Graduate School ETD Form 9
(Revised 01/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Howard Cheung

Entitled
INVERSE MODELING OF VAPOR COMPRESSION EQUIPMENT TO ENABLE SIMULATION OF FAULT IMPACTS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Prof. James E. Braun

Dr. Piotr A. Domanski

Prof. Eckhard Groll

Prof. W. Travis Horton

To the best of my knowledge and as understood by the student in the *Thesis/Dissertation Agreement. Publication Delay, and Certification/Disclaimer (Graduate School Form 32)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Prof. James E. Braun

Approved by Major Professor(s): _____

_____

Approved by: Prof. David C. Anderson 07/17/2014

Head of the Department Graduate Program Date

INVERSE MODELING OF VAPOR COMPRESSION EQUIPMENT TO ENABLE SIMULATION OF FAULT IMPACTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Howard Cheung

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2014

Purdue University

West Lafayette, Indiana

To my family.

ACKNOWLEDGMENTS

First of all, I would like to acknowledge my supervisor Prof. James E. Braun. I would not have the opportunities and training to finish the research projects during my five years of graduate study without his teaching and guidance. I would in particular like to thank him for his tolerance on my inexperienced writing and hands-on skills since the beginning of my graduate study. I would also like to thank my committee members Prof. Eckhard Groll, Prof. Travis Horton and Dr. Priotr Domanski for their advice in the research work, and everyone at Ray W. Herrick Laboratories to make the working environment much more friendly than I expected before I came here.

In particular, I would also like to thank Frank Lee, without whom most of my experiments would take much longer to complete, and David Yuill for all his opinions and discussion throughout my graduate study. His work related to this project, such as the extraction of steady state data from Breuker's system (Breuker, 1997) and his feedback on my model, also helps me to improve my dissertation. My thanks also go to Christian Bach who have been sharing the joy and pain since we started the graduate study together in the same student office, Simbarashe Nyika for all his patience and hard work in our collaboration, Ian Bell for his novel and inspiring ideas,

Woohyun Kim for working on laboratory experiments together, Stephen Caskey and Derek Kultgen for working together at the Purdue ASHRAE student chapter, Andrew Hjortland and Brandon Woodland for all group projects and Orkan Kurtulus and Tim Blatchley for all the laughs in the office.

I would also like to acknowledge the sponsors and stakeholders of my projects throughout my graduate studies, for their opinions in my work and without which I cannot have the materials to finish the dissertation. They include Dr. Vance Payne from National Institute of Standards and Technology that sponsors this modeling project on fault impacts on vapor compression system, Ben Larson from Ecotope, Inc., Dr. Jon Winkler from National Renewable Energy Laboratory and Dr. Ron Domitrovic from Electric Power Research Institute.

I would also like to take this opportunity to thank my friends and former supervisors outside Purdue University whom supported me to come to Purdue, including Prof. Wenjing Ye for her recommendation and supervision of research work during my last year of undergraduate, Prof. Kai Tang for inspiring my interest in numerical methods and programming, Prof. Christopher Chow for his recommendation in my graduate school application, Chi Wai Leung and Kawai Tam for facing the unknowns in our last undergraduate year together before I came to Purdue and Ho Ming Ku without whom I might not be confident to start my graduate study.

Finally, I would like to deliver my sincere thanks to my family. I would have no way to keep track of anything related to me in Hong Kong without my sister's smart

and careful management, and I would not have the time to pursue my graduate study if she did not stay in Hong Kong and helped my parents to manage all family issues. I would also like to thank my parents to nurture me everything from work ethics, sense of responsibility, morals and relationship and tolerance with people. Without them, I would not be able to interact and get help from anyone I acknowledge here and finish my graduate study.

TABLE OF CONTENTS

Page

APPENDICES

VOLUME 2

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

NOMENCLATURE

A	Surface area $[m^2]$
A_{flow}	Flow area $[m^2]$
C	Regression parameters [varies]
$COSP$	Coefficient of system performance [–]
$C_{EXV,\text{discharge}}$	Expansion valve discharge coefficient $[m^2]$
$Capr$	Capacitance rate in dry-coil analysis $[W/K]$
$Capr_{wet}$	Capacitance rate in wet-coil analysis $[kg/s]$
Cr	Capacity ratio [–]
D	Diameter $[m]$
Dew	Dewpoint [K]
F_{clear}	Clearance ratio [–]
J	Objective function [varies]
K	Tuning coefficients for heat transfer coefficients $[K/m^{0.8}]$
L	Length $[m]$

LL	Liquid line restriction fault level [–]
M	Mass of refrigerant $[kg]$
NC	Non-condensable fault level [–]
NTU	Number of transfer units [–]
N_{bin}	Number of data points in the bin [–]
Nu	Nusselt number [–]
P	Pressure $[kPa]$
Pr	Prandtl number [–]
Ra	Rayleigh number [–]
Re	Reynolds number [–]
SC	Subcooling [K]
SH	Superheat [K]
T	Temperature $[K]$
U	Heat transfer coefficient $[W/m^2 - K]$
UA	Heat transfer conductance $[W/K]$
UA^*	Heat and mass transfer conductance $[kg/s]$
V	Volume $[m^3]$

VL	Fault level for compressor flow fault [–]
ΔM	Mass adjustment factor $[kg]$
ΔP	Pressure drop $[kPa]$
$\dot{Q}$	Heat transfer rate $[W]$
$\dot{V}$	Volumetric airflow $[m^3/s]$
$\dot{W}$	Power consumption $[W]$
$\dot{m}$	Mass flow rate $[kg/s]$
ϵ	Threshold [–]
η_v	Volumetric efficiency [–]
η_{fan}	Fan efficiency [−]
η_{fin}	Fin efficiency [–]
γ	Void fraction [–]
μ	Viscosity $[kg/m - s]$
ρ	Density $[kg/m^3]$
ε	Heat exchanger effectiveness [–]
bp	Bypass mass flow ratio of a compressor [–]
c	Regression coefficients [varies]

c_{pa}	Specific heat capacity of air-water mixture $[J/kg-K]$
c_{pr}	Specific heat capacity of refrigerant $[J/kg-K]$
$f_{\Delta P,r}$	Friction factor [–]
f_{comp}	Compressor frequency $[Hz]$
f_{dry}	Fraction of dry portion in a phase section
f_{fan}	Fan rotational frequency $[Hz]$
h	Enthalpy $[J/kg]$
k	Thermal conductivity $[W/m-K]$
n	Exponential coefficient of airflow [–]
n_{poly}	Polytropic coefficient [–]
r	Residual [–]
w	Area ratio [–]
$weight$	Weighting function [–]
x	Thermodynamic quality [–]
x_{ind}	Independent variables [–]
y	Thickness $[m]$
δ	Distance to the boundary of applicability domain [–]

θ	leverage

Subscripts

1ϕ	single-phase
Δh	enthalpy change
ΔP	pressure drop
a	air-water mixture
act	actual
amb	ambient
$bulb$	bulb
$comp$	compressor
$cond$	condenser
$cric$	critical
$econ$	economizer
$evap$	evaporator
EXV	expansion valve
fan	fan/blower
fin	heat exchanger fin

$guess$	guess value
HL	heat loss
in	inlet
l	saturated liquid
$local$	local
M	charge tuning
mea	measured in experiment
nc	non-condensable
NF	mean at non-faulted operation
NF	non-faulted
$nozzle$	airflow measurement nozzle
ori	original
out	outlet
$overall$	overall
$pipeline$	refrigerant pipeline
$poly$	polytropic process
pre	predicted by component model

r	refrigerant
$rated$	rated condition
ret	return air duct
sc	subcooled
sh	superheated
sim	simulated by void fraction model
ss	surface
sup	supply air duct
$temp$	temporary
$total$	total refrigerant circuit
$train$	training data
TXV	thermostatic expansion valve
v	saturated vapor
liquidline	liquid line
specified	user-specified

ABSTRACT

Cheung, Howard Ph.D., Purdue University, August 2014. Inverse Modeling of Vapor Compression Equipment to Enable Simulation of Fault Impacts. Major Professor: James E. Braun, School of Mechanical Engineering.

This research is part of an overall effort to develop an evaluator of fault detection and diagnostics (FDD) tools of vapor compression systems (Yuill and Braun, 2012). The evaluator needs a large database of performance data of systems under both faulted and non-faulted conditions. The types of faults include non-standard charging, heat exchanger fouling, compressor flow fault, liquid line restriction and presence of non-condensable. However, conducting experiments to build the database is expensive and time-consuming. Empirical modeling may induce data outside the applicability domain of the model in the database. Forward modeling of vapor compression systems requires many details of the systems that may be unavailable. It may also require multiple tuning methods with experimental data for accuracy. Consequently, inverse modeling, where parameters of models are trained from experimental data directly without detailed knowledge of systems, is chosen to construct the models and to generate the database in this project.

Although models have been developed for simulating faulted impacts on vapor compression systems, they are not quick enough to generate the database and do not

cover all faults studied by the evaluator (Rossi, 1997; Harms, 2002; Shen, 2006). These models also require detailed specification of the systems in addition to the tuning of heat transfer coefficients and other models for accurate simulation. However, inverse modeling approaches need less knowledge of the system than the empirical approach and fewer tuning procedures and less time to build than the forward approach. It is also capable to simulate all the faults investigated by the evaluator and satisfies the needs of the evaluator.

Data from eleven cooling systems tested by different parties were collected. These systems were tested under various types of faults such as non-standard charging, heat exchanger fouling, compressor flow fault, liquid line restriction and presence of non-condensables. Semi-empirical component models were developed with data filtering to avoid predicting unrealistic outcomes. Weighted parameter estimation was carried out during the training process to reduce the effect of imbalanced test matrices on the coefficients. The leverage of the parameter estimation result and the range of training data were also studied to define the applicability domain of the models. Component models were joined together to form a system model. A quasi-Newton method and a constrained optimization algorithm were used to solve the system model with good speed and robustness. An existing charge tuning method was modified to increase the accuracy of charge inventory estimation. The final simulation results were validated with experimental data by comparing estimated performance variables with the experimental data and predicted changes of performance with the measured changes of performance with fault level. The validated simulation was used to study

the impacts of different faults on different types of sample systems (an fixed orifice (FXO) system, an FXO system with an accumulator and a thermostatic expansion valve (TXV) system) by plotting the change of coefficient of performance, evaporator heat transfer rate, compressor power consumption and SHR with increasing fault level.

CHAPTER 1. INTRODUCTION

1.1 Significance and Motivation

With increasing interest in LEED(Leadership in Energy and Environmental Design)-certified and net-zero-energy buildings, the number of studies of various energy-saving techniques on households and commercial buildings has grown. One important approach for improving energy efficiency is fault detection and diagnostics (FDD) for air conditioning and heating equipment. The FDD tools identify problems in situ and help technicians to discover and fix problems which reduce the energy efficiency of the equipment.

In 2010, 47% of energy use of buildings in the U.S. was devoted to space heating and cooling (U.S. Department of Energy, 2011). The significance of the energy consumption has sparked research to promote energy efficiency of space heating and cooling equipment such as the development of FDD tools of vapor compression systems. For example, Breuker (Breuker, 1997) studied the impact of charge leakage, heat exchanger fouling, compressor valve leakage and liquid line restriction on the performance of a fixed orifice (FXO) system. Kim et al. (Kim et al., 2009) presented the impact of the presence of non-condensables and other types of faults on a thermostatic expansion valve (TXV) system. All these researches show that

faults degrade system performance and enhance energy consumption, and FDD technology can be used to avoid unwanted energy consumption and cost.

Numerous parties proposed different algorithms to create FDD tools for vapor compression systems. Rossi and Braun (Rossi and Braun, 1997) proposed a statistical rule-based FDD algorithm with temperature and humidity measurement to identify several faults: refrigerant leakage, heat exchanger fouling, compressor valve leakage and liquid line restriction. California's Title 24 (CEC, 2008) included a rule-based approach using temperature sensors only as an example FDD tool to detect refrigerant leakage and heat exchanger fouling for residential and commercial systems. Schantz and Leeb (Schantz and Leeb, 2012) suggested a method to detect compressor valve leakage by examining electrical signals from the compressor. These FDD tools demonstrate that faults can be detected and diagnosed in different ways.

Although multiple FDD tools have been proposed, there was no standard evaluation scheme for the tools. A scheme is being developed to evaluate FDD tools that detect and diagnose faults based on measurement of system performance at steady state (Yuill, 2014). To conduct a fair and reliable evaluation, it is necessary to have a large database of steady-state performance under faulted and non-faulted conditions of different types of systems, and this raises interest to develop a method to describe system performance under these conditions accurately and quickly.

1.2 Objective

The primary objective of this project is to construct a fast, robust and accurate model for vapor compression systems that properly predicts the impact of faults on steady-state performance of various types of systems. This enables a fair and reliable evaluation of FDD tools with a comprehensive set of accurate performance data under a wide range of conditions and fault levels.

1.3 Methods for Database Construction

To construct a database of system performance under faulted and non-faulted conditions, various methods, including experimental study, empirical modeling, forward modeling and inverse modeling, were investigated.

1.3.1 Experimental Study

Experimental studies of system performance under fault impacts were conducted in different projects(Harms, 2002; Shen, 2006; Kim et al., 2009) where systems were tested with faults in the laboratories. Experimental studies usually take a long time to get steady state system data in a limited number of faulted cases. This restricts the size of the database available for an FDD evaluation approach. Evaluation of situations under different faulted and environmental conditions becomes impossible because the amount of data required for the evaluation is too large to be collected by experimental studies. If a small database is built, it can be gamed by FDD tool

manufacturers easily. To avoid gaming and to allow evaluation over a wide range of fault levels, conditions and systems, experimental observations cannot be used directly to construct a database.

1.3.2 Empirical Modeling

Empirical models allow the construction of a database quickly, but their reliability is limited to the range of experimental data available. In empirical models, the relationship between the system performance and the inputs is defined by empirical coefficients only, and no physics are involved. If system performance is estimated outside the applicability domain of the models, such as multi-fault scenarios, the results may not follow reality. This can be illustrated by modeling superheat with decreasing charge level in an FXO system by a cubic equation. For FXO systems, a decrease of charge level under a fixed environmental condition always leads to an increase of superheat. However, if a cubic equation is used to estimate the superheat at different charge levels and the model is trained by experimental data with charge levels between 60% to 100% only, the model may predict a decrease of superheat when estimating the change of superheat from a charge level of 50% to 40%. This shows that empirical models based on experimental data cannot be used to evaluate FDD tools accurately.

1.3.3 Forward Models

Forward models can usually predict performance accurately. For example, Shen (Shen, 2006) demonstrates that the use of finite segment models with detailed geometry descriptions and fine tuning could estimate the evaporator heat transfer rate with an average deviation of 2.0% in four different systems. Although the estimation is accurate, forward models require geometrical details and they may be unavailable in some experimental datasets. The construction and simulation of the model also requires high computational time because of the tuning process with experimental data for accuracy and detailed modeling methodology involved, and the modeling approach is unsuitable to create a large database.

1.3.4 Inverse Modeling

Inverse modeling is a modeling technique where physical models are simplified and key parameters are trained using experimental data. The model can then be used to predict the system behavior in non-experimented conditions. Optimization algorithms are usually involved to train the parameters. One early effort for thermal systems was made by Biegler (Biegler, 1993) who predicted the heat transfer conductance in a heat exchanger network by sequential quadratic programming. Gau (Gau, 2004) tested an optimization algorithm called the error-in-variable algorithm with the same network. However, these papers did not cover vapor compression systems.

Inverse modeling typically employs semi-empirical models that are constructed based on physical rules with a few empirical parameters. They have fewer empirical parameters than empirical models because part of the models is described by physical rules and does not require empirical rules. The semi-empirical models can also estimate off-design conditions with reasonable accuracy because the change of model outputs with input variables follow the embedded physical rules in off-design conditions. This is important to model fault impacts on different systems as the presence of faults implies operation at off-design conditions. Rabehl et al. (Rabehl et al., 1999) trained semi-empirical heat exchanger models to simulate the heat transfer and pressure drop characteristics by experimental data with water and 50% ethylene/glycol. Another vapor compression cycle model was constructed (Jin et al., 2002) to learn parameters from a water-to-water heat pump by a multi-variant optimization.

1.3.5 Choice of Techniques

Since experimental studies and empirical models are incapable to generate a large database with accurate and reliable system performance under faulted conditions, only forward modeling and inverse modeling can be used to generate an accurate database. Among them, inverse modeling requires less details of the systems than forward modeling and has the potential to require much less computation. Therefore inverse modeling is chosen as the technique to model the performance of the systems under faulted conditions.

1.4 Procedure of the Investigation

The steps of the investigation are shown in Figure 1.1.

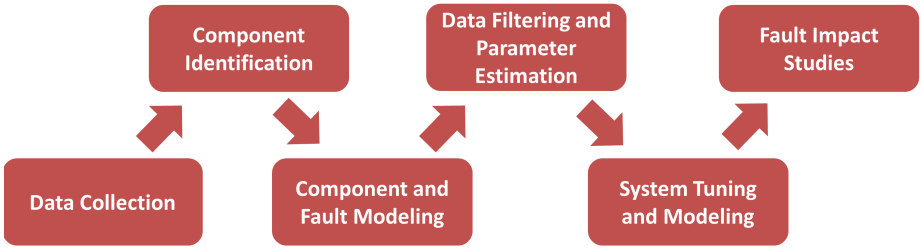

Figure 1.1. Flowchart of the investigation procedure.

The process begins with data collection with experimental data collected from faulted systems tested in laboratories were collected. The data were examined to identify the major components crucial to the system performance, and the effect of other minor components to the system performance was neglected. In the third step, semi-empirical models were developed for different major components and fault effects were described mathematically. To avoid corrupting the parameter estimation by inaccurate experimental data, unrealistic and invalid data were filtered before estimating the parameters of the component models. The limitation of the component models were also studied based on the results of the parameter estimation. The component models were then combined together to form a system model. Solution approach of the cycle model was also developed so that the model could be solved under both normal and faulted conditions quickly. Tuning was conducted on the cycle model to remove bias in charge estimation as a consequence of model simplifications.

The applicability domain of the system model was also studied based on the deviation between the experimental and simulation result and the reliability of the component models. The final simulation results were compared to the experimental data for model validation and parametric studies were conducted to examine fault impacts on different types of systems.

1.5 Organization of the Thesis

This chapter discusses the significance of the work and the direction of the research. In the next chapter, a literature review is presented on component modeling, charge tuning methods, system modeling and experimental work on faulted systems. The experimental setups along with uncertainty analysis and data filtering rules are described in Chapter 3. Chapter 4 discusses the mathematics and simplification of the semi-empirical component models. Fault models are also described in this chapter. The parameter estimation and the performance of component models are discussed in Chapter 5. Features of the system model, including its solution process with multiple component and fault models, its charge tuning method, its computational speed and its limitation, are discussed in Chapter 6. Chapter 7 analyzes the validity of the system by comparing the experimental data with simulation outputs. Chapter 8 describes the impacts of faults on various types of systems based on the simulation outputs, and Chapter 9 gives the conclusions of the dissertation.

CHAPTER 2. LITERATURE REVIEW

The literature review describes the previous work that is the basis of the development of the simulation within the dissertation. It summarizes the literature of different component models, heat transfer and pressure drop correlations, charge modeling and tuning, fault models and experimental work to study the fault impacts on vapor compression systems.

2.1 Component Modeling

In this dissertation, a system model is divided into a compressor model, heat exchanger models and an expansion valve model. The literature related to these models are presented here.

2.1.1 Compressor Modeling

AHRI Compressor Map

One common method to represent compressor performance is the 10-coefficient compressor map from AHRI standard 540 (AHRI, 2004). For refrigerant mass flow rate, the map is

$$\begin{aligned}\dot{m}_r = \; & c_0 + c_1 T_{r,evap} + c_2 T_{r,cond} + c_3 T_{r,evap}^2 + c_4 T_{r,evap} T_{r,cond} \\ & + c_5 T_{r,cond}^2 + c_6 T_{r,evap}^3 + c_7 T_{r,evap}^2 T_{r,cond} + c_8 T_{r,evap} T_{r,cond}^2 \\ & + c_9 T_{r,cond}^3\end{aligned} \tag{2.1}$$

and for compressor power consumption, it is

$$\begin{aligned}\dot{W}_{comp} = \; & c_0 + c_1 T_{r,evap} + c_2 T_{r,cond} + c_3 T_{r,evap}^2 + c_4 T_{r,evap} T_{r,cond} \\ & + c_5 T_{r,cond}^2 + c_6 T_{r,evap}^3 + c_7 T_{r,evap}^2 T_{r,cond} + c_8 T_{r,evap} T_{r,cond}^2 \\ & + c_9 T_{r,cond}^3.\end{aligned} \tag{2.2}$$

The method is widely supported by manufacturers and other modeling entities mainly because of its ease to use. It can be constructed by linear regression with manufacturers' data and can be programmed into electronic devices easily. It also represents the compressor performance accurately within the range of training data. Its problem of unreliable prediction outside the applicability domains is addressed by using a large operation range for the data.

However, in conditions which were not examined by compressor manufacturers, empirical compressor maps fail to predict the correct performance. One potential scenario is the simulation of system performance with evaporator fouling and charge leakage. These cases usually involve an evaporating temperature much lower than the minimum evaporating temperature at design, and compressor manufacturers may not have examined the situation. If their compressor maps are used to predict the performance, a large error can occur.

In addition, some manufacturers consider compressor maps as proprietary materials and do not make them available. The maps of these compressor can only be obtained from the system data collected in the laboratory. If the range of experimental data is not large, the compressor model will not accurately predict the performance in many feasible scenarios.

Polytropic Compressor Model

A polytropic compressor model (Kuehn et al., 1998) assumes a polytropic compression process as shown in

$$\frac{P_{r,comp,out}}{P_{r,comp,in}} = (\frac{\rho_{r,comp,out}}{\rho_{r,comp,in}})^{n_{poly}}. \tag{2.3}$$

By assuming polytropic processes during the compression and re-expansion processes in the compressor, the refrigerant mass flow rate can be analytically

related to the displacement volume, the pressure ratio and the volumetric efficiency using

$$\dot{m}_{r,comp} = \rho_{r,comp,in} f_{comp} V_{comp} \eta_{v,comp} \tag{2.4}$$

and

$$\eta_{v,comp} = 1 + F_{clear}(1 - (\frac{P_{r,comp,out}}{P_{r,comp,in}})^{n_{poly}}). \tag{2.5}$$

An expression estimating compressor power consumption can be derived assuming a reversible polytropic process to form

$$\dot{W}_{comp} = \frac{n_{poly}}{n_{poly} - 1} f_{comp} V_{comp} \eta_{v,comp} P_{r,comp,in}((\frac{P_{r,comp,out}}{P_{r,comp,in}})^{(\frac{n_{poly}-1}{n_{poly}})} - 1). \tag{2.6}$$

Although the model is analytical and few parameters are needed, the model has not accounted for other factors such as heat loss and internal leakage.

Semi-empirical Compressor Models

Jähnig et. al. (Jähnig et al., 2000) proposed the use of isentropic processes of ideal gas instead of polytropic processes to model the compressor behavior. In the model, the specific heat ratio of refrigerant at compressor suction is used to replace the polytropic coefficient. An overall isentropic efficiency, which is calculated from an empirical relationship with the evaporating temperature, is added to the power consumption model in Eqn. (2.6) to account for the irreversibilities and heat loss to

the surroundings. The model was validated with experimental data from 21 hermetic reciprocating compressors.

Kim and Bullard (Kim and Bullard, 2002) assumed an isentropic process to build a model similar to the polytropic compression model. Semi-empirical equations were built based on Newton's law of cooling and linear equations, and the isentropic efficiency of the compressor and hence the compressor discharge temperature were estimated. The model was validated with data from five different types of compressors running with different refrigerants.

Winandy et al. (Winandy et al., 2002) took another approach by splitting up the compressor power consumption into two parts: isentropic compression at designated volume ratio and change of power consumption as a result of operation at non-designated volume ratio. The paper argues that the compressor works at an optimal isentropic efficiency at a designated volume ratio and operation at other volume ratios degrades the isentropic efficiency. The parameters were learned from data of a scroll compressor and maximum errors obtained for mass flow rate, compressor power consumption and compressor discharge temperature were 3.5%, 3% and 5K respectively.

Zakula et al. (Zakula et al., 2011) modified the model from Jähnig et al. (Jähnig et al., 2000) with the addition of a model of back leakage loss to estimate the mass flow rate. The modeling methodology was validated based on quasi-steady state testing data from a 2.5kW ductless heat pump system. Root mean square errors at 1.57% and 3.88% were obtained for the mass flow rate and power predictions respectively.

2.1.2 Heat Exchanger Modeling

Heat exchanger models describe how heat is transferred from one stream of fluid to another as a result of the geometry of the heat exchanger, the flow rate and the properties of the streams. In this dissertation, only air-to-refrigerant heat exchangers are of interest, and only literature on how to model air-to-refrigerant condensers and evaporators are presented in this section.

ε-NTU Method

The ε-NTU method describes heat transfer rate across a heat exchanger by multiplying a heat exchanger effectiveness by the maximum possible heat transfer rate based on the conditions of the streams. The effectiveness is a function of a capacity ratio and a number of transfer units (NTU) that depend on the mass flow rates, the specific heat capacity of the streams and the geometry of the heat exchanger. Functions of the effectiveness have been analytically derived for different types of heat exchangers. For example, Kays and London (Kays and London, 1964) presented a relationship for a crossflow heat exchanger effectiveness from an analytical solution of a differential equation governing the temperature profiles of the streams. As the solution of the ε-NTU method is explicit, it can estimate heat exchanger performance with little computational effort.

Finite Segment Method

Finite segment methods divide heat exchangers into smaller segments and solve them individually in order to predict the heat exchanger performance more accurately than other models such as ε-NTU method. They divide the heat exchanger according to the flow distribution and heat exchanger geometry. For example, Shen (Shen, 2006) divided each refrigerant circuit of a heat exchanger into 10 segments and each segment was evaluated by ε-NTU method. The method also adjusts the air inlet conditions to each segment according to their locations within the heat exchanger. The heat transfer rates solved at each segment are summed up to estimate the performance of the heat exchanger. Zhou et al.(Zhou et al., 2007) considers each tube circuit of an air-to-water heat exchanger individually and used ε-NTU method to solve each segment.

Moving Boundary Method

The moving boundary method was first suggested to study the dynamics inside a heat exchanger (MacArthur and Grald, 1989). Since it divides a heat exchanger into two to three sections according to the phase of refrigerant and conducts lumped analyses on each section, it has the potential to predict steady-state performance of heat exchangers quickly. DOE/ORNL Mark VII Model (DOE/ORNL, 2005) uses the method to solve heat exchanger performance in their vapor compression system model. ACHP (Bell, 2010) uses the same idea to estimate steady state performance of

air-to-refrigerant heat exchangers. Zakula et al. (Zakula et al., 2011) used the same approach to estimate the performance of a condenser in a mini-split system. The trend shows that the method becomes more popular for heat exchanger modeling than the conventional finite segment method.

Wet Coil Analysis

One critical consideration for an air-to-refrigerant evaporator is the analysis of wet and dry coil conditions. A wet coil is an evaporator coil covered with a film of liquid water from condensation of water vapor in air. It occurs when the coil surface temperature is lower than the dewpoint in the air. In this scenario, not only the sensible heat is transferred to the refrigerant stream, but latent heat is also transferred as a result of the condensation of water vapor. To capture the effect, McQuiston et al. (McQuiston et al., 1989) suggested to estimate the heat transfer rate of wet coils by a modified Log Mean Temperature Difference (LMTD) method - a method used to estimate heat exchanger performance without humid air. The air temperature in the LMTD method is replaced by the air-water mixture enthalpy according to their humidity, and the coil surface and refrigerant temperature are replaced by the saturated air-water mixture enthalpy at their temperature. The model is solved with the enthalpy values to predict the heat transfer rate of the wet-coil heat exchangers.

Partial-wet-partial-dry Method

Some evaporators may have part of their coils covered by water only, and the heat transfer rate of the dry surface may be predicted incorrectly if wet coil analysis is applied to the entire evaporator coil. Braun (Braun, 1989) suggested modeling multi-row crossflow cooling coils as counterflow heat exchangers and divided the heat exchanger into dry and wet section according to the surface temperature. An air-to-refrigerant evaporator was considered as two counterflow heat exchangers modeled with the ε-NTU method. It was segmented by a point where the surface temperature equals the dewpoint of the inlet air as shown in Figure 2.1.

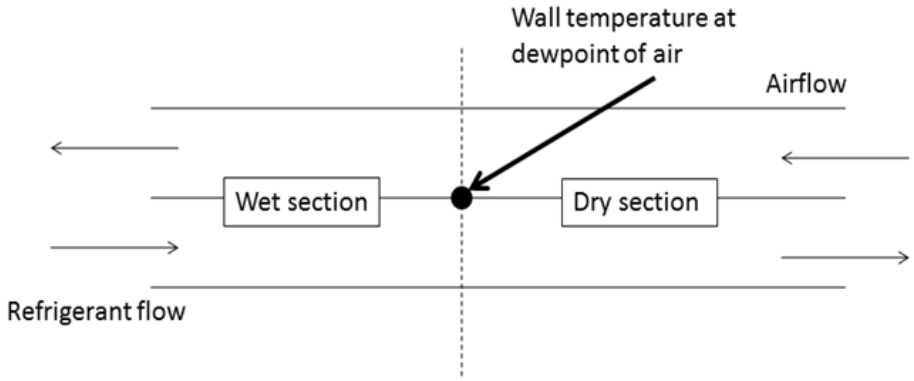

Figure 2.1. Schematic of partial-wet-partial-dry method.

An iterative solver was needed to solve the location of the segmentation and the performance of the heat exchanger. This method allows the coils to be analyzed in situations between the completely dry and wet analysis and enhances the accuracy of the air-to-refrigerant evaporator model.

2.1.3 Expansion Valve Modeling

At the expansion valve, liquid or two-phase refrigerant is expanded rapidly from a high-pressure regime into a two-phase condition at a lower pressure. A simple model of the refrigerant mass flow rate of an expansion valve is

$$\dot{m}_{r,EXV} = C_{EXV,\text{discharge}}\sqrt{2\rho_{r,EXV,in}(P_{r,EXV,in} - P_{r,EXV,out})}. \tag{2.7}$$

The expansion valve model in Eqn. (2.7) is insufficient to simulate the change of refrigerant mass flow rate of different types of valves with different input variables. Models of different types of expansion valve, including fixed orifice, thermostatic expansion valve and electronic expansion valves, were developed to estimate the refrigerant mass flow rate of expansion valves accurately and are described in this section.

Fixed Orifice Model

Aaron and Domanski (Aaron and Domanski, 1990) measured the subcooled flow of refrigerant R22 through a short tube and suggested a correlation for the mass flow rate similar to Eqn. (2.7). The formula uses an empirical equation which adjusts the pressure difference in Eqn. (2.7) based on subcooling, saturation pressure, length of the tube, inner diameter of the tube, inlet pressure and outlet pressure. Another empirical factor is introduced to Eqn. (2.7) to account for the effect of chamfer depth

of the short tube on the refrigerant mass flow rate. Kim and O'Neal (Kim and O'Neal, 1994) extended the applicable range of the model by conducting more experiments outside the range of data from the previous experiments (Aaron and Domanski, 1990) and modified the original model with a similar set of inputs and outputs.

A similar approach was used (Payne and O'Neal, 1999) to estimate the refrigerant mass flow rate of a short tube. The investigators neglected the relationship between the chafer depth and the mass flow rate and focused on the effect of two-phase refrigerant entering the valve on its mass flow rate. A two-phase flow adjustment factor, which is an empirical equation based on upstream thermodynamic quality, length of the tube, inner diameter of the tube and saturated vapor and liquid density, was made. The product of the adjustment factor with the refrigerant mass flow rate from a single-phase flow equation estimates the mass flow rate when a two-phase refrigerant flow enters the valve.

Singh et al. (Singh et al., 2001) studied the mass flow rate of refrigerant R-134a entering orifice tubes and constructed a correlation for both single-phase and two-phase flow. The single-phase flow in the model is considered in a similar manner as other methods. However, when it predicts flows with two-phase refrigerant at its inlet, it predicts the mass flow rate of a choked flow and a Fanno flow and adds them together to predict the total mass flow rate.

Payne and O'Neal (Payne and O'Neal, 2004) used a Buckingham-PI method to identify the necessary dimensionless groups and constructed an expansion valve mass flow rate model dependent on the dimensionless groups. To simplify the model,

statistical backward elimination was employed to reduce the number of parameters. The same technique was used to create an empirical equation of a two-phase flow adjustment factor. The analysis was conducted with data on different types of orifice with refrigerant R12, R134a, R502, R22, R407C and R410A.

Thermostatic Expansion Valve Model

Thermostatic expansion valve (TXV) is an expansion device that controls the evaporator outlet superheat by changing its opening area mechanically in response to the temperature at the evaporator outlet and the evaporating pressure. A TXV with an internal equalizer controls its opening based on the refrigerant pressure at the outlet of the valve, and a TXV with an external equalizer controls its opening based on the refrigerant pressure at the outlet of the evaporator.

TXV models are built on FXO models by adding equations to simulate the change of opening area in TXVs. Lenger (Lenger, 1998) modeled a TXV by estimating its flow rate at its maximum opening area and the opening area of the TXV separately. The valve was tested in experiments with its maximum opening under a variety of conditions to estimate the mass flow rate. The opening area was estimated by the details of its opening mechanisms such as spring control by the Hooke's law. The two models were combined together to form a TXV model.

Li and Braun (Li and Braun, 2008) proposed a method to model a TXV with catalog data only. A quadratic relationship between the superheat and the valve opening was found after simplifying the mathematics of the opening mechanism of

a TXV. The superheat corresponding to the maximum opening was estimated from common practices and the manufacturer's specification. A nonlinear model and a linear model of refrigerant mass flow rate were formed. Results show a maximum deviation of 6.9% for the nonlinear model and 8.82% for the linearized model.

Hariharan and Rasmussen (Hariharan and Rasmussen, 2010) presented a dynamic model of a TXV with a linear equation to describe the change of the discharge coefficient with the pressure difference across the TXV bulb. A lumped capacitance method was used to model the dynamic response of the TXV opening to temperature changes at the TXV bulb. The change of the TXV mass flow rate with time was modeled based on the dynamic changes of its opening.

Electronic Expansion Valve Model

Electronic expansion valve (EEV) is an expansion valve of which the opening is controlled by electronic signals, and controllers can change its opening according to different control setpoint. Park et al. (Park et al., 2001) created an EEV model by changing the diameter of the opening of a short-tube orifice model to simulate the operation of a multi-type inverter air conditioner. Shanwei et al. (Shanwei et al., 2005) built two correlations for the discharge coefficients of the EEVs after conducting experiments on six different electronic expansion valves with three refrigerants (R22, R407C and R410A): one from the Buckingham PI theorem and the other one by fitting polynomials. Park et al. (Park et al., 2007) and Qifang et al. (Qifang et al., 2007) developed their discharge coefficient correlations from previous models with

additional considerations: Park et al. (Park et al., 2007) considered the effect of surface tension of refrigerant, and Qifang et al. (Qifang et al., 2007) considered the effect of viscosity and size of taper. Zhifang et al. (Zhifang et al., 2008) used only two dimensionless groups - one on surface tension and the other one on viscosity - in their discharge coefficient correlation. Liang et al. (Liang et al., 2009) , unlike the previous models, suggested an empirical correlation to adjust $P_{r,EXV,out}$ in Eqn. (2.7) to estimate the mass flow rates of four different EEVs.

