_{\text{frac new}}] = P_Int_Euler(t, delta_t, X_{\text{frac}}, DX_{\text{frac}});
\[ t_{\text{new}} = t + \Delta t; \]
\[ X_{\text{frac}}_{\text{new}} = X_{\text{frac}} + DX_{\text{frac}} \cdot \Delta t; \]

\textbf{End Int} \_\textbf{Euler}

%%%=============================================================================%
%%% \textbf{P\_Normalize\_States} \textbf{Return} \textbf{Normalizes Plant States} \textbf{given} \textbf{Time: t; States: X\_frac\_Pr;}
%%%=============================================================================

\textbf{function} \[ [X_{\text{frac}}] = \text{P\_Normalize\_States}(t, X_{\text{frac}\_Pr}) \]

\textbf{Glob\_Decs;}

\textbf{Set size} \[ X_{\text{frac}} = X_{\text{frac}\_Pr}; \]

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Make the low compositions zero %%%%%%%%%%%%%%%%%%%%%%%%%
% Check for still pot, tray(s), reflux drum liquid compositions (moles/moles)
for \[ i=1:\text{NT}+2; \]
  for \[ j=1:\text{NC}; \]
    if \[ \text{isnan}(X_{\text{frac}\_Pr(i,j))}; \]
      \text{error}([\text{Stage no '}, \text{num2str(i)}, \text{ component '}, \text{num2str(j)}, \text{ liquid fraction is Nan'}]);
    end;
    if \[ X_{\text{frac}\_Pr(i,j) < \text{zero}\_\text{tolerance}}; \]
      \text{display}([\text{Stage no '}, \text{num2str(i)}, \text{ composition of comp. '}, \text{num2str(j)}, \text{ made zero'}]);
      \[ X_{\text{frac}(i,j) = 0.0}; \]
    end;
  end;
end;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Normalize the liquid compositions (moles/moles) %%%%%%%% 
% Normalize still pot, tray(s), reflux drum liquid compositions (moles/moles)
for \[ i=1:\text{NT}+2; \]
  \[ \text{dummy}_1 = \text{sum}(X_{\text{frac}\_Pr(i,:)}); \]
  if \[ \text{if} \text{(dummy}_1 < 0.0); \]
    \text{error}([\text{Sum of comp. fraction in the Stage no '}, \text{num2str(i)}, \text{ is zero. '}, \text{num2str(X_{frac\_Pr(i,:)}))}]];
  else
    \text{end};
  \end;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Check compositions are in the limit of \([0,1]\) %%%%%% 
% Check still pot, tray(s), reflux drum liquid compositions (moles/moles)
for \[ i=1:\text{NT}+2; \]
  if \[ \text{if} X_{\text{frac}(i,j)} < 0 \text{ or } X_{\text{frac}(i,j)} > 1); \]
    \text{error}([\text{Composition out of limit ! - [Normalize\_States] X_{frac'}}\text{,num2str(i)},', = ', \text{num2str(X_{frac\_Pr(i,:))})}]];
  end;
end;

\textbf{End P\_Normalize\_States}

%%%=============================================================================%
%%% \textbf{P\_Calc\_Tanks} \textbf{Return} \textbf{Storage Tank Holdups and Compositions} %%%%%%%%
%%%=============================================================================

\textbf{function} \[ [X, M] = \text{P\_Calc\_Tanks}(t, \Delta T, X, M, X\_Drum, D\_Rate, \text{Active}); \]

\textbf{if} \[ \text{Active} \neq 0; \]
  \text{Calculate increase in holdup}
\[ \Delta \text{M} = \text{D\_Rate} \cdot \Delta T; \]
\textbf{if} \[ \text{if} (\Delta \text{M} = 0) \text{ and } M(\text{Active}, 1) = 0) \]
\[ X(\text{Active}, :) = \text{zeros(size}(X(\text{Active}, :)))); \]
\textbf{else}
\[ \text{end}; \]
\[ X(\text{Active}, :) = (X(\text{Active}, :) + M(\text{Active}, 1) \cdot X\_\text{Drum}(1, :) \cdot \Delta \text{M}) / (M(\text{Active}, 1) + \text{M}); \]
\textbf{end};
\textbf{if} \[ \text{Calculate tank's current holdup} \]
\[ M(\text{Active}, 1) = M(\text{Active}, 1) + D\_\text{Rate} \cdot \Delta T; \]
\textbf{end};

\textbf{End P\_Calc\_Tanks}

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \textbf{Estimator Functions} %%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
% \textbf{INIT\_ESTIMATE} \textbf{(don't MODIFY GLOBAL VARIABLES not owned by this function)}
% % perform \textbf{Initialize estimator}
% % given \textbf{All Global variables}
% % output

149
% function [outputs] = INIT_ESTIMATE(Glob_Decs);
% outputs = [];
%end INIT_ESTIMATE

%function [outputs] = ESTIMATE(t) Glob_Decs;
%outputs = [];
%end ESTIMATE

%function [t_new, X_new] = EST_Int_Euler(t, delta_t, X, dX);
% t_new = t + delta_t;
% X_new = X + dX*delta_t;
%end EST_Int_Euler

%function [X_frac] = EST_Norm_States(t, X_frac_Pr) Glob_Decs;
% Set size
% X_frac = X_frac_Pr;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Make the low compositions zero %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for i=1:NT+2;
% for j=1:NC;
% if isnan(X_frac_Pr(i,j));
% error(’Stage no. ‘, num2str(i), ’ component ‘, num2str(j), ’ liquid fraction is Nan’);  
% end;
% if X_frac_Pr(i,j) < zero_tolerance;
% display(’Stage no.’, num2str(i), ’ composition of comp. ‘, num2str(j), ’ (’, num2str(X_frac_Pr(i,j)),’) made zero’);
% X_frac(i,j) = 0.0;
% end;
% end;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normalize the liquid compositions (moles/moles) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for i=1:NT+2;
% dummy1 = sum(X_frac_Pr(i,:));
% if (dummy1 > 0.0);
% error(’Sum of comp. fraction in the Stage no. ‘, num2str(i), ’ is zero. ’, num2str(X_frac_Pr(i,:))’));
% else
% X_frac(i,:) = X_frac_Pr(i,:) / dummy1;
% end;
% end;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check compositions are in the limit of [0,1] %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for i=1:NT+2;
% for j=1:NC;
% if (X_frac(i,j) < 0 | X_frac(i,j) > 1);
% error(’Composition out of limit! ‘, num2str(X_frac(i,:)), ’) ’);  
% end;
% end;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%end EST_Norm_States

%function [I, M] = EST_Tanks(t, DeltaT, X, M, k_rate, Active);
% if Active=’0’;
% % Calculate increase in holdup
%
\[
\delta M = D_Rate \times \Delta T;
\]

% Calculate tank's composition
if ((\delta M == 0) & M(Active, 1)==0)
    X(Active, :) = zeros(size(X(Active, :)));
else
    X(Active, :) = (X(Active, :) * M(Active, 1) + X_Drum(1, :) * \delta M) / (M(Active, 1) + \delta M);
end;

% Calculate tank's current holdup
M(Active, 1) = M(Active, 1) + D_Rate*\Delta T;
end;

% INIT_CONTROL
% (don't MODIFY GLOBAL VARIABLES not owned by this function)
% perform
% given
% All Global variables
% output
% any output required
 function [SetPoints, Num_Oper_Stage, DistillProfile] = INIT_CONTROL Glob_Decs;

 SetPoints = zeros(NC,1);
 SetPoints = [0.9; 0.81; 0.69];%[0.96; 0.87; 0.74];%[0.9; 0.81; 0.69];

 Num_Oper_Stage = 5;

 DistillProfile = zeros(Num_Oper_Stage,1);

%end INIT_CONTROL

% CONTROL_real (don't MODIFY GLOBAL VARIABLES not owned by this function)
% perform
% given
% Current Time, t; All Global variables at time t;
% output
% any output required
 function [Q_Boiler, R_Ratio_inv, Tank_Active, CONT_Curr_Stage_new] = CONTROL_real(t, X_frac, Q_Boiler_pr, R_Ratio_inv_pr, CONT_SetPoints, CONT_Num_Oper_Stage, CONT_DistillProfile, Tank_Active_prv, X_tank, CONT_Curr_Stage) Glob_Decs;

 if (CONT_Curr_Stage==0) %----------- Total Reflux operation
    if (X_frac(NT+2,1)<CONT_SetPoints(1,1))
        R_Ratio = -1;
        Tank_Active = 0;
        CONT_Curr_Stage = 0;
    elseif (X_frac(NT+2,1)>=CONT_SetPoints(1,1))
        '0->1';
        CONT_Curr_Stage = 1;
    end;
 end;

 if (CONT_Curr_Stage==1) %----------- 1st product-cut distillation to 1st product-cut tank
    if ((X_tank(1,1)==0) & (X_tank(1,1)==CONT_Sets(1,1)))
        R_Ratio = 13.92537313432837;
        Tank_Active = 2;
        CONT_Curr_Stage = 2;
    elseif (X_tank(1,1)<CONT_Sets(1,1))
        '1->2';
        CONT_Curr_Stage = 2;
    end;
 end;

 if (CONT_Curr_Stage==2) %----------- 1st slop-cut distillation to 1st slop-cut tank
    if ((X_frac(NT+2,2)<CONT_Sets(2,1))
        R_Ratio = 7.00000000000000;
        Tank_Active = 1;
        CONT_Curr_Stage = 1;
    elseif (X_frac(NT+2,2)>=CONT_Sets(2,1))
        '2->3';
    %pause;
 end;

%end CONTROL_real
CONT_Curr_Stage = 3;
end;

if (CONT_Curr_Stage==3) %------------- 2nd product-cut distillation to 2nd product-cut tank
if ((X_tank(3,2)==0) | (X_tank(3,2)>=CONT_SetPoints(2,1)))
    R_Ratio = 4.91715976331361;
    Tank_Active = 3;
    CONT_Curr_Stage = 3;
elseif (X_tank(3,2)<CONT_SetPoints(2,1))
    '3->4'
    %pause;
    CONT_Curr_Stage = 4;
end;
end;

if (CONT_Curr_Stage==4) %------------- 2nd slop-cut distillation to 2nd slop-cut tank
% In reality, the fraction of the total amount of 3th component in the whole column is to be checked
if (X_frac(1,3)<CONT_SetPoints(3,1))
    R_Ratio = 7.06451612903226;
    Tank_Active = 4;
    CONT_Curr_Stage = 4;
elseif (X_frac(1,3)>=CONT_SetPoints(3,1))
    '4->5'
    %pause;
    CONT_Curr_Stage = 5;
end;
end;

if (CONT_Curr_Stage==5) %------------- Distillation stops
    R_Ratio = 1/R_Ratio_inv_pr;
    Tank_Active = -1;
end;

% Keep Current Stage #
CONT_Curr_Stage_new = CONT_Curr_Stage;

% Convert Reflux ratio (L0/D) to One Over Reflux ratio (D/L0)
R_Ratio_inv = 1.0 / R_Ratio;

% Find Reboiler load (J/hour)
% Constant Reboiler Load
Q_Boiler = Q_Boiler_pr;

end CONTROL_real

function [Q_Boiler, R_Ratio_inv, Tank_Active] = CONTROL(t, X_frac,
    Q_Boiler_pr, R_Ratio_inv_pr, CONT_SetPoints, CONT_Num_Oper_Stage,
    CONT_DistillProfile) Glob_Decs;

% Find new Reflux ratio (L0/D)
if (t >= 0 & t<2.06)
    R_Ratio = 7.00000000000000;
    Tank_Active = 1;
elseif (t>=2.06 & t<3.4)
    R_Ratio = 10.23595505617978;
    Tank_Active = 1;
elseif (t>=3.4 & t<6.17)
    R_Ratio = 13.92537313432837;
    Tank_Active = 2;
elseif (t>=6.17 & t<6.51)
    R_Ratio = 4.91715976331361;
    Tank_Active = 3;
elseif (t >= 6.51)
    R_Ratio = 7.06451612903226;
    Tank_Active = 3;
end;

% Convert Reflux ratio (L0/D) to One Over Reflux ratio (D/L0)
R_Ratio_inv = 1.0 / R_Ratio;