2.1.4 Heat Transfer Coefficient, Fin Efficiency and Pressure Drop Correlations

Although heat transfer coefficient, fin efficiency and pressure drop correlations were not used directly for any calculation in the current work, they feature the important parameters to be included in an inverse heat exchanger model and are the bases of the semi-empirical models of heat transfer rate and pressure drop in different components.

Single-phase Heat Transfer Coefficient Correlation

Both refrigerant and air flow as a single-phase fluid in air-to-refrigerant heat exchangers. Single-phase refrigerant flows are considered as internal turbulent flow, and air flows are considered to be external and turbulent. For single-phase refrigerant internal flow, the Dittus-Boelter equation (Incropera et al., 2007) gives a

simple correlation between Nusselt number, Reynolds number and Prandtl number as shown by

$$Nu = 0.023 Re^{4/5} Pr^{0.4}. \tag{2.8}$$

The empirical correlation is a simple product of Reynolds number and Prandtl number with some exponents and is one of the simplest models in heat transfer correlations. However, it suffers from a maximum error of 25%. Gnielinksi (Incropera et al., 2007) built another correlation which is dependent on friction factor and is valid for a larger range of Reynolds and Prandtl numbers. Though the equation form is more complex than Eqn. (2.8), the maximum error is reduced to 10%.

Air usually flows across a bank of tubes externally in air-to-refrigerant heat exchangers. Grimison (Incropera et al., 2007) proposed a correlation between Nusselt, Reynolds and Prandtl numbers for 10 or more rows of coils as shown in

$$Nu = CRe^{m}. \tag{2.9}$$

C and m in Eqn. (2.9) change according to the coil alignment and other geometry features of a heat exchanger. An additional correction factor to the correlation was given if the number of coils is less than 10.

Two-phase Heat Transfer Coefficient

Two-phase internal flow always occurs in the refrigerant stream in the condenser and the evaporator. Cavallini and Zecchin (Cavallini and Zecchin, 1971) constructed

a correlation for condensing flow with a modified Reynolds number from experiments with condensing flows of R113, R12 and R22. Shah (Shah, 1979) suggested a correction factor to the heat transfer coefficient of saturated liquid flow to estimate the heat transfer coefficient of condensing flows under a variety of working fluids and flow geometries.

For evaporating flows, Chen (Chen, 1966) assumed a combination of nucleate boiling and forced convection and formed a two-part correlation for the heat transfer coefficient of evaporating flow. Shah (Shah, 1982) formulated rules to distinguish the dominant mode of heat transfer in evaporating flow. Adjustment factors were calculated based on the dominant mode of heat transfer, and the evaporating flow heat transfer coefficient was estimated by changing a heat transfer coefficient of a saturated liquid flow with the adjustment factors. Kattan et al. (Kattan et al., 1998) built another correlation for evaporating flow based on Chen (Chen, 1966) to form an implicit correlation of the evaporating flow heat transfer coefficient.

Fin Efficiency

Fin efficiency is the ratio between the real heat transfer rate to the maximum heat transfer rate if the temperature of the entire fin surface is the same as the base temperature. Fin efficiency for straight fins (McQuiston et al., 1998) is estimated using Eqns. (2.10) and (2.11).

$$\eta_{fin} = \frac{tanh(C_{fin}L_{fin})}{C_{fin}L_{fin}} \tag{2.10}$$

and

$$C_{fin} = (\frac{2U_a}{k_{fin}y_{fin}})^{0.5}. \tag{2.11}$$

For more complicated geometries, other expressions may be needed. Kern and Kraus (Kern and Kraus, 1972) proposed a method with Bessel functions to calculate the fin efficiency of circular fins by solving ordinary differential equations governing the temperature distribution on circular fins. For circular finned tube coils, Schmidt (McQuiston et al., 1998) developed an empirical correlation similar to Eqn. (2.11). Kuehn et al. (Kuehn et al., 1998) recommended a modification to C_{fin} in Eqn. (2.11) for wet fins using air-water mixture enthalpy. Zhou et al. (Zhou et al., 2007) added a multiplier to the fin efficiency according to the temperature and air-water enthalpy to improve the sensible heat ratio accuracy by 7.2%. Hong and Webb (Hong and Webb, 1996) suggested another solution with Bessel functions for fin efficiency of wet fins.

Pressure Drop Calculation and Correlations

Pressure drop of various types of internal flows can be divided into accelerational loss, gravitational loss and frictional loss according to Wallis (Wallis, 1969). Accelerational loss is induced as the average velocity of the flow change with its density along the tube. Gravitational loss is the loss of pressure as it flows against gravity. Frictional loss is the loss of pressure because of friction between the flow and any solid medium.

Frictional loss is usually estimated by empirical pressure drop correlations because the friction on the tube wall is difficult to be represented analytically. Moody (Moody, 1944) charted the dependence of friction factor with relative roughness of tube surface and the Reynolds number for single-phase flows. Multiple literature such as Churchill (Churchill, 1977) recommended mathematical correlations of the friction factor to avoid the use of graphical methods. The pressure drop of two-phase flows is usually estimated by multiplying a two-phase flow multiplier by a liquid flow pressure drop. Lockhart and Martinelli (Lockhart and Martinelli, 1949) proposed a correlation of the multiplier from observations of flows of different liquid and air mixtures. Steam-water flow was examined in Martinelli and Nelson (Martinelli and Nelson, 1948) to construct another correlation in a different pressure regime.

2.1.5 Fan Power Consumption Modeling

Fan power consumption modeling is important because fan power consumption is significant to the total power consumption for ducted systems and can be affected by faults. Berry (Berry, 1963) drew the change of total pressure and power consumption with change of airflow for centrifugal fans with forwardly-curved blades, backwardly-curved blades and radial-tip blades. The power consumption of fans with forwardly-curved blades or radial-tip blades increases with airflow, but the power consumption of fans with backwardly-curved blades reaches a maximum before it wanes with increasing airflow. Osborne (Osborne, 1977) derived the theoretical relationship between the power consumption, airflow and rotational

speed of centrifugal fans and axial flow fans based on their geometry, continuity equation and conservation of momentum. However, the analysis did not cover the effect of other features such as housing or turbulence, and its results are very different from the ones in ASHRAE (ASHRAE, 2008) , which described the change of power consumption and the total pressure with airflow of different types of fans from observations in practice. These observations include the stall of fan power consumption at zero airflow, total pressure drop with increasing airflow for centrifugal fans, etc.

2.2 Charge Modeling

To estimate the amount of refrigerant inside a refrigerant circuit, the refrigerant density at different parts of the circuit is needed. While the density of single phase flow can be estimated based on the temperature and pressure according to thermodynamic relationships, the density of two-phase flow refrigerant cannot be calculated in the same way due to the different flow velocities of the liquid and vapor phases. To estimate the correct velocities, research has been conducted on void fraction models.

However, the estimation of velocities in two-phase flow is not the only ambiguity in the estimation of its refrigerant density. For instance, the refrigerant dissolved in the oil may not be estimated correctly if the oil solubility is not clearly known. Charge may be underestimated if the heat exchanger volume inside the heat exchanger bends or non-uniform tube geometry is not measured. Uneven flow distribution and different flow patterns in parallel tubes may distort the charge distribution and may induce

bias in the estimation. To offset the bias, charge tuning methods were suggested in the literature.

2.2.1 Void Fraction Modeling

Void fraction is a ratio of the volume of vapor to the total volume of a mixture. Rice (Rice, 1987) gave a thorough literature review of different types of void fraction models of two-phase flows and described the simplest void fraction model, the homogeneous model, as shown in

$$\gamma = (1 + \frac{1-x}{x}\frac{\rho_v}{\rho_l})^{-1}. \tag{2.12}$$

Common void fraction models include Zivi (Zivi, 1964), Baroczy (Baroczy, 1965) and Hughmark (Hughmark, 1962). Zivi (Zivi, 1964) built a void fraction model for an annular flow with minimum entropy production. Hughmark (Hughmark, 1962) introduced a multiplier dependent on quality, density ratio, mass flux, inner diameter, viscosity at saturation and void fraction from Eqn. (2.12) to Eqn. (2.12) to estimate void fraction of bubbly flow. Baroczy (Baroczy, 1965) developed a void fraction model dependent on quality and the Loackhart-Martinelli parameter in the Loackhart-Martinelli correlation (Loackhart and Martinelli, 1949). While all these models estimate a local void fraction in a two-phase flow, an average void fraction is needed to estimate the density of a two-phase flow quickly. Wedekind et al. (Wedekind et al., 1978) compared the mean void fractions from three void fraction

models. While the model from Fujie (Fujie, 1964) gave the best estimation based on experimental data on R12, only the Zivi (Zivi, 1964) model produced an analytical solution of average void fraction for both condensing and evaporating flow.

2.2.2 Charge Tuning

Rossi (Rossi, 1995) built a vapor compression model called ACMODEL which estimates the heat transfer rate, pressure drop and amount of charge inside a vapor compression system using a finite segment method and parameter tuning. The study recommended a one-point tuning method as shown in

$$M_{\text{mea}} - M_{\text{sim}} = \Delta M. \tag{2.13}$$

ΔM in Eqn. (2.13) is calculated at a rated condition while $M_{\text{simulated}}$ is a mass of refrigerant calculated by the sum and products of volumes and average density of refrigerant in the pipelines and heat exchangers. ΔM is the difference between the measured charge of an experimental case and the estimated charge of a simulation case with the same subcooling and environmental conditions as the experimental case. The determination of ΔM removes the bias in charge estimation, and simulations can be conducted in scenarios with unknown unsubcooling.

Shen (Shen, 2006) formulated the tuning factor in Eqn. (2.13) to be a linear equation of the length of the subcooled section in condensers and recommended

$$M_{mea} - M_{sim} = C_0 + C_1 L_{cond,sc} \tag{2.14}$$

for a two-point charge tuning method. The two methods were compared in Shen (Shen, 2006) to show that the two-point tuning method is more accurate. The two-point tuning method reduces the maximum errors for cooling capacity and compressor power consumption predictions from 4.2% to 0.9% and 7.0% to 2.0%, respectively for an R410A split system.

2.3 Fault Definition and Modeling

There are multiple types of faults that degrade the performance of vapor compression systems. They include non-standard charging, heat exchanger fouling, compressor flow fault, liquid line restriction and presence of non-condensables.

2.3.1 Non-standard Charging

Charge levels of systems in the field vary from the standard level. For example, charging more refrigerant in the field than the specification may cause overcharging, while improper sealing of refrigerant pipelines causes undercharging. Shen (Shen, 2006) defined the standard charge level as the amount of charge corresponding to the maximum coefficient of performance (COP) at the rated condition. Since the

optimal charge level varies with the length of liquid line of the system, technicians may accidentally overcharge or undercharge a system. If the pipes are not tightly sealed, refrigerant will leak from the system and the system performance wanes. The fault level is usually identified with the charge level specified by the manufacturer as

$$\text{Charge fault level} = \frac{M}{M_{\text{Specified by manufacturer}}} \tag{2.15}$$

because optimal charge level is difficult to know beforehand.

The charge fault level in Eqn. (2.15) is 1 when there is no fault in the charge level of the system. When the system is undercharged, the charge fault level is smaller than 1. If it is overcharged, it will be greater than 1.

Various simulations for effects of charge levels on system performance were conducted. Rossi (Rossi, 1995), Leroy (Leroy, 1997), Harms (Harms, 2002) and Shen (Shen, 2006) continued the development of ACMODEL and charge tuning methods to simulate the amount of charge inventory inside a system. The studies simulated the effect of charge level with different types of systems. FXO and TXV systems with capacities from 2 to 5 tons and refrigerant R22, R410A and R407C were examined. Tuning methods for each component were also applied according to experimental data collected to simulate the effect of charge on system performance more accurately.

2.3.2 Heat Exchanger Fouling

Heat exchanger fouling occurs when dust, leaves or other debris accumulate between the fins of heat exchangers or filters upstream of the heat exchanger. Fouling restricts airflow going through heat exchangers and reduces heat transfer rate across heat exchangers. Some studies of heat exchanger fouling, such as condenser fouling studies (Shen, 2006; Kim et al., 2009) defined the fault level by the area of blockage as shown in

$$\text{Heat Exchanger Fouling level in Experiments} = \frac{A_{flow,\text{with fouling}}}{A_{flow,\text{without fouling}}}. \tag{2.16}$$

The fault level of heat exchanger fouling is also quantified in terms of airflow reduction. Yang et al. (Yang et al., 2007) and Bell et al. (Bell et al., 2012) showed that fouling reduces the heat transfer coefficient of the heat exchanger by increasing the pressure drop across the heat exchanger and inducing a reduction of airflow. These studies showed that heat exchanger fouling can be simulated by reducing the airflow crossing the heat exchangers, and the heat exchanger fouling level can be defined with the airflow as shown in

$$\text{Heat Exchanger Fouling level in simulation} = 1 - \frac{\dot{m}_{a,\text{with fouling}}}{\dot{m}_{a,\text{without fouling}}}. \tag{2.17}$$

2.3.3 Compressor Flow Fault

Compressor flow fault usually occurs when the valves of the compressor cannot be closed completely. When the discharge valve only partially closes, the high-pressure superheated refrigerant vapor at compressor discharge flows back to the compression chamber, and hence the fault is also widely known as compressor valve leakage. If the valve at the suction side is not closed securely, the refrigerant in the compression chamber will be forced into the suction line by the pressure difference. Either fault reduces the amount of refrigerant compressed by the compressor in every cycle and the volumetric efficiency of the compressor. The reduction in mass flow rate is used to define the fault level as in

$$VL = 1 - \frac{\text{Refrigerant mass flow entering the condenser with fault}}{\text{Refrigerant mass flow entering the condenser without fault}}. \quad (2.18)$$

Kim et al. (Kim et al., 2009) and Yuill et al. (Yuill et al., 2013) used this definition to define the fault level of compressor valve leakage.

Another definition was used by Breuker (Breuker, 1997) and Kim (Kim, 2013) based on an analogy of valve leakage to a compressor bypass. When compressor valves cannot be closed tightly, some refrigerant will flow backwards across the compressor. This raises the compressor suction superheat and reduces the refrigerant flow out of the compressor. This effect can be simulated by adding a bypass from the compressor discharge to the compressor suction, and the refrigerant mass flow rate entering the condenser will be the difference between the refrigerant mass flow rate leaving the

compressor and the refrigerant mass flow rate entering the bypass. The fault level can then be defined by a bypass mass flow bp ratio as

$$bp = \frac{\text{Refrigerant mass flow rate entering the bypass}}{\text{Refrigerant mass flow rate leaving the compressor}}. \tag{2.19}$$

Breuker (Breuker, 1997) used ACMODEL to simulate the effect of compressor valve leakage by reducing the volumetric efficiency of the compressor and the mass flow rate of the compressor from the ordinary case without modeling the increase of suction refrigerant enthalpy due to the refrigerant flow from the compression chamber to the suction line.

2.3.4 Liquid Line Restriction

A liquid line restriction is caused by sediments within a liquid line filter, drier or at the inlet to the expansion valve. It induces a larger pressure drop than the designated value and may lead to a larger power consumption of the compressor.

Breuker (Breuker, 1997) developed ACMODEL to simulate the effect of liquid line restriction by reducing the inner diameter of liquid line. The reduction of the inner diameter increases the Reynolds number for the same mass flow rate and the friction factor. This increases pressure drop across the liquid line, leading to a lower evaporating temperature and higher compressor power consumption.

Breuker (Breuker, 1997) defined liquid line restriction level as shown in

$$LL = \frac{\text{Pressure drop across restriction}}{\text{Original pressure drop from condenser outlet to evaporator inlet}}. \quad (2.20)$$

2.3.5 Presence of Non-condensable

When non-condensables such as air are accidentally mixed with refrigerant inside the system, the system no longer operates with the pure refrigerant. Since the systems are designed to operate with refrigerant having a specific set of thermodynamic properties for optimal efficiency and the non-condensables distort the thermodynamic properties of refrigerant in the system, the efficiency of the system may reduce.

As non-condensables condense nor dissolve in refrigerant, they can be assumed to accumulate in the vapor section in the condenser and hot gas line. With the compressor pumping vapor refrigerant to the high-pressure regime and the liquid refrigerant at condenser blocking non-condensables for further recirculation, the air accumulates at the compressor discharge and the superheated region in the condenser.

The fault level is defined in Kim et al. (Kim et al., 2009) by

$$NC = \frac{\text{Mass of non-condensables in the system}}{\text{Mass of non-condensables filling up the system at standard condition}} \quad (2.21)$$

where the standard condition in this case is defined with temperature at 26.7°C and pressure at 101kPa.

The denominator of Eqn. (2.21) is the maximum possible mass of non-condensable present in the system. In the field, non-condensables in the system is caused by poor vacuum procedure before the refrigerant charging process. In the most severe situation, technicians do not evacuate the system and air at atmospheric pressure fills up the system before charging. The maximum mass of the non-condensables inside the system is calculated by the volume inside the system and the specific volume of the non-condensables at atmospheric pressure.

Bendapudi (Bendapudi, 2002) simulated the effect of non-condensables in a dynamic chiller model by assuming accumulation of non-condensables at the highest point in the chiller. Ideal gas behavior is assumed for both non-condensables and refrigerant to calculate the partial pressure of the non-condensables through Dalton's law. The non-condensable partial pressure is added to the refrigerant pressure to simulate the effect of increased high-side pressure in the system when the condenser is the highest point in the chiller.

2.4 Experimental Studies on Fault Impacts on Vapor Compression Cycle

Breuker (Breuker, 1997) tested the effect of compressor valve leakage, liquid line restriction, condenser and evaporator fouling and charge leakage on a 3-ton R22 packaged FXO system in order to evaluate the performance of an FDD tool. The effects of the faults on evaporating temperature, compressor suction superheat,

condensing temperature, condenser subcooling, compressor discharge temperature, air temperature difference across the condenser and air temperature difference across the evaporator were measured. Their effects on system performance are tabulated in Table 2.1.

Table 2.1. Observations with significant changes under different faults according to Breuker (Breuker, 1997).

Fault	Variables with significant increase	Variables with significant decrease
Compressor valve leakage	Evaporating temperature	Superheat
Liquid line restriction	Superheat and compressor discharge temperature	N/A
Condenser fouling	Condensing temperature and air temperature difference across condenser	N/A
Evaporator fouling	Air temperature difference across evaporator	Compressor discharge temperature
Charge leakage	Superheat and compressor discharge temperature	Evaporating temperature

Harms (Harms, 2002) tested a 5-ton R22 packaged TXV system with different charge levels to compare the effects of different void fraction models on the system performance at the rated condition (outdoor temperature at 35°C, indoor dry-bulb temperature at 26.7°C, indoor dewpoint at 15.77°C and refrigerant charge level at 5.61kg) after charge tuning and system-level tuning. It was found that the combined use of the Yashar et al. model (Yashar et al., 2001) for separated flow, and Taitel and Barnea model (Taitel and Barnea, 1990) for slug flow gave the best prediction in COP and capacity with deviations at 0.68% and 0.00% respectively under the rated

condition. The accuracy was followed by Zivi model (Zivi, 1969) with deviations of COP and capacity from the experiment at 0.73% and 0.00% respectively. The deviations of the system performance using other void fraction models were also small, showing that the choice of void fraction models has negligible effect to the accuracy of the estimation of the system performance when charge tuning is employed.

Shen (Shen, 2006) tested a 3-ton R410A packaged FXO system, a 3-ton R410A split FXO system, a 3-ton R410A split TXV system and a 5-ton R407C packaged FXO system with evaporator and condenser fouling and non-standard charging. It was found that a higher level of fouling leads to a higher degree of maldistribution of refrigerant flow in parallel refrigerant circuits. The two-point charge tuning method in Eqn. (2.14) was also tested with the experimental results, and it predicted performance better than the one-point tuning method in Eqn. (2.13) at seriously overcharged or undercharged conditions. For example, for the R410A packaged system tested at ambient temperature 53°F, the use of the two-point charge tuning method reduced the subcooling prediction error by 2K to less than 0.5K at 130% charge level. The accuracy of the superheat prediction by the two-point tuning method was improved by 5K to an error smaller than 5K at 75% charge level.

Kim et al. (Kim et al., 2009) imposed different faults on a 2.5-ton R410A split TXV system in a laboratory to study fault impacts on vapor compression systems with TXVs. The variables identified to be influenced significantly by faults were tabulated in Table 2.2.

Table 2.2. Observations with significant changes under different faults according to Kim et al. (Kim et al., 2009).

Fault	Variables with significant increase	Variables with significant decrease
Compressor valve leakage	Evaporating temperature	Condensing temperature
Liquid line restriction	Superheat and compressor discharge temperature	N/A
Condenser fouling	Condensing temperature and compressor discharge temperature	N/A
Evaporator fouling	Air temperature difference across evaporator	Evaporating temperature
Charge leakage	N/A	Subcooling and evaporating temperature
Presence of non-condensable	Subcooling and condensing temperature	N/A

Palmiter et al. (Palmiter et al., 2011) and Kim et al. (Kim et al., 2008) tested a 3-ton R410A split TXV system, a 3-ton R22 split FXO system and a 3-ton R22 split TXV system. Among them, the 3-ton R410A split system and the 3-ton R22 split TXV system contained a by-flow device at the TXV to conduct heating operation. It was tested with different indoor airflows and different charging levels. Cycling tests were also performed with one of the systems and it was found that the coefficient of degradation decreased with decreasing indoor unit airflow and increasing amount of charge.

2.5 Summary

In this chapter, a literature review of component models was given. Different techniques to model compressors, heat exchangers and expansion valves are presented in terms of their applications and advantages over each other. Common heat transfer coefficient correlations, pressure drop correlations and fin efficiency correlations are reviewed to describe how heat exchanger performance is accounted for mathematically. The development of void fraction models and charge tuning methods in literature is also covered. Experiments and theories related to different types of faults on vapor compression systems are also listed to examine how impacts of faults on vapor compression systems were studied.

CHAPTER 3. EXPERIMENTAL TESTING AND PERFORMANCE ANALYSIS

To conduct inverse modeling, performance data under a variety of faults and environmental conditions were collected from multiple systems. The systems included two types of compressors (reciprocating and scroll), two types of units (packaged and split), three different refrigerants (R22, R410A and R407C), three types of expansion valves (FXO, TXV and EEV) and a range of capacities between 2.5 ton and 5 ton. They were tested with different types of faults and a range of fault levels under a variety of environmental conditions. The choices of systems mainly serve the following aims.

1. To test the generality of the modeling approach:

   The project aims at devising an inverse modeling approach to simulate the operation of different types of systems. By applying the technique to data from different systems, the applicability and generality of the approach can be tested in terms of accuracy, robustness and speed of the simulation.

2. To test the validity of the component and fault models:

When the fault models are validated for data from different systems under a variety of conditions, the likelihood of omitting errors in these models is reduced. The validity of the component models with different systems can also be studied.

3.1 Specifications of Systems

The specifications of the cooling systems are shown in Tables 3.1 and 3.2.

Table 3.1. List of experimental setups with general information.

Index	Source	Size	Refrigerant	Type
I	Breuker (1997)	3	R22	Packaged
II	Harms (2002)	5	R22	Packaged
III	Shen (2006)	3	R410A	Packaged
IV	Shen (2006)	3	R410A	Split
V	Shen (2006)	3	R410A	Split
VI	Shen (2006)	5	R407C	Packaged
VII	Kim et al. (2009)	2.5	R410A	Split
VIII	Palmiter et al. (2011)	3	R410A	Split
IX	Kim et al. (2008)	3	R22	Split
X	Kim et al. (2008)	3	R22	Split
XI	Kim (2013)	4	R410A	Packaged

The capacity of the systems in Table 3.1 ranged from 2.5 ton to 5 tons and three different refrigerants (R22, R407C and R410A) were used. Table 3.2 shows that the list covers two types of units (packaged and split), two types of compressors (reciprocating and scroll) and three types of expansion valves (FXO, TXV and EEV).

Table 3.2. List of experimental setups with details of components.

Index	Compressor	Expansion valve	Accumulator
I	Reciprocating	FXO	No
II	Scroll	TXV	No
III	Scroll	FXO	No
IV	Reciprocating	FXO	No
V	Reciprocating	TXV	No
VI	Scroll	FXO	No
VII	Scroll	TXV	No
VIII	Scroll	TXV	No
IX	Scroll	FXO	Yes
X	Scroll	TXV	Yes
XI	Reciprocating	EEV	No

3.2 Instrumentation

All experiments listed in this section were conducted in psychrometric chambers. The temperature and humidity in each of the chamber were controlled to collect data of system performance at steady state.

The uncertainties of instrumentation of various setups are listed in Table 3.3, with system VII as the only system which uncertainties were reported in 95% confidence level.

Different setups used different instrumentation. For example, system I utilized a relative humidity sensor for quick response in transient tests while all other systems used a chilled mirror sensor for steady state measurement of humidity.

On the air side, the air temperature across the heat exchangers was measured using T-type thermocouples. The airflow was measured with ASHRAE standard 41.2

Table 3.3. Uncertainty of instrumentation in various systems.

System	I	II	III to VI	VII	VIII to X	XI
T-type thermocouples	±0.5K	±0.2K	±0.5K	±0.3K	±1.0K	±0.5K
Airflow pressure differential transducers	±7.5Pa	±7.5Pa	N/A	±1.0Pa	N/A	±1.0Pa
Air nozzle temperature measurement	±0.26K	±0.26K	N/A	±0.3K	N/A	±0.1K
Overall uncertainty of evaporator airflow	±0.91%	±0.91%	±0.91%	±1.67%	±10%	±10g/s
Chilled mirror sensors for air dewpoint	N/A	±0.2K	±0.2K	±0.2K	±0.2K	±0.2K
Relative humidity sensor	±3%	N/A	N/A	N/A	N/A	N/A
Refrigerant pressure transducer	±17kPa	±20kPa	±0.8%	±0.5%	±1.0%	±0.8kPa
Coriolis mass flowmeter for refrigerant flow	±1.20kg/h	±1.45kg/h	±0.6%	±1.0%	±0.5%	±0.27g/s
Power transmitters	±32W	±11W	±10W	±0.5%	±0.2%	±10W

(ASHRAE, 1987) nozzles, T-type thermocouples, resistance temperature detectors (RTDs) and pressure differential sensors.

On the refrigerant side, the refrigerant mass flow was measured with Coriolis mass flowmeters and the refrigerant temperatures were measured by T-type thermocouples. The refrigerant pressure was measured by pressure transducers. Compressor and fan power consumption were measured by power transmitters except system I where the fan power consumption was not measured.

3.3 Test Matrices

The environmental conditions of experiments are tabulated in Table 3.4.

Table 3.4. Testing condition of various systems.

Index	Number of data points	Indoor dry-bulb temperature [K]	Indoor dewpoint [K]	Outdoor dry-bulb temperature [K]
I	202	293.2-301.1	275.9-290.2	288.5-311.1
II	24	299.7-299.9	266.4-289.0	300.9-322.1
III	93	292.9-300.3	269.6-294.3	292.6-324.9
IV	49	299.2-300.2	280.8-295.2	300.7-325.9
V	70	297.2-301.2	268.5-294.0	301.3-326.0
VI	78	292.8-300.4	277.3-295.1	292.5-319.6
VII	123	294.0-300.0	273.2-289.2	294.1-310.9
VIII	48	299.6-300.0	272.4-288.9	300.8-324.8
IX	36	299.6-299.9	272.3-288.8	300.9-324.9
X	36	299.7-300.0	272.5-288.8	300.8-324.8
XI	69	298.5-304.6	283.5-289.0	296.9-316.8

The range of fault levels imposed on systems during experiments are shown in Table 3.5.

Table 3.5. Faulted condition tested in various systems (with fault level in heat exchanger defined by airflow, unless otherwise specified).

Index	Evaporator fouling	Condenser fouling	Charge variation	Compressor valve leakage	Liquid Line Restriction	Non-condensable
I	0 - 35%	0 - 56% (in blockage area)	90 - 100%	0 - 28% (bypass ratio)	0 - 20%	–
II	–	–	75 - 140%	–	–	–
III	0 - 17.63%	0 - 45%	60 - 130%	–	–	–
IV	0 - 42%	0 - 32%	50 - 110%	–	–	–
V	0 - 42.58 %	0 - 32%	50 - 110%	–	–	–
VI	0 - 10.55	0 - 45%	60 - 130%	–	–	–
VII	0 - 30%	0 - 35% (in blockage area)	70 - 130%	0 - 12%	0 - 20%	0 - 20%
VIII	0 - 40%	–	80 - 120%	–	–	–
IX	0 - 40%	–	75 - 125%	–	–	–
X	0 - 40%	–	75 - 125%	–	–	–
XI	0 - 50%	0 - 50% (in blockage area)	50 - 130%	0 - 30% (bypass ratio)	–	–

Breuker (Breuker, 1997) measured the transient behavior of the system under faulted conditions. Among all the transient results collected, 202 data points were obtained at quasi-steady state. Kim et al. (Kim et al., 2009) collected 123 data points under faulted conditions. The number of steady state data points for the other systems ranged from 24 to 93. Systems II, IV, IX and X were not tested with different indoor dry-bulb temperatures. All systems were examined with different indoor dewpoint and outdoor dry-bulb temperature.

Additional tests with different levels of presence of non-condensable were conducted with system XI. The test matrix was tabulated in Table 3.6.

3.4 Experimental Setup

General schematics of the experimental setups of systems I to X in Table 3.1 are drawn in Figures 3.1 and 3.2. The notations in Figure 3.1 are explained in Table 3.7. Table 3.8 shows which the measurement stations in Figure 3.1 were used in different experimental setups.

Table 3.8 shows that the position of air-side temperature and humidity sensors across the evaporator is different between systems. Its effect on the data analysis is discussed in Section 3.6.2.

System XI in Table 3.1 has a different experimental setup because the system has an outdoor air economizer. An outdoor air economizer is a damper installed in a packaged unit upstream to its evaporator, and can be opened to mix outdoor air with return air from the indoor environment at the air inlet of the evaporator. Should

Table 3.6. Test conditions for additional test performed on system XI.

Test No.	Indoor dry-bulb temperature [K]	Indoor dewpoint [K]	Outdoor dry-bulb temperature [K]	Presence of non-condensable [%]	EEV opening control strategy
1	300.0	287.6	297.3	0.0	$SH_{comp,out}$ = 8.94K
2	300.0	287.6	297.3	9.9	$SH_{comp,out}$ = 8.87K
3	299.9	287.8	297.0	20.2	$SH_{comp,out}$ = 9.00K
4	300.0	288.0	297.2	58.9	$SH_{comp,out}$ = 8.71K
5	299.8	287.8	297.1	98.0	$SH_{comp,out}$ = 8.66K
6	300.0	288.0	308.1	0.0	$SH_{comp,out}$ = 8.86K
7	300.1	288.1	308.1	9.9	$SH_{comp,out}$ = 8.92K
8	300.0	287.9	308.2	20.2	$SH_{comp,out}$ = 8.88K
9	300.0	288.1	308.2	58.9	$SH_{comp,out}$ = 8.94K
10	299.9	287.8	308.3	98.0	$SH_{comp,out}$ = 8.68K
11	299.9	287.8	308.4	98.0	Same EEV opening as Test 6

Table 3.7. Notation representation in Figure 3.1.

Notation	Sensor
T	Temperature sensor
P	Pressure transducer
D	Dewpoint sensor/ relative humidity sensor
M	Coriolis mass flowmeter

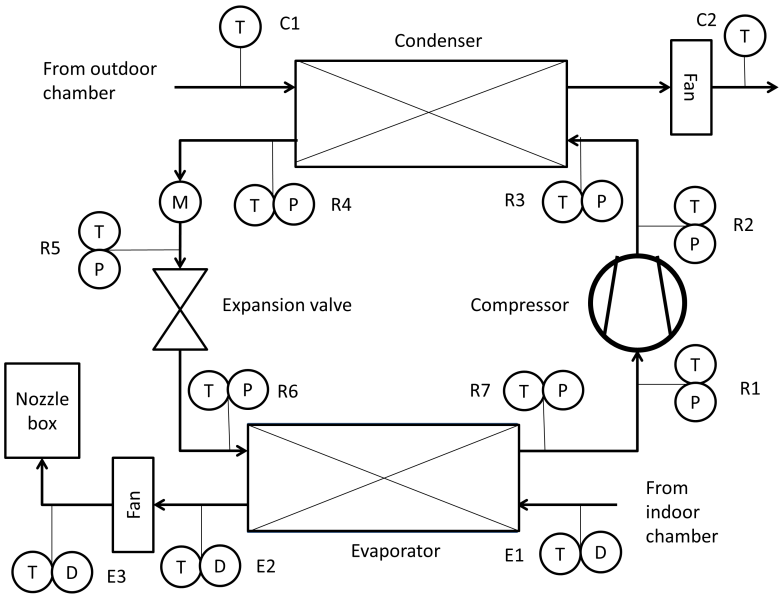

Figure 3.1. General experimental schematic of systems without an accumulator (systems I to VIII).

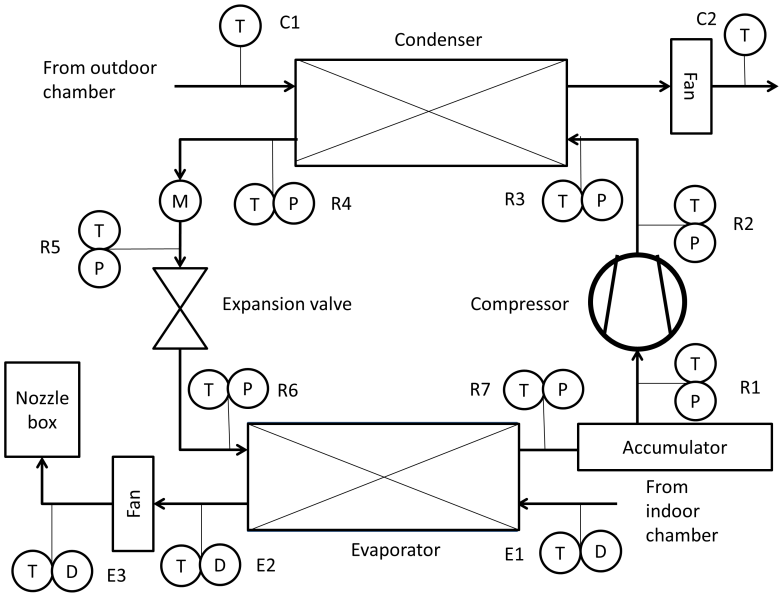

Figure 3.2. General experimental schematic of systems with accumulators (systems IX and X).

Table 3.8. Presence of sensors in Figure 3.1 for different systems.

System	I	II	III to VI	VII	VIII to X
C1	Yes	Yes	Yes	Yes	Yes
C2	Yes	Yes	Yes	Yes	Yes
R1	Yes	Yes	Yes	Yes	Yes
R2	Yes	Yes	Yes	Yes	Yes
R3	Yes (only T)	Yes	Yes	Yes	Yes (only T)
R4	Yes (only T)	Yes	Yes	Yes	Yes
R5	Yes (only P)	–	Yes	Yes	Yes
R6	–	–	Yes	Yes (only T)	–
R7	Yes (only T)	Yes	Yes	Yes	Yes
E1	Yes	Yes	Yes	Yes	Yes
E2	Yes	Yes	Yes	–	–
E3	–	–	–	Yes	Yes

the outdoor air have a higher air-water enthalpy than the return air, COP of the system can be enhanced. It also helps indoor ventilation by pulling outdoor fresh air into the indoor environment. Grids of thermocouples were installed at the return air inlet, evaporator air inlet and in the outdoor chamber to measure their dry-bulb temperature. Chilled mirror sensors were also installed at the return air duct and outdoor chamber to measure their dewpoints as shown in Figure 3.3.

Although the economizer in Figure 3.3 was closed in all experiments conducted under the conditions in Table 3.6, its effect on the system performance was considered because Hjortland (Hjortland, 2014) showed that the outdoor air enters the evaporator air inlet through leakages at the outdoor air economizer when the economizer is fully closed and the humidity ratio at the evaporator air inlet always depends on the humidity ratios in the indoor and outdoor environment. However,

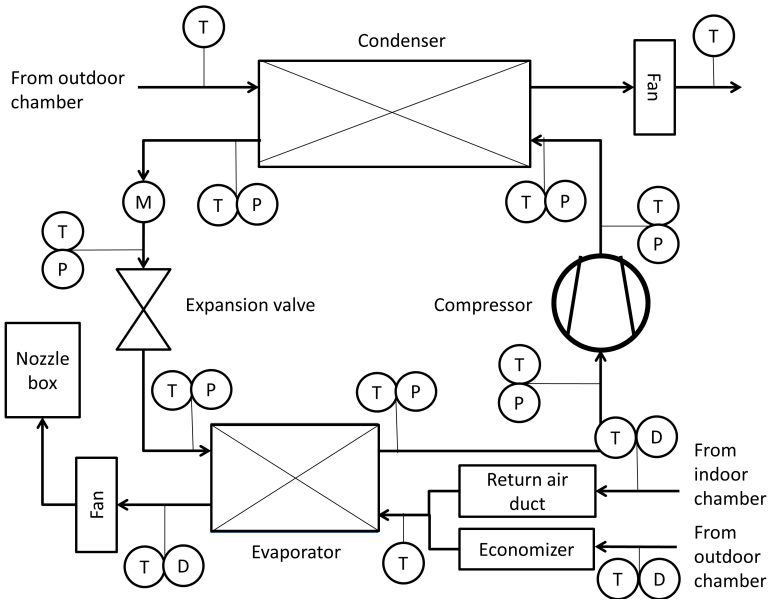

Figure 3.3. Experimental schematic of system XI with notations following Figure 3.7.

the economizer stratifies air at the evaporator inlet, and no well-mixed air is available at the evaporator inlet for any valid humidity measurement. The humid air properties at the evaporator inlet hence cannot be measured directly but are estimated by solving the continuity equation of airflow

$$\dot{m}_{a,evap} = \dot{m}_{a,econ} + \dot{m}_{a,ret}, \tag{3.1}$$

the continuity equation of humidity

$$\dot{m}_{a,evap}\omega_{a,evap} = \dot{m}_{a,econ}\omega_{a,econ} + \dot{m}_{a,ret}\omega_{a,ret} \tag{3.2}$$

and the energy equation

$$\dot{m}_{a,evap}h_{a,evap} = \dot{m}_{a,econ}h_{a,econ} + \dot{m}_{a,ret}h_{a,ret} \tag{3.3}$$

simultaneously.

Eqns. (3.1) and (3.2) are obtained by considering the airflow and humidity flow leaked through the economizer $\dot{m}_{a,econ}$ and $\dot{m}_{a,evap}\omega_{a,evap}$. Eqn. (3.3) represents the assumption of adiabatic mixing of air at the evaporator inlet. By solving these equations simultaneously with the temperature readings at the evaporator inlet, the humidity ratio and enthalpy of air-water mixture at the evaporator air inlet can be calculated. This gives all the information related to the properties of air at the evaporator inlet of system XI.

The experimental data of the tests on system XI with conditions in Table 3.6 and schematic in Figure 3.3 are tabulated in the Appendix.

3.5 Fault Testing

To create repeatable experiments for fault impacts, experimental setups were modified to simulate the faults in systems in the field according to the mechanisms of the faults.

3.5.1 Non-standard Charging

The tests of non-standard charge were conducted by operating the systems with different levels of charge inventory. The refrigerant in the system was recovered, followed by evacuating the system, before re-charging between tests at different refrigerant charge levels to ensure the correctness of the amount of charge inside the system.

3.5.2 Heat Exchanger Fouling

Multiple methods were developed to simulate heat exchanger fouling in laboratory experiments. One method is to block the upstream or downstream airflow path to induce a larger flow resistance. The cross-sectional area of the blockage was recorded to define the fault level on the system.

The other method was based on the study from Yang et al. (Yang et al., 2006) and Bell et al. (Bell et al., 2012). They showed that the decrease of airflow is the main impact of fouling on heat exchangers, and the fouling effect is equivalent to the reduction of airflow. Some experiments achieved the lower airflow by reducing the rotational speed of the heat exchanger fan. Other experiments ran the booster fan for airflow measurement at a lower rotational speed to adjust the pressure drop across the heat exchangers and to simulate the fouling effects on the heat exchangers.

3.5.3 Compressor Flow Fault

Breuker (Breuker, 1997) proposed a testing method for different levels of compressor flow fault by adding a refrigerant bypass across the compressor. This method was utilized in various studies (Breuker, 1997 ; Kim et al., 2009 ; Kim, 2013) to simulate the fault by a metering valve next to the compressor as shown in Figure 3.4.

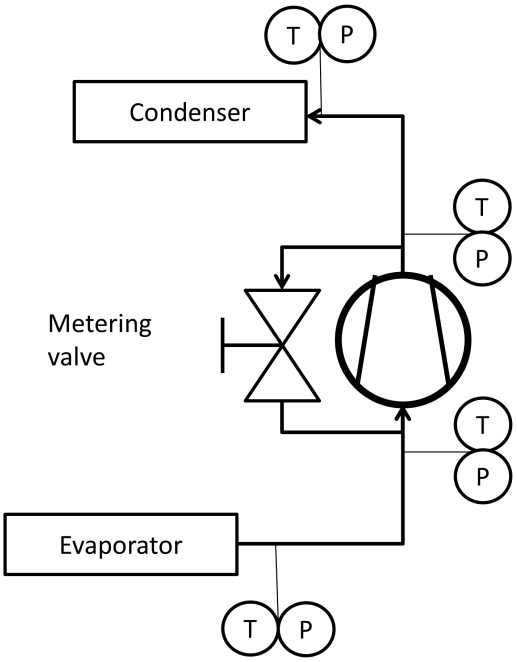

Figure 3.4. Modification on setup for testing of compressor valve leakage.

In Figure 3.4, some refrigerant flow from the compressor is allowed to bypass the rest of the system, depending on the opening of the adjustable valve. The rest of the system only receives a portion of refrigerant flow from the compressor as if the compressor is damaged and cannot supply the refrigerant flow to the system

normally. The bypass also directs refrigerant back to the suction port of the compressor as if refrigerant is leaking from the discharge port to the suction port through the compression chamber in a real compressor valve leakage scenario. The simulated level of fault is controlled by the opening of the metering valve, which determines the bypass mass flow rate and the fault level.

3.5.4 Liquid Line Restriction

Breuker (Breuker, 1997) and Kim et al. (Kim et al., 2009) used a restriction valve upstream to the expansion valve to simulate the effect of liquid line restriction. As the valve closes, the flow resistance and the pressure drop across the liquid line increases. This simulates the increase of pressure drop as a result of sediment accumulation or a clogged filter in a real liquid line restriction scenario.

The liquid line restriction level is based on the pressure drop of the non-faulted scenario as shown in Eqn. (2.20). To define a liquid line restriction fault level, a baseline case with the restriction valve fully opened is needed. The pressure drops in a non-faulted case and a faulted case under the same environmental condition defines the liquid line restriction level in the faulted experiment.

3.5.5 Presence of Non-condensable

Kim et al. (Kim et al., 2009) conducted tests with non-condensable gas. Although the non-condensable in the field consists of air and humidity, to have a controllable

and repeatable mass of non-condensables, dry nitrogen at a known mass was charged into the system to create the fault in the laboratory. Refrigerant was charged into the system afterwards to create a refrigerant-non-condensable mixture inside the system.

In the tests conducted according to the conditions in Table 3.6, a different testing method was used. The tests were conducted by first evacuating the refrigerant circuit. Then, a vacuum gage was connected to the system to measure the pressure inside the system. Nitrogen was injected to the system until a pressure corresponding to the required fault level is reached. Assuming nitrogen as ideal gas, the pressure value was calculated by

$$\begin{aligned} &\text{Required pressure} \\ = \ &\text{(Average temperature of temperature sensors on refrigerant in K)} \\ &\text{x } \tfrac{(101.325[\text{kPa}])}{299.85[\text{K}]}\text{x Non-condensable fault level.} \end{aligned} \tag{3.4}$$

After charging nitrogen into the system, refrigerant was charged into the system and the system was operated to carry out the experiment. This avoided the systematic bias on the fault level because of the difficulty to estimate the volume of heat exchanger from the geometrical information of the refrigerant circuit.

3.6 Data Filtering and Heat Exchanger Performance Analysis

Experimental results may contain data which violate physical principles due to sensor inaccuracy or other effects, and unrealistic data points must be removed to maintain the accuracy of the data analysis. One typical example of this type of data

removal can be found in AHRI standard 210/240 (AHRI, 2008) which states that a test result is only valid if the refrigerant-side and air-side heat transfer rates of a system are measured within 6% from each other. Test results violating this rule may be suspected to be violating the first law of thermodynamics and are considered to be invalid.

Besides the issue of validity, a consistent definition and calculation method of heat transfer rates of the heat exchangers were needed though the systems were tested with different experimental setups as illustrated in Table 3.8. This helps to train the models with the experimental data by the same method.

3.6.1 Data Filtering by System Heat Loss

All systems operating at steady state must obey the conservation of energy given by

$$\dot{Q}_{evap} + \dot{W}_{comp} - \dot{Q}_{cond} - \dot{Q}_{HL} = 0. \tag{3.5}$$

The heat loss in Eqn. (3.5) should be positive in cooling systems. Any data, exhibiting a heat gain larger than 8% of the evaporator heat transfer rate of the system, were rejected because of their unrealistic behavior.

3.6.2 Data Filtering by Condenser Fouling with No Subcooling

In experiments of cooling systems, the airflows across condensers were not measured and were estimated by energy balance as shown in

$$\dot{m}_{a,cond} = \frac{\dot{m}_r(h_{r,cond,in} - h_{r,cond,out}) + \dot{W}_{cond,fan}}{c_{pa,cond}(T_{a,cond,out} - T_{a,cond,in})}. \tag{3.6}$$

However, in cases when the subcooling at the condenser outlet was less than 3K, no reliable measurement of refrigerant mass flow rate could be obtained, and the condenser airflow could not be estimated from the information of the data point. In cases without condenser fouling, the condenser airflow could be estimated from other cases without condenser fouling because the airflow in these cases should be the same. However, in condenser fouling scenarios with condenser subcooling less than 3K, the airflow could not be estimated from other data points and the data point was rejected because of unknown condenser airflow rate.

3.6.3 Data Filtering by Liquid Line Heat Loss

The liquid line in all systems should be dissipating heat as it must be hotter than the surroundings, and the liquid line heat loss can be calculated by

$$\dot{Q}_{\text{liquidline},HL} = \dot{Q}_{evap} + \dot{m}_r(h_{r,cond,in} - h_{r,evap,out}) - \dot{Q}_{cond} \tag{3.7}$$

derived from the conservation of energy.

As Eqn. (3.7) requires refrigerant mass flow rate for calculation, the elimination was conducted after estimating the parameters of the compressor mass flow rate model according to Section 5.1.4. With the compressor mass flow rate model, the refrigerant mass flow rate of the data points with compressor suction superheat greater than 1K can be estimated, and the liquid line heat loss of these data points can be estimated by Eqn. (3.7).

Similar to system heat loss, the heat loss of the liquid line may be negative due to experimental uncertainty. In this project, the data points yielding a ratio heat loss from Eqn. (3.7) to condenser heat transfer rate lower than -0.01 should be removed.

3.6.4 Results of Data Filtering

A cumulative frequency diagram of ratios of system heat loss to evaporator heat transfer rate of all eleven cooling systems is plotted in Figure 3.5.

38 cases in Figure 3.5 show a heat gain which is higher than 8% of their evaporator heat transfer rate. These cases could not be accounted for by physical models and were rejected.

A cumulative frequency diagram of ratios of liquid line heat loss to condenser heat transer rate is plotted in Figure 3.6.

Figure 3.6 contains fewer data points than Figure 3.5 because of the removal of data points with excessive heat gain and inability to obtain valid refrigerant mass flow rate for data points with small superheat and subcooling. By setting the threshold

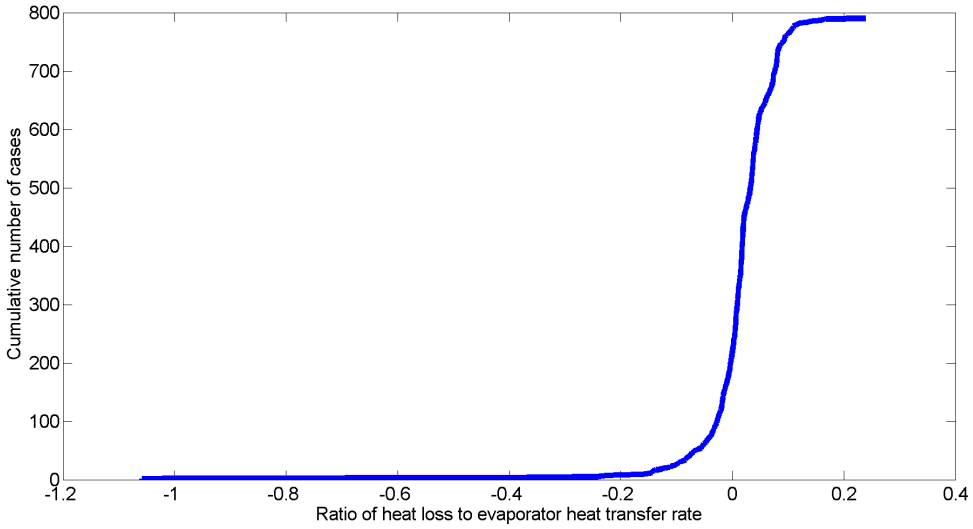

Figure 3.5. Cumulative frequency diagram of ratio of system heat loss to evaporator heat transfer rate of all 11 cooling systems.

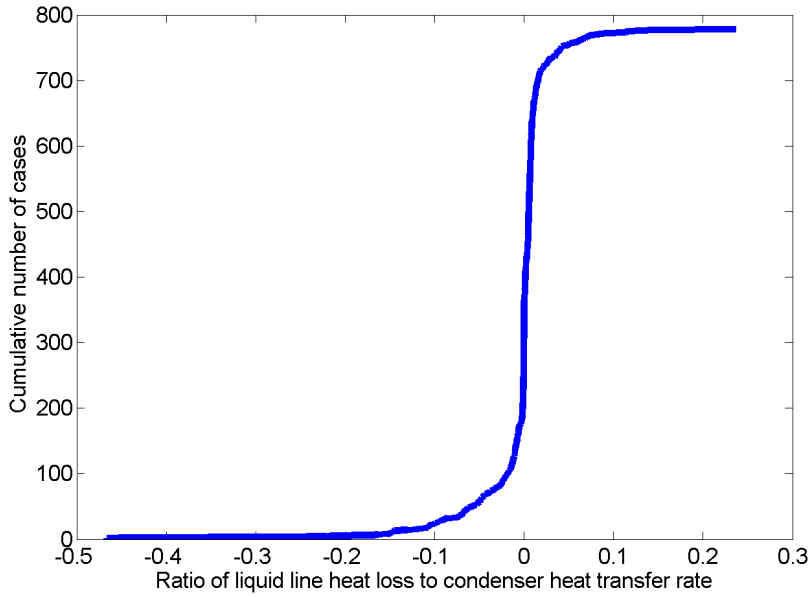

Figure 3.6. Cumulative frequency diagram of ratio of liquid line heat loss to condenser heat transfer rate of all 11 cooling systems.

ratio of rejection due to unrealistic liquid line heat gain to -0.01, 131 data points were removed according to Figure 3.6.

The number of data points filtered is shown in Table 3.9 which shows that sufficient data for parameter estimation is left in the systems according to Table 3.9 for inverse modeling.

Table 3.9. Number of filtered data points in each system.

System	Total number of data points measured at steady state	Number of data points filtered because of unknown condenser airflow	Number of data points filtered because of unrealistic system heat gain	Additional number of data points eliminated because of unrealistic liquid line heat gain
I	202	20	19	65
II	24	0	2	2
III	93	1	2	9
IV	49	3	0	5
V	70	0	0	1
VI	78	0	9	10
VII	123	9	0	10
VIII	48	0	0	1
IX	36	0	0	0
X	36	0	1	4
XI	69	5	5	24

3.6.5 Condenser Heat Transfer Performance Calculation

Since condenser airflows were not measured, they were estimated from the refrigerant-side measurement of the condensers by

$$\dot{Q}_{cond} = \dot{m}_r(h_{r,cond,in} - h_{r,cond,out}). \tag{3.8}$$

When condenser subcooling was greater than 3K, the measured refrigerant mass flow rate were used to calculate the heat transfer rate by Eqn. (3.8). The condenser airflow in these cases were obtained by Eqn. (3.6). The average airflow in the non-faulted conditions was used to calculate the condenser heat transfer rates of cases with no condenser fouling and subcooling below 3K by

$$\dot{Q}_{cond} = \overline{\dot{m}_{a,cond}}_{NF} c_{pa} (T_{a,cond,out} - T_{a,cond,in}) - \dot{W}_{cond,fan}. \tag{3.9}$$

Since most experiments of cooling systems did not involve the measurement of humidity in the outdoor environment and humidity has negligible effect to the heat dissipation of condensers, the outdoor environment was assumed to be dry and Eqn. (3.9) does not contain any humidity variables.

3.6.6 Evaporator Heat Transfer Performance Calculation

The measurement of evaporator heat transfer rate was more comprehensive than the condenser because both refrigerant-side and air-side cooling rates might be available. When the condenser subcooling was greater than 3K and the evaporator superheat was greater than 1K, the refrigerant mass flow rate, expansion valve inlet enthalpy and evaporator outlet enthalpy were valid. The evaporator heat transfer

rates were calculated, assuming adiabatic refrigerant flow from expansion valve inlet to evaporator inlet, by

$$\dot{Q}_{evap} = \dot{m}_r(h_{r,evap,out} - h_{r,evap,in}). \tag{3.10}$$

The airflows of these cases were calculated from an energy balance to equalize the air-side heat transfer rate with the refrigerant-side because previous experience shows that the airflow rate is easier to be subjected to systematic bias than other air-side measurement. In systems where sensor location E2 was used in Figures 3.1 and 3.2 and system XI, the temperature at the outlet of the system was measured upstream of the fan, and the evaporator airflows were given by

$$\dot{m}_{a,evap} = \frac{\dot{Q}_{evap}}{h_{a,evap,in} - h_{a,evap,out}}. \tag{3.11}$$

In systems where sensor location E3 in Figures 3.1 and 3.2 was used to measure the properties of air at the evaporator outlet, a fan power offset was needed and the airflow was given by

$$\dot{m}_{a,evap} = \frac{\dot{Q}_{evap} - \dot{W}_{evap,fan}}{h_{a,evap,in} - h_{a,evap,out}}. \tag{3.12}$$

In cases where the refrigerant-side evaporator heat transfer rates were unavailable, the air-side evaporator heat transfer rates were used. Due to the discrepancy between the measured air-side and refrigerant-side heat transfer rate, the measured airflow in

these scenarios were adjusted. The adjustment was done by a ratio of air mass flow rate as

$$\dot{m}_{a,evap} = \dot{m}_{a,evap,mea} \frac{\overline{\dot{m}_{a,evap,r-adj}}}{\overline{\dot{m}_{a,evap,a-adj}}}. \tag{3.13}$$

where $\overline{\dot{m}_{a,evap,r-adj}}$ is the average evaporator airflow calculated from the refrigerant-side heat transfer rate and $\overline{\dot{m}_{a,evap,a-adj}}$ is the average measured evaporator airflow of cases with valid refrigerant-side heat transfer rate.

With the adjusted evaporator airflow from Eqn. (3.13), the evaporator heat transfer rates could be calculated. For data points with air temperature at evaporator outlet measured upstream of the evaporator fan, the evaporator heat transfer rates were given by

$$\dot{Q}_{evap} = \dot{m}_{a,evap}(h_{a,evap,in} - h_{a,evap,out}). \tag{3.14}$$

The sensible heat ratios (SHR) in these cases were given by

$$SHR = \frac{\dot{m}_{a,evap} c_{pa,evap} (T_{a,evap,in} - T_{a,evap,out})}{\dot{Q}_{evap}}. \tag{3.15}$$

Evaporator heat transfer rates in systems with air outlet condition measured downstream of the evaporator fan were given by

$$\dot{Q}_{evap} = \dot{m}_{a,evap}(h_{a,evap,in} - h_{a,evap,out}) + \dot{W}_{evap,fan}. \tag{3.16}$$

The SHR in these cases were given by

$$SHR = \frac{\dot{m}_{a,evap} c_{pa,evap} (T_{a,evap,in} - T_{a,evap,out}) + \dot{W}_{evap,fan}}{\dot{Q}_{evap}} \tag{3.17}$$

because the fan power only contributes to sensible heat but not the latent heat of the heat transfer.

The COP of systems can then be defined as

$$COP = \frac{\dot{Q}_{evap}}{\dot{W}_{comp}} \tag{3.18}$$

The standard conditions defined with the analysis are tabulated in Table 3.10.

Table 3.10. Data for standard condition.

System	Airflow ratio in Eqn. (3.13)	Standard evaporator airflow $[m^3/s]$	Standard condenser airflow $[m^3/s]$	Standard charge level [kg]
I	1.15e+00	6.72e-01	1.63e+00	2.72e+00
II	1.03e+00	1.05e+00	1.97e+00	5.22e+00
III	9.98e-01	5.87e-01	1.07e+00	3.24e+00
IV	9.57e-01	5.70e-01	1.26e+00	2.86e+00
V	9.44e-01	5.63e-01	1.22e+00	2.91e+00
VI	1.06e+00	1.04e+00	1.85e+00	2.59e+00
VII	1.05e+00	5.09e-01	1.02e+00	4.65e+00
VIII	1.02e+00	6.00e-01	1.71e+00	3.36e+00
IX	1.00e+00	7.36e-01	1.43e+00	3.32e+00
X	1.04e+00	6.59e-01	1.46e+00	4.94e+00
XI	9.55e-01	6.70e-01	1.80e+00	5.22e+00

3.7 Uncertainty Analysis

Uncertainty propagation for output Y being a function of X can be calculated (Kline and McClintock, 1953) by

$$(\delta Y)^2 = \Sigma_i (\frac{\partial Y}{\partial X_i} X_i)^2. \tag{3.19}$$

The average uncertainties of important measurements in various systems are shown in Table 3.11.

The uncertainty of evaporator air-side heat transfer and SHR of system I is generally higher than other systems except system VIII because the uncertainty of the relative humidity sensors used in system I is higher than the chilled mirror sensors used in other systems. Since systems VIII, IX and X accounted for the uncertainty of duct leakage loss with a flow hood, it reported a ±10% uncertainty from the evaporator air flow and the uncertainties of the measurements of system VIII became higher than all the other systems.

3.8 Summary

The chapter summarizes the different experimental procedures and results of tests of faulted vapor compression systems. Different types of systems tested under different environmental and faulted conditions were chosen to prove the generality of the modeling approach and test various fault models. The detailed specification

Table 3.11. Uncertainty of major performance indicators of various systems.

System	Evaporator refrigerant-side heat transfer [%]	Evaporator air-side heat transfer [%]	SHR [%]	Condenser refrigerant-side heat transfer [%]	Condenser air mass flow rate [%]	SH [K]	SC [K]
I	0.74%	7.43%	9.89%	0.70%	10.75%	0.56	0.60
II	0.48%	3.92%	4.24%	0.42%	2.78%	1.04	0.54
III	0.89%	6.54%	8.64%	0.91%	6.67%	0.57	0.61
IV	0.93%	6.46%	8.85%	0.88%	7.33%	0.57	0.60
V	0.96%	6.07%	8.01%	0.89%	7.83%	0.57	0.61
VI	0.86%	6.61%	8.35%	0.79%	6.17%	0.56	0.61
VII	1.07%	4.70%	5.75%	1.06%	5.03%	0.34	0.37
VIII	1.50%	5.82%	9.32%	0.78%	10.85%	0.60	0.66
IX	1.01%	5.66%	12.45%	0.66%	8.84%	0.59	0.66
X	1.04%	4.98%	11.31%	0.65%	9.16%	0.60	0.65
XI	1.32%	6.65%	8.25%	0.71%	10.61%	0.50	0.50

of the systems and the instrumentation in the experiments are documented and the test matrices for environmental conditions and fault levels are tabulated. The procedures to test different types of faults are introduced to generate repeatable experiments of the faulted scenarios. Data analysis to calculate the capacities for all systems are introduced for a consistent description of system performance among different experimental data. Data filtering rules and data analysis results are described to remove unrealistic or invalid data. Uncertainty analysis of different instruments is also discussed.

CHAPTER 4. MODEL DEVELOPMENT

To simulate fault impacts on a vapor compression cycle with inverse modeling, the cycle was divided into several semi-empirical component models. Fault models were also developed according to the fault mechanisms. This chapter depicts the methodology and the mathematics to develop the semi-empirical component models and the fault models. This chapter fulfills the following purposes:

1. To describe the development of semi-empirical component models and the physical principles involved;
2. To explain the mathematics of the fault models.

4.1 Overview of the System Model

To form a generic model for different types of vapor compression systems, minor components in different systems were grouped with major components to form a generalized vapor compression system. Major component models are shown in Figure 4.1: compressor model, hot gas line model, condenser model, liquid line model, expansion valve model, evaporator model and suction line model.

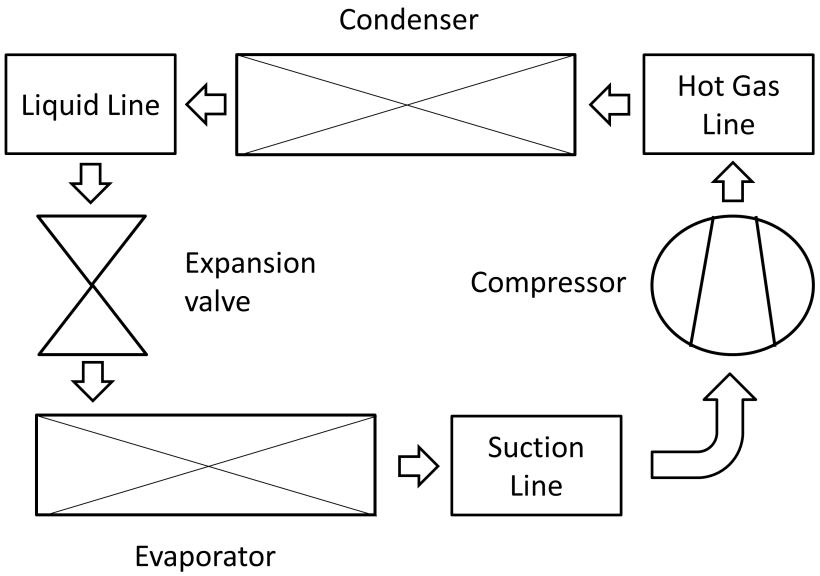

Figure 4.1. Schematic of component models in a cycle model.

The component models in Figure 4.1 were aligned according to the direction of refrigerant flow in the refrigerant circuit. Superheated refrigerant flows from compressor discharge to a hot gas line and dissipates heat from the hot gas line to the surroundings by forced convection of refrigerant and natural convection of air around the pipeline. It enters the condenser and condenses into a two-phase or subcooled state in the condenser by transferring heat to the airflow across the condenser. The heat dissipation process continues in the liquid line with a smaller heat transfer rate than the rate of the condenser. Entering an expansion valve, the refrigerant is expanded to a much lower pressure through an adiabatic process. The refrigerant then enters an evaporator where it absorbs heat from air by forced convection and is boiled to become superheated vapor. It is further heated up in a

suction line as heat is absorbed from the ambient. At compressor suction, refrigerant is compressed to superheated refrigerant and the vapor compression cycle is repeated.

To model systems with an accumulator at the compressor suction, an accumulator model is inserted between the suction line model and the compressor model as shown in Figure 4.2.

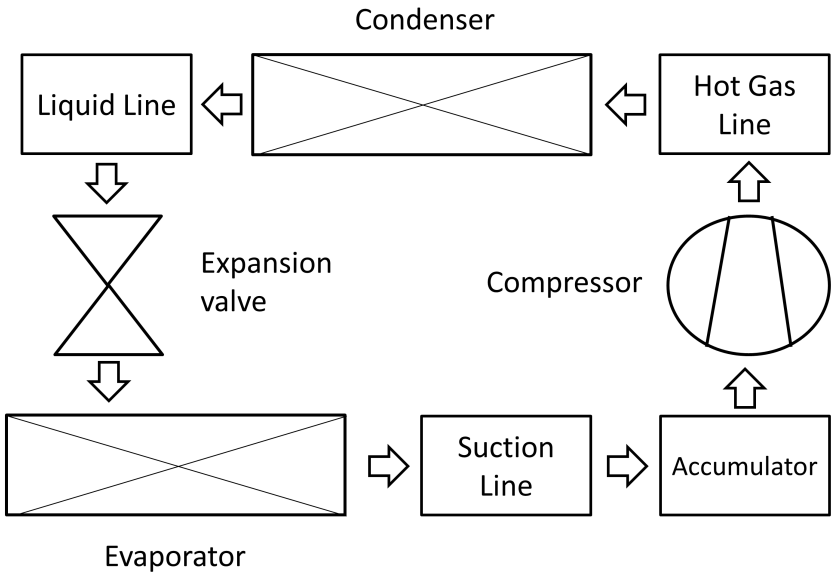

Figure 4.2. Schematic of component models in a cycle model with an accumulator model.

For systems with configuration in Figure 4.2, refrigerant passes through an accumulator first before entering the compressor from the suction line. If the refrigerant flow contains liquid and vapor, the liquid will be stored in the accumulator and only the refrigerant vapor will be expelled to the compressor. If

the refrigerant flow contains vapor only, the refrigerant flow will vaporize any liquid in the accumulator and will leave the accumulator in vapor form only. More details of the operation of an accumulator are discussed in Section 4.2.7.

Unlike models such as ACHP (Bell, 2010) where the evaporator outlet superheat is a user-defined quantity, the system model simulates the mass flow rate across expansion valves to predict the rapid change of superheat in faulted scenarios. Heat dissipation from pipelines was modeled to avoid biased estimation of heat transfer rates of heat exchangers. However, the pipeline between the evaporator and expansion valve was not modeled because of the unavailability of data at the inlet and exit of the pipeline.

4.2 Mathematical Modeling of Components

This section covers the mathematics of the component models, including the compressor model, the condenser model, the evaporator model, the expansion valve model and the refrigerant pipeline model. The parameters of the component models are divided into several types depending on the estimation method.

4.2.1 Types of Parameters

During the development of the component models, several types of parameters are identified as shown in Table 4.1.

Table 4.1. Types of parameters in component models.

Types of parameters	Estimation procedure
Existing empirical parameters	Inherited from the original literature
Unit-specific normalization parameters	By assigning rated values or averaging the experimental data
Regression parameters	By minimizing the difference between the model prediction and experimental data

Existing empirical parameters in Table 4.1 are parameters discovered by previous investigators and documented in other literature such as exponential coefficients in expansion valve models. To save the effort in estimating the coefficients again, these parameters remain unchanged.

The other type of parameters is the unit-specific normalization parameters. These parameters normalize inputs to the equations to prevent scaling issues during the estimation of regression parameters.

The last type of parameters is the regression parameter. These parameters are estimated by minimizing objective functions which depend on the difference between experimental measurements and model predictions.

4.2.2 Compressor Model

The semi-empirical compressor model for refrigerant mass flow rate and power consumption was originated from Zakula et al. (Zakula et al., 2011). The mass flow rate model remains the same as

$$\begin{aligned}\dot{m}_{r,comp} = \quad & \rho_{r,comp,in}(C_{comp,0} + C_{comp,1}(\tfrac{P_{r,comp,out}}{P_{r,comp,in}})^{(\frac{1}{n_{poly}})} \\ & + C_{comp,2}(P_{r,comp,out} - P_{r,comp,in})).\end{aligned} \tag{4.1}$$

The first two terms in Eqn. (4.1) account for the mass flow rate as a result of a reversible polytropic process, and the third term in Eqn. (4.1) accounts for internal leakages inside the compression chamber. The regression parameters in Eqn. (4.1) are $C_{comp,0}$, $C_{comp,1}$ and $C_{comp,2}$.

The power consumption model involves

$$\dot{W}_{comp}\eta_{comb} = \frac{n_{poly}}{n_{poly}-1}\frac{\dot{m}_{r,comp}}{\rho_{r,comp,in}}P_{r,comp,in}(1000[Pa/kPa])((\frac{P_{r,comp,out}}{P_{r,comp,in}})^{(\frac{n_{poly}-1}{n_{poly}})} - 1) \tag{4.2}$$

and

$$\eta_{comb} = C_{comp,3} + C_{comp,4}\exp(C_{comp,5}\frac{P_{r,comp,in}}{P_{r,comp,in,rated}}) + C_{comp,6}exp(C_{comp,7}\frac{P_{r,comp,out}}{P_{r,comp,in,rated}}). \tag{4.3}$$

It is the same as Zakula et al. (Zakula et al., 2011) except the terms with regression parameters $C_{comp,6}$ and $C_{comp,7}$.

Eqn. (4.2) explains the power consumption of refrigerant compression in reversible polytropic process, and Eqn. (4.3) accounts for the effect of refrigerant pressures on the irreversibilities. In the original model, only the first two terms in Eqn. (4.3) exist. To account for the effect of high pressure on the efficiency empirically, additional empirical terms with $C_{comp,6}$ and $C_{comp,7}$ were added to form Eqn. (4.3).

The heat loss of the compressor is modeled by

$$\dot{Q}_{HL} = C_{comp,8}(T_{r,comp,in} - T_{amb}) + C_{comp,9}(T_{r,sat}(P_{r,comp,out}) - T_{amb}) \tag{4.4}$$

and

$$h_{r,comp,out} = h_{r,comp,in} + \frac{\dot{W}_{comp} - \dot{Q}_{HL}}{\dot{m}_r} \tag{4.5}$$

with regression parameters from $C_{comp,8}$ and $C_{comp,9}$.