% Find Reboiler load (J/hour)
% Constant Reboiler Load
Q_Boiler = Q_Boiler_pr;

end CONTROL

% ------------------------------------------------------------------------------------------------------------------ %
% ---------------------------------------------- End Controller Functions ---------------------------------------- %
% ################################################################################################################## %
% ------------------------------ Simulation loop control user interface functions ------------------------- %

function write_plant_to_scr(t, X_frac) Glob_Decs;
% Reflux Drum Liquid composition
data = X_frac(NT+2,:); l = prod(size(data)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(['%9.4f ' Real; ' %s'; ', t, reshape(data, 1, prod(size(data))));
end write_plant_to_scr

function write_estcont_to_scr(t, EST_Xp) Glob_Decs;
% Estimated Reflux Drum Liquid composition
dummy = EST_Xp(NC*(NT+1)+1:NC*(NT+1)+NC,1); l = prod(size(dummy)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf([' Estd: ' formats '
'], reshape(dummy, 1, prod(size(dummy))));
end write_estcont_to_scr

function write_plant_to_file(t, X_frac, Y_frac, Temp, M_Holdup, L_flow, V_flow) Glob_Decs;
% Liquid Profile file l = prod(size(X_frac)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_lprofile, ['%9.4f ' formats '
'], t, reshape(X_frac, 1, l));

% Vapor Profile file l = prod(size(Y_frac)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_vprofile, ['%9.4f ' formats '
'], t, reshape(Y_frac, 1, l));

% Temperature Profile file l = prod(size(Temp)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_tprofile, ['%9.4f ' formats '
'], t, reshape(Temp, 1, l));

% Holdup Profile file l = prod(size(M_Holdup)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_holdup, ['%9.4f ' formats '
'], t, reshape(M_Holdup, 1, l));

% Liquid and Vapor Flowrate Profile file l = prod(size(L_flow)) + prod(size(V_flow)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_lvflow, ['%9.4f ' formats '
'], t, reshape([L_flow; V_flow], 1, l));
end write_plant_to_file

function write_estcont_to_file(t, CONT_QBoiler, CONT_RRatioinv, EST_Xp, EST_Pp); Glob_Decs;
% Controller Outputs file l = prod(size(CONT_QBoiler)) + prod(size(CONT_RRatioinv)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_control, ['%9.4f ' formats '
'], t, CONT_QBoiler, CONT_RRatioinv);

% Estimator Outputs file l = prod(size(EST_Xp)) + prod(size(EST_Pp)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_estimator, ['%9.4f ' formats '
'], t, reshape(EST_Xp, 1, prod(size(EST_Xp))));
end write_estcont_to_file

function write_tank_to_file(t, X_actual, M_actual, X_est, M_est); Glob_Decs;
% Tank data file l = prod(size(X_actual)) + prod(size(M_actual)) + prod(size(X_est)) + prod(size(M_est)); formats = ' %f'; for i=1:l; formats = [formats '; %f']; end;
fprintf(FID_tank, ['%9.4f ' formats '
'], t, reshape(X_actual, 1, prod(size(X_actual))));
end write_tank_to_file
D.1.4 PressureProfile.m

% PressureProfile
% Return pressure profile through the column (Pa)
% for a given Still pot and reflux drum pressures (Pa)

function [fP_Tray] = PressureProfile(fP_Pot, fP_Drum)

Glob_decs;

if (size(fP_Pot) ~= 1 | size(fP_Drum) ~= 1)
    error('fP_Pot and/or fP_Drum are not scalar(s). (PressureProfile)');
end;

fP_Tray = zeros(NT, 1);

fdelP_Tray = (fP_Pot - fP_Drum) / NT;

for i = 1:NT;
    fP_Tray(i) = fP_Pot - i * fdelP_Tray;
end;

end PressureProfile

D.2 Thermodynamic Library MATLAB Interface Code

D.2.1 thermo_Init.m

function thermo_Init(check_input_parameters)

% function thermo_Init(check_input_parameters)
% Thermophysical and physical property calculation MEX File Interface
% ...

thermo_LIBRARY('init', check_input_parameters);

D.2.2 thermo_Equilibrium.m

function [Tequi, y] = thermo_Equilibrium(T, P, x)

% function [Tequi, y] = thermo_Equilibrium(T, P, x)
% Thermophysical and physical property calculation MEX File Interface
% ...

[Tequi, y] = thermo_LIBRARY('equilibrium', T, P, x);

D.2.3 thermo_Enthalpy.m

function [hl, hv] = thermo_Enthalpy(T, P, x, y)

% function [hl, hv] = thermo_Enthalpy(T, P, x, y)
% Thermophysical and physical property calculation MEX File Interface
% ...

[hl, hv] = thermo_LIBRARY('enthalpy', T, P, x, y);

D.2.4 thermo_Density.m

function [mwa, densa] = thermo_Density(T, P, x)

% function [mwa, densa] = thermo_Density(T, P, x)
% Thermophysical and physical property calculation MEX File Interface
% ...

[mwa, densa] = thermo_LIBRARY('density', T, P, x);

D.3 Thermodynamic Library Fortran DLL Code

D.3.1 thermo_LIBRARY.f
```fortran
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
  include 'thermo_LIBRARY.h'
  include 'parameter.h'
  integer plhs(*), prhs(*) ! pointer to left-hand and right-hand side variables
  integer nlhs, nrhs ! # of variables in plhs, prhs
  integer mxCreateFull, mxGetString ! mx Functions declarations
  integer mxGetM, mxGetN, mxIsNumeric, mxIsString ! mx Functions declarations
  integer m, n, size, status, alloc_err ! Dummy variables
  integer Func_name_ptr ! Function name fortran pointers
  character*100 Func_name ! Function name for fortran use

  ! Input fortran pointers
  integer Input1_pr, Input2_pr, Input3_pr, Input4_pr, Input5_pr, Input6_pr, x_pr, y_pr, z_pr

  ! Output fortran pointers
  integer Output1_pr, Output2_pr, Output3_pr, Output4_pr, Output5_pr, Output6_pr

  ! Input arguments for fortran use
  integer, allocateble, dimension (:) :: int_Input1, int_Input2, int_Input3, int_Input4, int_Input5, int_Input6
  real*8, allocateble, dimension (:) :: real_Input1, real_Input2, real_Input3, real_Input4, real_Input5, real_Input6
  integer Input1_sz, Input2_sz, Input3_sz, Input4_sz, Input5_sz, Input6_sz

  ! Output arguments for fortran use
  integer, allocateble, dimension (:) :: int_Output1, int_Output2, int_Output3, int_Output4, int_Output5, int_Output6
  real*8, allocateble, dimension (:) :: real_Output1, real_Output2, real_Output3, real_Output4, real_Output5, real_Output6
  integer Output1_sz, Output2_sz, Output3_sz, Output4_sz, Output5_sz, Output6_sz

  real*8 x, y(3,3), z(3,3), ugur

  ! ----------------------------------------------------- Check
  ! for at least one function is requested.
  if (nrhs .lt. 1) then
    call mexErrMsgTxt('Not a proper function selected. - [thermo_LIBRARY.dll]')
  endif

  ! Check if mxGetString is successful.
  if (mxGetM(prhs(1)) .ne. 1) then
    call mexErrMsgTxt('Function name parameter must be a row vector. - [thermo_LIBRARY.dll]')
  endif

  ! Call the requested function. C Get the string contents (dereference the input integer).
  status = mxGetString(prhs(1), Func_name, 100)
  if (status .ne. 0) then
    call mexErrMsgTxt('String length must be less than 100. - [thermo_LIBRARY.dll]')
  endif

  ! Call initialization function
  if (Func_name.eq.'init') then
    if (nrhs .lt. 1) then
      call mexErrMsgTxt('One input (number of components) is required for the initialization. - (init) [thermo_LIBRARY.dll]')
    endif

    ! Check if mxGetNumeric is successful.
    if (mxIsNumeric(prhs(2)) .ne. 1) then
      call mexErrMsgTxt('Input #1 is not a numeric. - (init) [thermo_LIBRARY.dll]')
    endif
  endif
```

if (status .ne. 0) then
    if (allocated(int_Input1)) deallocate(int_Input1)
call mexErrMsgTxt('Memory allocation error. - (init) [thermo_LIBRARY.dll]')
endif

call mxCopyPtrToInteger4(Input1_pr,int_Input1,1)
status = 1
call init(int_Input1,status)
if (status.eq.0) then
call mexPrintf('thermo_LIBRARY is initialized. - (init) [thermo_LIBRARY.dll]')
else
call mexErrMsgTxt('thermo_LIBRARY can not be initialized. - (init) [thermo_LIBRARY.dll]')
endif

! Call enthalpy function
elseif ((Func_name.eq.'enthalpy') .and. (lib_Inited.eq.1)) then
    if (nrhs .ne. 5) then
call mexErrMsgTxt('Four inputs (T,P,x,y) is required. - (enthalpy) [thermo_LIBRARY.dll]')
eendif
    if (mxIsNumeric(prhs(2)) .ne. 1) call mexErrMsgTxt('Input #1 is not a numeric. - (enthalpy) [thermo_LIBRARY.dll]')
    if (mxIsNumeric(prhs(3)) .ne. 1) call mexErrMsgTxt('Input #2 is not a numeric. - (enthalpy) [thermo_LIBRARY.dll]')
    if (mxIsNumeric(prhs(4)) .ne. 1) call mexErrMsgTxt('Input #3 is not a numeric. - (enthalpy) [thermo_LIBRARY.dll]')
    if (mxIsNumeric(prhs(5)) .ne. 1) call mexErrMsgTxt('Input #4 is not a numeric. - (enthalpy) [thermo_LIBRARY.dll]')

    m = mxGetM(prhs(2))
    n = mxGetN(prhs(2))
    Input1_sz = m*n
    if (n .ne. 1 .or. m .ne. 1) call mexErrMsgTxt('Input #1 is not a scalar. - (enthalpy) [thermo_LIBRARY.dll]')

    m = mxGetM(prhs(3))
    n = mxGetN(prhs(3))
    Input2_sz = m*n
    if (n .ne. 1 .or. m .ne. 1) call mexErrMsgTxt('Input #2 is not a scalar. - (enthalpy) [thermo_LIBRARY.dll]')

    m = mxGetM(prhs(4))
    n = mxGetN(prhs(4))
    Input3_sz = m*n
    if (n .ne. nj .or. m .ne. 1) call mexErrMsgTxt('Input #3 is not a NC-element row vector. - (enthalpy) [thermo_LIBRARY.dll]')

    m = mxGetM(prhs(5))
    n = mxGetN(prhs(5))
    Input4_sz = m*n
    if (n .ne. nj .or. m .ne. 1) call mexErrMsgTxt('Input #4 is not a NC-element row vector. - (enthalpy) [thermo_LIBRARY.dll]')

    Input1_pr = mxGetPr(prhs(2))
    Input2_pr = mxGetPr(prhs(3))
    Input3_pr = mxGetPr(prhs(4))
    Input4_pr = mxGetPr(prhs(5))
    plhs(1) = mxCreateFull(1,1,0)
    plhs(2) = mxCreateFull(1,1,0)
    Output1_pr = mxGetPr(plhs(1))
    Output2_pr = mxGetPr(plhs(2))

    status = 0
    allocate (real_Input1(1),STAT = alloc_err)
    status = status + alloc_err
    allocate (real_Input2(1),STAT = alloc_err)
    status = status + alloc_err
    allocate (real_Input3(nj),STAT = alloc_err)
    status = status + alloc_err
    allocate (real_Input4(nj),STAT = alloc_err)
    status = status + alloc_err
    allocate (real_Output1(1),STAT = alloc_err)
    status = status + alloc_err
    allocate (real_Output2(1),STAT = alloc_err)
    status = status + alloc_err
    if (status .ne. 0) then
        if (allocated(real_Input1)) deallocate(real_Input1)
        if (allocated(real_Input2)) deallocate(real_Input2)
        if (allocated(real_Input3)) deallocate(real_Input3)
        if (allocated(real_Input4)) deallocate(real_Input4)
        if (allocated(real_Output1)) deallocate(real_Output1)
        if (allocated(real_Output2)) deallocate(real_Output2)
call mexErrMsgTxt('Memory allocation error. - (enthalpy) [thermo_LIBRARY.dll]')
endif

call mxCopyPtrToReal8(Input1_pr,real_Input1,1)
call mxCopyPtrToReal8(Input2_pr,real_Input2,1)
call mxCopyPtrToReal8(Input3_pr,real_Input3,1)
call mxCopyPtrToReal8(Input4_pr,real_Input4,1)
call enth(real_Input1, real_Input2, real_Input3, real_Input4, real_Output1, real_Output2)
call mexReadToPtr(real_Output1,Output1_pr,1)
call mexReadToPtr(real_Output2,Output2_pr,1)