Eqn. (4.4) estimates the heat loss from the compressor that depends on the temperature difference between the refrigerant and the surrounding ambient. Since the effective surface temperature on the compressor for heat dissipation to the surroundings is unknown, both compressor suction temperature and refrigerant saturation temperature at compressor discharge are included in Eqn. (4.4) to estimate the heat loss of the compressor. The resultant compressor discharge enthalpy is calculated by Eqn. (4.5)

These equations can be generalized into an input and output diagram of the compressor model as shown in Figure 4.3.

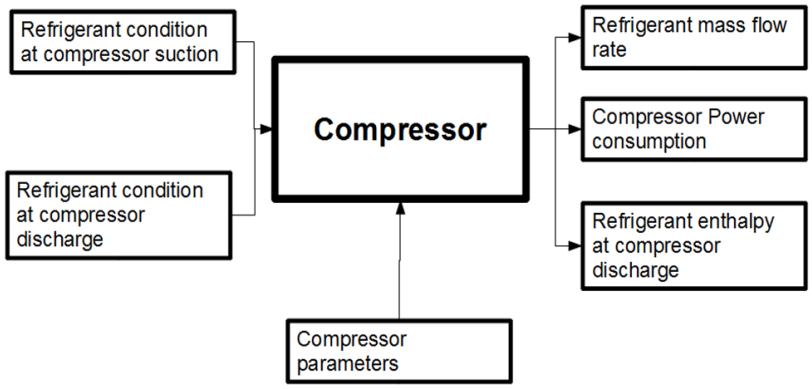

Figure 4.3. Compressor input and output diagram.

The input and output diagram shows that the model is implicit because the polytropic exponent in Eqns. (4.1) and (4.2) requires a compressor outlet temperature to estimate. The solution process of the cycle model to deal with the situation is discussed in Section 6.1.

4.2.3 Condenser Model

The condenser model is developed from the moving boundary method used in ACHP (Bell, 2010) with modification to achieve higher accuracy in estimating the system performance under faulted conditions. The general schematic of the condenser model is shown in Figure 4.4.

The condenser schematic in Figure 4.4 is an air-to-refrigerant crossflow condenser segmented by three sections according to the thermodynamics phases of refrigerant. Despite potential mal-distribution of refrigerant flow in a real condenser, the refrigerant flow is assumed to be flowing in a single circuit for

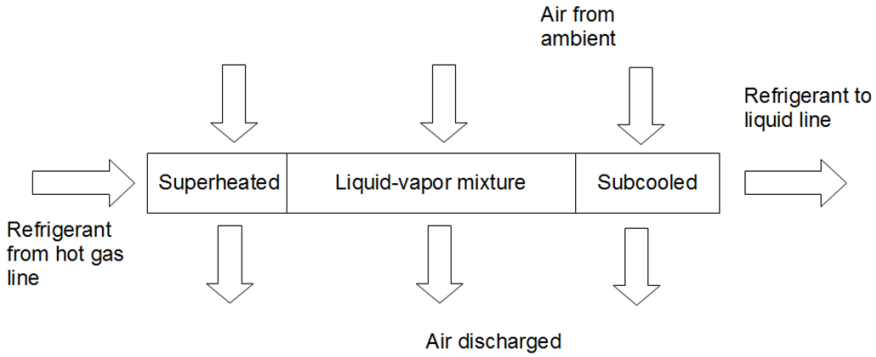

Figure 4.4. Condenser model schematic.

simplicity. The change of refrigerant pressure across the coil is also assumed to be negligible to its heat transfer rate and pressure drop. Since condensation of water vapor is impossible on the air side of a condenser coil, the air is assumed to be dry with dewpoint at 230K.

The resultant input and output diagram is shown in Figure 4.5.

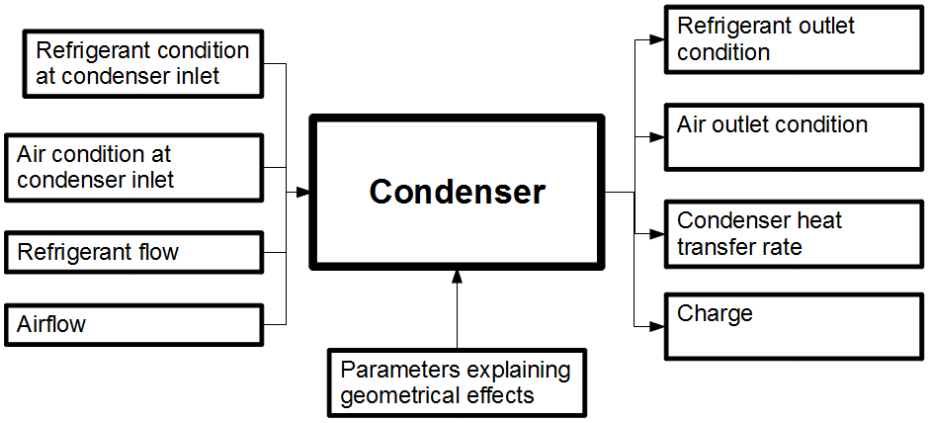

Figure 4.5. Condenser model input and output diagram.

Figure 4.5 shows that the mass flow rates, the air condition at the inlet and the refrigerant condition at the inlet are the inputs to the model, and the outputs of the model can be calculated by a heat transfer rate model, a pressure drop model and a charge estimation model.

By using moving boundary method, the heat transfer rate estimation of the condenser is separated into three different sections as shown in Figure 4.4. The heat transfer rate of the superheated section is estimated by assuming negligible effect of pressure drop on the calculation of the heat transfer rate and using the refrigerant mass flow rate, the enthalpy at the condenser inlet and the enthalpy of saturated vapor. After estimating the heat transfer conductance from the inputs and the parameters, the size of the superheated section is solved by the ε-NTU method. The size of the two-phase section is estimated based on a similar analysis by assuming the refrigerant at the inlet of the two-phase section be saturated vapor and the refrigerant at the exit of the two-phase section to be saturated liquid. The remaining section of the condenser is the subcooled section. The heat transfer rate of the subcooled section is estimated with the airflow across the condenser, the air properties at the condenser inlet, the size of the subcooled section, the heat transfer conductance of the subcooled section, the refrigerant flow and the refrigerant properties at the outlet of the two-phase section. If the combined size of the estimated two-phase and superheated sections exceeds the size of the condenser, the refrigerant will exit the condenser as a two-phase mixture. In this case, the heat transfer rate of the two-phase section is estimated in a similar calculation method as

the heat transfer rate of the subcooled section. The condenser heat transfer rate is calculated by summing up the heat transfer rates of all three sections. The calculation procedure is shown in Figure 4.6, and the mathematics involved is listed in Appendix A.

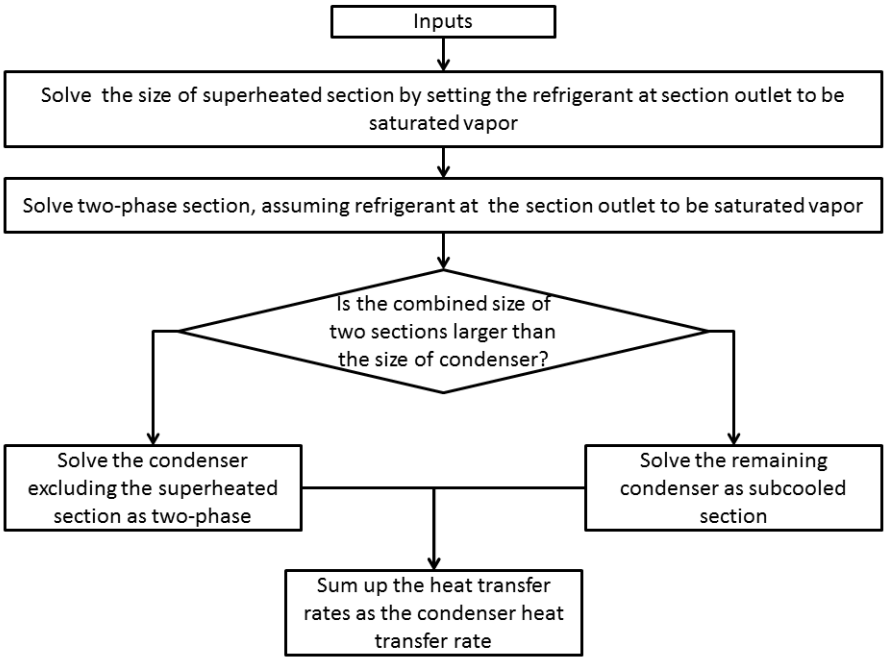

Figure 4.6. Solution procedure of refrigerant phase sections in the condenser model.

In forward models, the overall heat transfer conductance of each section is estimated based on the flow conditions and the geometry of the heat exchanger. While the flow conditions in inverse modeling are given, the geometry is unavailable. To study the effect of the geometry of heat exchanger on heat transfer conductance and create their semi-empirical models, the correlations of heat transfer coefficients and fin efficiency from the literature were investigated.

On the air side, Grimison (Incropera et al., 2007) provides a correlation between the Nusselt number (dimensionless heat transfer coefficient) and Reynolds number (dimensionless airflow) as Eqn. (2.9) for heat transfer coefficients of airflows across banks of horizontal tubes.

As air properties do not vary significantly in air conditioning applications, air properties were assumed to be constant and the air-side heat transfer conductance was simplified as

$$UA_{a,cond} = U_{a,cond,rated}(\frac{\dot{m}_a}{\dot{m}_{a,cond,rated}})^{n_{a,cond,rated}} A_{r,cond,rated}, \tag{4.6}$$

where $U_{a,cond,rated}$ and $n_{a,cond,rated}$ are regression parameters and $A_{r,cond,rated}$ and $\dot{m}_{a,cond,rated}$ are unit-specific normalization parameters.

On the refrigerant side, the heat transfer conductance of condensing flow is simply assumed to be directly proportional to the refrigerant mass flow rate because the refrigerant-side heat transfer conductance is not a dominant factor to the heat transfer rate across the two-phase section, and the refrigerant-side heat transfer conductance of the condensing flow is modeled by

$$UA_{r,cond,tp} = U_{r,cond,tp,rated}(\frac{\dot{m}_r}{\dot{m}_{r,rated}})A_{r,cond,rated}. \tag{4.7}$$

Eqn. (4.7) consists of $U_{r,cond,tp,rated}$ as a regression parameter and $\dot{m}_{r,rated}$ and $A_{r,cond,rated}$ as unit-specific normalization parameters.

For the single-phase flows, the development of the semi-empirical model of the heat transfer conductance started with Dittus-Boelter equation (Incropera et al., 2007) Eqn. (2.8) for internal turbulent flow. The Nusselt number, Reynolds number and Prandtl number in Eqn. (2.8) were expanded and the parameters related to geometry were grouped together. The final result is shown in

$$UA_{r,cond,1\phi} = \frac{U_{r,cond,1\phi,rated}}{K_{r,cond,1\phi}}\left(\frac{c_{pr,1\phi}\mu_{r,1\phi}}{k_{r,1\phi}}\right)^{0.4}\left(\frac{\dot{m}_r}{\mu_{r,1\phi}}\right)^{0.8}A_{r,cond,rated}. \tag{4.8}$$

Regression parameters in Eqn. (4.8) are $U_{r,cond,1\phi,rated}$ and $K_{r,cond,1\phi}$ and $A_{r,cond,rated}$ are unit-specific normalization parameters.

With Eqns. (4.6), (4.7) and (4.8), the heat transfer conductances of the air side and refrigerant side for different sections can be computed and the heat transfer rates of the condenser can be estimated according to Appendix A.

Pressure drop of refrigerant flow is estimated to avoid bias in the estimation of condensing pressure and the heat transfer rate in a cycle model. The development of pressure drop inverse models began with the definition of pressure drop friction factor as

$$\Delta P_{r,1\phi} = \frac{f_{\Delta P,r}}{2\rho_r D_{cond}}\left(\frac{4\dot{m}_r}{\pi {D_{cond}}^2}\right)^2 L_{r,1\phi}. \tag{4.9}$$

By grouping the geometrical terms, the pressure drop of the single-phase section can be written as

$$\Delta P_{r,cond,1\phi} = w_{r,cond,1\phi} C_{r,cond,\Delta P,1\phi} (\frac{\dot{m}_r^2}{\rho_{r,cond,1\phi}}) (\frac{\rho_{r,cond,1\phi,rated}}{\dot{m}_{r,rated}^2}) \tag{4.10}$$

with a regression parameter $C_{r,cond,\Delta P,1\phi}$ and unit-specific normalization parameters $\rho_{r,cond,\Delta P,1\phi}$ and $\dot{m}_{r,rated}$.

For pressure drop across the two-phase section which involves a two-phase friction factor multiplier with viscosity, a viscosity term was added to the single-phase pressure drop model to form the pressure drop equation for two-phase flow as

$$\Delta P_{r,cond,tp} = w_{r,cond,tp} C_{r,cond,\Delta P,tp,1} (\frac{\dot{m}_r^2}{\rho_{r,cond,tp}}) (\frac{\rho_{r,cond,tp,rated}}{\dot{m}_{r,rated}^2}) (\frac{\mu_{r,cond,v}}{\mu_{r,cond,v,rated}})^{C_{r,cond,\Delta P,tp,2}} \tag{4.11}$$

with regression parameters $C_{r,cond,\Delta P,tp,1}$ and $C_{r,cond,\Delta P,tp,2}$ and the unit-specific normalization parameters $\rho_{r,cond,tp,rated}$, $\dot{m}_{r,rated}$ and $\mu_{r,cond,v,rated}$.

Since the refrigerant mass flow rate throughout the system is the same at steady state and the density of refrigerant at the entrance of a condensing flow is much lower than that of the exit, the velocity of the condensing flow decreases significantly along its flow direction. This induces an accelerational component of pressure drop in the flow, and the accelerational effect to the pressure drop is modeled as

$$\Delta P_{r,cond,acc} = C_{r,cond,\Delta P,acc} (\frac{\dot{m}_r^2}{\dot{m}_{r,rated}^2}) \rho_{r,cond,rated} (\frac{1}{\rho_r(P_{r,cond,in}, h_{r,cond,out})} - \frac{1}{\rho_{r,cond,in}}), \tag{4.12}$$

with regression parameters $C_{r,cond,\Delta P,acc}$ and unit-specific normalization parameters $\dot{m}_{r,rated}$ and $\rho_{r,cond,rated}$. Eqn. (4.12) is made to be explicit to avoid iterative calculation by approximating the refrigerant outlet density by the refrigerant pressure at the condenser inlet and the refrigerant enthalpy at the condenser outlet.

The overall pressure drop equation is

$$\begin{aligned}\Delta P_{r,cond} = \; & w_{r,cond,sh} C_{r,cond,\Delta P,sh} \left(\frac{\dot{m}_r^2}{\rho_{r,cond,v}}\right)\left(\frac{\rho_{r,cond,rated}}{\dot{m}_{r,rated}^2}\right) \\ & + w_{r,cond,tp} C_{r,cond,\Delta P,tp,1} \left(\frac{\dot{m}_r^2}{\rho_{r,cond,tp}}\right)\left(\frac{\rho_{r,cond,rated}}{\dot{m}_{r,rated}^2}\right)\left(\frac{\mu_{r,cond,v}}{\mu_{r,cond,v,rated}}\right)^{C_{r,cond,\Delta P,tp,2}} \\ & + w_{r,cond,sc} C_{r,cond,\Delta P,sc} \left(\frac{\dot{m}_r^2}{\rho_{r,cond,l}}\right)\left(\frac{\rho_{r,cond,out,rated}}{\dot{m}_{r,rated}^2}\right) \\ & + C_{r,cond,\Delta P,acc} \left(\frac{\dot{m}_r^2}{\dot{m}_{r,rated}^2}\right) \rho_{r,cond,rated} \left(\frac{1}{\rho_r(P_{r,cond,in}, h_{r,cond,out})} - \frac{1}{\rho_{r,cond,in}}\right)\end{aligned} \tag{4.13}$$

with regression parameters $C_{r,cond,\Delta P,sh}$, $C_{r,cond,\Delta P,tp,1}$, $C_{r,cond,\Delta P,tp,2}$, $C_{r,cond,\Delta P,tp,sc}$ and $C_{r,cond,\Delta P,acc}$.

To model the effect of non-standard charging in the systems, the charge inside a condenser is estimated by

$$M_{cond} = (\rho_{r,cond,sh} w_{r,cond,sh} + \rho_{r,cond,tp} w_{r,cond,tp} + \rho_{r,cond,sc} w_{r,cond,sc}) V_{r,cond}. \tag{4.14}$$

The density of a single phase section can be solved with the inlet pressure and the average temperature across the section assuming thermodynamic equilibrium. However, in the two-phase section, void fraction models are needed to solve for the liquid and vapor volume in the two-phase flow and hence the average density. Harms (Harms, 2002) showed that the accuracy of charge inventory estimation is independent

of the choice of void fraction model after charge tuning and hence the simplest void fraction model can be used. Among the void fraction model examined by Harms (Harms, 2002), Zivi (Zivi, 1969) model is the simplest and the average density of the two-phase region in the condenser is estimated by

$$\overline{\gamma_{cond}} = -\frac{S_{\gamma,cond}(\ln(-\frac{1}{S_{\gamma,cond}(x_{r,cond,tp,out}-1)-x_{r,cond,tp}}) + 1 - x_{r,cond,tp,out}) + 1 - x_{r,cond,tp,out}}{(S_{\gamma,cond}^2 - 2S_{\gamma,cond} + 1)(1 - x_{r,cond,tp,out})}, \tag{4.15}$$

$$S_{\gamma,cond} = (\frac{\rho_{r,cond,v}}{\rho_{r,cond,l}})^{2/3} \tag{4.16}$$

and

$$\rho_{r,cond,tp} = (1 - \overline{\gamma_{cond}})\rho_{r,cond,l} + \overline{\gamma_{cond}}\rho_{r,cond,v}. \tag{4.17}$$

4.2.4 Evaporator Model

The evaporator model consists of three parts: the heat and mass transfer rate model, the pressure drop model and the charge model. The heat and mass transfer rate model estimates the heat transfer rate of the evaporator and the rate of condensation of humidity on the evaporator coil. Similar to the condenser model, the pressure drop model and the charge model require area ratios of the two-phase and the superheated sections from the heat and mass transfer rate model, and the heat and mass transfer rate model is solved prior to the estimation of the pressure drop and the amount of charge in the evaporator.

The heat and mass transfer rate model uses the moving boundary method from ACHP (Bell, 2010) and the partial-wet-partial-dry method (Braun, 1989) to estimate the heat and mass transfer rate of the evaporator. Since the surface temperature of evaporators may be lower than the saturation temperature at the evaporator inlet, solving the latent heat transfer of evaporators with evaporator inlet pressure may result in systematic bias in the estimation of sensible heat ratio, and the evaporator outlet pressure is used to solve the heat and mass transfer rate model.

The partial-wet-partial-dry method simulates the air-side heat and mass transfer performance between the completely dry and wet coil analyses to enhance the accuracy of the model. Since the method is developed for counterflow heat exchangers but evaporators in real systems are usually crossflow heat exchangers, the evaporator model combines these two types of geometries together as shown in Figure 4.7.

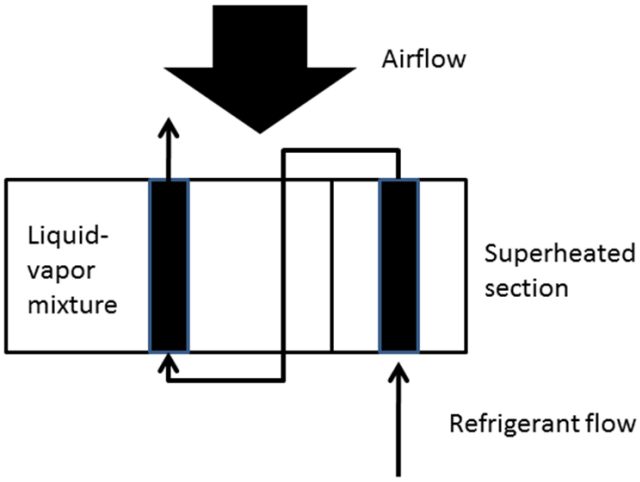

Figure 4.7. Evaporator model schematic.

To solve the evaporator model in Figure 4.7, the heat and mass transfer rates of each section are estimated by using the ε-NTU method of a counterflow heat exchanger and the partial-wet-partial-dry method.

The partial-wet-partial-dry method starts with the estimation of the heat and mass transfer rate of the counterflow heat exchanger in a completely dry coil configuration. The surface temperature at the air outlet is determined with Newton's law of cooling, and the solution will be accepted if the surface temperature at the air outlet is higher than the dewpoint at the air inlet. Otherwise the heat and mass transfer rate of the counterflow heat exchanger is solved with the configuration of a completely wet coil, and the wet-coil solution will be accepted if the surface temperature at the air inlet in this configuration is lower than the dewpoint in the wet-coil solution. If both conditional statements are violated, the heat and mass transfer rate will be estimated with the configuration shown in Figure 2.1, which the dry and wet sections join together in series. The size of both the dry coil and wet coil area are solved iteratively so that the surface temperature at the intersection equaled to the dewpoint of the inlet air. Detailed mathematical procedures are described in Appendix C.

To determine the size of the two-phase and superheated sections, the heat exchanger is first analyzed assuming that only two-phase section exists in the evaporator. The assumption will be accepted if the heat transfer rate calculated is

lower than the maximum possible two-phase heat transfer rate of the two-phase section as shown in

$$\dot{Q}_{r,evap,max} = \dot{m}_r(h_{r,evap,v} - h_{r,evap,in}). \tag{4.18}$$

If the estimated heat transfer rate is higher than the rate in Eqn. (4.18), the refrigerant at the outlet of the two-phase section will be saturated vapor, and the remaining part of the evaporator will be the superheated section. Otherwise the evaporator will have a two-phase section only. The total heat transfer rate is solved by adding the heat transfer rate of the two sections together. The humidity ratio at the evaporator outlet is calculated by mass continuity of humidity at the evaporator. The detailed mathematical procedure is shown in Appendix B and the flowchart of the procedure is sketched as Figure 4.8.

The evaporator heat and mass transfer model requires the heat transfer conductance on the air side and the refrigerant side. Literature was investigated to build semi-empirical models of the conductances. On the air side, by modifying the air-side heat transfer coefficients in Eqn. (4.6) with the consideration of the fin efficiency correlations in Eqns. (2.10) and (2.11), the air-side heat transfer conductance formula are devised as

$$U_{a,evap} = U_{a,evap,rated}(\frac{\dot{m}_a}{\dot{m}_{a,evap,rated}})^{n_{a,evap,rated}}, \tag{4.19}$$

$$UA_{a,evap} = U_{a,evap}\frac{\tanh(C_{a,evap,1}U_{a,evap}^{0.5})}{C_{a,evap,1}U_{a,evap}^{0.5}}A_{r,evap,rated}, \tag{4.20}$$

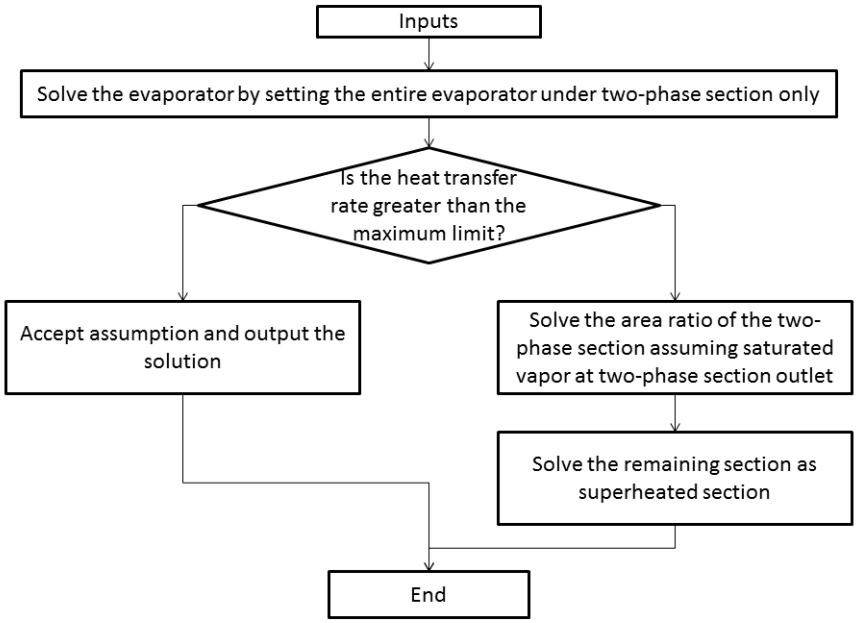

Figure 4.8. Solution procedure of refrigerant phase sections in the evaporator model.

$$UA^*_{a,evap} = \frac{U_{a,evap}}{c_{pa,evap}} \frac{\tanh(C_{a,evap,1}(U_{a,evap}c_{s,evap}/c_{pa,evap})^{0.5})}{C_{a,evap,1}(U_{a,evap}c_{s,evap}/c_{pa,evap})^{0.5}} A_{r,evap,rated} \tag{4.21}$$

and

$$c_{s,evap} = \frac{dh_{a,sat}(T = T_{r,sat}(P_{r,evap,out}))}{dT}. \tag{4.22}$$

On the refrigerant side, the heat transfer conductance of the superheated section is calculated with the same correlation as Eqn. (4.8). The correlation of the conductance of the two-phase section was developed from Shah (1982) which described a correlation

of a multiplier to convert the heat transfer coefficient of saturated liquid into the heat transfer coefficient of evaporating flow. The resultant correlation is

$$\begin{aligned} UA_{r,evap,tp} = \quad & \frac{U_{r,evap,tp,rated}}{K_{r,evap,tp}}\left(\frac{\dot{m}_r}{\rho_{r,evap,l}\mu_{r,evap,l}}\right)^{0.8}\left(\frac{\rho_{r,evap,l}c_{pr,evap,l}}{k_{r,evap,l}}\right)^{0.4} \\ & k_{r,evap,l}A_{r,evap,rated}\int_{x_{r,evap,in}}^{x_{r,evap,out}}\Phi(x_r)(1-x_r)^{0.8}dx_r \end{aligned} \tag{4.23}$$

where Φ in Eqn. (4.23) is integrated numerically by trapezoidal rules with 200 equal divisions following the instructions from Shah (Shah, 1982) listed in Appendix D.

The resultant regression parameters of the heat and mass transfer rate model are $U_{r,evap,tp,rated}$, $U_{r,evap,sh,rated}$, $U_{a,evap,rated}$, $n_{a,evap,rated}$ and $C_{a,evap,1}$.

The pressure drop model of the evaporator is similar to that of the condenser model which is divided into frictional and accelerational parts. However, since the inlet and outlet qualities of an evaporating flow are different from a condensing flow, the viscosity term to adjust the two-phase flow pressure drop follows Dukler et al. (Dukler et al., 1964) as

$$\mu_{r,evap,in} = \frac{x_{r,evap,in}\mu_{r,evap,v}/\rho_{r,evap,v} + (1-x_{r,evap,in})\mu_{r,evap,l}/\rho_{r,evap,l}}{x_{r,evap,in}/\rho_{r,evap,v} + (1-x_{r,evap,in})/\rho_{r,evap,l}}. \tag{4.24}$$

After summing up the frictional pressure drop and the accelerational pressure drop, the resultant pressure drop model for the evaporator is shown in

$$\begin{aligned}\Delta P_{r,evap} = \quad & w_{r,evap,sh} C_{r,evap,\Delta P,sh} \left(\frac{\dot{m}_r^2}{\rho_{r,evap,v}}\right)\left(\frac{\rho_{r,evap,out,rated}}{\dot{m}_{r,evap,rated}^2}\right) \\ & + w_{r,evap,tp} C_{r,evap,\Delta P,tp,1} \left(\frac{\dot{m}_r^2}{\rho_{r,evap,tp}}\right)\left(\frac{\rho_{r,evap,rated}}{\dot{m}_{r,evap,rated}^2}\right)\left(\frac{\mu_{r,evap,in}}{\mu_{r,evap,v,rated}}\right)^{C_{r,evap,\Delta P,tp,2}} \cdot \\ & + C_{r,evap,\Delta P,acc} \left(\frac{\dot{m}_r^2}{\dot{m}_{r,rated}^2}\right) \rho_{r,evap,rated} \left(\frac{1}{\rho_{r,evap,out}} - \frac{1}{\rho_r(P_{r,evap,out}, h_{r,evap,in})}\right)\end{aligned} \tag{4.25}$$

Similar to the condenser pressure drop equation Eqn. (4.13), the evaporator pressure drop model is made to be explicit to prevent iterative calculation by approximating the refrigerant density at the evaporator inlet with evaporator outlet pressure and evaporator inlet enthalpy.

The charge inside the evaporator is estimated by

$$M_{evap} = (w_{r,evap,sh}\rho_{r,evap,sh} + w_{r,evap,sh}\rho_{r,evap,sh})V_{evap}. \tag{4.26}$$

The density in the superheated section of the evaporator is estimated with the average temperature and evaporator outlet pressure assuming thermodynamic equilibrium, and the density of the evaporator two-phase section is given by

$$\rho_{r,evap,tp} = \rho_{r,evap,l}(1 - \overline{\gamma_{evap}}) + \rho_{r,evap,v}\overline{\gamma_{evap}} \tag{4.27}$$

where the average void fraction $\overline{\gamma_{evap}}$ in Eqn. 4.27 is given by

$$\overline{\gamma_{evap}} = -\frac{\begin{array}{l} S_{\gamma,evap}(ln(\frac{S_{\gamma,evap}(x_{r,evap,out}-1)-x_{r,evap,out}}{S_{\gamma,evap}(x_{r,evap,in}-1)-x_{r,evap,in}}) + x_{r,evap,out} - x_{r,evap,in}) \\ -x_{r,evap,out} + x_{r,evap,in} \end{array}}{({S_{\gamma,evap}}^2 - 2S_{\gamma,evap} + 1)(x_{r,evap,out} - x_{r,evap,in})} \tag{4.28}$$

and

$$S_{\gamma,evap} = (\frac{\rho_{r,evap,v}}{\rho_{r,evap,l}})^{2/3}. \tag{4.29}$$

The resultant input and output diagram of the evaporator model is shown in Figure 4.9.

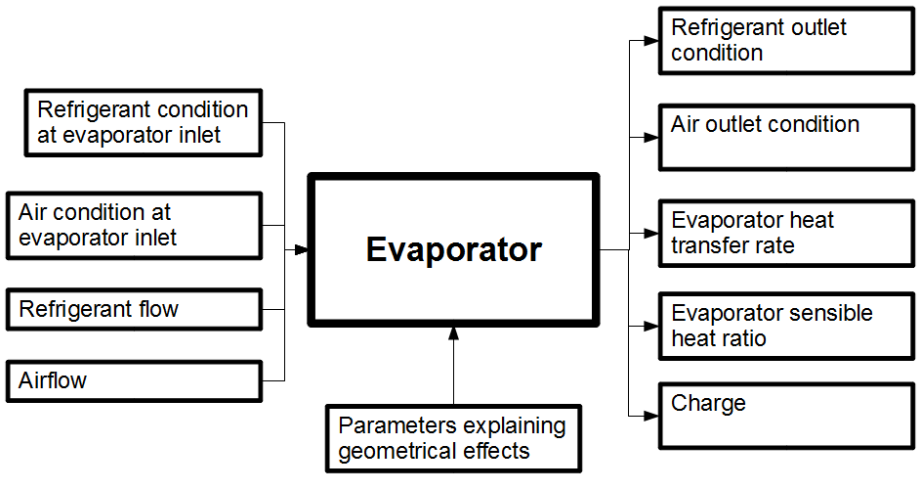

Figure 4.9. Evaporator model input and output diagram.

4.2.5 Refrigerant Pipeline Model

The refrigerant pipelines act similarly as heat exchangers. Both of them induce pressure drop on refrigerant flow and conduct heat transfer with the surroundings.

Although the magnitudes of heat transfer and pressure drop of refrigerant pipelines are smaller than the exchangers, they are simulated because their heat transfer rates, pressure drop and amount of charge are significant to a vapor compression system.

Heat transfer between refrigerant pipelines and the surroundings is dominated by natural convection of air. By assuming air as ideal gas, considering natural convection of horizontal cylinder in ambient air and referencing the Morgan correlation (Incropera, 2007) as

$$Nu = CRa^m = C(\frac{g\rho_a^2 c_{pa}(T - T_{amb}D^3)}{k_a \mu_a T})^m, \tag{4.30}$$

the heat transfer rates of the pipelines are modeled by

$$\begin{aligned} & \dot{m}_r(h_{r,pipeline,out} - h_{r,pipeline,in}) \\ = \; & C_{pipeline,\Delta h,1}(\frac{|T_{r,pipeline,in} - T_{amb}|}{T_{r,pipeline,in}})^{C_{pipeline,\Delta h,2}} \dot{m}_{r,pipeline,rated} \\ & \frac{(T_{r,pipeline,in} - T_{amb})}{\Delta T_{r,pipeline,rated}} \dot{m}_{r,pipeline,rated} \Delta h_{r,pipeline,rated} \end{aligned} \tag{4.31}$$

with $C_{pipeline,\Delta h,1}$ and $C_{pipeline,\Delta h,2}$ as regression parameters.

The pressure drop model is similar to the pressure drop model of the heat exchanger that it is developed with frictional pressure drop. The accelerational

pressure drop is assumed to be negligible to the pressure drop of the system, and the resultant equation is

$$\begin{aligned}\Delta P_{r,pipeline} = \quad & C_{pipeline,\Delta P,1}\left(\frac{\dot{m}_r}{\dot{m}_{r,pipeline,rated}}\right)^{C_{pipeline,\Delta P,2}} \\ & \left(\frac{\mu_{r,pipeline,in}}{\mu_{r,pipeline,rated}}\right)^{C_{pipeline,\Delta P,3}}\left(\frac{\rho_{r,pipeline,in}}{\rho_{r,pipeline,rated}}\right)^{-1}\end{aligned} \tag{4.32}$$

with $C_{pipeline,\Delta h,1}$ to $C_{pipeline,\Delta h,3}$ as regression parameters.

The charge inside the pipeline is estimated by

$$M_{pipeline} = \frac{V_{pipeline}}{2}(\rho_{r,pipeline,in} + \rho_{r,pipeline,out}). \tag{4.33}$$

The models in Eqns. (4.31), (4.32) and (4.33) simulate the operation of the hot gas line, the liquid line and the suction line. The input and output diagram of the model is shown in Figure 4.10.

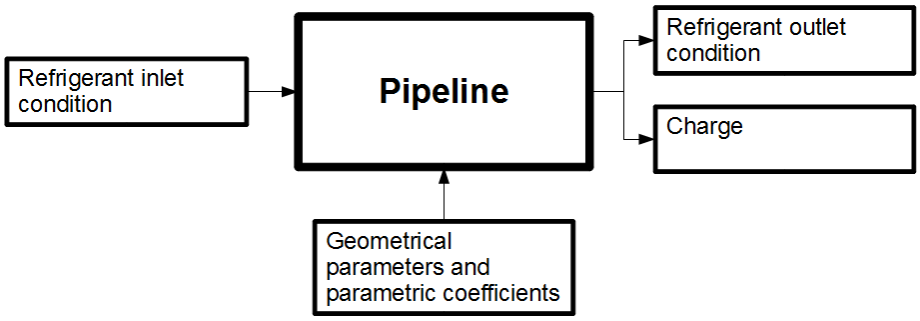

Figure 4.10. Refrigerant pipeline model input and output diagram.

4.2.6 Expansion Valve Model

There are three types of expansion valve models in the simulated systems: FXO, TXV and EEV. FXO operates with a fixed opening area, TXV varies its opening to maintain a superheat by mechanical control with a TXV bulb, and the EEV to be modeled in system XI operates to maintain a user-specified superheat by electronic signals.

FXO Model

The FXO model was developed from Payne and O'Neal (Payne and O'Neal, 2004) which is an FXO model suitable for different types of refrigerants. The modified model

$$\begin{aligned}\frac{4\dot{m}_{r,EXV}}{\pi D_{EXV}^2} = & \frac{\sqrt{\rho_{r,EXV,in} P_{r,cric}}}{1+a_{EXV,6}\pi_3+C_{EXV,1}(\pi_9)^2}(a_{EXV,1} + a_{EXV,2}\pi_3 + C_{EXV,2}\pi_9 \\ & +a_{EXV,4}\pi_6 + a_{EXV,5}\ln(\tfrac{L_{EXV}}{D_{EXV}})))\end{aligned} \tag{4.34}$$

is only suitable to estimate mass flow rate with subcooled or saturated liquid at the inlet, where the definition of Π groups and existing empirical coefficients a_i are shown in Appendix E.

Eqn. (4.34) includes regression parameters D_{EXV}, L_{EXV}, $C_{EXV,1}$ and $C_{EXV,2}$. Since mass flow rate readings are only valid when subcooling at the meter inlet is greater than 3K and Payne and O'Neal (Payne and O'Neal, 1999) showed that the mass flow rate decreases with subcooling, to avoid the violation of the observation by

the model when it estimates the refrigerant mass flow rate with inlet subcooling less than 3K, the mass flow rate model is adjusted by

$$\begin{aligned} & \dot{m}_{r,EXV,in}(SC<3K) \\ = & \begin{cases} \dot{m}_{r,EXV,in}(SC) & \text{if } \dot{m}_{r,EXV,in}(SC) < \dot{m}_{r,EXV,in}(SC=3K) \\ \dot{m}_{r,EXV,in}(SC=3K) & \text{otherwise} \end{cases} . \end{aligned} \tag{4.35}$$

Payne and O'Neal (Payne and O'Neal, 2004) also provides a two-phase flow multiplier to the FXO model as shown in Appendix E.

In some systems, there are insufficient data points to obtain all regression parameters in Eqn. (4.34). Under this situation, the FXO refrigerant mass flow rate is modeled by the original FXO model from Payne and O'Neal (Payne and O'Neal, 2004) as shown by

$$\begin{aligned} & \frac{4\dot{m}_{r,EXV}}{\pi D_{EXV}^2} \\ = & \sqrt{\rho_{r,EXV,in} P_{r,cric}} \left(\frac{a_{EXV,1}+a_{EXV,2}\pi_3+a_{EXV,3}\pi_9+a_{EXV,4}\pi_6+a_{EXV,5}\ln(\frac{L_{EXV}}{D_{EXV}})}{1+a_{EXV,6}\pi_3+a_{EXV,7}(\pi_9)^2} \right). \end{aligned} \tag{4.36}$$

TXV Model

The TXV model was developed to simulate the dependence of the opening area with the superheat at the outlet of the evaporator or the inlet of the compressor by modifying the FXO model. Only TXVs with external equalizer are modeled because other types of TXVs are not used in the systems in Table 3.1.

Li and Braun (Li and Braun, 2009), assuming that the evaporator superheat is directly proportional to the pressure difference across the TXV bulb, approximated the relationship between the evaporator outlet superheat and the opening area by a quadratic relationship. This approximation can help to estimate the opening area by

$$\frac{\pi D_{EXV}^2}{4} = \begin{cases} C_{TXV,1} + C_{TXV,2}\Delta P_{TXV} + C_{TXV,3}\Delta P_{TXV}^2 & \text{if } \Delta P_{TXV} < \mathrm{C}_{TXV,4} \\ A_{TXV,max} & \text{otherwise} \end{cases}. \tag{4.37}$$

Eqn. (4.37) estimates the change of opening area with pressure difference across the TXV bulb. However, when the pressure difference exceeds a certain value, the opening area is maximized and the maximum opening area is given by

$$A_{TXV,max} = C_{TXV,1} + C_{TXV,2}\mathrm{C}_{TXV,4} + C_{TXV,3}\mathrm{C}_{TXV,4}. \tag{4.38}$$

D_{EXV} is changed from a regression parameter to a variable dependent on the superheat at the evaporator outlet as shown by Eqn. (4.37). The pressure difference at the TXV bulb is given by

$$\Delta P_{TXV} = P_{TXV,bulb} - P_{r,evap,out} \tag{4.39}$$

and

$$P_{TXV,bulb} = \begin{cases} P_{r,sat}(T_{r,evap,out}) & \text{if } T_{evap,out} < 294\text{K} \\ P_r(T_{r,evap,out}, \rho_{r,sat}(294\text{K})) & \text{otherwise} \end{cases} . \tag{4.40}$$

A temperature 294K in Eqn. (4.40) is set as the dry-out temperature of the refrigerant in the TXV bulb according to the specification of the TXV (Kim et al., 2006).

To estimate the refrigerant mass flow rate of the TXV, the FXO model was modified to include three more regression parameters and D_{EXV} from Eqn. (4.37) is substituted into the modified model to create

$$\begin{aligned} \dot{m}_{r,EXV} = \; & \frac{\sqrt{\rho_{r,EXV,in} P_{r,cric}}}{1+a_{EXV,6}\pi_3+C_{EXV,1}(\pi_9)^2}(a_{EXV,1} + C_{EXV,2}\pi_9 + C_{EXV,3}\ln(\frac{L_{EXV}}{D_{EXV}}) \\ & +C_{EXV,4}\pi_3 + C_{EXV,5}\pi_6)\frac{\pi D_{EXV}^2}{4} \end{aligned} \tag{4.41}$$

where three more empirical coefficients in the literature were changed to regression parameters $C_{EXV,3}$, $C_{EXV,4}$ and $C_{EXV,5}$.

EEV Model

Since the opening of EEV is controlled by electronic signals and can be controlled according to any control setpoints, the EEV model requires a user-specified control setpoint as an input to determine the EEV opening. Since system XI in Table 3.1 was tested with a compressor suction superheat setpoint at 8.7K in the experiments and system XI is the only system with EEV in Table 3.1, the simulation in this

project only considers compressor suction superheat as the control setpoint for the EEV opening. Although EEV openings are controlled in discrete steps in reality, the opening area of the EEV is assumed to be continuous to simplify the model, and this assumption means that the EEV can control the system to reach any user-specified setpoint as far as its opening is not maximized.

The modeling of an EEV begins with the estimation of its opening area. With refrigerant mass flow rate and the refrigerant condition at its inlet and outlet as inputs, the opening area of the EEV can be calculated. If the estimated opening area exceeds its maximum opening area, the maximum opening area will be used to calculate the correct refrigerant mass flow rate across the EEV. However, as the experimental data of system XI does not contain valid data points with maximum EEV opening, the EEV in system XI is assumed to be able to achieve any compressor outlet superheat setpoint given by the user and the use of EEV model in the simulation of system XI does not involve the maximum opening area. The effect of this assumption to the system simulation process is discussed in Section 6.1.3.

4.2.7 Accumulator Model

An accumulator is a device installed at the compressor suction which avoids liquid refrigerant from entering the compressor. Its schematic is shown in Figure 4.11.

When the accumulator receives two-phase refrigerant, it stores the liquid portion in its body and expels the vapor out of the accumulator. However, if the accumulator is full of liquid, it cannot hold any more liquid, and two-phase refrigerant will enter

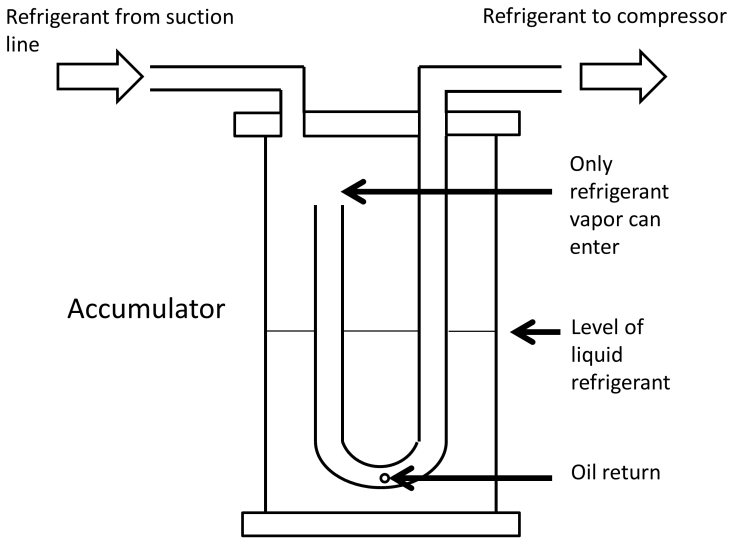

Figure 4.11. Schematic of an accumulator.

the compressor. When pure vapor enters the accumulator, the vapor will vaporize any liquid refrigerant inside the accumulator. If the accumulator is dry when vapor enters it, the vapor will leave the accumulator without any interaction with the accumulator.

In this project, the accumulator model in Cheung and Braun (Cheung and Braun, 2014) is used. The model assumes that the accumulator has a negligible heat transfer surface and heat transfer rate with the surroundings comparing with the rest of the system, and the refrigerant enthalpy values at the inlet and outlet of an accumulator are equal. The model assumes that the accumulator never fills completely with liquid, which is a reasonable assumption for the range of charge levels in practice. Since an accumulator expels two-phase refrigerant only when it is full of liquid, the aforementioned assumptions mean that the accumulator outlet

refrigerant can only be pure vapor when the system is operating at steady state. By assuming that the amount of refrigerant vapor inside an accumulator is negligible comparing to the total amount of charge inside a system, it is not necessary to calculate the amount of charge inside the accumulator when the accumulator contains refrigerant vapor only. Hence the accumulator model does not have any unknown parameters and the model can be embedded to the models of systems with accumulators directly. The procedure to embed the model is also discussed in Section 6.1.7.

4.2.8 Fan Power Consumption Model

Fan power consumption modeling is important to model the impact of heat exchanger fouling on vapor compression systems because heat exchanger fouling changes the pressure drop across the fans and the fan power consumption. Since this dissertation only considers systems with single-speed fans and the system model only takes volumetric airflow as an input related to fan power consumption, the fan speed is assumed to be constant at all volumetric airflow. Under constant fan speed and decreasing airflow, pressure difference across the fan increase and fan power consumption decrease, as exemplified by the manufacturer data of the blowers in two 3-ton packaged systems in Tables 4.2 and 4.3.

Table 4.2. Fan curve data from Unit A when the fan operates in low speed mode.

Airflow [cfm]	Brake Horse Power	External Static Pressure (inch. of water)
900	0.21	0.46
1000	0.24	0.44
1100	0.25	0.4
1200	0.27	0.32
1300	0.29	0.28
1400	0.3	0.22
1500	0.31	0.16

Table 4.3. Fan curve data from Unit B when the fan operates at 600RPM.

Airflow [cfm]	Brake Horse Power	External Static Pressure (inch. of water)
900	0.17	0.3
1000	0.19	0.28
1100	0.20	0.24
1200	0.22	0.21
1300	0.24	0.17
1400	0.25	0.13

Since condenser fans are usually axial fans, axial fan models are used to model the condenser fan power consumption. Osborne (Osborne, 1977) suggested that the fan power consumption of axial fans in theory can be described by

$$\dot{W}_{cond,fan} = \frac{\Delta P_{a,cond}\dot{V}_{a,cond}}{\eta_{cond,fan}} = \frac{\rho_{a,cond}\frac{D_{fan}}{2}2\pi f_{fan}}{\eta_{cond,fan}}(\frac{D_{fan}}{2}2\pi f_{fan}V_{a,cond} - \frac{4V_{a,cond}^2}{\pi D_{fan}^2 C_{geometry}}). \tag{4.42}$$

As all geometrical parameters in Eqn. (4.42) are constants, by assuming constant fan efficiency, the condenser fan power consumption can be simplified to

$$\dot{W}_{cond,fan} = C_{cond,fan,1}\dot{V}_{a,cond} + C_{cond,fan,1}\dot{V}_{a,cond}^2. \tag{4.43}$$

Since evaporator fans are usually centrifugal fans, the power consumption model of centrifugal fans with forward curved blades is used, and its prediction is given in Figure 4.12 (ASHRAE, 2008).

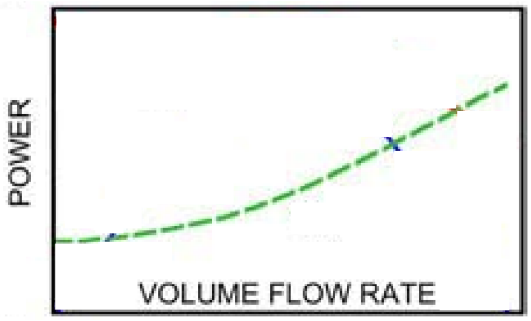

Figure 4.12. Fan power consumption of a centrifugal fan with forward curved blades (ASHRAE, 2008).

By using a quadratic curve with a positive y-axis interval to model the fan power consumption curve with a stall power consumption in Figure 4.12 , an evaporator fan power consumption model can be written mathematically as

$$\dot{W}_{evap,fan,temp} = C_{evap,fan,1} + C_{evap,fan,2}\dot{V}_{a,evap} + C_{evap,fan,3}\dot{V}^2_{a,evap} \tag{4.44}$$

and

$$\dot{W}_{evap,fan} = \begin{cases} \dot{W}_{evap,fan,rated} & \text{if } \dot{W}_{evap,fan,temp} < \dot{W}_{evap,fan,rated} \\ \dot{W}_{evap,fan,temp} & \text{otherwise} \end{cases} . \tag{4.45}$$

4.3 Fault Modeling

This section discusses the models of different types of faults, including non-standard charging, heat exchanger fouling, compressor flow fault, liquid line restriction and presence of non-condensables.

4.3.1 Non-standard Charging

To simulate the effect of non-standard charging, an accurate estimation of charge inside the system is needed. The modeling of charge inventory inside the system assumes that the amount of refrigerant inside the expansion valve and the compressor to be negligible to the total charge level, and the charge estimation in other components follows the discussion in Sections 4.2.3, 4.2.4 and 4.2.5. The amount of charge estimated from all component models is tuned by a charge tuning method to enhance the accuracy of the charge estimation, and the method is discussed in Section 6.2. With accurate estimation of the amount of charge inside a system, simulation users can impose a charge level as an input to the simulation, and an implicit solver can be used to solve the system performance at a given charge level. The details of the implicit solver is discussed in Section 6.1.

In some systems, the dimensions of heat exchangers or pipelines were unavailable from the record. Their dimensions were estimated based on the rated capacity and type of the system in order to estimate the charge inside the heat exchangers and pipelines. For example, if the dimensions of a 3-ton split system were unavailable, the

dimensions of heat exchangers and pipelines would be estimated from another 3-ton split system, and the bias as a result of dimension deviation is removed by using charge tuning.

4.3.2 Heat Exchanger Fouling

Yang et al. (Yang et al., 2007) and Bell et al. (Bell et al., 2012) found that airflow reduction is the most important impact of heat exchanger fouling on the performance of the heat exchanger. The airflow of the condenser and evaporator is reduced according to the heat exchanger fouling level defined in Eqn. (2.16) to simulate the impact of heat exchanger fouling on the system.

4.3.3 Compressor Flow Fault

Compressor flow fault is simulated similarly as the valve leakage experiments described in Section 3.5.3. Following the fault definition in Eqn. (2.18), the refrigerant mass flow rate entering the condenser is reduced according to

$$\dot{m}_{r,cond,in} = (1 - VL)\dot{m}_{r,comp,ori}(P_{r,suctionline,in}, h_{r,comp,in}, P_{r,comp,out}, h_{r,comp,out}). \quad (4.46)$$

The $\dot{m}_{r,comp,ori}$ in Eqn. (4.46) is the original mass flow rate without compressor flow fault from Eqn. (4.1). $\dot{m}_{r,comp}$ is the final mass flow rate to be used in the other component models when the fault occurs. This simulates how the fault reduces compressor mass flow rate.

The compressor suction enthalpy observed by the compressor is also raised as a result of refrigerant backflow across the compressor. This is described by

$$\begin{aligned} & \dot{m}_{r,cond,in} \\ = \; & (1-bp)\dot{m}_{r,comp}(P_{r,comp,in}, (1-bp)h_{r,comp,in} + bph_{r,comp,out}, P_{r,comp,out}, h_{r,comp,out}) \end{aligned} \tag{4.47}$$

where bp in Eqn. (4.47) is the bypass mass flow ratio which is different from the fault level VL in Eqn. (4.46). To calculate the fault level VL, an implicit function solver is used to solve Eqns. (4.47) and (4.46) simultaneously. The bypass mass flow ratio bp, and hence the compressor inlet enthalpy and the compressor power consumption, can be found.

4.3.4 Liquid Line Restriction

Liquid line restriction level is defined with reference to the pressure drop in the non-faulted case as shown in Eqn. (2.20). To simulate the restriction impact on the system performance, a case without restriction is first calculated. Eqn. (2.20) is then used to estimate the additional pressure drop due to liquid line restriction. The additional pressure drop is subtracted from the pressure at the outlet of the liquid line in a repeated simulation. After adjusting the refrigerant density of the pipeline with the new outlet pressure, the new simulation outputs show the impact of the restriction on the system.

4.3.5 Presence of Non-condensables

The presence of non-condensables inside the system is simulated by developing the model proposed in Bendapudi (Bendapudi, 2002). When the fault level is 100%, non-condensable, represented by pur nitrogen, occupies the system at 1 atmospheric pressure (101.325[kPa]) and 299.85K. The amount of non-condensable in the system at the fault level can be calculated by

$$M_{nc,total} = \frac{V_{total}(101.325[kPa])}{(0.2968[kPa - m^3/kg - K])(299.85[K])} \tag{4.48}$$

The amount of non-condensable in the system can then be defined as

$$M_{nc} = M_{nc,total} NC \tag{4.49}$$

The model assumes that non-condensable is trapped in the vapor section between compressor discharge and expansion valve inlet. The volume of the vapor section is estimated by

$$V_{\mathrm{nc}} = V_{r,cond} w_{r,nc} + V_{HG} \tag{4.50}$$

and

$$w_{r,nc} = w_{sh,cond} + w_{tp,cond}(1 - x_{r,cond,tp,out})\bar{\gamma}_{r,cond,tp}. \tag{4.51}$$

By Dalton's law, the volume of the vapor section and the amount of non-condensable in the system, the partial pressure of the non-condensables can be calculated according to the ideal gas law as and

$$P_{nc} = \frac{M_{nc} T_{r,comp,out} (0.2968[kPa - m^3/kg - K])}{V_{nc}} \tag{4.52}$$

where $0.2968[kPa - m^3/kg - K]$ is the gas constant of dry nitrogen.

The partial pressure of refrigerant is computed by

$$P_r = P_{r,total} - P_{nc}. \tag{4.53}$$

The hot gas line model and the condenser model are solved by the refrigerant partial pressure P_r in Eqn. (4.53) while the compressor model, the liquid line model and the expansion valve model are solved by the total pressure $_{r,total}$ in Eqn. (4.53). The compressor discharge temperature is estimated using the equation of state by the refrigerant partial pressure at the compressor discharge and the compressor discharge enthalpy from the compressor model.

The heat exchanger volume required for solving the non-condensable model is tabulated in Table 4.4.

Table 4.4. Information of heat exchanger volume.

System	Total volume of refrigerant circuit $[m^3]$	Volume of hot gas line $[m^3]$	Volume of condenser $[m^3]$
I	6.23e-03	4.97e-05	4.09e-03
II	1.03e-02	0.00e+00	6.47e-03
III	6.12e-03	4.97e-05	3.82e-03
IV	7.55e-03	1.45e-04	4.09e-03
V	7.55e-03	1.45e-04	4.09e-03
VI	9.94e-03	0.00e+00	4.83e-03
VII	4.83e-03	0.00e+00	1.58e-03
VIII	9.01e-03	0.00e+00	4.09e-03
IX	9.02e-03	0.00e+00	4.09e-03
X	1.08e-02	0.00e+00	5.90e-03
XI	9.60e-03	0.00e+00	5.70e-03

4.4 Summary

Various mathematical models for components are given in this chapter. Semi-empirical models were created based on fundamental theories and empirical models in literature. For instance, the heat loss model of refrigerant pipelines was developed based on natural convection of air with some empirical coefficients. The pressure drop models were created based on the correlations of frictional pressure drop and principles of accelerational pressure drop. Existing empirical coefficients of TXV and FXO models are listed in Appendix E, and the list of unit-specific normalization parameters and regression parameters in the component models are tabulated in Table 4.5 and Table 4.6.

Table 4.5. List of unit-specific normalization parameters in different component models.

Models	Unit-specific parameters
Compressor mass flow rate model	None
Compressor power consumption model	None
Compressor heat loss model	None
Condenser heat transfer rate model	$\dot{m}_{a,cond,rated}$, $A_{r,cond,rated}$, $\dot{m}_{r,rated}$, $K_{r,cond,sh}$ and $K_{r,cond,sc}$
Condenser pressure drop model	$\rho_{r,cond,rated}$, $\rho_{r,cond,out,rated}$ and $\mu_{r,cond,v,rated}$
Condenser fan power consumption model	None
Condenser charge model	$V_{r,cond}$
Evaporator heat and mass transfer rate model	$\dot{m}_{a,evap,rated}$, $A_{r,evap,rated}$, $\dot{m}_{r,evap,rated}$,$K_{r,evap,tp}$ and $K_{r,evap,sh}$
Evaporator pressure drop model	$\rho_{r,evap,out,rated}$, $\mu_{r,evap,v,rated}$ and $\rho_{r,evap,rated}$
Evaporator fan power consumption model	$\dot{W}_{evap,fan,rated}$
Evaporator charge model	V_{evap}
FXO mass flow rate model	None
TXV mass flow rate model	None
Refrigerant pipeline heat loss model	$\Delta T_{r,pipeline,rated}$, $\dot{m}_{r,pipeline,rated}$, $\Delta h_{r,pipeline,rated}$
Refrigerant pipeline pressure drop model	$\rho_{r,pipeline,rated}$ and $\mu_{r,pipeline,rated}$
Refrigerant pipeline charge model	$V_{pipeline}$

Mathematical models for different types of faults were also built based on deductive and inductive discoveries in literature. For example, non-standard charging is modeled by imposing different charge levels in the simulation inputs. Heat exchanger fouling is simulated by reduction of airflow across heat exchangers. The effect of refrigerant backflow in cases of compressor flow fault is modeled in terms of reduction of mass flow rate and changes of refrigerant enthalpy at compressor discharge and suction. Liquid line restriction is modeled based on the

Table 4.6. List of regression parameters in different component models.

Models	Regression parameters
Compressor mass flow rate model	$C_{comp,0}$, $C_{comp,1}$ and $C_{comp,2}$
Compressor power consumption model	$C_{comp,3}$, $C_{comp,4}$, $C_{comp,5}$, $C_{comp,6}$ and $C_{comp,7}$
Compressor heat loss model	$C_{comp,8}$ and $C_{comp,9}$
Condenser heat transfer rate model	$U_{a,cond,rated}$, $n_{a,cond,rated}$, $U_{r,cond,tp,rated}$, $U_{r,cond,sh,rated}$ and $U_{r,cond,sc,rated}$
Condenser pressure drop model	$C_{r,cond,\Delta P,sh}$, $C_{r,cond,\Delta P,tp,1}$, $C_{r,cond,\Delta P,tp,2}$, $C_{r,cond,\Delta P,sc}$ and $C_{r,cond,\Delta P,acc}$
Condenser fan power consumption model	$C_{cond,fan,1}$ and $C_{cond,fan,2}$
Condenser charge model	None
Evaporator heat and mass transfer rate model	$U_{a,evap,rated}$, $n_{a,evap,rated}$, $C_{a,evap,1}$, $U_{r,evap,tp,rated}$ and $U_{r,evap,sh,rated}$
Evaporator pressure drop model	$C_{r,evap,\Delta P,sh}$, $C_{r,evap,\Delta P,tp,1}$, $C_{r,evap,\Delta P,tp,2}$ and $C_{r,evap,\Delta P,acc}$
Evaporator fan power consumption model	$C_{evap,fan,1}$, $C_{evap,fan,2}$ and $C_{evap,fan,3}$
Evaporator charge model	None
FXO mass flow rate model	D_{EXV}, L_{EXV}, $C_{EXV,1}$ and $C_{EXV,2}$
TXV mass flow rate model	$C_{TXV,1}$, $C_{TXV,2}$, $C_{TXV,3}$, $C_{TXV,4}$, L_{EXV}, $C_{EXV,1}$, $C_{EXV,2}$, $C_{EXV,3}$, $C_{EXV,4}$, $C_{EXV,5}$
Refrigerant pipeline heat loss model	$C_{pipeline,\Delta h,1}$ and $C_{pipeline,\Delta h,2}$
Refrigerant pipeline pressure drop model	$C_{pipeline,\Delta P,1}$, $C_{pipeline,\Delta P,2}$ and $C_{pipeline,\Delta P,3}$
Refrigerant pipeline charge model	None

definition of the fault level, and presence of non-condensables is modeled by applying Dalton's law on the refrigerant-non-condensable mixture.

CHAPTER 5. PARAMETER ESTIMATION

In Chapter 4, the relationship between the component models and the parameters are described. However, the regression parameters inside the models are unknowns and are necessary to be estimated by regression with the experimental data. This chapter describes the estimation of the regression parameters in the component models to achieve the following goals.

1. To obtain realistic and valid data for parameter estimation of component models:

   Most experiments collected data for system performance and did not aim at collecting data for individual components. However, parameter estimation of component models needed data at the components only, and some data points were not useful to estimate the regression parameters in the component models. They should be removed to avoid corrupting the estimation process.

2. To estimate component model parameters in an unbiased manner:

   The test matrices of the experiments were not balanced for this study. Their test results may overemphasize the performance at certain testing conditions.

A weighting scheme for the data points during parameter estimation is needed to avoid overemphasizing the significance of these conditions.

3. To bound the models with the laws of physics:

   When the effect of an input to a system is small, the change of the system behavior with respect to the input may be smaller than the uncertainty of the experimental observation. This may result in misleading trends in experimental observations that violate the laws of physics. If a model is trained with these observations without constraints from laws of physics, they may predict behaviors which violate the laws. This should be prevented by setting appropriate constraints to the parameters during the estimation process.

4. To validate the semi-empirical component models:

   To examine if the component models in Chapter 4 are valid, the estimated outputs from the component models are compared with the experimental data after parameter estimation.

5. To define the applicability domains of the component models:

   Since the component models were trained by experimental data collected within limited ranges of conditions, the models will only be reliable with a limited range of inputs. To avoid using the models accidentally with inputs outside the applicability domains of the models, the inputs should be compared to the applicability domains of the models when the models are used. However,

conventional methods to define the domain by inequalities only do not work because the experimental data were collected with an imbalanced test matrix. Other methods are developed to define the domains.

5.1 Component-level Parameter Estimation

5.1.1 General Methodology

To avoid estimating parameters with heavy computational effort, instead of estimating the parameters of all component models simultaneously, the parameters for each component were estimated separately in small groups. For example, when estimating the parameters of the compressor model, only the inlet and outlet conditions of the compressor from the measurement are considered and other regression parameters such as the parameters of the liquid line pressure drop model are ignored. Invalid data points for component models are also eliminated to avoid unreliable inputs to the estimation process. For example, if the compressor suction superheat is measured as 0K, the possible range of compressor suction enthalpy will be too large for the data point to act as a reliable input to parameter estimation, and the data point should not be used. Valid data are passed to the component models to produce different model predictions with different sets of parameters. Differences between the model predictions and the measured data are then calculated as objective functions and the sets of parameters which give the smallest differences are the parameters of the component models. To implement the search

method for the parameters corresponding to the minimum objective functions quickly, optimization algorithms are used. Some minimization processes bound the parameters to avoid the resultant model violating the laws of physics. The flowchart of the component-level parameter estimation process is shown in Figure 5.1.

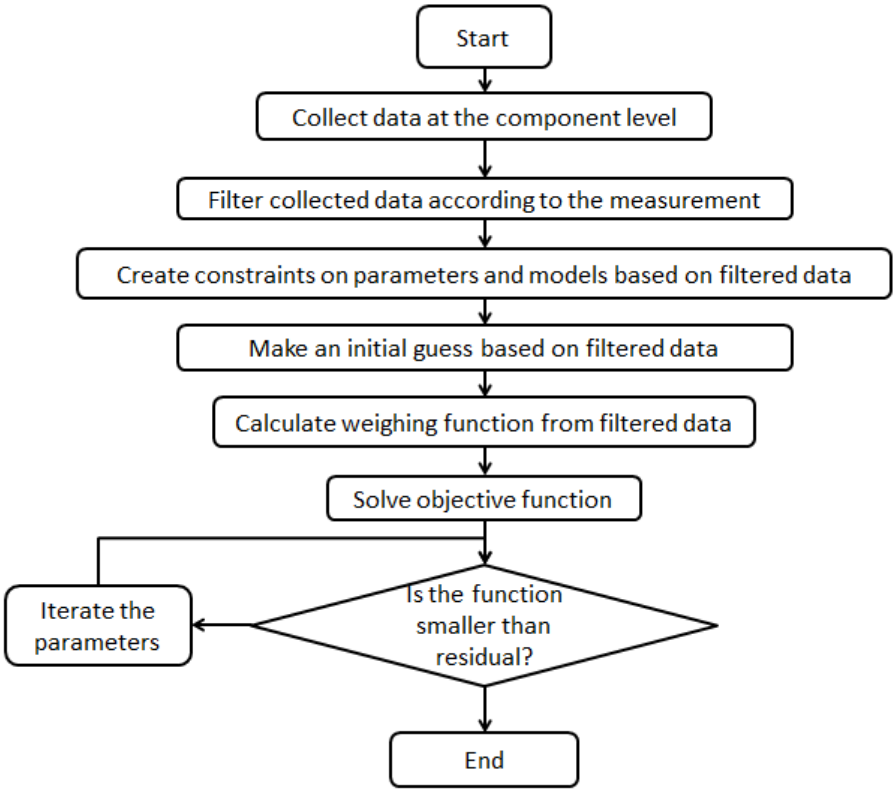

Figure 5.1. Flowchart of parameter estimation for each component model.

5.1.2 Weighting Function for Parameter Estimation

Since the test matrices of the experiments were not balanced with all levels of faults and environmental conditions, parameter estimation of regression parameters

was carried out using a weighted objective function. A general weighting function is given by

$$weight_{\text{variable},i} = \frac{1}{N_{bin,\text{variable},i}}. \tag{5.1}$$

The factor $weight_{\text{variable},i}$ is a weighting factor for the i^{th} data point in a parameter estimation process, 'variable' is the data variable being weighted and N_{bin} is the number of data points sharing the same bin as the i^{th} data point. For example, for data points weighted by compressor mass flow rates where its histogram is plotted in Figure 5.2, the weighting factor of a data point with a compressor mass flow rate at 0.041kg/s is 0.125 because the data point is assigned to the bin with compressor mass flow rate at 0.045kg/s and the number of data points in the bin is 8.

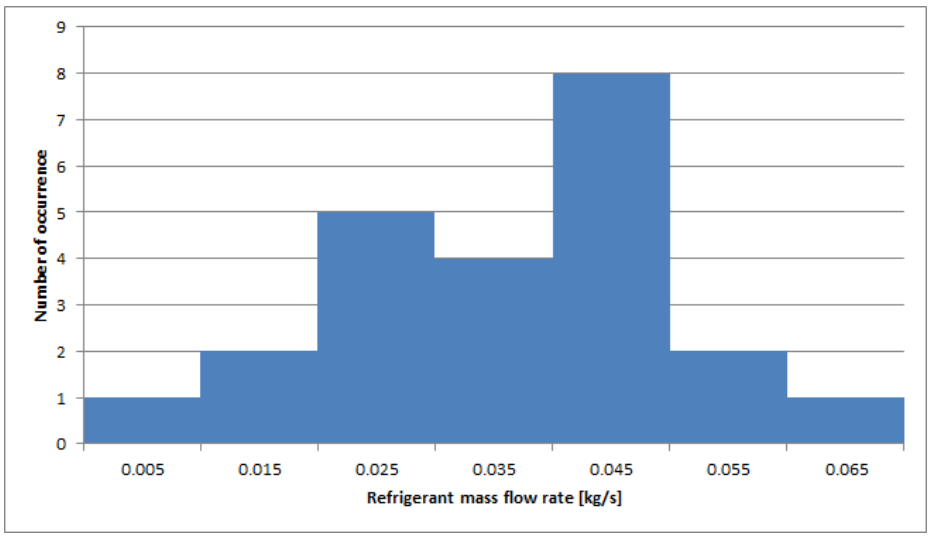

Figure 5.2. Example histogram of compressor mass flow rates in a system.

5.1.3 Validity of Input Variables to Parameter Estimation

Parameter estimation requires reliable measurement data to calculate its objective functions with reasonable uncertainties. However, some data points consist of unreliable or undefined measurements. For instance, if the temperature and pressure measurements were taken from a condensing flow with unknown thermodynamic quality, the measurements would be insufficient to provide the state of refrigerant. This section presents how the parameter estimation processes can obtain valid inputs and how they remove undefined data points to generate valid component models.

Refrigerant Properties

Refrigerant properties are estimated based on the temperature and pressure measurement of refrigerant along the refrigerant circuit. However, measurements of temperature and pressure are insufficient to estimate refrigerant properties along refrigerant circuits with two-phase refrigerant flow. Considering the uncertainty of temperature and pressure measurements, any data points with superheat or subcooling less than 1K are categorized to be insufficient to provide accurate refrigerant properties and are regarded as invalid for parameter estimation.

Refrigerant Mass Flow Rate

Refrigerant mass flow rate measurements from Coriolis mass flowmeters are only reliable when the subcooling upstream is higher than 3K or potential bubbly flow may cause incorrect readings. When the subcooling is below 3K, the evaporator outlet superheat and the subcooling upstream of the expansion valve are examined. If both of them are above 1K, the refrigerant enthalpies at the two ends of the evaporator will be used to estimate the refrigerant mass flow rate by an energy balance equation at the evaporator. Otherwise the condenser outlet subcooling is examined. If the subcooling is higher than 1K, the refrigerant enthalpy at the condenser outlet will be valid, and the refrigerant mass flow rate will be estimated by energy balance at the condenser. If both methods cannot be used to estimate the mass flow rate and the subcooling is below 3K, the refrigerant mass flow rate of the data point will be invalid for parameter estimation.

The rules for valid refrigerant mass flow rate are summarized in Table 5.1.

Increasing the Number of Valid Data Points using Compressor Mass Flow Rate Model

To increase the number of data points available for parameter estimation, after estimating the regression parameters of the compressor mass flow rate model, the mass flow rate model is used to estimate the refrigerant mass flow rate of data points that were collected with a compressor suction superheat greater than 1K and do not satisfy the conditions in Table 5.1. The refrigerant mass flow rates of these data

Table 5.1. Rules to estimate valid refrigerant mass flow rate before establishing the compressor mass flow rate model.