! Call density function
elseif ((Func_name.eq.'density') .and. (lib_Inited.eq.1)) then
    if (nrhs .ne. 4) then
call mexErrMsgTxt('Three inputs (T, P, x) is required. - (density) [thermo_LIBRARY.dll]')
else if (nlhs .ne. 2) then
call mexErrMsgTxt('Two outputs (Avg. mol. weight and density) are required. - (density) [thermo_LIBRARY.dll]')
endif
if (mxIsNumeric(prhs(2)) .ne. 1) call mexErrMsgTxt('Input #1 is not a numeric. - (density) [thermo_LIBRARY.dll]')
    m = mxGetM(prhs(2))
    n = mxGetN(prhs(2))
    Input1_sz = m*n
    if (n .ne. 1 .or. m .ne. 1) call mexErrMsgTxt('Input #1 is not a scalar. - (density) [thermo_LIBRARY.dll]')
    m = mxGetM(prhs(3))
    n = mxGetN(prhs(3))
    Input2_sz = m*n
    if (n .ne. 1 .or. m .ne. 1) call mexErrMsgTxt('Input #2 is not a scalar. - (density) [thermo_LIBRARY.dll]')
    m = mxGetM(prhs(4))
    n = mxGetN(prhs(4))
    Input3_sz = m*n
    if (n .ne. nj .or. m .ne. 1) call mexErrMsgTxt('Input #3 is not a NC-element row vector. - (density) [thermo_LIBRARY.dll]')
    Input1_pr = mxGetPr(prhs(2))
    Input2_pr = mxGetPr(prhs(3))
    Input3_pr = mxGetPr(prhs(4))
    plhs(1) = mxCreateFull(1,1,0)
    plhs(2) = mxCreateFull(1,1,0)
    Output1_pr = mxGetPr(plhs(1))
    Output2_pr = mxGetPr(plhs(2))
endif
if (status .ne. 0) then
allocate (real_Input1(1),STAT = alloc_err)
allocate (real_Input2(1),STAT = alloc_err)
allocate (real_Input3(nj),STAT = alloc_err)
allocate (real_Output1(1),STAT = alloc_err)
allocate (real_Output2(1),STAT = alloc_err)
if (status .ne. 0) then
if (allocated(real_Input1)) deallocate(real_Input1)
if (allocated(real_Input2)) deallocate(real_Input2)
if (allocated(real_Input3)) deallocate(real_Input3)
if (allocated(real_Output1)) deallocate(real_Output1)
if (allocated(real_Output2)) deallocate(real_Output2)
call mexErrMsgTxt('Memory allocation error. - (density) [thermo_LIBRARY.dll]')
endif
endif
c subroutine pr_dens(t,p,x,mwa,densa)
call pr_dens(real_Input1, real_Input2, real_Input3, real_Output1, real_Output2)
call mxCopyReal8ToPtr(real_Output1,Output1_pr,1)
call mxCopyReal8ToPtr(real_Output2,Output2_pr,1)
c subroutine pr_dens(t,p,x,mwa,densa)
call pr_dens(real_Input1, real_Input2, real_Input3, real_Output1, real_Output2)
call mxCopyReal8ToPtr(real_Output1,Output1_pr,1)
call mxCopyReal8ToPtr(real_Output2,Output2_pr,1)
c subroutine pr_dens(t,p,x,mwa,densa)
call pr_dens(real_Input1, real_Input2, real_Input3, real_Output1, real_Output2)
call mxCopyReal8ToPtr(real_Output1,Output1_pr,1)
call mxCopyReal8ToPtr(real_Output2,Output2_pr,1)
c subroutine pr_dens(t,p,x,mwa,densa)
call pr_dens(real_Input1, real_Input2, real_Input3, real_Output1, real_Output2)
call mxCopyReal8ToPtr(real_Output1,Output1_pr,1)
call mxCopyReal8ToPtr(real_Output2,Output2_pr,1)
Output2_pr = mxGetPr(plhs(2))

status = 0
allocate (real_Input1(),STAT = alloc_err)
status = status + alloc_err
allocate (real_Input2(),STAT = alloc_err)
status = status + alloc_err
allocate (real_Input3(nj),STAT = alloc_err)
status = status + alloc_err
allocate (real_Output1(),STAT = alloc_err)
status = status + alloc_err
allocate (real_Output2(nj),STAT = alloc_err)
status = status + alloc_err
if (status .ne. 0) then
  if (allocated(real_Input1)) deallocate(real_Input1)
  if (allocated(real_Input2)) deallocate(real_Input2)
  if (allocated(real_Input3)) deallocate(real_Input3)
  if (allocated(real_Output1)) deallocate(real_Output1)
  if (allocated(real_Output2)) deallocate(real_Output2)
call mexErrMsgTxt('Memory allocation error. - (equilibrium) [thermo_LIBRARY.dll]\')
endif

c subroutine pr_equil(t,p,x,yy) ' t is also an output
call pr_equil(real_Input1, real_Input2, real_Input3, real_Output2)
real_Output1 = real_Input1
call mxCopyReal8ToPtr(real_Output1,Output1_pr,1)
call mxCopyReal8ToPtr(real_Output2,Output2_pr,nj)
c
subroutine init(check_input,st)
integer check_input, st
include 'thermo_LIBRARY.h'
include 'parameter.h'
include 'common_plant.h'
C ------ Initialization of the 'plant' common statement in 'common_plant.h' ------ written by MTD
(Revised by Uur Yldz)
integer :: i,j,k,l,m,n,
integer :: thermo_LIBRARY_dummy_pr, thermo_LIBRARY_dummy_pi
C tolerance = 1.d-7
open(5, file='thermo_data.dat', IOSTAT=I_O_err, ERR = 100)
read(5, *)
read(5, *) tolerance
read(5, *)
do i=1,nj
read(5, *) mw(i), tc(i), tboil(i), pc(i), wc(i)
enddo
read(5, *)
read(5, *)
do i=1,nj
read(5, *) (del(i,j), j=1,nj)
enddo
read(5, *)
read(5, *)
do i=1,nj
read(5, *) cenhi(i), cenh2(i), cenh3(i), cenh4(i)
enddo
close(5)
C
if (check_input .eq. 1) then
  open(6, file='thermo_data_check.dat', IOSTAT=I_O_err, ERR = 100)
write(6, *) 'tolerance'
write(6, 1) tolerance
write(6, *)
write(6, *) 'Mw(kg/mol) Tc(K) Tboil(K) Pc(Pa) w'
do i=1,nj
write(6, 2) mw(i), tc(i), tboil(i), pc(i), wc(i)
enddo
write(6, *)
write(6, *) 'del(binary interaction parameters)'
do i=1,nj
write(6, 3) (del(i,j), j=1,nj)
enddo
write(6, *)
write(6, *) ' cenh1 cenh2 cenh3 cenh4(J/molK)'
do i=1,nj
write(6, 4) cenh1(i), cenh2(i), cenh3(i), cenh4(i)
enddo
close(6)
endif
lib_Inited = 1
st = 0
return
100 if (I_O_err.ne.0) then
  call mexErrMsgTxt('"thermo_data.dat" couldn't be opened. - (init) [thermo_LIBRARY.dll]')
  lib_Inited = 0
  st = 1
  return
endif
1 format(d11.3) 2 format(5d15.3) 3 format(4d15.3)
end subroutine
C
C Write statements in these routines are exchanged with mexErrMsgTxt and mexPrintf c and also 'parameter.h' and 'plant_data.dat' are modified. C
C
C Peng-Robinson EOS Subroutines Written by Mustafa T. DOKUCU C
Date : 16-05-2001 C
C
C c
C subroutine enth(t,p,x,y,hl,hv)
!Usage: ! to calculate the ideal gas mixture enthalpy !Record of revisions: ! date programmer description of change ! 18/03/2001 MTD original code !
! implicit none ! include 'parameter.h' ! include 'common_plant.h' !
! ! Inputs ! real*8 :: t ! temperature
real*8 :: p ! pressure
real*8 :: x(nj) ! liquid phase fractions
real*8 :: y(nj) ! vapour phase fractions
!
! Locals
! ========
real*8 :: hl1 ! ideal liquid mixture enthalpy
real*8 :: hv1 ! ideal vapour mixture enthalpy
real*8 :: dhhl ! liquid enthalpy departure
real*8 :: enigl ! ideal gas enthalpy
real*8 :: enigv ! ideal gas enthalpy
real*8 :: cl1,cv1
real*8 :: cl2,cv2
real*8 :: cl3,cv3
real*8 :: cl4,cv4
!
! Outputs
! =========
real*8 :: hl ! liquid enthalpy
real*8 :: hv ! vapour enthalpy
cl1 = 0.d0
cl2 = 0.d0
cl3 = 0.d0
cl4 = 0.d0
cv1 = 0.d0
cv2 = 0.d0
cv3 = 0.d0
cv4 = 0.d0
do i = 1,nj
  cl1 = cl1 + cenh1(i) * x(i)
  cl2 = cl2 + cenh2(i) * x(i)
  cl3 = cl3 + cenh3(i) * x(i)
  cl4 = cl4 + cenh4(i) * x(i)
  cv1 = cv1 + cenh1(i) * y(i)
  cv2 = cv2 + cenh2(i) * y(i)
  cv3 = cv3 + cenh3(i) * y(i)
  cv4 = cv4 + cenh4(i) * y(i)
endo
enigl = cl1 * (t-trf) +
  & * (1.d0/2.d0) * cl2 * (t**2 - trf**2)
  & + (1.d0/3.d0) * cl3 * (t**3 - trf**3)
  & + (1.d0/4.d0) * cl4 * (t**4 - trf**4)
enigv = cv1 * (t-trf) +
  & * (1.d0/2.d0) * cv2 * (t**2 - trf**2)
  & + (1.d0/3.d0) * cv3 * (t**3 - trf**3)
  & + (1.d0/4.d0) * cv4 * (t**4 - trf**4)
ifase = 0
  call pr_enth(t,p,x,ifase,dhl)
ifase = 1
  call pr_enth(t,p,y,ifase,dhv)
hl = enigl + dhl + 20000.d0
hv = enigv + dhv + 20000.d0
return
end subroutine

-------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
! Locals
! ========
complex*8:: z_vap_cplx
complex*8:: z_liq_cplx

---------------------------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
! Locals
! ========
complex*8:: z_vap_cplx
complex*8:: z_liq_cplx

-------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
! Locals
! ========
complex*8:: z_vap_cplx
complex*8:: z_liq_cplx

-------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
! Locals
! ========
complex*8:: z_vap_cplx
complex*8:: z_liq_cplx

-------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
! Locals
! ========
complex*8:: z_vap_cplx
complex*8:: z_liq_cplx

-------------------------------------------------------------------
subroutine pr_compr(a_mixture,b_mixture,z_liq,z_vap)
!
! Usage:
! to solve the cubic equation for the liquid and vapor compressibility factors used for the estimation of species fugacities
! Record of revisions:
! date programmer description of change
! ==== ========== ================
! 14/02/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: a_mixture
real*8 :: b_mixture
!
complex*8:: s1
complex*8:: a
complex*8:: b

! Outputs
! =========
real*8:: z_vap
real*8:: z_liq

! convert the type declaration of the input variables to complex
a = cmplx(a_mixture,0.d0)
b = cmplx(b_mixture,0.d0)