Estimation method	$SC_{cond,out} > 3K$	$SC_{cond,out} > 1K$ & $SH_{evap,out} > 1K$	$SC_{cond,out} > 1K$
Measured refrigerant mass flow rate	Yes	–	–
Mass flow rate from energy balance on evaporator	No	Yes	–
Mass flow rate from energy balance on condenser	No	No	Yes

points become valid, and these data points can be used to estimate the parameters in the remaining component models in the systems.

The enthalpies of data points without sufficient superheat and subcooling can also be estimated after defining the compressor mass flow rate model. If the condenser outlet subcooling for cases without condenser fouling is less than 1K, the condenser outlet enthalpy can be estimated by solving the energy balance equation at the condenser. In cases where the expansion valve inlet subcooling is less than 1K and the evaporator outlet superheat is higher than 1K, the enthalpy at the expansion valve inlet can be estimated by solving the energy balance equation at the evaporator.

5.1.4 Compressor Model

The parameter estimation of the compressor mass flow rate model, the compressor power consumption model and the compressor heat loss model was conducted separately.

Compressor Mass Flow Rate Model

The compressor mass flow rate model shown in Eqn. (4.1) is obtained by minimizing the objective function

$$J_{\dot{m}_{r,comp}} = \Sigma_i weight_{\dot{m}_{r,comp,mea},i}(\frac{\dot{m}_{r,comp,mea,i} - \dot{m}_{r,comp,pre,i}}{\dot{m}_{r,comp,mea,i}})^2. \tag{5.2}$$

Eqn. (5.2) calculates the relative difference between the measured and predicted mass flow rate with a weighting function of refrigerant mass flow rate where $\dot{m}_{r,comp,pre}$ comes from Eqn. (4.1). Unconstrained minimization was used to estimate the regression parameters in Eqn. (4.1). The parameter estimation of the component sub-model only used data points with compressor suction superheat greater than 1K and valid refrigerant mass flow rates.

During the minimization, $C_{comp,1}$ is restricted to be negative so that the volumetric efficiency is always smaller than or equal to 1. $C_{comp,2}$ should also be restricted to be negative so that the mathematical term related to internal leakages always reduces the refrigerant mass flow rate of a compressor. The constraints are implemented by

minimizing the objective function Eqn. (5.2) with interior-point method (Nocedal and Wright, 2006).

Compressor Power Consumption Model

The compressor power consumption model in Eqn. (4.3) estimates the compressor power consumption by inlet and outlet pressures and a polytropic coefficient. Since the polytropic coefficient is estimated from the compressor suction density, only data with a compressor suction superheat higher than 1K can be used to estimate the regression parameters of the compressor power consumption model. The objective function for the process is

$$J_{\dot{W}_{comp}} = \Sigma_i weight_{\dot{W}_{comp},i}(\frac{\dot{W}_{comp,mea,i} - \dot{W}_{comp,pre,i}}{\dot{W}_{comp,mea,i}})^2. \tag{5.3}$$

The minimization of the objective function Eqn. (5.3) only iterated the regression parameters from $C_{comp,3}$, $C_{comp,4}$, $C_{comp,5}$, $C_{comp,6}$ and $C_{comp,7}$. Since Eqn. (4.1) was used to calculate the refrigerant mass flow rate during the minimization, the validity of refrigerant mass flow rate in the training data did not matter in the minimization of Eqn. (5.3).

Compressor Heat Loss Model

The compressor heat loss model, consisting of equations listed from Eqns. (4.4) to (4.5), takes refrigerant mass flow rate, the refrigerant temperature and pressure

of the compressor and the ambient temperature as inputs. Its parameter estimation can only use data points with compressor suction superheat greater than 1K so as to have valid compressor suction enthalpy as inputs. The weighted objective function for its parameter estimation is

$$J_{\dot{\Delta H}_{comp}} = \Sigma weight_{\dot{\Delta H}_{comp},i}(\frac{\dot{Q}_{HL,pre,i} - \dot{Q}_{HL,mea,i}}{\dot{Q}_{HL,mea,i}})^2, \tag{5.4}$$

where $\dot{\Delta H}_{comp}$ is defined by

$$\dot{\Delta H}_{comp} = \dot{m}_r(h_{r,comp,out} - h_{r,comp,in}). \tag{5.5}$$

Since the model aims at estimating the power getting into the refrigerant flow, the objective function Eqn. (5.4) is weighted according to the power input to the refrigerant flow defined by Eqn. (5.5). The minimization also restricts $C_{comp,8}$ and $C_{comp,9}$ to be positive so that they can represent the heat transfer coefficients of the surface of the compressor correctly. The restriction is applied by using interior-point method which is a constrained optimization algorithm to minimize the objective function Eqn. (5.4) (Nocedal and Wright, 2006).

5.1.5 Condenser Model

Two sub-models of the condenser model contain regression parameters: the heat transfer rate model and the pressure drop model. As the pressure drop model takes

the area ratios from the heat transfer model as inputs, the regression parameters of the heat transfer rate model are estimated before the parameters in the pressure drop model.

Condenser Heat Transfer Rate Model

5 regression parameters($U_{a,cond,rated}$, $n_{a,cond,rated}$, $U_{r,cond,tp,rated}$, $U_{r,cond,sh,rated}$ and $U_{r,cond,sc,rated}$) are needed in the heat transfer rate model as described by Eqns. (4.6), (4.7) and (4.8). The sub-model requires refrigerant mass flow rate as inputs, and training data are filtered for valid refrigerant mass flow rates according to Section 5.1.3. The objective function for estimating the regression parameters of the condenser heat transfer rate model, weighted according to the enthalpy difference across the condenser, is

$$J_{\dot{Q}_{cond}} = \Sigma weight_{h_{r,cond,in}-h_{r,cond,out},i}(\frac{\dot{Q}_{cond,pre,i} - \dot{Q}_{cond,mea,i}}{\dot{Q}_{cond,mea,i}})^2. \tag{5.6}$$

During the parameter estimation, constraints were applied to avoid unrealistic parameters. First of all, upper limits were applied on the refrigerant-side heat transfer coefficients to avoid unrealistically large parameters. For the single-phase sections, the upper limit, approximated by the Dittus-Boelter equation (Incropera et al., 2007), is

$$U_{r,cond,1\phi,rated,\max} = 0.023(\frac{c_{pr,1\phi}\mu_{r,1\phi}}{k_{r,1\phi}})^{0.4}(\frac{4\dot{m}_r}{\pi D_{cond}\mu_{r,1\phi}})^{0.8}\frac{k_{r,1\phi}}{D_{cond}}. \tag{5.7}$$

The upper limit of $U_{r,cond,tp,rated}$ was calculated by equations based on a correlation for the heat transfer coefficients of condensing flows (Shah, 1979). The equations to calculate the upper limit are

$$U_{r,cond,tp,rated,max} = \frac{2U_{r,cond,sc,rated,max}}{(\frac{P_{r,cond,in}}{P_{r,cric}})^{0.38}((1-x_{max})^{0.8} + 3.8x_{max}^{0.76}(1-x_{max})^{0.04})} \tag{5.8}$$

and

$$\frac{d}{dx}(-0.8(1-x)^{-0.2} + 3.8(0.76x^{-0.24}(1-x)^{0.04} - 0.04x^{0.76}(1-x)^{-0.96}))|_{x=x_{max}} = 0. \tag{5.9}$$

$U_{r,cond,sc,rated}$ and $U_{r,cond,sh,rated}$ were also limited to be smaller than or equal to $U_{r,cond,tp,rated}$ as it is common knowledge that the condensing flow heat transfer coefficient is higher than the single-phase ones.

To limit the heat transfer coefficient on the air side, the overall heat exchanger effectiveness of the simulation result with the largest subcooled section was limited to be less than 0.9999. The calculation of the heat exchanger effectiveness is depicted in Appendix A.

$n_{a,cond,rated}$ was restricted to be between 0.4 and 0.84 since the coefficient came from the Grimison correlation (Incropera et al., 2007) and the exponent of air mass flow rate in the correlation lie between 0.4 and 0.84 only.

To apply the constraints in the parameter estimation, the constrained optimization routine written in sequential quadratic programming (Nocedal and Wright, 2006) was used to minimize the objective function Eqn. (5.6).

Condenser Pressure Drop Model

The condenser pressure drop model is described by Eqn. (4.13) and requires the same inputs as the heat transfer model. The same set of data points was used to estimate the regression parameters of the pressure drop model: $C_{r,cond,\Delta P,sh}$, $C_{r,cond,\Delta P,tp,1}$, $C_{r,cond,\Delta P,tp,2}$, $C_{r,cond,\Delta P,sc}$ and $C_{r,cond,\Delta P,acc}$. The objective function is

$$J_{\Delta P_{r,cond}} = \Sigma weight_{\Delta P_{r,cond},i}(\frac{\Delta P_{r,cond,pre,i} - \Delta P_{r,cond,mea,i}}{\Delta P_{r,cond,mea,i}})^2. \tag{5.10}$$

Constrained optimization with sequential quadratic programming (Nocedal and Wright, 2006) was used to restrict the regression parameters $C_{r,cond,\Delta P,sh}$, $C_{r,cond,\Delta P,tp,1}$, $C_{r,cond,\Delta P,sc}$ and $C_{r,cond,\Delta P,acc}$ to be positive because negative frictional and positive accelerational pressure drop component in the condenser are impossible.

5.1.6 Evaporator Model

Similar to the condenser model, the evaporator model has regression parameters in two of its sub-models: the heat and mass transfer rate model and the pressure drop model. The regression parameters of heat and mass transfer rate model should

be estimated before the pressure drop model to provide the area ratios as inputs of the pressure drop model.

Evaporator Heat and Mass Transfer Rate Model

In the evaporator heat and mass transfer rate model, the regression parameters are $U_{a,evap,rated}$, $n_{a,evap,rated}$, $C_{a,evap,1}$, $U_{r,evap,sh,rated}$ and $U_{r,evap,tp,rated}$ as defined by Eqns. (4.8), (4.19), (4.20), (4.21) and (4.23). Its parameter estimation filtered the data points according to Section 5.1.3 to obtain data points with valid refrigerant mass flow rate and valid refrigerant enthalpy at the inlet of the expansion valve. As the model estimates both the sensible heat ratio and evaporator heat transfer rate, the objective function involves both outputs and is weighted according to the refrigerant enthalpy difference across the evaporator as shown in

$$\begin{aligned} & J_{\dot{Q}_{evap}} \\ = \; & \Sigma_i weight_{h_{r,evap,in}-h_{r,evap,out},i}[(\frac{\dot{Q}_{evap,pre,i}-\dot{Q}_{evap,mea,i}}{\dot{Q}_{evap,mea,i}})^2 + (\frac{SHR_{pre,i}-SHR_{mea,i}}{SHR_{mea,i}})^2]. \end{aligned} \tag{5.11}$$

The objective function Eqn. (5.11) was minimized by sequential quadratic programming (Nocedal and Wright, 2006) to restrict the regression parameters within realistic values. $U_{r,evap,sh,rated}$, representing the heat transfer coefficient of the

superheated section, is restricted by an upper limit estimated by the Dittus-Boetler equation (Incropera et al., 2007) as

$$U_{r,evap,sh,rated,\max} = 0.023(\frac{c_{pr,evap,sh}\mu_{r,evap,sh}}{k_{r,evap,sh}})^{0.4}(\frac{4\dot{m}_r}{\pi D_{evap}\mu_{r,evap,sh}})^{0.8}\frac{k_{r,evap,sh}}{D_{evap}}. \quad (5.12)$$

Representing the heat transfer coefficient of the evaporating flow, $U_{r,evap,tp,rated}$ is given an upper limit $U_{r,evap,tp,rated,max}$ estimated by

$$\begin{aligned} U_{r,evap,tp,rated,max} &= 30U_{r,evap,l,rated,\max}(\frac{\dot{m}_r}{\rho_{r,evap,l}\mu_{r,evap,l}})^{0.8}(\frac{\rho_{r,evap,l}c_{pr,evap,l}}{k_{r,evap,l}})^{0.4} \\ & k_{r,evap,l}\int_{x_{r,evap,in}}^{x_{r,evap,out}}\Phi(x_r)(1-x_r)^{0.8}dx_r \end{aligned} \quad (5.13)$$

and

$$U_{r,evap,l,rated,\max} = 0.023(\frac{c_{pr,evap,l}\mu_{r,evap,l}}{k_{r,evap,l}})^{0.4}(\frac{4\dot{m}_r}{\pi D_{evap}\mu_{r,evap,l}})^{0.8}\frac{k_{r,evap,l}}{D_{evap}}. \quad (5.14)$$

Eqns. (5.13) and (5.14) are originated from an evaporating flow correlation (Shah, 1982). The calculation of $\Phi(x_r)$ in Eqn. (5.13) is given in the same literature and is outlined in Appendix D.

To limit the magnitude of the air-side heat transfer coefficient, the heat exchanger effectiveness of the simulation result with the largest superheated section was limited to 0.9999. The calculation of the heat exchanger effectiveness is listed in Appendix C.

Limits were also imposed on Eqns. (4.20) and (4.21). Being analogous to fin efficiency, they should range between 0.01 and 0.09. This was maintained by imposing inequalities

$$0.01 \leq \frac{\tanh(C_{a,evap,1}U_{a,evap,rated}^{0.5})}{C_{a,evap,1}U_{a,evap,rated}^{0.5}} \leq 0.99 \tag{5.15}$$

and

$$0.01 \leq \frac{\tanh(C_{a,evap,1}(U_{a,evap,rated}c_{s,evap}/c_{pa,evap})^{0.5})}{C_{a,evap,1}(U_{a,evap,rated}c_{s,evap}/c_{pa,evap})^{0.5}} \leq 0.99 \tag{5.16}$$

in the parameter estimation process.

$n_{a,evap,rated}$ was also limited between 0.4 and 0.84 because of the same reason as the limitation on $n_{a,cond,rated}$ in Section 5.1.5.

The parameters $U_{a,evap,rated}$, $C_{a,evap,1}$, $U_{r,evap,tp,rated}$ and $U_{r,evap,sh,rated}$ were restricted to be positive to maintain their physical interpretations as heat transfer coefficients.

Evaporator Pressure Drop Model

The evaporator pressure drop model was trained with the same set of experimental data as the evaporator heat and mass transfer rate model. The sub-model Eqn. (4.25) contains regression parameters $C_{r,evap,\Delta P,sh}$, $C_{r,evap,\Delta P,tp,1}$, $C_{r,evap,\Delta P,tp,2}$ and $C_{r,evap,\Delta P,acc}$, and they were estimated by minimizing the objective function

$$J_{\Delta P_{r,evap}} = \Sigma weight_{\Delta P_{r,evap},i}(\frac{\Delta P_{r,evap,pre,i} - \Delta P_{r,evap,mea,i}}{\Delta P_{r,evap,mea,i}})^2. \tag{5.17}$$

To maintain the physical meaning of the regression parameters, sequential quadratic programming (Nocedal and Wright, 2006) was used to maintain all regression parameters to be positive.

5.1.7 Refrigerant Pipeline Model

Each of the hot gas line model, the liquid line model and the suction line model contains two sub-models: the pressure drop model and the heat loss model.

Refrigerant Pipeline Pressure Drop Model

The refrigerant pipeline pressure drop model consists of regression parameters $C_{pipeline,\Delta P,1}$, $C_{pipeline,\Delta P,2}$ and $C_{pipeline,\Delta P,3}$ as shown in Eqn. (4.32). The objective function is weighted by the pressure drop values as shown in

$$J_{\Delta P_{r,pipeline}} = \Sigma weight_{\Delta P_{r,pipeline},i}(\frac{\Delta P_{r,pipeline,pre,i} - \Delta P_{r,pipeline,mea,i}}{\Delta P_{r,pipeline,mea,i}})^2. \tag{5.18}$$

Sequential quadratic programming (Nocedal and Wright, 2006) was carried out to ensure $C_{pipeline,\Delta P,1}$ to be greater than zero, $C_{pipeline,\Delta P,2}$ to be within 1 and 2 and $C_{pipeline,\Delta P,3}$ to be within 0 and 1.

$C_{pipeline,\Delta P,1}$ represents friction factors which must be positive.

The limits of $C_{pipeline,\Delta P,2}$ and $C_{pipeline,\Delta P,3}$ are recognized by considering the pressure drop correlation of single-phase flow in Eqn. (4.9) and the friction factor calculation in the Moody's chart (Moody, 1944). In Eqn. (4.9), the exponent of

refrigerant mass flow rate is 2. In the Moody's chart, the steepest drop of friction factor with Reynolds number is caused by the friction factor calculation with laminar flow as shown in

$$f_{\Delta P,r} = \frac{64\pi D\mu_r}{\dot{m}_r}. \tag{5.19}$$

If Eq. (5.19) is substituted into Eqn. (4.9), the pressure drop calculation becomes

$$\Delta P_{r,1\phi} = \frac{512\mu_r \dot{m}_r L_{r,1\phi}}{\rho_r \pi} \tag{5.20}$$

Since the exponents of refrigerant mass flow rate and viscosity in Eqn. (5.20) are 1 and the exponents of refrigerant mass flow rate and viscosity in Eqn. (4.9) are 2 and 0 respectively, they form the limits on $C_{pipeline,\Delta P,2}$ and $C_{pipeline,\Delta P,3}$.

Refrigerant pipeline heat loss model

The refrigerant heat loss model is described in Eqn. (4.31) where $C_{pipeline,\Delta h,1}$ and $C_{pipeline,\Delta h,2}$ are regression parameters. The objective function is weighted with the heat loss values as shown in

$$J_{\dot{Q}_{HL,pipeline}} = \Sigma weight_{\dot{Q}_{HL,pipeline},i}(\frac{\dot{Q}_{HL,pipeline,pre,i} - \dot{Q}_{HL,pipeline,mea,i}}{\dot{Q}_{HL,pipeline,mea,i}})^2. \tag{5.21}$$

To identify the limits on the regression coefficients in the heat loss model, it was recognized that $C_{pipeline,\Delta h,2}$ is analogous to the exponent of Rayleigh number in Eqn. (4.30) and the exponent in Morgan correlation ranges between 0.058 and 0.333.

Hence 0.058 and 0.333 set the bounds of $C_{pipeline,\Delta h,2}$ during the minimization of the objective function Eqn. (5.21).

The limits on $C_{pipeline,\Delta h,1}$ were estimated by recognizing that it represents a heat transfer coefficient and heat transfer coefficient of natural convection increases with the temperature difference between the surface and the surroundings. Its lower limit was given by the lower limit of a heat transfer coefficient which is zero. Its upper limit was calculated from the data point with the largest $|T_{r,pipeline,in} - T_{amb}|$ in the experimental data. Since this data point should have the highest heat transfer coefficient among all data points, it was used to estimate the maximum value of $C_{pipeline,\Delta h,1}$. The upper limit of $C_{pipeline,\Delta h,1}$ was given by substituting the data from data point with the largest $|T_{r,pipeline,in} - T_{amb}|$ into Eqn. (4.31) to form

$$\frac{\dot{Q}_{loss,pipeline}}{(\frac{|T_{r,pipeline,in}-T_{amb}|}{T_{r,pipeline,in}})^{C_{pipeline,\Delta h,2,temp}}(T_{r,pipeline,in} - T_{amb})} \tag{5.22}$$

where $C_{pipeline,\Delta h,2,temp}$ in Eqn. (5.22) was obtained by estimating m in Eqn. (4.30) with the temperature difference $T_{r,pipeline,in} - T_{amb}$ and the tube diameter according to the Morgan correlation (Incropera et al., 2007).

The limits on $C_{pipeline,\Delta h,1}$ and $C_{pipeline,\Delta h,2}$ were applied to the parameter estimation process by sequential quadratic programming (Nocedal and Wright, 2006) .

5.1.8 FXO Model

The FXO model in Eqn. (4.34) has regression parameters D_{EXV}, L_{EXV}, $C_{EXV,1}$ and $C_{EXV,2}$. Its parameter estimation filtered the training data for valid refrigerant mass flow rate with the assistance of the compressor mass flow rate model. Data which showed inlet quality higher than the applicable range specified in Payne and O'Neal (2004) were eliminated because they were inapplicable to the two-phase flow adjustment model. The estimation process minimized the objective function weighted according to the refrigerant mass flow rate and the inlet quality as shown in

$$J_{\dot{m}_{r,FXO}} = \Sigma(weight_{x_{r,EXV,in},i} + weight_{\dot{m}_r,i})(\frac{\dot{m}_{r,EXV,pre,i} - \dot{m}_{r,mea,i}}{\dot{m}_{r,mea,i}})^2. \tag{5.23}$$

The minimization is weighted according to the inlet quality to avoid overemphasis on the non-faulted conditions because only a few data points with two-phase flow at valve inlet are available in each system. Since the number of data points with respect to refrigerant mass flow rate is imbalanced in each system, the weighting function Eqn. (5.23) also involved refrigerant mass flow rate.

To avoid the FXO model from violating the laws of physics, multiple constraints were identified to apply to its parameter estimation. Geometrical parameters in the FXO model must be positive. The increase of refrigerant mass flow rate with increasing inlet subcooling, as observed in experiments (Payne and O'Neal, 1999),

should also be predicted by the FXO model. These rules are enforced to the FXO model by applying the inequalities

$$D_{EXV} > 0, \tag{5.24}$$

$$L_{EXV} > 0 \tag{5.25}$$

and

$$\frac{d\dot{m}_{r,EXV,pre}(P_{r,FXO,mea}, h_{r,FXO,mea}, SC = 20[K])}{dSC} > 0 \text{ for all valid data points} \tag{5.26}$$

to the sequential quadratic programming process which estimates the regression parameters by minimizing Eqn. (5.23). The first two inequalities Eqns. (5.24) and (5.25) ensures that all geometrical parameters in the FXO model are positive. The third inequality Eqn. (5.26) ensures that the increase of refrigerant mass flow rate with increasing subcooling is predicted by the FXO model when the subcooling values are large. While the increasing trend of mass flow rate with increasing subcooling was observed in all experimental data, few experimental data show subcooling observations as large as 20K. Since the monotonic trend may not be true in Eqn. (4.34) for some sets of regression parameters, the inequality Eqn. (5.26) was enforced to the parameter estimation of the FXO model to reduce the probability of the model to predict the unrealistic decrease of refrigerant mass flow rate with increasing inlet subcooling.

5.1.9 TXV model

The TXV model in Section 4.2.6 contains regression parameters $C_{TXV,1}$, $C_{TXV,2}$, $C_{TXV,3}$, $C_{TXV,4}$, L_{EXV}, $C_{EXV,1}$, $C_{EXV,2}$, $C_{EXV,3}$, $C_{EXV,4}$ and $C_{EXV,5}$. Its parameter estimation required the same validity in its training data as the FXO model and the same data filtering rule was used. The objective function is

$$J_{\dot{m}_{r,TXV}} = \Sigma(weight_{\Delta SH_{evap},i} + weight_{x_{r,EXV,in},i} + weight_{\dot{m}_r,i})(\frac{\dot{m}_{r,EXV,pre,i} - \dot{m}_{r,mea,i}}{\dot{m}_{r,mea,i}})^2. \tag{5.27}$$

An additional weighting function on evaporator superheat is added to Eqn. (5.23) to form Eqn. (5.27) in order not to overemphasize data with controlled superheat. With sequential quadratic programming to minimize Eqn. (5.27), the inequalities Eqns. (5.25) and (5.26) were applied so as to create a TXV model that predicts the refrigerant mass flow rate with the changes of inputs correctly.

Since $C_{TXV,4}$ is situated inside the conditional statement in Eqn. (4.37) and its derivative is very hard to obtain correctly, $C_{TXV,4}$ cannot be minimized together with other regression parameters where their derivatives are estimated in the optimization algorithm. To estimate $C_{TXV,4}$, $C_{TXV,4}$ was first fixed at the pressure difference across the TXV bulb in the first data point in the training data set, and Eqn. (5.27) was minimized to estimate other regression coefficients. The procedure was repeated by using the pressure difference across the TXV bulb for all other data points as $C_{TXV,4}$ to implement the minimization. The $C_{TXV,4}$ which gave the smallest value of the

cost function Eqn. (5.27) was the value of $C_{TXV,4}$ and the minimization result given by this $C_{TXV,4}$ was the solution for the rest of the regression coefficients in the TXV model.

5.1.10 Fan Power Consumption Model

The fan power consumption models predict the change of power consumption with the heat exchanger fouling levels. However, the heat exchanger fouling cases in some systems were not tested by blocking the heat exchanger, and their fan power consumption measurement did not reflect the correct change of fan power with heat exchanger fouling levels. Therefore the estimation of the regression parameters in fan power consumption models should be independent of the faulted data from the laboratory, and the parameter estimation of the fan power consumption models should only depend on assumptions and catalog data from manufacturers.

Condenser Fan Power Consumption Model

The condenser fan power consumption model Eqn. (4.43) contains two regression parameters, and two different pieces of information are needed to determine the parameters. By assuming that the fan power consumption peaks at the non-faulted operation, the regression parameters can be obtained by solving

$$\frac{d\dot{W}_{cond,fan}}{d\dot{V}_{a,cond}}|_{\dot{V}_{a,cond}=\dot{V}_{a,cond,NF}} = C_{cond,fan,1} + 2C_{cond,fan,1}\dot{V}_{a,cond,NF} = 0 \qquad (5.28)$$

and

$$\dot{W}_{cond,fan,NF} = C_{cond,fan,1}\dot{V}_{a,cond,NF} - C_{cond,fan,1}\dot{V}^2_{a,cond,NF} \tag{5.29}$$

simultaneously, where $\dot{V}_{a,cond,NF}$ is the mean condenser airflow under non-faulted condition and $\dot{W}_{cond,fan,NF}$ is the mean condenser fan power consumption under non-faulted condition.

Evaporator Fan Power Consumption Model

To estimate the regression parameters in the evaporator fan power consumption model Eqn. (4.45), it was assumed that normalizing Eqn. (4.45) with the fan power consumption and evaporator airflow at the non-faulted condition made the model become applicable to all evaporator fans. A stall fan power consumption was also assumed to be greater than or equal to 10% of the fan power consumption at the non-faulted condition. This transforms the model Eqns. (4.44) and (4.45) into

$$\frac{\dot{W}_{evap,fan,temp}}{\dot{W}_{evap,fan,NF}} = C_{evap,fan,temp,1} + C_{evap,fan,temp,2}\frac{\dot{V}_{a,evap}}{\dot{V}_{a,evap,NF}} + C_{evap,fan,temp,3}(\frac{\dot{V}_{a,evap}}{\dot{V}_{a,evap,NF}})^2 \tag{5.30}$$

and

$$\frac{\dot{W}_{evap,fan}}{\dot{W}_{evap,fan,NF}} = \begin{cases} 0.1 & \text{if } \frac{\dot{W}_{evap,fan,temp}}{\dot{W}_{evap,fan,NF}} < 0.1 \\ \frac{\dot{W}_{evap,fan,temp}}{\dot{W}_{evap,fan,NF}} & \text{otherwise} \end{cases} . \tag{5.31}$$

Since the model was assumed applicable to all evaporator fans, the regression parameters $C_{evap,fan,temp,1}$, $C_{evap,fan,temp,2}$ and $C_{evap,fan,temp,3}$ in Eqn. (5.30) can be esitmated from the evaporator fan power consumption data in the catalogs of any

manufacturers. The data from two different 3-ton packaged unit were used to estimate the regression parameters. They were given in Tables 4.2 and 4.3.

After normalizing the data in Tables 4.2 and 4.3 with the performance at 1200cfm, the regression parameters in Eqn. (5.30) were estimated with the normalized data by linear regression. The results showed a small deviation of 2.57% between the predicted normalized fan power consumption and the normalized fan power consumption from the manufacturer data.

The change of the estimated normalized evaporator fan power consumption with the normalized evaporator airflow is shown in Figure 5.3.

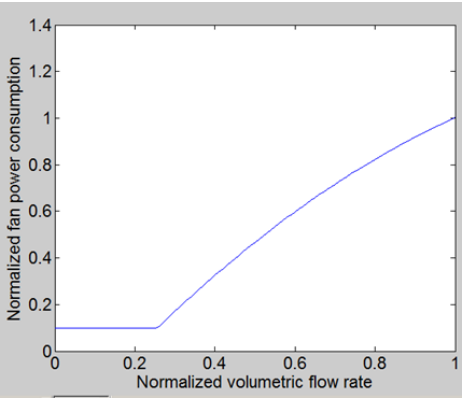

Figure 5.3. Change of the estimated normalized evaporator fan power consumption with the normalized evaporator airflow.

Figure 5.3 shows the fan is stalled when normalized evaporator airflow is smaller than 0.26 and that the evaporator fan power consumption is increasing with the evaporator airflow. This curve also follows the trend shown in Figure 4.12. This reveals that the model is realistic to estimate the fan power consumption without training the model with experimental data.

When the evaporator fan power consumption model is used in system simulation, $\dot{V}_{a,evap,NF}$ in Eqn. (5.30) is given by the mean evaporator airflow at non-faulted condition in the experimental data, and $\dot{W}_{evap,fan,NF}$ in Eqn. (5.31) is given by the mean evaporator fan power consumption in the experimental data.

5.2 Component Model Validation

To validate the component models constructed in Chapter 4, the estimation results of various component models are compared with the experimental data of all cooling systems tabulated in Table 3.1. Since the fan power consumption measurement in some heat exchanger fouling cases was not conducted to reflect the realistic change of fan power consumption with heat exchanger fouling, the fan power consumption models were not validated with the measured fan power consumption. The numerical values of unit-specific normalization and regression parameters are tabulated in Appendix F, and the comparison results are discussed in this section.

5.2.1 Statistical Indicators for Comparison

Several statistical indicators are used to compare the experimental and simulation results. The first one is the coefficient of determination whose definition is illustrated as

$$\text{Coefficient of Determination} = \frac{\Sigma_i (X_{pre,i} - X_{mea,i})^2}{\Sigma_i (X_{mea,i} - \overline{X_{mea}})^2}. \tag{5.32}$$

Coefficient of determination in Eqn. (5.32) shows how well the model accounts for the experimental data. An ideal model should have a coefficient of determination equal to one.

The other indicator is the average absolute deviation defined as

$$\text{Average Absolute Deviation} = |\overline{\frac{X_{pre} - X_{mea}}{X_{mea}}}|. \tag{5.33}$$

The average absolute deviation in Eqn. (5.33) shows the average magnitude of the relative deviation between the simulation and experimental results.

The third indicator is the absolute maximum deviation, defined by

$$\text{Absolute Maximum Deviation} = \max |\frac{X_{pre} - X_{mea}}{X_{mea}}|. \tag{5.34}$$

The absolute maximum deviation in Eqn. (5.34) describes the behavior of outliers in the simulation.

The final indicator is the average deviation as shown in

$$\text{Average Deviation} = \overline{\frac{X_{pre} - X_{mea}}{X_{mea}}}. \tag{5.35}$$

The average deviation in Eqn. (5.35) indicates the magnitude of bias in the estimation results.

For most of the models, their performance is investigated by Eqns. (5.33), (5.34) and (5.35). However, because the accuracy of all pressure drop models and the

heat loss models of pipelines should not be compared to their own magnitude, their statistical indicators are expressed in absolute magnitude shown in

$$\text{Average Absolute Deviation} = \overline{|X_{pre} - X_{mea}|}, \tag{5.36}$$

$$\text{Absolute Maximum Deviation} = \max |X_{pre} - X_{mea}| \tag{5.37}$$

and

$$\text{Average Deviation} = \overline{X_{pre} - X_{mea}}. \tag{5.38}$$

5.2.2 Component Models Not Established due to Insufficient Data

Some components did not have enough experimental data for model construction. For example, the pressure drop across the hot gas line of system I was not recorded and could not be modeled. Table 5.2 summarizes the component sub-models which were not constructed.

5.2.3 Compressor Model

Compressor Mass Flow Rate Model

The estimation of the compressor mass flow rate model in experimental scenarios is compared with measurement data in Figure 5.4, with values non-dimensionalized by the average measured compressor mass flow rate.

Table 5.2. List of component sub-models not established.

System	Sub-models
I	Hot gas line pressure drop, hot gas line heat loss, condenser pressure drop, liquid line pressure drop, liquid line heat loss, suction line pressure drop, suction line heat loss, evaporator fan power consumption and condenser fan power consumption
II	Hot gas line pressure drop and hot gas line heat loss
III	None
IV	None
V	None
VI	Evaporator pressure drop
VII	None
VIII	Evaporator pressure drop
IX	Evaporator pressure drop
X	Evaporator pressure drop
XI	Hot gas line pressure drop, hot gas line heat loss, expansion valve model, suction line pressure drop and suction line heat loss

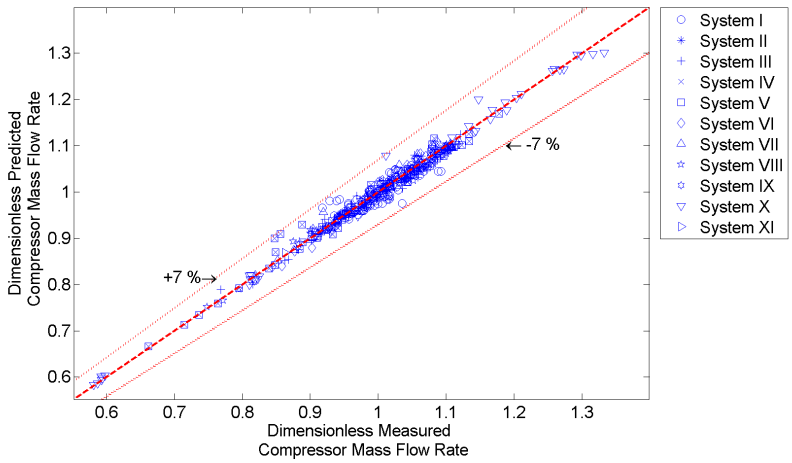

Figure 5.4. Comparison of mass flow rate between the predictions of the compressor model and the measurements.

Figure 5.4 shows that the compressor refrigerant mass flow rates of all systems are estimated within 6.67%. Detailed analysis reveals that only systems I and II

show coefficients of determination smaller than 0.93. However, the average absolute deviation of both systems are small (1.85% and 1.07%), showing that the low coefficients of determination do not mean that the models are too inaccurate to be used.

Compressor Power Consumption Model

The comparison between the simulation and experimental results of the compressor power is plotted as Figure 5.5, with values non-dimensionalized by the average measured compressor power consumption.

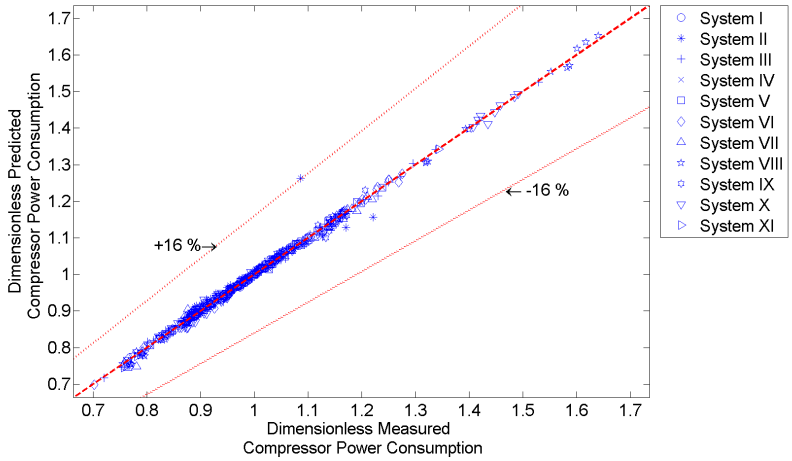

Figure 5.5. Comparison of compressor power consumption between the predictions of the compressor model and the measurements.

Figure 5.5 shows only one outlier from system II exists while all data points are estimated within 5.31% from the measured power consumption. This shows that the model can account for the power consumption of compressors under most situations accurately, and the outlier is only an anomaly.

Modeling of Enthalpy Gain of Refrigerant across Compressor

The simulation and experimental results of enthalpy gain by refrigerant through the compressor are shown in Table 5.3.

Table 5.3. Results of enthalpy gain by refrigerant through compressor.

System	Coefficient of determination	Mean value [J/kg]	Average absolute deviation [%]	Absolute maximum deviation [%]	Average deviation [%]
I	0.8256	4.47E+004	1.45	4.30	-0.25
II	0.9348	3.81E+004	2.15	4.82	1.02
III	0.9495	4.29E+004	2.66	10.29	-0.41
IV	0.9515	4.96E+004	1.32	10.89	-0.34
V	0.9515	4.96E+004	1.32	10.89	-0.34
VI	0.9156	4.98E+004	1.86	10.34	-0.50
VII	0.9794	3.38E+004	1.88	7.57	0.44
VIII	0.9948	4.53E+004	1.86	4.57	-0.85
IX	0.4723	4.50E+004	7.19	22.23	1.33
X	0.9424	3.97E+004	4.66	9.87	2.42
XI	0.9932	3.36E+004	1.00	2.13	-0.23

The average absolute deviations are all smaller than 5% in Table 5.3 except system IX. A parity plot for the refrigerant enthalpy gain through the compressor of system IX is given in Figure 5.6.

Four outliers exist in Figure 5.6, and all the other data points in Figure 5.6 are estimated within 11% from the experimental observations. As the model accounts for the enthalpy difference across other systems within 12% as shown in Table 5.3, the accuracy of the model is on par as the others.

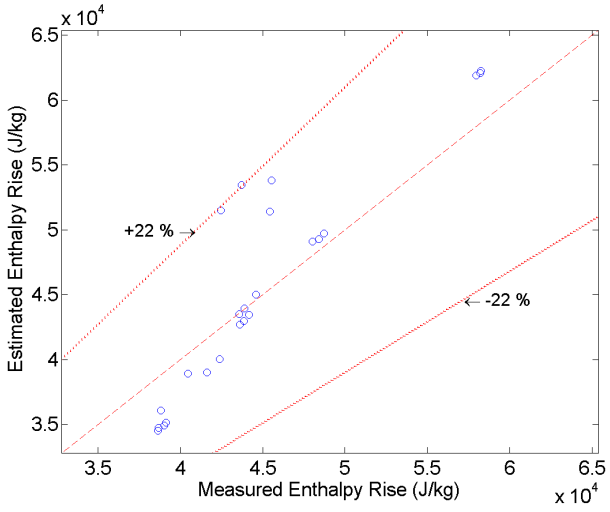

Figure 5.6. Parity plot of experimental and simulation enthalpy gain across compressor of system IX.

5.2.4 Condenser Model

Condenser Heat Transfer Rate Model

The estimated condenser heat transfer rates are compared to experimental results in Table 5.4.

In Table 5.4, the largest absolute maximum deviation and the lowest coefficient of determination are found for system VIII with values at 8.20% and 0.7919 respectively. The deviation is further studied by a parity plot in Figure 5.7.

The 8.20% absolute maximum deviation and 0.7919 coefficient of determination are a consequence of three overestimated outliers in Figure 5.7, and other data points do not exhibit outlying behavior. If these outliers are removed, the absolute maximum deviation will be 4.20% and the coefficient of determination will be 0.8390. These

Table 5.4. Results of condenser heat transfer model.

System	Coefficient of determination	Mean value [W]	Average absolute deviation [%]	Absolute maximum deviation [%]	Average deviation [%]
I	0.8589	12488	1.23	5.97	-0.01
II	0.9322	22493	0.69	2.01	0.04
III	0.9808	12303	1.04	4.26	0.01
IV	0.9840	11696	1.11	5.36	0.23
V	0.9840	11696	1.11	5.36	0.23
VI	0.9784	23622	0.84	3.38	-0.26
VII	0.9857	10239	0.70	2.87	0.25
VIII	0.7919	12434	2.93	8.20	-0.93
IX	0.9572	12469	1.98	4.33	-0.71
X	0.9920	12692	0.55	2.50	0.05
XI	0.9520	12903	0.88	3.66	0.10

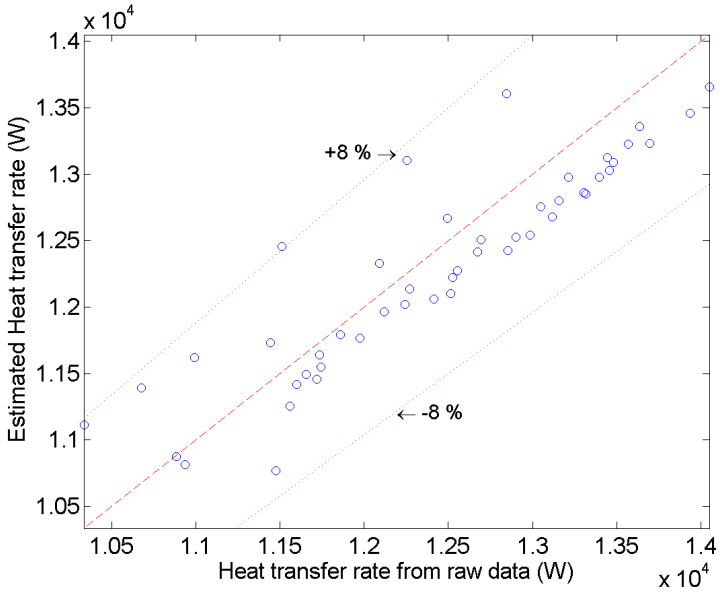

Figure 5.7. Parity plot of experimental and simulation condenser heat transfer rate of system VIII.

values are similar to the ones from the other systems, and this shows that the outliers

are the major reason for the large deviation and the model of system VIII performs as accurate as the models of the other systems.

Condenser Pressure Drop Model

The estimated pressure drop across the condenser is compared with experimental results in Table 5.5.

Table 5.5. Results of condenser pressure drop model.

System	Mean value [kPa]	Average absolute deviation [kPa]	Maximum deviation [kPa]	Average deviation [kPa]	Mean pressure rise across compressor [kPa]
I	39.34	0.05917	0.5879	0.02027	996.9
II	100.81	0.07383	0.1508	0.01669	1123
III	65.56	0.06699	0.5413	0.02064	1944
IV	110.08	0.1175	1.034	0.08376	2115
V	110.08	0.1175	1.034	0.08376	2115
VI	170.91	0.03367	0.3029	-0.008964	1884
VII	60.32	0.03335	0.2391	-0.01519	1574
VIII	67.39	0.1308	0.4202	0.07059	1829
IX	58.43	0.9891	4.142	0.9585	1268
X	21.29	0.9721	2.851	0.6873	1149
XI	66.34	0.0203	0.1564	-0.004308	1318

The maximum magnitude of deviation of the condenser pressure drop model is 0.33% of the pressure gain across the compressor in Table 5.5 and is negligible comparing to the pressure increase experienced by the compressors in the systems.

5.2.5 Expansion Valve Model

FXO model

The validation of the FXO model is conducted by comparing the mass flow rate estimated by the FXO model with the reference values obtained by data filtering in Section 5.1.3. The reference values of mass flow rates consist of measured refrigerant mass flow rates, estimated refrigerant mass flow rate by energy balance of heat exchangers and mass flow rates estimated by the compressor mass flow rate model based on experimental observations across the compressor. The comparison is shown in Table 5.6.

Table 5.6. Results of FXO mass flow rate model.

System	Coefficient of determination	Mean value [kg/s]	Average absolute deviation [%]	Absolute maximum deviation [%]	Average deviation [%]
I	0.7266	0.05719	1.47	5.48	0.59
III	0.9655	0.06531	0.96	3.53	-0.01
IV	0.9455	0.06516	0.85	4.03	-0.19
VI	0.9600	0.1149	1.24	5.68	0.28
IX	0.9970	0.06853	0.26	0.59	0.01

The coefficient of determination of system I in Table 5.6 is smaller than that of other systems. The FXO model of system I is further examined with the residual plot Figure 5.8 that studies the deviation between the estimation and reference values with subcooling at expansion valve inlet.

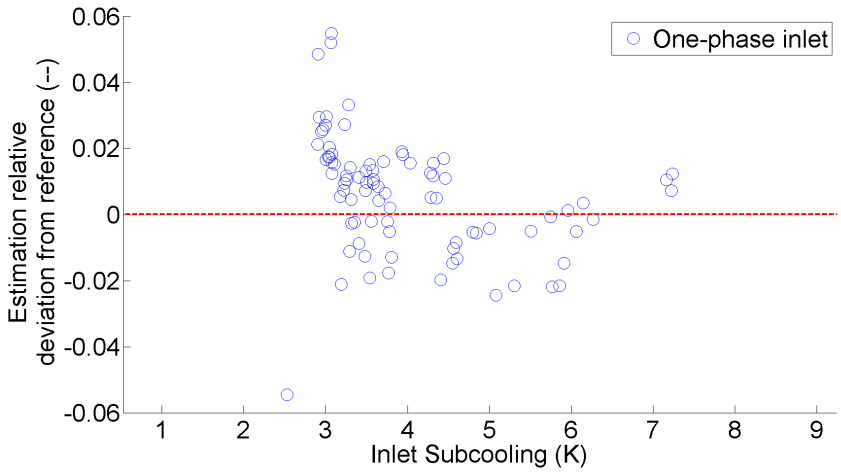

Figure 5.8. Residual plot of the relative difference between reference and simulated FXO mass flow rate with FXO inlet subcooling of system I.

The outlier in Figure 5.8 occurs at a subcooling smaller than 3K. As the reference refrigerant mass flow rate of the outlier was calculated instead of being measured and the uncertainties of the reference values are larger than the uncertainties of the measured values, the larger uncertainty leads to a higher deviation at the outlier in Figure 5.8.

TXV model

The TXV models are validated by comparing the estimated mass flow rate of the TXV model with reference values obtained by the data filtering method in Section 5.1.3. The results are tabulated in Table 5.7.

Table 5.7. Results of TXV mass flow rate model.

System	Coefficient of determination	Mean value [kg/s]	Average absolute deviation [%]	Absolute maximum deviation [%]	Average deviation [%]
II	0.7808	0.1082	1.47	4.17	0.15
V	0.8133	0.06045	3.42	14.27	-1.15
VII	0.9172	0.04916	1.87	6.77	-0.39
VIII	0.8225	0.06138	1.97	8.33	-0.56
X	0.6119	0.0645	4.63	12.99	-1.38

System X in Table 5.7 shows a lower coefficient of determination compared with other systems, and this is examined by creating a residual plot of the deviation of the mass flow rates of system X as shown in Figure 5.9.

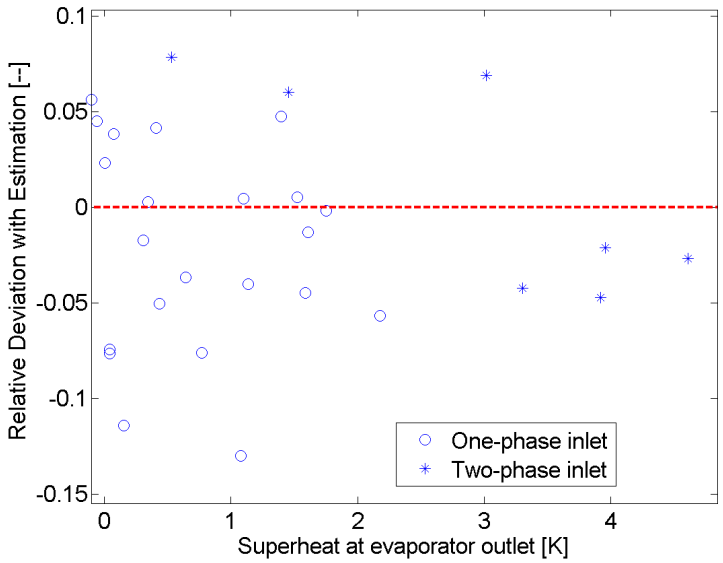

Figure 5.9. Residual plot of the deviation of TXV mass flow rate of system X with evaporator outlet superheat.

Figure 5.9 shows that the system has multiple data points measured with zero evaporator outlet superheat. This is unusual because TXVs are used to maintain a positive superheat that is higher than 3K. Since the data do not conform the expected behavior of normal TXVs, it is hard for the TXV model to predict the abnormal behavior, resulting in the large deviation.

5.2.6 Evaporator Model

Evaporator Heat and Mass Transfer Rate Model

The comparison between the estimated and measured evaporator heat transfer rate is conducted with heat transfer rates non-dimensionalized by the average measured evaporator heat transfer rate and the results are shown in Figure 5.10.

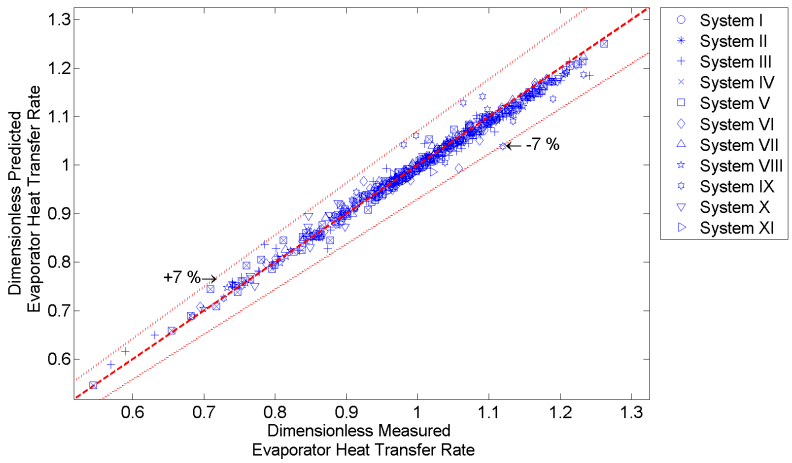

Figure 5.10. Comparison of evaporator heat transfer rate between estimations and measurements.

The evaporator heat transfer rates in Figure 5.10 were estimated within 7% of the measured value without systematic bias.

The comparison of the estimated and experimental evaporator SHR is conducted in Table 5.8.

Table 5.8. Results of evaporator sensible heat ratio model.

System	Coefficient of determination	Mean value [–]	Average absolute deviation [%]	Absolute maximum deviation [%]	Average deviation [%]
I	0.9237	0.8119	1.87	7.45	0.55
II	0.9981	0.8267	0.51	1.42	-0.24
III	0.9907	0.8319	1.77	18.22	0.07
IV	0.9854	0.7911	2.77	13.68	-0.30
V	0.9854	0.7911	2.77	13.68	-0.30
VI	0.9790	0.7943	2.10	14.66	-0.59
VII	0.9956	0.8452	1.53	14.46	-0.71
VIII	0.9939	0.8937	1.17	7.96	-0.08
IX	0.9874	0.8478	2.10	9.51	-0.33
X	0.9961	0.8914	1.08	5.54	-0.36
XI	0.9245	0.8448	2.67	10.35	0.65

Although the coefficient of determination of all systems are higher than 0.92, the absolute maximum deviation for five systems are higher than 10%. As the objective function Eqn. (5.11) to estimate the regression parameters of the heat and mass transfer rate model is not weighted with the sensible heat ratio, the estimated sensible heat ratios are prone to larger scattering at extreme cases, leading to high absolute maximum deviations for various systems in Table 5.8.

Evaporator Pressure Drop Model

The values of estimated and experimental pressure drop are tabulated in Table 5.9.

Table 5.9. Results of evaporator pressure drop model.

System	Mean value [kPa]	Average absolute deviation [kPa]	Maximum deviation [kPa]	Average deviation [kPa]	Mean pressure rise across compressor [kPa]
I	40.15	0.9346	3.796	0.1891	996.9
II	57.16	2.383	8.759	0.4352	1123
III	34.91	2.852	12.69	-1.794	1944
IV	11.85	2.075	8.136	1.293	2115
V	11.85	2.075	8.136	1.293	2115
VII	69.73	8.64	25.84	1.189	1574
XI	105.83	15.75	43.99	2.802	1318

The deviation values in Table 5.9 are at maximum 3.34% relative to the overall pressure increase across the compressor, showing that the deviation is negligible.

5.2.7 Refrigerant Pipeline Model

Refrigerant Pipeline Pressure Drop Model

The experimental and simulation results of the pressure drop across the hot gas line, the liquid line and the suction line are shown in Table 5.10, 5.11 and 5.12 respectively.

Table 5.10. Results of pressure drop by hot gas line.

System	Mean value [kPa]	Average absolute deviation [kPa]	Maximum deviation [kPa]	Average deviation [kPa]	Mean pressure rise across compressor [kPa]
I	115.48	3.69	10.89	-0.9454	996.9
III	31.54	7.745	17.85	-1.886	1944
IV	30.50	12.29	32.64	-7.681	2115
V	30.50	12.29	32.64	-7.681	2115
VI	98.71	14.92	39.78	7.911	1884
VII	6.55	1.508	4.072	-0.3982	1574
VIII	14.62	15.29	26.07	7.246	1829
IX	2.07	2.613	4.015	1.525	1268
X	17.54	2.823	6.31	0.9278	1149

Table 5.11. Results of liquid line pressure drop model.

System	Mean value [kPa]	Average absolute deviation [kPa]	Maximum deviation [kPa]	Average deviation [kPa]	Mean pressure rise across compressor [kPa]
I	11.29	2.603	7.244	-0.6775	996.9
II	8.11	12.45	14.92	6.33	1123
III	3.32	11.13	15.13	10.01	1944
IV	0.03	2.613	3.121	2.613	2115
V	0.03	2.613	3.121	2.613	2115
VI	30.77	13.01	32.56	7.33	1884
VII	82.40	9.738	24.09	-2.701	1574
VIII	6.67	4.488	8.504	1.804	1829
IX	24.18	2.702	5.918	0.009771	1268
X	53.51	19.38	39.56	2.1	1149
XI	10.55	2.042	7.206	-1.045	1318

The deviation values in Table 5.10 are at least two orders smaller than the pressure rise across the compressor in each system, showing that the deviations are negligible. The results of the liquid line pressure drop model in Table 5.11 and the suction

Table 5.12. Results of suction line pressure drop model.

System	Mean value [kPa]	Average absolute deviation [kPa]	Maximum deviation [kPa]	Average deviation [kPa]	Mean pressure rise across compressor [kPa]
II	4.50	3.546	12.25	-2.496	1123
III	13.69	1.358	3.785	-0.06522	1944
IV	11.13	2.109	6.269	-0.003257	2115
V	11.13	2.109	6.269	-0.003257	2115
VII	15.51	0.6088	1.388	0.05522	1574
VIII	25.40	2.882	7.514	0.2668	1829
IX	32.95	3.253	13.65	-1.002	1268
X	38.40	2.234	4.479	0.4379	1149

line pressure drop model in Table 5.12 are similar to the hot gas line results. The magnitude of their maximum deviations are 3.4% and 1.1% relative to the pressure gain across the compressor.

Refrigerant Pipeline Heat Loss Model

The results of simulation and experiment of the heat loss from the hot gas line, the liquid line and the suction line are tabulated in Tables 5.13, 5.14 and 5.15.

The magnitude of deviation in Table 5.13 is at maximum 3.52% compared to the mean condenser heat transfer rate, showing that the deviation of heat loss model is negligible. The maximum values of the absolute maximum deviation estimated by the liquid line model in Table 5.14 and the suction line model in Table 5.15 are 1.2% and 4.6% relative to the heat transfer rates of the adjacent heat exchangers. Although the deviation of the liquid line model is insufficient to cause bias in the estimation of heat

Table 5.13. Results of heat loss by hot gas line.

System	Mean value [W]	Average absolute deviation [W]	Maximum deviation [W]	Average deviation [W]	Mean Condenser Heat Transfer Rate [W]
I	0.00	0.00	0.00	0.00	12488
III	481.36	186.54	407.66	-180.34	12303
IV	836.44	118.71	411.53	-95.85	11696
V	836.44	118.71	411.53	-95.85	11696
VI	0.00	0.00	0.00	-0.00	23622
VII	151.29	7.79	24.94	-1.78	10239
VIII	274.09	7.59	25.64	-0.61	12434
IX	280.23	84.81	286.54	-4.99	12469
X	106.01	49.09	66.72	30.21	12692

Table 5.14. Results of liquid line heat loss model.

System	Mean value [W]	Average absolute deviation [W]	Maximum deviation [W]	Average deviation [W]	Mean Condenser Heat Transfer Rate [W]
II	28.44	5.73	13.50	-1.53	22493
III	49.44	38.58	146.75	-36.33	12303
IV	96.48	42.20	115.38	-40.16	11696
V	96.48	42.20	115.38	-40.16	11696
VI	3.48	2.29	6.94	-0.81	23622
VII	71.31	28.10	58.30	20.83	10239
VIII	65.38	4.56	16.69	0.07	12434
IX	91.89	4.93	12.96	1.38	12469
X	11.93	3.75	7.28	1.80	12692

transfer rate in a cycle simulation, the one of the suction line model is significant. The deviation of the suction line model is further studied by plotting the measured and estimated suction line heat loss of system IX in Figure 5.11.

Table 5.15. Results of suction line heat loss model.

System	Mean value [W]	Average absolute deviation [W]	Maximum deviation [W]	Average deviation [W]	Mean Evaporator Heat Transfer Rate [W]
II	-11.6563	4.295	8.941	-1.388	1.844e+004
III	-58.2464	14.06	104.2	8.577	9815
IV	-105.6048	16.26	59.17	-5.87	9501
V	-105.6048	16.26	59.17	-5.87	9501
VI	-78.5521	41.98	99.49	10.39	1.789e+004
VII	-171.2915	23.66	72.1	8.872	8549
VIII	-286.3124	160.3	278.7	160.3	9795
IX	-398.5486	276.8	446	276.8	9640
X	-368.5845	237.5	356.4	237.5	1.017e+004

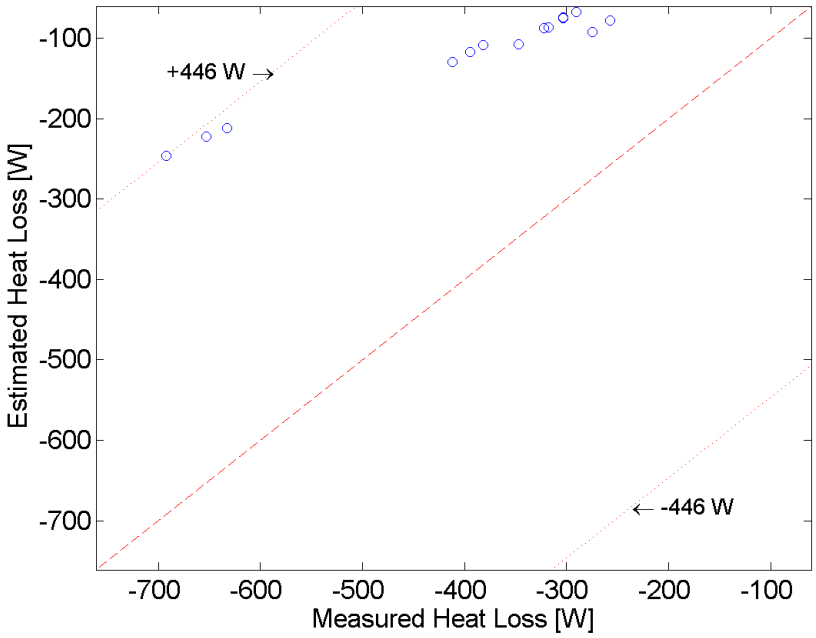

Figure 5.11. Parity plot of measured and estimated suction line heat loss of system IX.

Figure 5.11 shows an underestimation of suction line heat gain. This is a result to restrict the magnitude of the suction line heat transfer coefficient by Eqn. (5.22) during the parameter estimation process. Although the limitation causes overestimation of suction line heat loss, this avoids the model from predicting unreasonably large suction line heat gain in other conditions. Although the model cannot predict the suction line heat loss accurately, it can still provide reasonable estimation of the heat loss of the suction lines.

5.3 Applicability Domains of Component Models

An applicability domain of a mathematical model is a space consisting of input vectors to a model with which the model can estimate its outputs without significant modeling uncertainty. There are two methods to define the applicability domains of component models: leverage calculation and limits defined in literature. After defining the domains, the distance of an operating point of a component model to the boundary of its applicability domain can be defined.

5.3.1 Calculation of Leverage

For most component models which are trained from training data, their applicability domains can be defined by the calculation of leverage (Atkinson, 1985)

which can be defined differently for a linear regression model and a nonlinear regression model. For a linear regression model

$$\vec{y} = \mathbf{X}\vec{\beta} \tag{5.39}$$

where $\vec{y}$ is the output vector,

$$\mathbf{X} = [\vec{x}_1 ... \vec{x}_N]^T \text{for N input vectors} \tag{5.40}$$

and $\vec{\beta}$ is the regression coefficient vector, the leverage of any output

$$y = \vec{x}^T \vec{\beta} \tag{5.41}$$

with an input vector $\vec{x}$ is defined as

$$\theta = \vec{x}^T (\mathbf{X}_{train}^T \mathbf{X}_{train})^{-1} \vec{x}. \tag{5.42}$$

Leverage is an estimate of the influence of a data point on the result of the regression. The higher the leverage of an input vector, the higher the influence of the data point associated with the input vector to the result of the regression relative to the training data of the model. If a leverage of an input vector $\vec{x}$ is higher than any leverages calculated from the input vectors of the training data, a data point associated with the input vector is expected to have a significant

influence on the regression result. This means that the data point should be included in the regression to estimate $\vec{\beta}$ in Eqn. (5.39), or Eqn. (5.39) cannot estimate the output corresponding to the input vector reliably. Since component models are trained by regression with experimental data, by calculating the leverage of an input vector to a component model, the position of the estimation result relative to the applicability domain of a component model can be determined. If the leverage of an estimation output from the component model parameters is higher than the maximum leverage calculated from the input vectors within the training data of the model, the component model is said to operate outside its applicability domain to obtain the estimation result.

For non-linear regression models, the leverage of an input vector $\vec{x}$ can be calculated by

$$y = f(\vec{x}, \vec{\beta}), \tag{5.43}$$

$$\theta = \vec{\phi_i}^T (\mathbf{\Phi}_{train}^T \mathbf{\Phi}_{train})^{-1} \vec{\phi}, \tag{5.44}$$

$$\vec{\phi_i} = \frac{\partial f(\vec{x}, \vec{\beta})}{\partial \vec{\beta}} \tag{5.45}$$

and

$$\Phi = [\vec{\phi}_1 ... \vec{\phi}_N]^T \text{for N input vectors.} \tag{5.46}$$

To calculate leverages with any input vectors, the covariance matrix ($(\mathbf{X}_{train}^T \mathbf{X}_{train})^{-1}$ for linear models and $(\mathbf{\Phi}_{train}^T \mathbf{\Phi}_{train})^{-1}$) for each component model

are needed. The covariance matrices are calculated after the parameter estimation of the models.

5.3.2 Limits from Literature

While the applicability domains for component models containing regression parameters can be defined by the calculation of leverage, the applicability domains of the models that only contain existing empirical parameters were not defined by this method. In this case, only the applicability domain of the equation to calculate the mass flow rate adjustment factor in the expansion valve model for two-phase flow entering the valve in Appendix E was not defined. Since the literature of the equation (Payne and O'Neal, 2004) gave the maximum thermodynamic quality at the expansion valve inlet in the training data of the equation, these maximums were used to define the applicability domain of the two-phase adjustment factor equation. If the equation receives any expansion valve inlet thermodynamic quality higher than the maximums as its inputs, the equation will be considered as operating out of their applicability domain. These maximums are tabulated in Table 5.16.

Table 5.16. Maximum thermodynamic quality in the training data for the refrigerant mass flow rate adjustment factor in Appendix E (Payne and O'Neal, 2004).

Refrigerant	Maximum thermodynamic quality
R410A	0.087
R407C	0.047
R22	0.102

5.3.3 Distance to the Boundary of the Applicability Domains

For the applicability domains defined by leverage calculation, the boundary of the domains is defined by the maximum leverage calculated within the training data points, and the distance of an estimation result to the boundary of the applicability domain is defined by

$$\delta = \theta - max(\theta_{train,1}, \theta_{train,2}...). \tag{5.47}$$

The boundary of the applicability domain of the refrigerant mass flow rate adjustment factor equation is defined by the maximum thermodynamic quality at the expansion valve inlet in its training data as shown in Table 5.16. The distance of an estimation result to the boundary is calculated by subtracting the thermodynamic quality at the expansion valve inlet to the maximum thermodynamic quality according to the type of refrigerant as listed in Table 5.16.

In both cases, a positive distance to the boundary means that the component model is operating outside its applicability domain, and a larger distance means that the component model is operating further away from its applicability domain. However, a component model operating outside its applicability domain does not mean that the system model becomes unreliable. Further investigation of the effect of the component models being unreliable on the system model result is discussed in Section 6.4.

5.4 Summary

The chapter depicts how parameter estimation was carried out at the component level and the additional data filtering required to estimate valid parameters. Objective functions for different component models are introduced. A weighting scheme dependent on the experimental data was used to avoid biased parameters due to imbalanced test matrices. Results of the component-level parameters are compared with the experimental results to validate the component models. The applicability domains of component models are also defined by the calculation of leverage of linear and non-linear models and the range of training data given in the literature.

CHAPTER 6. SYSTEM TUNING AND MODELING

With the estimated parameters from Chapter 5, the component models can be joined together to form an implicit model of a vapor compression system. To solve the implicit model, an iterative solver was used. Bias in estimating the amount of charge inside a system based on measured refrigerant volume, thermodynamic equilibrium densities and void fraction models was removed by a charge tuning method (Shen, 2006). To form a system model to generate a database of performance data, the following is described in this chapter.

1. The construction of a fast and robust solver of the system model:

   After combining the component models to create a system model, an iterative solver is needed to solve the implicit model. Since the aim of the project is to develop a fast system solver to generate a large database in a short time, the solver should be designed for a short computational time. Also, the system model aims at solving off-design conditions and may result in calculation with off-design and unrealistic variables. This may cause divergence of the solver. The speed and the robustness of the solver should be examined to see if the system model achieve the goal of the project.

2. To include fault models in the system model:

To simulate the fault impacts on system performance, the fault models developed in Chapter 4 are embedded to the system model and its solution process.

3. To tune the charge estimation of the model:

   It is validated that charge tuning helps to eliminate the bias in charge estimation due to inaccuracies in void fraction models, equation of states and the geometry of the system (Shen, 2006). The two-point tuning method is re-visited to examine if improvements can be made for higher estimation accuracy and if the modified method is better the two-point tuning method.

4. To assess the reliability of the system model according to the applicability domain of component models:

   Although Section 5.3 describes how to determine the applicability domains of component models, the effect of component models operating outside their applicability domains to the accuracy of the system model is unknown. Its effect should be quantified to understand when a system simulation output should be rejected because of simulation outside the applicability domains of component models.

6.1 System Modeling

To simulate the performance of a system, the component models trained in Chapter 5 were connected together and were solved simultaneously. Since some

component model inputs depended on the outputs of other models, the connection of component models created a system model in the form of an implicit mathematical function, and the component model inputs which are independent to other models became the inputs to the system model. The independent variables and residuals were arranged so that the system model could be solved with implicit function solvers quickly and robustly. A pre-tuning simulation algorithm was first created to conduct charge tuning on the system model with experimental data. This improved the accuracy of the model to estimate system charge inventory, and the final simulation model that estimated charge inventory accurately could be developed from the pre-tuning simulation algorithm.

6.1.1 Inputs to the System Model

The component models trained in Chapter 5 were connected together according to the alignment of the models shown in Figure 4.1 or 4.2 depending on the presence of an accumulator in the system. After connecting the models together, the refrigerant-side inputs of the component models were provided by the outputs of other component models upstream to them. Other inputs to the component models, including the heat exchanger inlet conditions on the air side, the total amount of refrigerant mass inside the components and the fault levels, were specified by the user. This is illustrated by the input-output diagram of the simulation Figure 6.1.

Since system XI has an EEV, its system model requires a user-specified control setpoint as an input the system model. As system XI was controlled manually to

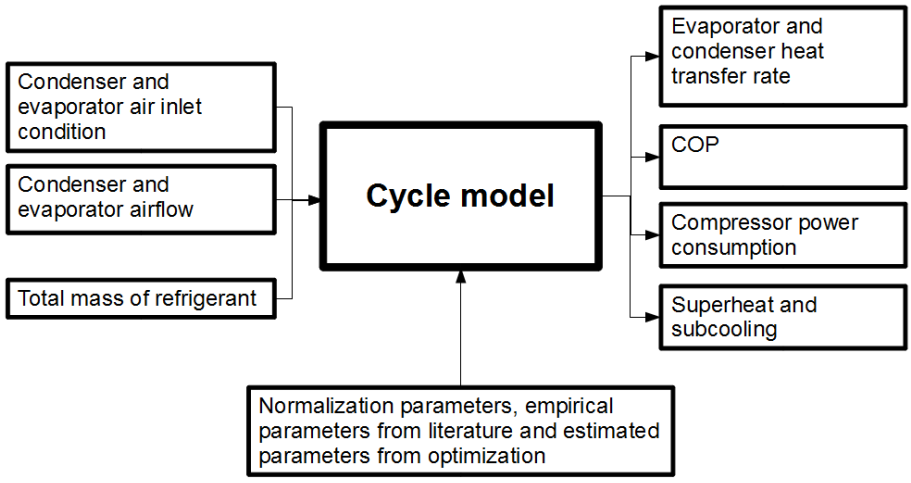

Figure 6.1. Input-output diagram of the final simulation model.

reach an 8.7K compressor suction superheat, the model of system XI is configured to require a compressor suction superheat setpoint as an input.

Before charge tuning, the mass of charge in the system estimated by the model is inaccurate and another input from experimental observations is required to substitute the total mass of refrigerant in Figure 6.1 for charge tuning. In the two-point charge tuning method (Shen, 2006), the experimental observation chosen is condenser outlet subcooling. The disadvantage to use subcooling is that data without subcooling cannot be used and their behavior may not be covered by the charge tuning equation. An experimental observation which is valid in all data points should be used so that the charge tuning equation can be trained with all experimental results. After trial-and-error with different refrigerant-side measurement such as condenser inlet pressure and compressor discharge superheat, the density at the compressor discharge, which is valid for all data points, is chosen and the pre-tuning simulation can converge under

most experimental scenarios for both FXO and TXV systems. The input and output diagram of the pre-tuning simulation model is shown in Figure 6.2.