! calculate the liquid phase compressibility
s1 = -(-36.0d0*a+144.0d0*a*b-48.0d0*b**2-224.0d0*b**3+48.0d0*b+8.0d0+12.0d0*sqrt(24.0d0*a*b-24.0d0*b**2-192.0d0*b**3+264.0d0*a*b**2-3.0d0*a**2+24.0d0*a**2*b**2-120.0d0*a**2*b-48.0d0*a*b**4+336.0d0*a*b**3-480.0d0*b**4+12.0d0*a**3-96.0d0*b**6-384.0d0*b**5))**(1.d0/3.d0)/12.0d0+(a-10.d0/3.d0)*b**2-4.d0/3.d0*B-1.d0/3.d0)/(-36.0d0*a+144.0d0*a*b-48.0d0*b**2-224.0d0*b**3+48.0d0*b+8.0d0+12.0d0*sqrt(24.0d0*a*b-24.0d0*b**2-192.0d0*b**3+264.0d0*a*b**2-3.0d0*a**2+24.0d0*a**2*b**2-120.0d0*a**2*b-48.0d0*a*b**4+336.0d0*a*b**3-480.0d0*b**4+12.0d0*a**3-96.0d0*b**6-384.0d0*b**5))**(1.d0/3.d0)

z_liq_cplx = s1+1.d0/3.d0-b/3.d0+cmplx(0.d0,1.d0)*sqrt(3.d0)*((-36.d0*a+144.d0*a*b-48.d0*b**2-224.d0*b**3+48.d0*b+8.d0+12*d0*sqrt(24.d0*a*b-24.d0*b**2-192.d0*b**3+264.d0*a*b**2-3.d0*a**2+24.d0*a**2*b**2-120.d0*a**2*b-48.d0*a*b**4+336.d0*a*b**3-480.d0*b**4+12.d0*a**3-96.d0*b**6-384.d0*b**5))**(1.d0/3.d0)/6.d0+(2.d0*a-20.d0/3.d0)*b**2-8.d0/3.d0)/(-36.d0*a+144.d0*a*b-48.d0*b**2-224.d0*b**3+48.d0*b+8.d0+12.d0*sqrt(24.d0*a*b-24.d0*b**2-192.d0*b**3+264.d0*a*b**2-3.d0*a**2+24.d0*a**2*b**2-120.d0*a**2*b-48.d0*a*b**4+336.d0*a*b**3-480.d0*b**4+12.d0*a**3-96.d0*b**6-384.d0*b**5))**(1.d0/3.d0)

! calculate the vapor phase compressibility
z_vap_cplx =(-36.d0*a+144.d0*a*b-48.d0*b**2-224.d0*b**3+48.d0*b+8.d0+12.d0*sqrt(24.d0*a*b-24.d0*b**2-192.d0*b**3+264.d0*a*b**2-3.d0*a**2+24.d0*a**2*b**2-120.d0*a**2*b-48.d0*a*b**4+336.d0*a*b**3-480.d0*b**4+12.d0*a**3-96.d0*b**6-384.d0*b**5))**(1.d0/3.d0)/6.d0-(2.d0*a-20.d0/3.d0)*b**2-8.d0/3.d0)/(-36.d0*a+144.d0*a*b-48.d0*b**2-224.d0*b**3+48.d0*b+8.d0+12.d0*sqrt(24.d0*a*b-24.d0*b**2-192.d0*b**3+264.d0*a*b**2-3.d0*a**2+24.d0*a**2*b**2-120.d0*a**2*b-48.d0*a*b**4+336.d0*a*b**3-480.d0*b**4+12.d0*a**3-96.d0*b**6-384.d0*b**5))**(1.d0/3.d0)+1.d0/3.d0-b/3.d0

! there is no liquid phase if the liquid compressibility root is a complex 
! in this case the compressibility root returned as equal to vapor phase
! compressibility
if (aimag(z_liq_cplx) > tolerance) then
  z_liq = real(z_vap_cplx)
else
  z_liq = real(z_liq_cplx)
endif

! the root found for the vapor compressibility is erroneous if it is
! a complex # in this case the compressibility root returned as zero to
! the mainprogram
if (aimag(z_vap_cplx) > tolerance) then
  call mexPrintf('vapor phase compressibility can not be calculated. - (pr_compr) [thermo_LIBRARY.dll]
)
  c write(*,*) 'vapor phase compressibility can not be calculated'
z_vap = 0.d0
else
  z_vap = real(z_vap_cplx)
endif
return
end subroutine

---------------------------------------------------------------------------------------
subroutine pr_cons(t,a,aij,b)
!
! Usage:
! to calculate the constants A and B of the
! Peng-Robinson EOS which is explained in p239 (Sandler)
! Record of revisions:
! date programmer description of change
! ==== ========== =====================
! 12/03/2001 MTD original code
!
!
! implicit none
! include 'parameter.h'
! include 'common_plant.h'
!
!
! Inputs
! ========
real*8:: t !temperature [K]
!
! Locals
! ========
real*8 :: ac(nj) !constant
real*8 :: xk !constant
real*8 :: alsqr !constant
real*8 :: alpha !constant
!
! the main program
if (aimag(z_vap_cplx) > tolerance) then
c write(*,*) 'vapor phase compressibility can not be calculated' write(6,*) 'vapor phase compressibility can not be calculated'
z_vap = 0.d0
else
  z_vap = real(z_vap_cplx)
endif
return
end subroutine pr_cons(t,a,aij,b)
integer:: j
!
! Outputs
!
real*8 :: a(nj)  !a of the species
real*8 :: b(nj)  !b of the species
real*8 :: aij(nj,nj)  !interacting a's of the species

do i=1,nj
   ac(i) = 0.457235529d0 * ((rg * tc(i))**2) / pc(i) !eqn 4.7-1(first part)
   b(i) = 7.779607650000000d-2 * rg * tc(i) / pc(i) !eqn 4.7-2
   xk = 0.37464d0 + (1.54226d0 - 0.26992d0 * wc(i)) * wc(i) !eqn 4.7-4
   tr = t / tc(i)
   alorg = 1.d0 + xk * (1.d0 - dsqrt(tr))
   alpha = alorg * alorg !eqn 4.7-3
   a(i) = alpha * ac(i) !eqn 4.7-1(whole)
enddo

return
end subroutine

---------------------------------------------------------------------------------------
subroutine pr_dens(t,p,x,mwa,densa)
!
!Usage:
! to calculate the average molecular weight and the
! density of the liquid phase using Peng-Robinson EOS
!Record of revisions:
! date programmer description of change
! ==== ========== =====================
! 25/03/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! =======
real*8 :: t
real*8 :: p
real*8 :: x(nj)
!
! Locals
! ======
real*8 :: aa
real*8 :: bb
real*8 :: ca
real*8 :: cb
real*8 :: a(nj)
real*8 :: b(nj)
real*8 :: zx(nj)
real*8 :: aij(nj,nj)
real*8 :: zz
real*8 :: vv
integer:: i
integer:: j
!
! Outputs
! ========
real*8 :: mwa
real*8 :: densa
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!


aa = aa + zx(i) * zx(i) * a(i)
else
aa = aa + zx(i) * zx(j) * aij(i,j)
endif
enddo
endo
do i = 1,nj
bb = bb + zx(i) * b(i)
do j = 1,nj
if (i == j) then
aa = aa + zx(i) * zx(i) * a(i)
else
aa = aa + zx(i) * zx(j) * aij(i,j)
endif
endo
ca = aa * p / ((rg * t)**2)
cb = bb * p / (rg * t)
call pr_compr(ca,cb,z_liq,z_vap)
zz = z_liq
vv = zz * rg * t / p
densa = mwa / vv
return
end subroutine

--------------------------------------------------------------------------------------
subroutine pr_enth(t,p,zx,ifase,dh)
!
!Usage:
!to calculate the enthalpy departure of a mixture
!as explained in Sandler p425
!Peng-Robinson EOS is explained in p239
!
!Record of revisions:
! date programmer description of change
! ==== ========== =======================
!12/03/2001 MTD original code
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
! Inputs
! ========
real*8 :: t
real*8 :: p
real*8 :: zx(nj)
!
! Locals
! ========
real*8 :: zz
real*8 :: a(nj)
real*8 :: b(nj)
real*8 :: xk(nj)
real*8 :: aij(nj,nj)
real*8 :: c1,c2,c3,c4,c5,c6,c7,c8,c9
real*8 :: a1,a2,a3,a4
real*8 :: anum,aden
real*8 :: cnum,cden
real*8 :: damdt
real*8 :: aa
real*8 :: ca
real*8 :: bb
real*8 :: cb
real*8 :: z_liq
real*8 :: z_vap
real*8 :: tr(nj)
real*8 :: dh0,dh1,dh2
integer:: i
integer:: j
integer:: ifase
!
! Outputs
!=========
real*8:: dh
!
call pr_cons(t,a,aij,b)
aa = 0.d0
bb = 0.d0
do i = 1,nj
bb = bb + zx(i) * b(i)
do j = 1,nj
if (i == j) then
aa = aa + zx(i) * zx(i) * a(i)
else
aa = aa + zx(i) * zx(j) * aij(i,j)
endif
endo
do i = 1,nj
ca = aa * p / ((rg * t)**2)
do j = 1,nj
if (i == j) then
ca = ca + bb * p / ((rg * t)**2)
call pr_compr(ca,cb,z_liq,z_vap)
if (ifase == 0) then
zz = z_liq
else
zz = z_vap
endif

do \( i = 1, nj \)
\[
\text{xk}(i) = 0.37464d0 + (1.54226d0 - 0.26992d0 \times wc(i)) \times wc(i) \quad \text{!eqn 4.7-4}
\]
enddo

damdt = 0.0d0

do \( i = 1, nj \)
do \( j = 1, nj \)
\[
tr(i) = t / tc(i)
\]
\[
tr(j) = t / tc(j)
\]
c1 = (-0.457235529d0 / 2.d0) \times (-1 + del(i,j)) \times rg**(4)
c2 = tc(i) \times (-1.d0 - xk(i) + xk(i) \times dsqrt(tr(i)))
c3 = tc(j) \times (-1.d0 - xk(j) + xk(j) \times dsqrt(tr(j)))
c4 = -tc(j) \times xk(i) \times dsqrt(tr(j))
c5 = 2.d0 \times xk(i) \times t \times xk(j)
c6 = -tc(i) \times xk(j) \times dsqrt(tr(i)) \times xk(i)
c7 = (-1.d0 - xk(i) + xk(i) \times dsqrt(tr(i)))**2
\]
\[
c8 = (-1.d0 - xk(j) + xk(j) \times dsqrt(tr(j)))**2 / pc(i) / pc(j)
\]
c9 = pc(i) \times pc(j) \times dsqrt(tr(i)) \times dsqrt(tr(j))
cnum = c4 + c5 + c6
cden = dsqrt(rg**4 * tc(i)**2 * c7 * tc(j)**2 * c8)
\[
damdt = damdt + zx(i) \times zx(j) \times (c1 \times c2 \times c3 \times cnum / cden)
\]
enddo

enddo

a1 = dsqrt(2.d0)
a2 = a1 + 1.d0
a3 = a1 - 1.d0
a4 = a1 \times 2.d0
anum = zz + (a2 \times cb)
aden = zz - (a3 \times cb)
dh0 = (rg \times t) \times (zz -1.d0)
dh1 = (t*damdt - aa)/a4/bb
dh2 = dlog(anum/aden)
dh = (dh0 + dh1*dh2)

return

c

!-------------------------------------------------------------------
! subroutine pr_equil(t,p,x,yy)
! ! Usage:
! ! to calculate the bubble point temperature using
! ! Peng-Robinson EDS similar to VLMU.BAS of Sandler
! ! Record of revisions:
! ! date programmer description of change
! ! ===== ======== ============
! ! 12/03/2001 MTD original code
! ! 12/06/2001 UGUR to be able to find equilibrium staff
! ! some checks were performed before calculation.
! ! Implicit none
! ! include 'parameter.h'
! include 'common_plant.h'
!
! Inputs
! ========
! \( t \) \quad \text{! t is also an output}
real*8:: t
real*8:: p
real*8:: x(nj)
!
! Locals
! ========
real*8 :: s(2),sum_xumy
real*8 :: dt1,dt2
real*8 :: dlnp
real*8 :: ps(nj)
real*8 :: xk(nj)
real*8 :: a(nj)
real*8 :: aij(nj,nj)
real*8 :: b(nj)
real*8 :: zx(nj)
real*8 :: fugacity(nj)
real*8 :: f1(nj),f2(nj)
real*8 :: xx1,xx2
real*8 :: yl(nj),y2(nj)
real*8 :: yk
real*8 :: test,tttest
real*8 :: ddadt,ddt,dd
real*8 :: neg_dd,neg_dlt
real*8 :: tbg,tcg
integer:: i
integer:: j
integer:: k
integer:: nc
integer:: kkk
integer:: nloop
integer:: iconv

integer:: ifase
integer:: itest
integer:: kvalue
logical:: reguess
!
! Outputs
! =========
real*8 yy(nj)
integer:: comp_index(nj)

common /nc/ nc
!
! zero component check
j=0
do i=1,nj
  if (x(i).gt.0.0d0) then
    j=j+1
    comp_index(j) = i
  else
    yy(i) = 0.0d0
  endif
end do
nc = j
k = 0
kvalue = 0
iconv = 0
sum = 0.d0
reguess = .false.
!
do i = 1,nc
  sum = sum + x(comp_index(i))
enddo
!
do i = 1,nc
  x(comp_index(i)) = x(comp_index(i))/sum
enddo
!
!Initial guess procedure for Ki = yi / xi and yi
3870 if (k > 30) then
  call mexErrMsgTxt('too many calculations. - (pr_equil) [thermo_LIBRARY.dll]')
c write(*,*) 'too many calculations'
goto 4880
endif
!
do i = 1,nc
  if (kvalue == 1) then
    yy(comp_index(i)) = x(comp_index(i)) * xk1(comp_index(i))
  else
    dt1 = (1.d0 / t) - (1.d0 / tboil(comp_index(i)))
dt2 = (1.d0 / tc(comp_index(i))) - (1.d0 / tboil(comp_index(i)))
dlnp = dlog(pc(comp_index(i)))
ps(comp_index(i)) = dexp(dlnp * dt1 / dt2)
xk1(comp_index(i)) = ps(comp_index(i)) / p
    yy(comp_index(i)) = x(comp_index(i)) * xk1(comp_index(i))
  endif
enddo
!
!Calculation of liquid fugacities
kkk = 0
3980 nloop = 1
!
if (t < 50.d0) goto 4590
if (t > 1200.d0) goto 4590
!
!
do i = 1,nc
  zx(comp_index(i)) = x(comp_index(i))
enddo
!
!phase liquid 1
ifase = 1
!call pr_fuga
!
!
do i = 1,nc
  if (comp_index(i)) = fugacity(comp_index(i))
enddo
!
!
!
!call pr_cons
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
do i = 1,nc
  yy(comp_index(i)) = yy(comp_index(i)) / sumy
enddo
4160  sumy = 0.d0 4170  kkk = kkk + 1
do i = 1,nc
  zx(comp_index(i)) = yy(comp_index(i))
enddo
!phase vapor 0
ifase = 0
!call pr_fuga {UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}
call pr_fuga(t,p,ifase,zx,zz,a,aij,b,fugacity)
!{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}{UGUR}
doi = 1,nc
  f2(comp_index(i)) = fugacity(comp_index(i))
enddo
zz2 = zz
do i = 1,nc
  yy(comp_index(i)) = f1(comp_index(i)) * yy(comp_index(i)) / f2(comp_index(i))
  y1(comp_index(i)) = yy(comp_index(i))
  sumy = sumy + yy(comp_index(i))
enddo
ittest = 0
do i = 1,nc
  test = dabs(y1(comp_index(i)) - y2(comp_index(i)))
  if (test > tolerance) then
    itest = itest + 1
  endif
  yy(comp_index(i)) = yy(comp_index(i)) / sumy
enddo
if (kkk > 25) goto 4400
if (itest <= 0) goto 4400
do i = 1,nc
  y2(comp_index(i)) = y1(comp_index(i))
enddo
goto 4160
4400  s(nloop) = sumy
       kkk = 0
if ((nloop - 1) > 0) goto 4460
t = t - 0.005d0
goto 4020
4460  dsdt = (s(2) - s(1)) / (0.005d0)
if (dabs(dsdt) < tolerance) goto 4590
dlt = (s(1) - 1.d0) / dsdt
if (dabs(dlt) < 0.0026) goto 4670
if (k > 50) goto 4650
if (k < 11) then
  dd = 20.d0
endif
if (k >= 11) then
  dd = 5.d0
endif
if (dlt > dd) then
  t = t + dd
endif
if (dlt > dd) goto 3980
neg_dd = -1.d0 * dd
if (dlt < neg_dd) then
  t = t - dd
endif
neg_dlt = -1.d0 * dlt
if (neg_dlt > dd) goto 3980
t = t + dlt + 0.0025
goto 3980
4590  if (k > 2) goto 4630
if (zz1 >= 0.307d0) then
  t = t - 10.d0
endif
if (zz1 <= 0.307d0) then
  t = t + 10.d0
endif
goto 3980
4630 call mexErrMsgTxt('not converging: one-phase region or poor initial guess. - (pr_equil) [thermo_LIBRARY.dll]')
write(*,*) 'not converging: one-phase region or poor initial guess'
if (reguess == .false.) then
  reguess = .true.
  tbg = 0.d0
  tcg = 0.d0
  do i = 1,nc
    tbg = tbg + x(comp_index(i)) * tboil(comp_index(i))
    tcg = tcg + x(comp_index(i)) * tc(comp_index(i))
  enddo
  t = 0.5d0 * (tbg + tcg)
call mexPrintf('it should print "t" here. - (pr_equil) [thermo_LIBRARY.dll]')
c write(*,*) t
goto 3870
else
goto 4880
endif
4650 call mexErrMsgTxt('bubble point temperature calculation did not converge. - (pr_equil) [thermo_LIBRARY.dll]')
c write(*,*) 'bubble point temperature calculation did not converge'
goto 4880
4670 yk = k
test = (zz1 - zz2)**2
if (ttest > tolerance) goto 4730
if (zz1 >= 0.307d0) then
  t = t - 25.d0 / dsqrt(yk)
endif
if (zz1 < 0.307d0) then
  t = t + 25.d0 / dsqrt(yk)
endif
kvalue = 0
goto 3870
4730 do i = 1,nc
  xk1(comp_index(i)) = yy(comp_index(i)) / x(comp_index(i))
endo
return
call mexErrMsgTxt('unsuccessful bubble point calculation. - (pr_equil) [thermo_LIBRARY.dll]')
c write(*,*) 'unsuccessful bubble point calculation'
end subroutine

c---------------------------------------------------------------------------------------
subroutine pr_fuga(t,p,ifase,zx,zz,a,aij,b,fugacity)
!
! Usage:
! to calculate the species fugacity f(T,P,xi)
! as explained in Sandler p409
! Peng-Robinson EOS is explained in p239
! Record of revisions:
! date programmer description of change
! === ======= ===============
! 12/03/2001 MTD original code
! 12/06/2001 UGUR to be able to find equilibrium staff
! ! when a zero-fraction component exist.
! ! some checks were performed before calculation.
!
implicit none
include 'parameter.h'
include 'common_plant.h'
!
Inputs
!
real*8 :: t
real*8 :: p
real*8 :: zz(nj)
real*8 :: a(nj)
real*8 :: b(nj)
real*8 :: aij(nj,nj)
integer:: ifase
!
Locals
!
real*8 :: c1
real*8 :: c2
real*8 :: c3
real*8 :: sa(nj)
real*8 :: aa
real*8 :: bb
real*8 :: cb
real*8 :: ca
real*8 :: zz
real*8 :: z_liq
real*8 :: z_vap
real*8 :: fox(nj)
real*8 :: ag1
real*8 :: ag2
real*8 :: ag3
integer:: nc
integer:: i
integer:: j
!
! Outputs
! =========
real*8 :: fugacity(nj)
integer:: comp_index(nj)

common /nc/ nc

! zero component check
j=0
do i=1,nj
  if (zx(i) .gt. 0.d0) then
    j=j+1
    comp_index(j) = i
  endif
end do
nc = j

c1 = dsqrt(2.d0)
c2 = 1.d0 + c1
c3 = c1 - 1.d0

do i = 1,nc
  sa(comp_index(i)) = 0.d0
enddo
aa = 0.d0
bb = 0.d0

do i = 1,nc
  bb = bb + zx(comp_index(i)) * b(comp_index(i))
doi j = 1,nc
  if (i == j) then
    aa = aa + zx(comp_index(i)) * zx(comp_index(i)) * a(comp_index(i))
    sa(comp_index(j)) = sa(comp_index(j)) + zx(comp_index(j)) * a(comp_index(j))
  else
    aa = aa + zx(comp_index(i)) * zx(comp_index(j)) * aij(comp_index(i),comp_index(j))
    sa(comp_index(j)) = sa(comp_index(j)) + zx(comp_index(i)) * aij(comp_index(i),comp_index(j))
  endif
end do
ca = aa * p / ((rg*t)**2)
cb = bb * p / (rg*t)
call pr_compr(ca,cb,z_liq,z_vap)

if (ifase == 0) then
  zz = z_vap
else
  zz = z_liq
endif
ag1 = (zz + c2 * cb) / (zz - c3 * cb)
ag1 = dlog(ag1)
ag2 = ca / (2.d0 * cb * c1)
do i = 1,nc
  ag3 = (2.d0 * sa(comp_index(i)) / aa) - (b(comp_index(i)) / bb)
  fox(comp_index(i)) = (b(comp_index(i)) * (zz-1.d0) / bb)-dlog(zz - cb) - ag1 * ag2 * ag3
  fugacity(comp_index(i)) = zz*fox(comp_index(i))
  fugacity(comp_index(i)) = zz*fox(comp_index(i))
  enddo
return
end subroutine

D.3.2 thermo_LIBRARY.h

common /thermo_LIBRARY/ lib_Initd
integer :: lib_Initd ! Toggle for checking whether thermo_LIBRARY.dll is initialized.

D.3.3 common_plant.h

common /plant/
D.3.4 parameter.h

This parameters were modified as the common statement labeled as 'parameter'

integer ,parameter :: nj = 3 ! number of components
real*8 ,parameter :: rg = 8.313999999999999d0 ! ideal gas constant
real*8 ,parameter :: trf= 273.15d0 ! reference temperature

D.3.5 thermo_data.dat

tolerance (Component order: cyclohexane, n-Heptane, toluene)
1.000d-7

Mw(kg/mol) Tc(K) Tboil(K) Pc(Pa) w
84.141d-3 553.68d0 353.15d0 4.100d6 0.212d0
100.204d-3 540.20d0 371.55d0 2.736d6 0.346d0
92.141d-3 591.80d0 383.95d0 4.113d6 0.262d0

del(binary interaction parameters) ( for NC = 3 , 3x3 ; for NC = 4 , 4x4 )
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0

D.4 EKF Nonlinear Model Code

D.4.1 KF_NonlinearModel_Sys.m

function [KF_NonlinearModel_XDot, KF_NonlinearModel_YVapors, KF_NonlinearModel_Tempers] = KF_NonlinearModel_Sys(Compositions, StillPotHoldup, DistillateFlowrate, t, NC, NT, Vb, Holdups, Press, Antonies)
KF_NonlinearModel_XLiquids = zeros(NT+2,NC);
KF_NonlinearModel_Holdups = zeros(NT+2,1);
KF_NonlinearModel_YVapors = zeros(NT+2,NC);
KF_NonlinearModel_Tempers = zeros(NT+2,1);
KF_NonlinearModel_Presses = zeros(NT+2,1);
KF_NonlinearModel_D = zeros(1,1);
KF_NonlinearModel_Yb = zeros(1,1);
KF_NonlinearModel_NC = zeros(1,1);
KF_NonlinearModel_NT = zeros(1,1);
KF_NonlinearModel_AntoniesA = zeros(1,NC);
KF_NonlinearModel_AntoniesB = zeros(1,NC);
KF_NonlinearModel_AntoniesC = zeros(1,NC);

D.4 EKF Nonlinear Model Code
KF_NonlinearModel_GuessTemp = zeros(1,1);

% Time derivative of states in the order of Estimation States (Output)
KF_NonlinearModel_EDot = zeros(NT+2, NC);

% ------------------------ Distribute the function INPUTS to PHYSICAL Variables --
% Input States
% Compositions : [xSPot1, xSPot2, ... xSPotNC; x11, x12, ... x1NC; x21, x22, ... x2NC; ... xNT1, xNT2, ... xNTNC; xRDrum1, xRDrum2, ... xRDrumNC]
% StillPotHoldup : [ Hb ]
% Output State derivative order
% Estimation States : [xSPot1 xSPot2 ... xSPotNC; x11 x12 ... x1NC; x21 x22 ... x2NC; ... xNT1 xNT2 ... xNTNC; xRDrum1 xRDrum2 ... xRDrumNC]
KF_NonlinearModel_XLiquids = Compositions;
% Parameters : [ Hb ]
KF_NonlinearModel_Holdups(1,1) = StillPotHoldup;
% Inputs : [DistillateFlowrate]
KF_NonlinearModel_D = DistillateFlowrate;
% Parameters : # of components, # of trays, Vapor Boilup rate, [Tray Holups, RefluxDrum Holdup], [Reboiler Pressure, RefluxDrum Pressure], [AntoniesConstants: A1...ANC, B1...BNC, C1...CNC]
KF_NonlinearModel_NC = NC; KF_NonlinearModel_NT = NT;
KF_NonlinearModel_Vb = Vb; KF_NonlinearModel_Holdups(2:NT+2,1) = Holdups; KF_NonlinearModel_Presses(1,1) = Press(1);
KF_NonlinearModel_Presses(NT+2,1) = Press(2);
KF_NonlinearModel_AntoniesA = Antonies(1:NC);
KF_NonlinearModel_AntoniesB = Antonies(NC+1:2*NC);
KF_NonlinearModel_AntoniesC = Antonies(2*NC+1:3*NC);

% First Bubble Point Temperature Guess
KF_NonlinearModel_GuessTemp = 360.0;
% Pre-Calculations (New RefluxFlowrate, Pressures, Temperatures and Vapor compositions)
KF_NonlinearModel_R = KF_NonlinearModel_Vb - KF_NonlinearModel_D;
KF_NonlinearModel_Presses(2:NT+1,1) = KF_NonlinearModel_Press(NT, KF_NonlinearModel_Presses(1,1), KF_NonlinearModel_Presses(NT+2,1));
KF_NonlinearModel_Tempers(1:NT+2,1) = KF_NonlinearModel_Equilib(KF_NonlinearModel_GuessTemp, KF_NonlinearModel_XLiquids(1,:), KF_NonlinearModel_Presses(1), KF_NonlinearModel_AntoniesA, KF_NonlinearModel_AntoniesB, KF_NonlinearModel_AntoniesC);

% Time derivative of states in the order of Estimation States (Output)
KF_NonlinearModel_EDot = zeros(NT+2, NC);

% ------------------------ State Derivatives Calculation
% for Still Pot
KF_NonlinearModel_EDot(1,:) = ( KF_NonlinearModel_R * ( KF_NonlinearModel_XLiquids(2,:) - KF_NonlinearModel_XLiquids(1,:)) - KF_NonlinearModel_Vb * ( KF_NonlinearModel_YVapors(1,:) - KF_NonlinearModel_XLiquids(1,:)) ) / KF_NonlinearModel_Holdups(1);
end;
% for Trays
for i=2:NT+1;
    for j=1:NC;
        KF_NonlinearModel_EDot(i,j) = ( KF_NonlinearModel_Vb * ( KF_NonlinearModel_YVapors(i-1,j)- KF_NonlinearModel_YVapors(i,j)) + KF_NonlinearModel_R * ( KF_NonlinearModel_XLiquids(i+1,j)- KF_NonlinearModel_XLiquids(i,j)) ) / KF_NonlinearModel_Holdups(i);
    end;
end;
% for Drum
i = NT+2; for j=1:NC;
    KF_NonlinearModel_EDot(i,j) = ( KF_NonlinearModel_Vb * ( KF_NonlinearModel_YVapors(NT+1,j) - KF_NonlinearModel_XLiquids(i,j)) ) / KF_NonlinearModel_Holdups(NT+2);
end;

% ****************************
% Output States
% Still Pot Liquid compositions (State)
KF_NonlinearModel_XLiquids(1,:) = zeros(NT+2,NC);
% Trays Liquid compositions (State)
KF_NonlinearModel_XLiquids(2:NT+2,:) = zeros(NT+2, NC);
% Reflux Drum Liquid compositions (State)
KF_NonlinearModel_XLiquids(NT+2,:) = zeros(NT+2, NC);
% Still Pot Holdup (State)
KF_NonlinearModel_Holdups(1) = zeros(NT+2, 1);
% Trays Holdups (Parameter)
KF_NonlinearModel_Holdups(2:NT+2) = zeros(NT+2, 1);
% Reflux Drum Holdup (Parameter)
KF_NonlinearModel_Holdups(NT+2) = zeros(NT+2, 1);

D.4.2 KF_NonlinearModel_Out.m

function [KF_NonlinearModel_h] = KF_NonlinearModel_Out(Compositions, StillPotHoldup, DistillateFlowrate, t, NC, NT, Vb, Holdups, Press, Antonies, MeasurementOrder)
KF_NonlinearModel_XLiquids = zeros(NT+2, NC);
% Still Pot Liquid compositions (State)
KF_NonlinearModel_EDot(1,:) = zeros(NT+2, NC);
% Trays Liquid compositions (State)
KF_NonlinearModel_EDot(2:NT+2,:) = zeros(NT+2, NC);
% Reflux Drum Liquid compositions (State)
KF_NonlinearModel_EDot(NT+2,:) = zeros(NT+2, NC);
% Still Pot Holdup (State)
KF_NonlinearModel_Holdups(1) = zeros(NT+2, 1);
% Trays Holdups (Parameter)
KF_NonlinearModel_Holdups(2:NT+2) = zeros(NT+2, 1);
% Reflux Drum Holdup (Parameter)
KF_NonlinearModel_Holdups(NT+2) = zeros(NT+2, 1);
% Still Pot Vapor compositions (PreCalculation)
% Trays Vapor compositions (PreCalculation)
% Reflux Drum Vapor compositions (PreCalculation)
KF_NonlinearModel_YVapors = zeros(NT+2,NC);

% Still Pot Temperature (PreCalculation)
% Trays Temperatures (PreCalculation)
% Reflux Drum Temperature (PreCalculation)
KF_NonlinearModel_Tempers = zeros(NT+2,1);

% Still Pot Pressure (Parameter)
% Trays Pressures (PreCalculation)
% Reflux Drum Pressure (Parameter)
KF_NonlinearModel_Presses = zeros(NT+2,1);

% Distillate flowrate (Input)
KF_NonlinearModel_D = zeros(1,1);
% Reflux flowrate (PreCalculation)
KF_NonlinearModel_R = zeros(1,1);
% Vapor boilup flowrate (Parameter)
KF_NonlinearModel_Vb = zeros(1,1);
% # of components (Parameter)
KF_NonlinearModel_NC = zeros(1,1);
% # of trays (Parameter)
KF_NonlinearModel_NT = zeros(1,1);
% Antonies Constants A, B and C (Parameter)
KF_NonlinearModel_AntoniesA = zeros(1,NC);
KF_NonlinearModel_AntoniesB = zeros(1,NC);
KF_NonlinearModel_AntoniesC = zeros(1,NC);
% First BubblePoint Temperature Guess (Parameter)
KF_NonlinearModel_GuessTemp = zeros(1,1);
% Desired Temperature Measurement Numbers (1 for StillPot, 2 ... NT for Trays, NT+2 for RefluxDrum)
MeasurementOrder

% System Output (Output)
KF_NonlinearModel_h = zeros(size(MeasurementOrder,2),1);

% --------------------------------------------------------------------------------------------------------
% ********************************************************************************************************
% First BubblePoint Temperature Guess
KF_NonlinearModel_GuessTemp = 360.0;
% ******************************************************************************************************
% --- Pre-Calculations (New RefluxFlowrate, Pressures, Temperatures and Vapor compositions)
% Reflux Flowrate
KF_NonlinearModel_R = KF_NonlinearModel_Vb - KF_NonlinearModel_D;
% Trays Pressures
KF_NonlinearModel_Presses(2:NT+1,1) = KF_NonlinearModel_Press(NT, KF_NonlinearModel_Presses(1,1), KF_NonlinearModel_Presses(NT+2,1));
% Still Pot Vapor compositions, Still Pot Temperature
% Trays Vapor compositions, Trays Temperatures
% Reflux Drum Vapor compositions, Reflux Drum Temperature
for i=1:NT+2;
[KF_NonlinearModel_Tempers(i,1), KF_NonlinearModel_YVapors(i,:)] = KF_NonlinearModel_Equilib(KF_NonlinearModel_GuessTemp, KF_NonlinearModel_XLiquids(i,:), KF_NonlinearModel_Presses(i), KF_NonlinearModel_AntoniesA, KF_NonlinearModel_AntoniesB, KF_NonlinearModel_AntoniesC);
end;
% ******************************************************************************************************
% *************************************% *************************************% *************************
% -------------------------------------------------------------- Output Calculation
for i=1:size(MeasurementOrder,2);
k = MeasurementOrder(i);
KF_NonlinearModel_h(i) = KF_NonlinearModel_Tempers(k,1);
end;
function [Temper, Vapors] = KF_NonlinearModel_Equilib(GuessTemper, Liquids, Press, AntoniesA, AntoniesB, AntoniesC)
% Equilibrium temp and composition calculation by Rault's law and Three constant Antoine's equation
% using Newton-Raphson method

% INPUTS ******************************************************
% GuessTemper (K) : First Bublepoint Temperature Guess
% Liquids (mol/mol) : Liquid mol fractions
% Press (Pa) : Ambient pressure
% AntoniesA, AntoniesB, AntoniesC : Antoine's Equation Constants
% log10(Pvap) = A - B/(C+T), T(degree C), Pvap(Torr)
% OUTPUTS ******************************************************
% Temper (K) : Bublepoint Temperature
% Vapors (mol/mol) : Bublepoint vapor phase mol fractions

% convert GuessT from Kelvin to degree of C
GuessTemper_C = GuessTemper - 273.15;
% convert Pressure from Pa to Torr
Press_Torr = Press / 1.333E2;
% # of components
NC = size(Liquids,2);
% ln(10)
ln_10 = log(10);
% Newton-Raphson Parameters
% zero finding tolerance
toler = Press_Torr / 100000.0;
% Max # of iterations allowed
limit = 50;

% *******************************************************************************************
% Newton-Raphson root finding ***************************************************************
% Temper_C = GuessTemper_C; flag = -1; count = 0; while flag<0 & count<limit;
% Pcalc = Equilib(Temper_C, NC, Liquids, AntoniesA, AntoniesB, AntoniesC);
% F_obj = Pcalc-Press_Torr;
% if abs(F_obj)<toler;
% flag = 1;
% else;
% count = count + 1;
% DF_obj = D_Equilib(Temper_C, NC, Liquids, AntoniesA, AntoniesB, AntoniesC, ln_10);
% if DF_obj == 0;
% Temper_C = Temper_C;
% else
% Temper_C = Temper_C - F_obj/DF_obj;
% end;
% end;
% end;
% *******************************************************************************************
% *******************************************************************************************
% *******************************************************************************************
% Result check and return ************************************************************************
if flag>0;
% Calculate vapor compositions
for i=1:NC;
Vapors(i) = Liquids(i)*power(10, AntoniesA(i) - AntoniesB(i)/ (Temper_C + AntoniesC(i)))/Pcalc;
end;
else
'GuessTemper, Liquids, Press, AntoniesA, AntoniesB, AntoniesC'
error(['Buble point calculation did not converge. count:' num2str(count) ' [KF_NonlinearModel_Equilib]']);
end;
% convert T from degree of C to Kelvin
Temper = Temper_C + 273.15;
% *******************************************************************************************
%end KF_NonlinearModel_Equilib

% Total Pressure Calculations from Vapor Pressures
function [Y] = Equilib(T, NC, X, A, B, C)
Y = 0.0; for j=1:NC;
Y = Y + x(j)*10^((A(j)-B(j)/(T+C(j)))) ;
end;
end Equilib

% function [Y] = D_Equilib(T, NC, X, A, B, C, ln_10)
Y = 0.0; for i=1:NC;
Y = Y + x(i)*10^((A(i)-B(i)/(T+C(i))))*B(i)/(T+C(i)^2)*ln_10;
end;
end D_Equilib
D.4.4 KF_NonlinearModel_Press.m

```matlab
function [Press] = KF_NonlinearModel_Press(NT, SPotPress, RDrumPress);
% Trays pressures calculations
if (size(SPotPress)~=1 | size(RDrumPress)~=1)
    error('SPotPress and/or RDrumPress are not scalar(s). [KF_NonlinearModel_Press]');
end;
Press = zeros(NT,1);
fdelP_Tray = (SPotPress-RDrumPress)/NT;
for i=1:NT;
    Press(i) = SPotPress - i*fdelP_Tray;
end;
% end KF_NonlinearModel_Press
```

D.5 EKF Linearized Model Code

D.5.1 KF_LinearizedModel_SysF.m

```matlab
function [KF_LinearizedModel_F] = KF_LinearizedModel_SysF(Compositions, StillPotHoldup, DistillateFlowrate, t, NC, NT, Vb, Holdups, Press, Antonies)
%KF_LinearizedModel_HbDot,
% Still Pot Liquid compositions (State)
% Trays Liquid compositions (State)
% Reflux Drum Liquid compositions (State)
KF_LinearizedModel_XLiquids = zeros(NT+2,NC);
% Still Pot Holdup (State)
% Trays Holdups (Parameter)
% Reflux Drum Holdup (Parameter)
KF_LinearizedModel_Holdups = zeros(NT+2,1);
% Still Pot Vapor compositions (PreCalculation)
% Trays Vapor compositions (PreCalculation)
% Reflux Drum Vapor compositions (PreCalculation)
KF_LinearizedModel_YVapors = zeros(NT+2,NC);
% Still Pot Temperature (PreCalculation)
% Trays Temperatures (PreCalculation)
% Reflux Drum Temperature (PreCalculation)
KF_LinearizedModel_Tempers = zeros(NT+2,1);
% Still Pot Pressure (Parameter)
% Trays Pressures (PreCalculation)
% Reflux Drum Pressure (Parameter)
KF_LinearizedModel_Presses = zeros(NT+2,1);
% Distillate flowrate (Input)
KF_LinearizedModel_D = zeros(1,1);
% Vapor boilup flowrate (Parameter)
KF_LinearizedModel_Vb = zeros(1,1);
% # of components (Parameter)
KF_LinearizedModel_NC = NC; KF_LinearizedModel_NT = NT;
% Antonies Constants A, B and C (Parameter)
KF_LinearizedModel_AntoniesA = Antonies(1:NC);
KF_LinearizedModel_AntoniesB = Antonies(NC+1:2*NC);
KF_LinearizedModel_AntoniesC = Antonies(2*NC+1:3*NC);
% First BublePoint Temperature Guess (Parameter)
KF_LinearizedModel_GuessTemp = zeros(1,1);
% Linearized system matrix for Estimation States (Output) in the order of Estimation States (Output)
KF_LinearizedModel_F = zeros(NC*(NT+2), NC*(NT+2));
```

% *******************************************************************************************************************
% ------------------------------------ Disturibute the function INPUTS to PHYSICAL Variables ----------------
% Input States
% Compositions : [xSPot1, xSPot2, ... xSPotNC; x11, x12, ... x1NC; x21, x22, ... x2NC; ... xNT1, xNT2, ... xNTNC;...
% xRDrum1, xRDrum2, ... xRDrumNC]
% StillPotHoldup : [ Hb ]
% Output Linearized system matrix order
% KF_LinearizedModel_XLiquids = Compositions;
% KF_LinearizedModel_Holdups(1,1) = StillPotHoldup;
% Inputs : [ DistillateFlowrate ]
% KF_LinearizedModel_D = DistillateFlowrate;
% Parameters : # of components, # of trays, Vapor Boilup rate, [Tray Holdups, RefluxDrum Holdup, ...]
% [Reboiler Pressure, RefluxDrum Pressure], [AntoniesConstants: A1...ANC, B1...BNC, C1...CNC]
% KF_LinearizedModel_NC = NC; KF_LinearizedModel_NT = NT;
% KF_LinearizedModel_YVapors = zeros(NT+2,NC);
% KF_LinearizedModel_NHoldups(2:NT+1,1) = Holdups; KF_LinearizedModel_Presses(NT+2,1) = Press(2);
% KF_LinearizedModel_AntoniesA = Antonies(1:NC);
% KF_LinearizedModel_AntoniesB = Antonies(NC+1:2*NC);
% KF_LinearizedModel_AntoniesC = Antonies(2*NC+1:3*NC);
% % *******************************************************************************************************************
% First Bubble Point Temperature Guess
KF_LinearizedModel_GuessTemp = 370.0;

% ******************************************************************************************************
% ------------------- Pre-Calculations (New RefluxFlowrate, Pressures, Temperatures and Vapor compositions)
% Reflux Flowrate
KF_LinearizedModel_R = KF_LinearizedModel_Vb - KF_LinearizedModel_D;

% Trays Pressures
KF_LinearizedModel_Presses(2:NT+1,1) = KF_LinearizedModel_Press(NT, KF_LinearizedModel_Presses(1,1), KF_LinearizedModel_Presses(NT+2,1));

% Still Pot Vapor compositions, Still Pot Temperature
% Trays Vapor compositions, Trays Temperatures
% Reflux Drum Vapor compositions, Reflux Drum Temperature
for i=1:NT+2;
    [KF_LinearizedModel_Tempers(i,1), KF_LinearizedModel_YVapors(i,:)] = KF_LinearizedModel_Equilib...
    (KF_LinearizedModel_GuessTemp, KF_LinearizedModel_XLiquids(i,:), KF_LinearizedModel_Presses(i), ...
    KF_LinearizedModel_AntoniesA, KF_LinearizedModel_AntoniesB, KF_LinearizedModel_AntoniesC);
end;

% ******************************************************************************************************
% *************************************% *************************************% *************************************
% Calculate dy/dx matrix ( size : NT+2, NC, NC ) ( referrence : tray #, y index, x index)
[DyDx] = KF_LinearizedModel_CalcDyDx(KF_LinearizedModel_XLiquids,
KF_LinearizedModel_Tempers, KF_LinearizedModel_Presses,...
KF_LinearizedModel_AntoniesA, KF_LinearizedModel_AntoniesB, KF_LinearizedModel_AntoniesC);
% *************************************% *************************************% *************************************
% *************************************% *************************************% *************************************
% Calculate Linearized system matrix for Estimation States (Output)
% Variable copying for simplifying syntax
% Reflux Flow Rate, RF <- KF_LinearizedModel_R
RF = KF_LinearizedModel_R;

% Vapor Bull-up Rate, Vboil <- KF_LinearizedModel_Vb
Vboil = KF_LinearizedModel_Vb;

% Holdups, H <- KF_LinearizedModel_Holdups
H = KF_LinearizedModel_Holdups;

%---------------------------------
for k=1:NT+1;
    %---------------------------------
    n=1; i=kn2i(k,n,NC); p=1; r=1; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - RF/H(1) - (Vboil(1,1,1)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - (Vboil(1,1,2)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - RF/H(1) - (Vboil(1,1,3)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - (Vboil(1,2,1)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - RF/H(1) - (Vboil(1,2,2)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = - RF/H(1) - (Vboil(1,2,3)-1); r=2; j=kn2i(p,r,NC);

%---------------------------------
%---------------------------------
%---------------------------------
for k=2:NT+2;
    %---------------------------------
    n=1; i=kn2i(k,n,NC); p=1; r=1; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,1,1)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,1,2)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,1,3)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,2,1)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,2,2)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,2,3)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,3,1)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,3,2)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,3,3)-1); r=2; j=kn2i(p,r,NC);

%---------------------------------
%---------------------------------
%---------------------------------
for k=NT+2:
    %---------------------------------
    n=1; i=kn2i(k,n,NC); p=1; r=1; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,1)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,2)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,3)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,4)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,5)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,6)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,7)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,8)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,9)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,10)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,11)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,12)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,13)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,14)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,15)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,16)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,17)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,18)-1); r=3; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,19)-1); r=2; j=kn2i(p,r,NC);
    KF_LinearizedModel_F(i,j) = (Vboil(1,k,20)-1); r=3; j=kn2i(p,r,NC);

%---------------------------------
%---------------------------------
%---------------------------------
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,1,2) ;

p=k;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,1,3) ;

r=2; j=kn2i(2,r,NC);
KF_LinearizedModel_F(i,j) = - (Vboil/H(k))* DyDx(k-1,1,1) - RF/H(k) ;

r=3; j=kn2i(3,r,NC);
KF_LinearizedModel_F(i,j) = - (Vboil/H(k))* DyDx(k-1,1,2) ;

p=k+1;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = RF/H(k) ;

r=2; j=kn2i(2,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(3,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

end;

%---------------------------------
%---------------------------------

n=2; i=kn2i(1,n,NC);

p=k;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,2,1) ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,2,2) ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,2,3) ;

p=k+1;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

%---------------------------------
%---------------------------------

n=3; i=kn2i(1,n,NC);

p=k;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,3,1) ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,3,2) ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = (Vboil/H(k))* DyDx(k-1,3,3) ;

p=k+1;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

%---------------------------------
%---------------------------------

k=NT+2; n=1; i=kn2i(1,n,NC);

p=NT;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

p=NT+1;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

%---------------------------------
%---------------------------------

k=NT+2; n=2; i=kn2i(1,n,NC);

p=NT;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

p=NT+1;

r=1; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=2; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

r=3; j=kn2i(1,r,NC);
KF_LinearizedModel_F(i,j) = 0 ;

%---------------------------------
}
\[ \frac{\partial H}{\partial N_{T+2}} \cdot \frac{\partial x_{PQ}}{\partial N_{T+1,2,3}}; p=NT+2; r=1; \]
\[ j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0; \]
\[ r=2; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = \left( \frac{\partial H}{\partial N_{T+2}} \right) \cdot \frac{\partial x_{PQ}}{\partial N_{T+1,2,3}}; r=2; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0; \]
\[ r=3; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = - \left( \frac{\partial H}{\partial N_{T+2}} \right) ; \]
\[ k=NT+2; n=3; i=kn2i(k,n,N_{C}); p=NT; r=1; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0 ; \]
\[ r=2; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0; \]
\[ r=3; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0; \]
\[ p=NT+1; r=1; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = \left( \frac{\partial H}{\partial N_{T+2}} \right) \cdot \frac{\partial x_{PQ}}{\partial N_{T+1,2,3}}; r=2; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = 0; \]
\[ r=3; j=kn2i(p,r,N_{C}); F_{i,j}^{\text{LinearizedModel}} = - \left( \frac{\partial H}{\partial N_{T+2}} \right) ; \]
\[ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \]
\[ function i = kn2i(k,n,N_{C}) i = (k-1)\cdot N_{C} + n; \]
\[ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \]
\[ D.5.2 \text{ KF LinearizedModel SysH.m} \]
\[ function [\text{KF LinearizedModel H}] = \text{KF LinearizedModel SysH}(\text{Compositions, StillPotHoldup, DistillateFlowrate, t, NC, NT, Vb, Holdups, Press, Antonies, MeasurementOrder}) \]
\[ \text{KF LinearizedModel HbDot}, \]
\[ \% Still Pot Liquid compositions \text{(State)} \]
\[ \% Trays Liquid compositions \text{(State)} \]
\[ \% Reflux Drum Liquid compositions \text{(State)} \]
\[ \text{KF LinearizedModel XLiquids} = \text{zeros}(NT+2,N_{C}); \]
\[ \% Still Pot Holdup \text{(State)} \]
\[ \% Trays Holdups \text{(Parameter)} \]
\[ \% Reflux Drum Holdup \text{(Parameter)} \]
\[ \text{KF LinearizedModel Holdups} = \text{zeros}(NT+2,1); \]
\[ \% Still Pot Vapor compositions \text{(PreCalculation)} \]
\[ \% Trays Vapor compositions \text{(PreCalculation)} \]
\[ \% Reflux Drum Vapor compositions \text{(PreCalculation)} \]
\[ \text{KF LinearizedModel YVapors} = \text{zeros}(NT+2,N_{C}); \]
\[ \% Still Pot Temperature \text{(PreCalculation)} \]
\[ \% Trays Temperatures \text{(PreCalculation)} \]
\[ \% Reflux Drum Temperature \text{(PreCalculation)} \]
\[ \text{KF LinearizedModel Tempers} = \text{zeros}(NT+2,1); \]
\[ \% Still Pot Pressure \text{(Parameter)} \]
\[ \% Trays Pressures \text{(PreCalculation)} \]
\[ \% Reflux Drum Pressure \text{(Parameter)} \]
\[ \text{KF LinearizedModel Presses} = \text{zeros}(NT+2,1); \]
\[ \% Distillate flowrate \text{(Input)} \]
\[ \text{KF LinearizedModel D} = \text{zeros}(1,1); \]
\[ \% Reflux flowrate \text{(PreCalculation)} \]
\[ \text{KF LinearizedModel R} = \text{zeros}(1,1); \]
\[ \% Vapor boilup flowrate \text{(Parameter)} \]
\[ \text{KF LinearizedModel Vb} = \text{zeros}(1,1); \]
\[ \% # of components \text{(Parameter)} \]
\[ \text{KF LinearizedModel NC} = \text{zeros}(1,1); \]
\[ \% # of trays \text{(Parameter)} \]
\[ \text{KF LinearizedModel NT} = \text{zeros}(1,1); \]
\[ \% Antonies Constants A, B and C \text{(Parameter)} \]
\[ \text{KF LinearizedModel AntoniesA} = \text{zeros}(1,N_{C}); \]
\[ \text{KF LinearizedModel AntoniesB} = \text{zeros}(1,N_{C}); \]
\[ \text{KF LinearizedModel AntoniesC} = \text{zeros}(1,N_{C}); \]
\[ \% First BubblePoint Temperature Guess \text{(Parameter)} \]
\[ \text{KF LinearizedModel GuessTemp} = \text{zeros}(1,1); \]
\[ \% Linearized measurement matrix \]
\[ \text{KF LinearizedModel H} = \text{zeros(size(MeasurementOrder,2),(NT+2)\cdot N_{C});} \]
\[ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \]
\[ x = \text{kn2i(k,n,N_{C}); i = (k-1)\cdot N_{C} + n;} \]
\[ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \]
% Parameters : # of components, # of trays, Vapor Boilup rate, (Tray Holdups, Reflux Drum Holdups),...
% (Reboiler Pressure, Reflux Drum Pressure), (AntoniesConstants: A1...ANC, B1...BNC, C1...CNC)
KF_LinearizedModel_SC = NC; KF_LinearizedModel_NT = NT;
KF_LinearizedModel_Vb = Vb; KF_LinearizedModel_Holdups(2:NT+2,1) = Holdups;
KF_LinearizedModel_Presses(1,1) = Press(1);
KF_LinearizedModel_Presses(NT,1) = Press(2);
KF_LinearizedModel_Tempers = RefluxDrum_Tempers; KF_LinearizedModel_YVapors = RefluxDrum_YVapors;
KF_LinearizedModel_AntoniesA = Antonies(1:NC);
KF_LinearizedModel_AntoniesB = Antonies(NC+1:2*NC);
KF_LinearizedModel_AntoniesC = Antonies(2*NC+1:3*NC);
% -------------------------------------------------------------------------------------------------------------
% *************************************************************************************************************
% First Bubblepoint Temperature Guess
KF_LinearizedModel_GuessTemp = 370.0;
% ******************************************************************************************************
% ------------------- Pre-Calculations (New RefluxFlowrate, Pressures, Temperatures and Vapor compositions)
% Reflux Flowrate
KF_LinearizedModel_R = KF_LinearizedModel_Vb - KF_LinearizedModel_D;
% Trays Pressures
KF_LinearizedModel_Presses(2:NT+1,1) = KF_LinearizedModel_Press(NT, KF_LinearizedModel_Presses(1,1),
KF_LinearizedModel_Presses(NT+2,1));
% Still Pot Vapor compositions, Still Pot Temperature
% Trays Vapor compositions, Trays Temperatures
% Reflux Drum Vapor compositions, Reflux Drum Temperature
for i=1:NT+2;
[KF_LinearizedModel_Tempers(i,1), KF_LinearizedModel_YVapors(i,:)] = KF_LinearizedModel_Equilib(KF_LinearizedModel_GuessTemp, KF_LinearizedModel_XLiquids(i,:), KF_LinearizedModel_Presses(i),
KF_LinearizedModel_AntoniesA, KF_LinearizedModel_AntoniesB, KF_LinearizedModel_AntoniesC);
end;
% ******************************************************************************************************
% *************************************% *************************************% *************************************
% Calculate dT/dx matrix ( size : NT+2, NC ) ( reference : tray temperature # , tray component #)
[TXD] = KF_LinearizedModel_CalcTXD( KF_LinearizedModel_XLiquids,
KF_LinearizedModel_Tempers, KF_LinearizedModel_Presses,
KF_LinearizedModel_AntoniesA, KF_LinearizedModel_AntoniesB, KF_LinearizedModel_AntoniesC);
% *************************************% *************************************% *************************************
% Calculate Linearized measurement matrix
% *************************************% *************************************% *************************************
% for measured Temperatures
for i=1:size(MeasurementOrder,2);
p = i;
for r=1:NC;
j = kn2i(p,r,NC);
KF_LinearizedModel_H(i,j) = TXD(MeasurementOrder(i),r);
end;
end;
% *************************************% *************************************% *************************************
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function i = kn2i(k,n,NC) i = (k-1)*NC + n;
%end kn2i
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

D.5.3 KF_LinearizedModel_Equilib.m

function [Temper, Vapors] = KF_LinearizedModel_Equilib(GuessTemper, Liquids, Press, AntoniesA,
AntoniesB, AntoniesC)
% Equilibrium temp and composition calculation by Rault's law and Three constant Antoine's equation
% using Newton-Raphson method

% INPUTS ****************************
% GuessTemper (K) : First Bubblepoint Temperature Guess
% Liquids (mol/mol) : Liquid mol fractions
% Press (Pa) : Ambient pressure
% AntoniesA, AntoniesB, AntoniesC : Antoine's Equation Constants
% log10(Pvap) = A - B/(C+T), T(degree C), Pvap(Torr)
% OUTPUTS ***************************
% Temper (K) : Bubblepoint Temperature
% Vapors (mol/mol) : Bubblepoint vapor phase mol fractions

% convert GuessT from Kelvin to degree of C
GuessTemp_C = GuessTemper - 273.15;
% convert Pressure from Pa to Torr
Press_Torr = Press / 1.333E2;
% # of components
NC = size(Liquids,2);
ln10 = log(10);
% Newton-Raphson Parameters
% zero finding tolerance
toler = Press_Torr / 100000.0;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function i = kn2i(k,n,NC) i = (k-1)*NC + n;
%end kn2i
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Max # of iterations allowed
limit = 50;

% Newton-Raphson root finding

Temper_C = GuessTemper_C; flag = -1; count = 0; while flag < 0 & count < limit;

Pcalc = Equilib(Temper_C, NC, Liquids, AntoniesA, AntoniesB, AntoniesC);
F_obj = Pcalc - Press_Torr;
if abs(F_obj) < toler;
    flag = 1;
else;
    count = count + 1;
    DF_obj = D_Equilib(Temper_C, NC, Liquids, AntoniesA, AntoniesB, AntoniesC, ln_10);
    if DF_obj == 0;
        Temper_C = Temper_C;
    else;
        Temper_C = Temper_C - F_obj/DF_obj;
    end;
end;
end;

% Result check and return

if flag > 0;
    % Calculate vapor compositions
    for i=1:NC;
        Vapors(i) = Liquids(i)*power(10, AntoniesA(i) - AntoniesB(i)/(Temper_C + AntoniesC(i)))/Pcalc;
    end;
else
    'GuessTemper, Liquids, Press, AntoniesA, AntoniesB, AntoniesC'
    'Buble point calculation did not converge. count:' num2str(count) ' [KF_LinearizedModel_Equilib]';
end;

% convert T from degree of C to Kelvin
Temper = Temper_C + 273.15;

D.5.4 KF_LinearizedModel_Press.m

function [Press] = KF_LinearizedModel_Press(NT, SPOTPress, RDrumPress);
% Trays pressures calculations
if (size(SPotPress)~=1 | size(RDrumPress)~=1)
    error('SPOTPress and/or RDrumPress are not scalar(s). [KF_LinearizedModel_Press]');
end;
Press = zeros(NT,1);
fdelP_Tray = (SPOTPress-RDrumPress)/NT;
for i=1:NT; Press(i) = SPOTPress - i*fdelP_Tray; end;
end KF_LinearizedModel_Press

D.5.5 KF_LinearizedModel_CalcDyDx.m

function [DyDx] = KF_LinearizedModel_CalcDyDx(XLiquids, Tempers, Presses, AntoniesA, AntoniesB, AntoniesC)
% Calculate dy/dx matrix ( size : NT+2, NC, NC ) ( reference : tray #, y index, x index)
%XLiquids : (NT+2,NC)
%Tempers : (NT+2,1)
%Presses : (NT+2,1)
% Calculation of alfa(i,k) and 10^alfa(i,k)
[a_l_f_a] = alfa_calc(T, AntoniesA, AntoniesB, AntoniesC);
ten_to_alfa = power(10, a_l_f_a);
% Calculation of theta(i,k)
[t_h_e_t_a] = theta_calc(T, AntoniesB, AntoniesC);
% Calculation of fi(i,k)
fi = ln_10 .* X .* ten_to_alfa .* t_h_e_t_a;
% Calculation of TOT(i)
TOT = sum(fi,2);
% Initialize DyDx
DyDx = zeros(NT+2,NC,NC);
% Calculation of DyDx(i,k,n)
for i=1:NT+2;
    for k=1:NC;
    for n=1:NC;
        if k==n;
            DyDx(i,k,k) = (1/P(i,1)) * ( ten_to_alfa(i,k) + fi(i,k)*(-ten_to_alfa(i,k)/TOT(i)) );
        elseif k~=n;
            DyDx(i,k,n) = (1/P(i,1)) * ( fi(i,k)*(-ten_to_alfa(i,n)/TOT(i)) );
        end;
    end;
end;
end;
% end KF_LinearizedModel_CalcDyDx

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation of alfa(i,k)
function [a] = alfa_calc(T, AntoniesA, AntoniesB, AntoniesC)
    for i=1:size(T,1);
        for k=1:size(AntoniesA, 2);
            a(i,k) = AntoniesA(1,k) - (AntoniesB(1,k)/(AntoniesC(1,k)+T(i,1)));
        end;
    end;
% end alfa_calc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation of theta(i,k)
function [t] = theta_calc(T, AntoniesB, AntoniesC)
    for i=1:size(T,1);
        for k=1:size(AntoniesB, 2);
            t(i,k) = AntoniesB(1,k)/((AntoniesC(1,k)+T(i,1))^2);
        end;
    end;
% end alfa_calc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

D.5.6 KF_LinearizedModel_CalcTXD.m

function [TXD] = KF_LinearizedModel_CalcTXD( XLiquids, Tempers, Presses, AntoniesA, AntoniesB, AntoniesC)
% Calculate dT/dx matrix ( size : NT+2, NC ) ( reference : tray temperature # , tray component #)
%XLiquids : (NT+2,NC)
%Tempers : (NT+2,1)
%Presses : (NT+2,1)
%AntoniesA : (1,NC)
%AntoniesB : (1,NC)
%AntoniesC : (1,NC)
NT = size(Tempers,1) - 2; NC = size(AntoniesA,2);
X = [ XLiquids ]; T = [ Tempers ]; P = [ Presses ];
ln_10 = log(10.0);
% Calculation of alfa(i,k) and 10^alfa(i,k)
[a_l_f_a] = alfa_calc(T, AntoniesA, AntoniesB, AntoniesC);
ten_to_alfa = power(10, a_l_f_a);
% Calculation of theta(i,k)
[t_h_e_t_a] = theta_calc(T, AntoniesB, AntoniesC);
% Calculation of fi(i,k)
fi = ln_10 .* X .* ten_to_alfa .* t_h_e_t_a;
% Calculation of $\text{TOT}(i)$
TOT = sum($fi$, 2);

% Initialize $\text{TED}$
TED = zeros($NT+2$, $NC$);

% Calculation of $\text{TED}(p,r)$
for $p$=1:$NT+2$;
    for $r$=1:$NC$;
        $TED(p, r) = - \text{ten}_\text{to}_\text{alfa}(p, r) / \text{TOT}(p);$;
    end;
end;

% end KF_LinearizedModel_CalcTED

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation of $\text{alfa}(i,k)$
function $[a]$ = alfa_calc($T$, AntoniesA, AntoniesB, AntoniesC)
for $i$=1:size($T$, 1);
    for $k$=1:size(AntoniesA, 2);
        $a(i, k) = \text{AntoniesA}(1, k) - (\text{AntoniesB}(1, k) / (\text{AntoniesC}(1, k) * T(1, 1)))$;
    end;
end;

% end alfa_calc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation of $\theta(i,k)$
function $[t]$ = theta_calc($T$, AntoniesB, AntoniesC)
for $i$=1:size($T$, 1);
    for $k$=1:size(AntoniesB, 2);
        $t(i, k) = \text{AntoniesB}(1, k) / ((\text{AntoniesC}(1, k) * T(1, 1))^2)$;
    end;
end;

% end theta_calc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%