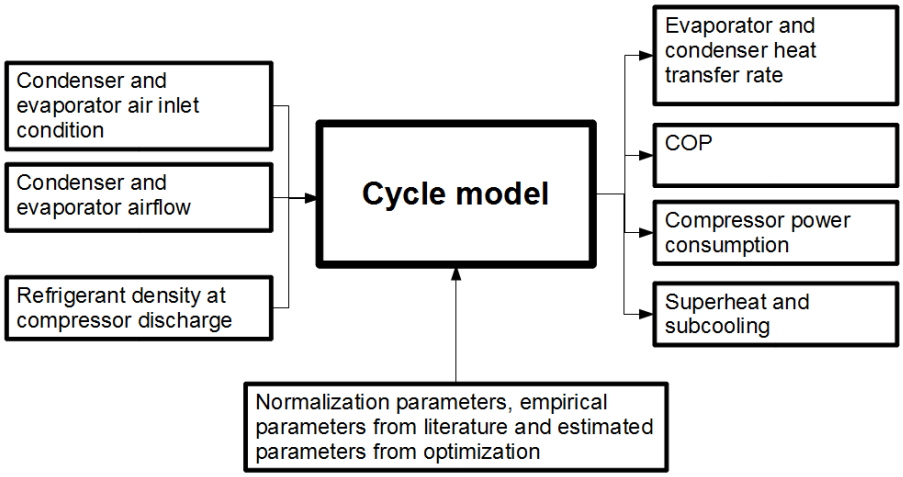

Figure 6.2. Input-output diagram of the pre-tuning simulation model.

The pre-tuning simulation of system XI requires compressor suction superheat as an input in addition to the inputs listed in Figure 6.2. When the experimental cases are simulated, the input is the measured compressor suction superheat.

6.1.2 Independent Variables

To obtain solutions of an implicit mathematical problem, the problem is usually formulated as a residual function

$$f(\vec{x}_{ind}) = \vec{r} \tag{6.1}$$

with an independent variable vector $\vec{x}_{ind}$. At the solution of the problem, the magnitude of the residual vector is zero. This solution can be obtained by employing numerical methods to reduce the magnitude of the residual vector $\vec{r}$ in Eqn. (6.1) to a threshold equivalent to zero as shown in

$$||\vec{r}|| < \epsilon. \tag{6.2}$$

In the system model, 5 independent variables were identified. The independent variables, which are the same in the pre-tuning simulation and final simulation, are listed in Table 6.1.

Table 6.1. List of independent variables of the cycle solver.

Pressure at suction line inlet
Pressure at compressor discharge
Enthalpy at suction line inlet
Temperature at compressor discharge
Refrigerant mass flow rate

Pressure and enthalpy at the suction line inlet are independent variables because both the evaporator and suction line models need the variables as inputs. The evaporator model takes the outlet pressure to solve its heat and mass transfer rate model and the suction line model uses the inlet condition to solve the pressure drop and heat loss model. To avoid additional computational effort to solve any sub-models implicitly, the thermodynamic state of the suction line inlet should be

available before the solution of any component models and is included in the list of the independent variables.

Refrigerant mass flow rate is also listed as one of the independent variables in Table 6.1. Although the suction line model requires a refrigerant mass flow rate as an input, neither pressure or enthalpy at the suction line inlet can be used to solve the expansion valve model or the compressor model for a refrigerant mass flow rate. Hence a refrigerant mass flow rate is provided in Table 6.1 for the solution of the suction line model.

After solving the suction line model, a compressor discharge pressure and temperature are required to solve the compressor mass flow rate model, and both of them are listed as independent variables in Table 6.1. Since all refrigerant-side inputs of other component models can be estimated after solving the suction line and compressor model as illustrated in Section 6.1.3, no other independent variables are needed.

To apply constraints on the independent variables consistently in the simulation of different systems, the independent variables in Table 6.1 are normalized as

$$x_{ind,1} = \frac{T_{a,evap,in} - T_{r,l}(P_{r,suctionline,in})}{T_{a,evap,in} - 250[\mathrm{K}]}, \tag{6.3}$$

$$x_{ind,2} = \frac{T_{r,v}(P_{r,comp,out}) - T_{a,cond,in}}{T_{r,cric} - T_{a,cond,in}}, \tag{6.4}$$

$$x_{ind,3} = \frac{h_{r,suctionline,in} - h_{r,sat}(P_{r,suctionline,in})}{c_{pr,v}(P = P_{r,suctionline,in})(10[\mathrm{K}])}, \tag{6.5}$$

$$x_{ind,4} = \frac{T_{r,comp,out,guess} - T_{a,cond,in}}{T_{r,cric} - T_{a,cond,in}}, \tag{6.6}$$

and

$$x_{ind,5} = \frac{\dot{m}_{r,guess}}{\dot{m}_{r,cond,rated}}. \tag{6.7}$$

In the pre-tuning simulation, experimental measurements are used as the initial guess of the independent variables of the solver as

$$x_{ind,1,guess} = \frac{T_{a,evap,in} - T_{r,suctionline,in,l,mea}}{T_{a,evap,in} - 250[\mathrm{K}]}, \tag{6.8}$$

$$x_{ind,2,guess} = \frac{T_{r,comp,out,v,mea} - T_{a,cond,in}}{T_{r,cric} - T_{a,cond,in}}, \tag{6.9}$$

$$x_{ind,3,guess} = \frac{T_{r,suctionline,in,mea} - T_{r,suctionline,in,v}}{10[\mathrm{K}]}, \tag{6.10}$$

$$x_{ind,4,guess} = \frac{T_{r,comp,out,mea} - T_{a,cond,in}}{T_{r,cric} - T_{a,cond,in}}, \tag{6.11}$$

and

$$x_{ind,5,guess} = \frac{\dot{m}_{r,mea}}{\dot{m}_{r,cond,rated}}. \tag{6.12}$$

In the final simulation, no measurements are available for the initial guesses of the solver. The initial guesses are obtained from regression equation

$$\begin{Bmatrix} P_{r,suctionline,in,l,guess} \\ P_{r,comp,out,v,guess} \\ (T_{r,suctionline,in,mea} - T_{r,suctionline,in,v})_{guess} \\ T_{r,comp,out,guess} \\ \dot{m}_{r,guess} \end{Bmatrix} = \mathbf{C}_{5x7} \begin{Bmatrix} 1 \\ T_{a,evap,in} \\ T_{a,evap,in,sat} \\ \dot{V}_{a,evap,in} \\ T_{a,cond,in} \\ \dot{V}_{a,cond,in} \\ M_{mea} \end{Bmatrix} \tag{6.13}$$

and the coefficients in Eqn. (6.13) are estimated by linear regression with experimental data as listed in the Appendix G.

The outputs of Eqn. (6.13) are converted to guess values of the independent variables listed from Eqns. (6.8) to (6.12). The saturation pressures in Eqn. (6.13) are converted to saturation temperature in Eqns. (6.8) and (6.9). The rest of the outputs from Eqn. (6.13) can be used directly in Eqns. (6.11) and (6.12).

6.1.3 Calculation of Residuals

To calculate the residual vector in Eqn. (6.1), the component models where inputs are available from the independent variables are solved first to generate inputs to the other component models. Eventually some component models will generate outputs which are inconsistent with the independent variables or user-defined inputs. The

inconsistencies form the residual of the system model. The calculation procedure to solve all component models is shown in Figure 6.3.

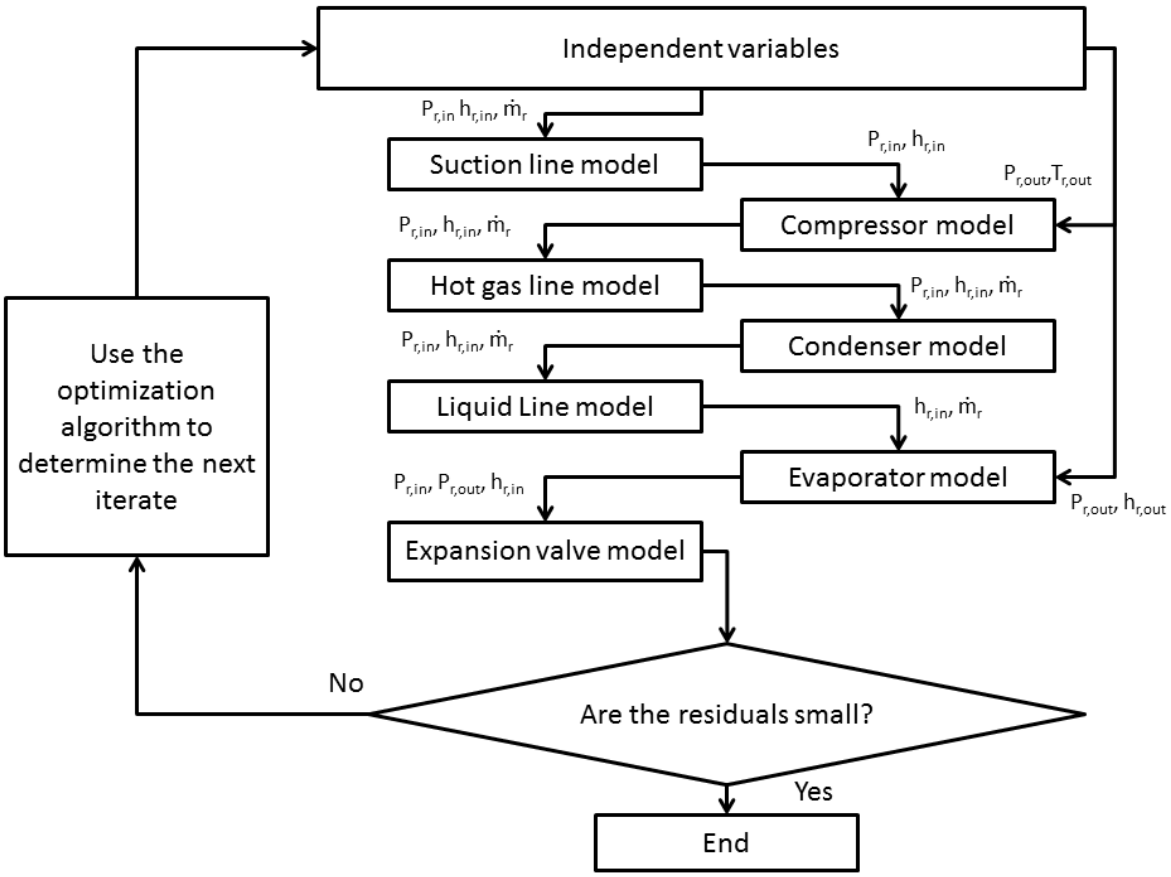

Figure 6.3. Flowchart to solve all component models from independent variables.

As discussed in the previous section, the solution process in Figure 6.3 starts with the suction line because all the inputs to the suction line are available from the user inputs and the independent variables. The outlet state of the suction line serves as the inlet state of the compressor and the compressor models can be solved with the temperature and pressure at the compressor discharge from the independent

variables. The component models between the compressor model and the expansion valve model are solved in the order of the refrigerant flow direction. Assuming an adiabatic expansion process across the valve, the enthalpy at the evaporator inlet is the same as the enthalpy at the liquid line outlet. Since the evaporator model uses the outlet pressure as an input, the evaporator model can be solved to give the pressure at the inlet of the evaporator. The inlet pressure of the evaporator is available as the outlet pressure of the valve and the expansion valve model can be solved. After solving all component models, the residual vector in Eqn. (6.1) can be calculated. If the norm of the residual is not smaller than a threshold as shown by Eqn. (6.2), an optimization algorithm is used to iterate the independent variables until the residual becomes smaller than the threshold. This gives the solution of the system model.

Since the variables in the residual vector such as refrigerant mass flow rate and refrigerant enthalpy are in different engineering units and scales, in order to compare the entries in the residual vector fairly and terminate the numerical method appropriately according to the conditions in Eqn. (6.2), the entries in the residual are normalized. The five dimensionless residuals are tabulated as Table 6.2.

The first normalized residual in Table 6.2 is the dimensionless enthalpy difference between the evaporator outlet and the suction line inlet. The enthalpy at the suction line inlet is obtained from the independent variables while the enthalpy at the evaporator outlet is given by the evaporator model. They are not guaranteed to be equal in advance and their difference forms one of the residuals.

Table 6.2. List of residuals of the cycle solver.

Residuals of pre-tuning simulation	Residuals of final simulation
$\frac{h_{r,evap,out}-h_{r,suctionline,in}}{c_{pr}(T=T_{a,evap,in},x=0)10[\mathrm{K}]}$	
$\frac{\dot{m}_{r,comp}-\dot{m}_{r,EXV}}{\dot{m}_{r,cond,rated}}$	
$\frac{T_{r,comp,out}-T_{r,comp,out,guess}}{10[\mathrm{K}]}$	
$\frac{\dot{m}_{r,guess}-\dot{m}_{r,comp}}{\dot{m}_{r,cond,rated}}$	
$\frac{\rho_{r,comp,out}-\rho_{r,comp,out,mea}}{\rho_{r,comp,out,mea}}$	$\frac{M_{\mathrm{specified}}-(M_{sim}+\Delta M)}{M_{\mathrm{specified}}}$

The second and forth residual in Table 6.2 involve the refrigerant mass flow rate. The refrigerant mass flow rates from the independent variables, from the compressor model and from the expansion valve model during the iteration process in Figure 6.3 may not be equal. Their differences form two residuals. However, if an electronic expansion valve model is used and its opening area is not maximized, the refrigerant mass flow rate from the electronic expansion valve model will be equal to the mass flow rate from the compressor model. In this case, if the opening of the electronic expansion valve is controlled to reach a user-specified compressor suction superheat, the second residual in Table 6.2 will be replaced by

$$\frac{SH_{comp,in}-SH_{\mathrm{specified}}}{10[\mathrm{K}]} \tag{6.14}$$

so that the compressor suction superheat at solution equals to that of the user-specified setpoint.

The temperature at the compressor discharge from the independent variables may be different from the solution of the compressor heat loss model, and their difference forms the third residual in Table 6.2.

The fifth residual in Table 6.2 differs between the pre-tuning simulation and the final simulation. In the pre-tuning simulation, the system model finds a solution that its compressor discharge density equals to that of the experimental observations. Since the simulated compressor discharge density may be different from that of the experimental ones during the iterations, the difference between the simulated and the observed densities during the iterations forms a residual. In the final simulation, the difference between the user-specified charge amount and the simulated charge amount after tuning forms the last residual because the simulation does not depend on any experimental data.

6.1.4 Minimization of Residual

In Figure 6.3, it is shown that an optimization algorithm is used to minimize the residual. An optimization algorithm is used because the cycle solver can be considered as a multi-variable optimization problem and can be solved by minimizing the objective function

$$J = \vec{\mathbf{r}}^T \vec{\mathbf{r}} \tag{6.15}$$

When the objective function Eqn. (6.15) is below a threshold ϵ close to zero, the objective function and all the residuals in $\vec{r}$ can be said to be equivalent to zero. To reduce the amount of computational effort needed, the solution process is executed first by the trust-region dogleg method, which is a type of quasi-Newton method (Nocedal and Wright (2006)). However, during the iteration process of the independent variables, the independent variables may iterate with unrealistic values and the objective function Eqn. (6.15) may output undefined results that terminate the iteration. To circumvent the problem, if an error occurs in the trust-region dogleg method, the minimization will be conducted by the trust region reflective method (Nocedal and Wright (2006)), which is a constrained minimization algorithm, to restrict the independent variables within some limits and avoid the error. The constraints of various independent variables are tabulated in Table 6.3.

Table 6.3. List of constraints on the dimensionless variables.

Independent variables	Upper limit	Lower limit
$x_{ind,1}$	2	0.03
$x_{ind,2}$	0.95	0.03
$x_{ind,3}$	6.67	N/A
$x_{ind,4}$	N/A	0
$x_{ind,5}$	N/A	1.e-8

The independent variables $x_{ind,1}$ and $x_{ind,2}$ in Table 6.3 refer to the saturation temperatures of the high-pressure and low-pressure regimes. The constraints are established to avoid solving heat exchangers with unrealistic heat transfer direction between refrigerant and air. They also prevent the solution of heat exchanger models

with saturation temperatures above the critical temperature or below 250K which is impossible for the application. $x_{ind,3}$ in Table 6.3 is restricted to prevent unrealistic cases with evaporator outlet superheat above 40K. $x_{ind,4}$ and $x_{ind,5}$ are constrained to avoid solving with compressor discharge temperature below the air inlet temperature of the condenser and negative refrigerant mass flow rate.

6.1.5 Additional Steps to Model Compressor Flow Fault

To model the compressor flow fault in the system model, the refrigerant enthalpy at the compressor suction and the refrigerant mass flow rate observed by component models other than the compressor model are changed. During each iteration in Figure 6.3, the method to solve a bypass factor bp at a user-specified fault level VL in Section 4.3.3 is implemented. The refrigerant mass flow rate entering the condenser can be calculated by Eqn. (4.46), and this is the refrigerant mass flow rate used by component models other than the compressor model in their inputs and the calculation of the residuals in Table 6.2.

6.1.6 Modification of the System Model to Simulate Presence of Non-condensables in the System

To simulate system operation with non-condensable, the partial pressure of the non-condensable in the hot gas line and the condenser is needed. However, since the calculation of the partial pressure needs the area ratios from the condenser model as

an input to Eqn. (4.48) and the area ratios cannot be solved without using the partial pressure as an input to the condenser heat transfer model, an additional independent variable and residual is added to the implicit function of the system model for cases with a non-condensable fault. The additional independent variable $x_{ind,6}$ is $w_{r,nc}$ in Eqn. (4.51) and the residual is the difference between $x_{ind,6}$ and $w_{r,nc}$ calculated by Eqn. (4.51).

In the beginning of the solution process, an initial guess of 0.2 is assigned to $x_{ind,6}$. After solving the compressor model in Figure 6.3, $x_{ind,6}$ is used to solve Eqn. (4.52) for P_{nc}. The compressor discharge temperature is updated with the refrigerant partial pressure at the compressor discharge and the compressor discharge enthalpy. P_{nc} is also available for the solution of the hot gas line model and the condenser model. The condenser model also provides another $w_{r,nc}$, and a new residual $x_{ind,6} - w_{r,nc}$ can be calculated. During constrained optimization, $x_{ind,6}$ is limited between 10^{-8} and 1 to avoid division by zero and to preserve its physical meaning.

6.1.7 Steps to Include the Accumulator Model in the System model

To append the accumulator model discussed in Section 4.2.7 to the system model, the procedures to conduct the simulation are modified so that the simulation of systems with accumulator follow the assumption in Section 4.2.7.

In the pre-tuning simulation, the system simulation is first implemented in the same way as other systems without the accumulator model. Upon convergence, the compressor suction superheat of the solution will be examined. If it equals to zero,

the compressor suction may have refrigerant in two-phase, and the simulation result will violate the assumption of the accumulator model in Section 4.2.7. The simulation will be repeated with the accumulator model by replacing the fifth residual in Table 6.2 with

$$\frac{h_{r,comp,in} - h_{r,v}(P_{r,comp,in})}{c_{pr}(T = T_{a,evap,in}, x = 1)10[\mathrm{K}]}. \tag{6.16}$$

After solving for the solution with the new residual, the refrigerant at its outlet is saturated vapor and the accumulator model is acting as if the accumulator is storing refrigerant liquid and expels saturated vapor according to the assumptions in Section 4.2.7.

In the final simulation, the steps of simulating systems with accumulators differ between FXO systems and TXV systems. For FXO systems with accumulators, it is more likely for the accumulator to hold liquid refrigerant and its system simulation is first executed by assuming that the accumulator holds liquid refrigerant and replacing the fifth residual in Table 6.2 with Eqn. (6.16). The amount of refrigerant outside the accumulator is estimated and the solution is accepted if it is less than the user-specified charge level. This method is applicable because the accumulator contains liquid, and the amount of refrigerant inside the accumulator is significant to the total charge level in the other parts of the system. Otherwise, it is impossible for the accumulator to hold any significant amount of refrigerant and the assumption that the accumulator holds liquid refrigerant is revoked. The simulation is then repeated

with the original residuals in Table 6.2 and the simulation output should have a positive compressor suction superheat.

For TXV systems with accumulators, it is very unlikely for the accumulator to hold liquid refrigerant because the superheat of the evaporator outlet or the compressor suction is maintained to be positive most of the time. The system simulation is first carried out in the same way as the simulation of systems without accumulators. After the simulation, if the compressor suction superheat is zero with possible two-phase flow at the compressor suction, the assumption of the accumulator model to have pure vapor at its outlet is violated and the simulation is repeated by replacing the fifth residual in Table 6.2 with Eqn. (6.16). Otherwise the solution is accepted and the accumulator is simulated without any liquid refrigerant in it.

When liquid line restriction is simulated on systems with accumulators, the non-faulted case is first simulated with the algorithm to determine whether the accumulator contains liquid refrigerant. After that, the liquid line restriction pressure drop is calculated with the imposed fault level and the pressure drop from the condenser outlet and the evaporator inlet by Eqn. (2.20). The restriction pressure drop is imposed to the system model to simulate the performance under liquid line restriction and the algorithm to determine if the accumulator contains liquid refrigerant is executed again to examine if the accumulator holds any liquid refrigerant in the liquid line restriction case.

The discussion also shows that it is unnecessary to calculate the amount of charge inside the accumulator. When the system model is solved assuming a dry

accumulator, the amount of charge inside the accumulator is negligible to the system charge level and is assumed to be zero. When the system model is solved assuming the accumulator to contain liquid refrigerant, only the amount of charge outside the accumulator is needed to determine if the assumption is true. Therefore the amount of charge inside the accumulator is not needed to solve the model and there is no need to have a parameter related to the inner volume of the accumulator.

6.2 Charge Tuning

Estimation of charge inventory by models based on density at thermodynamic equilibrium, refrigerant volume inside the system and void fraction models may have a bias relative to the real charge inventory because the model misses the following features (Shen, 2006).

1. Unaccounted volume inside the refrigerant circuit:

   A vapor compression system usually consists of many sections of tubing and return bends at different locations such as condenser and liquid line. These components contain a significant amount of refrigerant inventory in the system but their volume is often not known accurately. A small deviation in the measurement of their volumes may lead to a large deviation in the estimation of the charge inventory inside the system.

2. Deviation in void fraction estimation:

Although void fraction models for different flow patterns (Hughmark, 1962; Baroczy, 1965) are available, a variety of conditions may still cause deviation in void fraction estimation. For example, the flow quality of refrigerant flow in the heat exchanger model may be different from the thermodynamic quality calculated based on energy balance calculation. The flow pattern of refrigerant flow inside the heat exchanger model may be different from the real situation due to maldistribution along multiple heat exchanger circuits. This causes deviation in the estimation of void fraction and hence charge inventory.

3. Lubricating oil mixed with refrigerant flow:

   The refrigerant inside the system may be circulated together with the lubricating oil of the compressor. However, there is no model to estimate the amount of oil flowing with the refrigerant in most simulations because the volume of compressor sump and solubility of oil are not always measured. The inability of the model to estimate the amount of refrigerant dissolved in the oil causes a deviation in the estimation of charge inventory.

One example of the deviation is shown in Figure 6.4, obtained from the pre-tuning simulation output of system III.

Figure 6.4 shows that the pre-tuning simulation of system III underestimates the nominal charge level 4kg by 2kg, revealing the importance of removing the estimation bias in charge inventory by methods such as charge tuning.

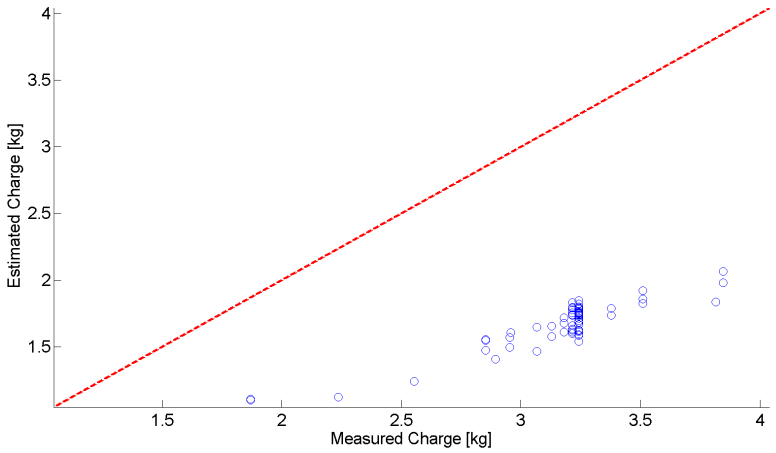

Figure 6.4. Parity plot of measured charge and estimated charge in pre-tuning simulation of system III before charge tuning.

To tune the charge level, a two-point charge tuning method was developed (Shen, 2006). To understand the origin of the bias,

$$\begin{aligned}\Delta M_{cond} = \quad & (\rho_{cond,tp,act} A_{cond,act} L_{cond,act} - \rho_{cond,tp,sim} A_{cond,mea} L_{cond,mea}) \\ & +(\rho_{cond,sc,act} A_{cond,act} \frac{U_{cond,sc,pre}}{U_{cond,sc,act}} - \rho_{cond,tp,,act} A_{cond,act} \frac{U_{cond,sc,pre}}{U_{cond,sc,act}} \\ & +\rho_{cond,tp,sim} A_{cond,mea} - \rho_{cond,sc} A_{cond,mea}) L_{cond,sc,sim}\end{aligned} \tag{6.17}$$

and

$$\begin{aligned}\Delta M_{evap} = \quad & (\rho_{evap,tp,act} A_{evap,act} L_{evap,act} - \rho_{evap,tp,sim} A_{evap,mea} L_{evap,mea}) \\ & +(\rho_{evap,sh,act} A_{evap,act} \frac{U_{evap,sh,pre}}{U_{evap,sh,act}} - \rho_{evap,tp,act} A_{evap,act} \frac{U_{evap,sh,pre}}{U_{evap,sh,act}} \\ & +\rho_{evap,tp,sim} A_{evap,mea} - \rho_{evap,sh} A_{evap,mea}) L_{evap,sh,sim}\end{aligned} \tag{6.18}$$

were derived. Eqns. (6.17) and (6.18) were added together, and the sum were simplified the sum by assuming some variables to be constant. The lengths of the

superheated sections are also assumed to be a function of the length of the condenser subcooled section. The resultant equation is

$$\Delta M = M_{\text{specified}} - M_{sim} = C_{M,0} + C_{M,1} L_{cond,sc,sim}. \tag{6.19}$$

To estimate the coefficients in Eqn. (6.19), pre-tuning simulations are carried out with two experimental conditions and are solved to give the same condenser outlet subcooling as the experiments. The deviation between the actual charge and the simulated charge for experimental condition 1 is considered as $C_{M,0}$ and $C_{M,1}$ in Eqn. (6.19) is given by

$$C_{M,1} = \frac{C_{M,0} - (M_{mea} - M_{sim})_{\text{Experimental condition 2}}}{L_{cond,sc,sim,\text{Experimental condition 1}} - L_{cond,sc,sim,\text{Experimental condition 2}}}. \tag{6.20}$$

The advantage of this method is its small computational effort because only two pre-tuning simulation cases are needed to form the charge tuning equation. However, the strategy cannot learn the parameters from cases with zero subcooling and may raise reliability issues when the simulation gives results with zero subcooling.

To account for cases with zero subcooling, a new tuning method is derived by first dismissing the assumption that the length of the superheated section is a function of the subcooled section. Since the heat transfer rate models estimate area ratios rather

than lengths of the sections, the lengths of the sections in Eqns (6.17) and (6.18) are replaced by the area ratios. The resultant equation is

$$\Delta M = M_{mea} - M_{sim} = C_{M,0} + C_{M,1} w_{cond,sc} + C_{M,2} w_{cond,sh} + C_{M,3} w_{evap,sh}. \quad (6.21)$$

By considering Eqn. (6.21) as an equation to adjust the simulated charge level from the measured charge level at a rated condition, Eqn. (6.21) is written as

$$\begin{aligned} \Delta M = \; & C_{M,0} + C_{M,1}(w_{cond,sc} - w_{cond,sc,rated}) + C_{M,2}(w_{cond,sh} - w_{cond,sh,rated}) \\ & + C_{M,3}(w_{evap,sh} - w_{evap,sh,rated}). \end{aligned} \quad (6.22)$$

By definition, $C_{M,0}$ in Eqn. (6.22) can be calculated as the difference between the measured charge level and the charge level in the pre-tuning simulation at a rated condition as

$$C_{M,0} = M_{mea,rated} - M_{sim,rated}. \quad (6.23)$$

The pre-tuning simulation at the rated condition also gives $w_{cond,sc,rated}$, $w_{cond,sh,rated}$ and $w_{evap,sh,rated}$ in Eqn. (6.22). The other regression coefficients in Eqn. (6.22) can be estimated by using the pre-tuning simulation results of all valid experimental cases other than the result at the rated condition to minimize an

objective function weighted by charge levels, evaporator fouling levels and condenser fouling levels as

$$J_M = \Sigma_i(weight_{M,i} + weight_{\dot{V}_{a,evap,i}} + weight_{\dot{V}_{a,cond,i}})(\frac{M_{sim,i} + \Delta M_{tuned,i} - M_{mea,i}}{M_{mea,i}})^2. \tag{6.24}$$

The objective equation Eqn. (6.21) is weighted because the airflows of both heat exchangers and the charge levels may affect the length of the phase sections.

To choose a valid experimental case as the rated condition for charge tuning, valid experimental cases with zero condenser outlet subcooling or zero evaporator outlet superheat were first eliminated. Then, each remaining case was specified as the rated condition to estimate $C_{M,0}$, $w_{cond,sc,rated}$, $w_{cond,sh,rated}$ and $w_{evap,sh,rated}$ in Eqn. (6.22) and the regression coefficients in Eqn. (6.22) were estimated by minimizing Eqn. (6.24) with all pre-tuning simulation result. The corresponding coefficients of determination for charge level estimation were also calculated by Eqn. (5.32). After calculating the coefficients of determinations for all cases, the case which gave the higher coefficient of determination was the rated condition, and the regression coefficients calculated from that case were the regression coefficients to tune the charge model of the system model.

Since the governing equations of the tuning method, Eqns. (6.17) and (6.18), do not involve the charge inside the accumulator and the amount of charge inside the accumulator is not estimated, the charge estimation model and the charge tuning method do not explain the charge inventory inside accumulators. Hence the pre-

tuning simulation results with liquid refrigerant in accumulators are excluded from the charge tuning process.

The result of charge tuning is tabulated in Table 6.4.

Table 6.4. Results of new charge tuning equation in Eqn. (6.22).

System	Coefficient of Determination	Average Absolute Deviation [%]	Absolute Maximum Deviation [%]	Average Deviation [%]
I	0.6384	1.58	6.06	1.58
II	0.9801	2.45	7.68	2.45
III	0.9083	2.32	7.61	2.32
IV	0.9365	2.26	9.68	2.26
V	0.9118	2.83	14.55	2.83
VI	0.6936	1.99	5.34	1.99
VII	0.9462	2.59	8.73	2.59
VIII	0.9511	2.60	9.99	2.60
IX	0.8311	5.10	21.38	5.10
X	0.8169	7.12	19.70	7.12
XI	0.8932	4.86	20.33	4.86

The result in Table 6.4 is compared to the ones conducted with the two-point charge tuning method as shown in Table 6.5.

All deviations in Table 6.4 are lower than that of Table 6.5 and all coefficients and all coefficients of determination in Table 6.4 are higher than that of Table 6.5. This shows that the method with Eqn. (6.22) can help to estimate charge in the system more accurately than the two-point charge tuning method.

The parameters of the charge tuning equations for different systems are listed in Appendix H.

Table 6.5. Results of old charge tuning equation in Eqn. (6.19).

System	Coefficient of Determination	Average Absolute Deviation [%]	Absolute Maximum Deviation [%]	Average Deviation [%]
I	0.1436	2.48	6.19	2.48
II	0.9748	2.61	9.46	2.61
III	0.7705	3.69	15.85	3.69
IV	0.1256	7.43	29.13	7.43
V	0.7809	4.44	28.42	4.44
VI	0.3734	2.75	11.65	2.75
VII	0.8222	4.27	16.44	4.27
VIII	0.5982	8.25	13.95	8.25
IX	0.6429	5.71	36.72	5.71
X	0.4468	13.33	29.98	13.33
XI	0.6627	12.52	35.39	12.52

6.3 Convergence and Computational Speed

Since the system model is designed to simulate the system operation quickly, its convergence rate and computational speed are evaluated. The number of data points converged in the final simulation for each system under experimental conditions is tabulated in Table 6.6.

System I, III, VI, VII, VIII and XI failed to achieve convergence for some valid data points in Table 6.6. Four data points diverged in system I, three data points diverged in system III and one data point diverged in each of the system V, VIII and XI.

All data points that diverged in system I were subjected to the highest fault level of the compressor flow fault in the experiment which is 28%. The diverged data

Table 6.6. Summary on the number of converged data points with old charge tuning equation.

System	Number of converged data points	Percentage of converged data points [%]
I	107	93.04
II	21	100.00
III	80	98.77
IV	41	100.00
V	69	100.00
VI	64	94.12
VII	103	99.04
VIII	47	100.00
IX	35	97.22
X	31	100.00
XI	39	97.50

points in system III and system VI were all overcharged with 30% more refrigerant than the standard charge level. The diverged data point in system VII was subjected to an evaporator fouling level at 55%. The diverged data point in system XI was overcharged with 36% more refrigerant at standard charge level. All these diverged cases are seriously faulted cases, and the simulation algorithm is not robust enough to solve these scenarios.

Since two algorithms were used to solve the system models, the number of function evaluations, instead of simulation time, is examined for the speed of calculation. The results are tabulated in Table 6.7.

The simulation of systems I, VI, IX and XI requires more function evaluations on average than other systems in Table 6.7. However, the high averages for systems I, VI and XI are a result of the diverged data points in these systems only because

Table 6.7. Summary on the number of function evaluations per data point with old charge tuning equation.

System	Average function evaluations per data point	Median function evaluations per data point	Standard deviation of function evaluations per data point
I	336.33	42.00	925.40
II	29.62	30.00	3.57
III	78.68	30.00	334.41
IV	29.71	30.00	8.49
V	29.57	30.00	4.97
VI	360.76	36.00	1270.44
VII	47.81	30.00	101.22
VIII	36.09	31.00	13.62
IX	134.25	84.00	286.18
X	47.55	36.00	21.49
XI	119.60	30.00	542.25

the medians of function evaluations of these systems in Table 6.7 are similar to the medians of the simulation of other systems. System IX is an FXO system with an accumulator, and its system model is operated twice for each case with no liquid refrigerant in the accumulator as discussed in Section 6.1.7. Since 43% of the simulation in Table 6.7 is simulated to have no liquid refrigerant in the accumulator, the simulation time of these cases is higher than the cases of the other systems. This results in higher average and median function evaluations for the simulation of system IX than that of other systems in Table 6.7.

6.4 Applicability Domain Study for the System Model

Although the applicability domains of the component models are defined in Section 5.3, their effects on the reliability of the system model results are not discussed. To study the effect, the relationship between the deviations of the simulation outputs with the experimental data and the distance of the simulation outputs to the applicability domain boundary as defined in Section 5.3 is examined. To investigate the change of the deviation between the measured and estimated COP with the distance of the simulation output to the applicability domain of the two-phase adjustment factor equation, Figure 6.5 is plotted.

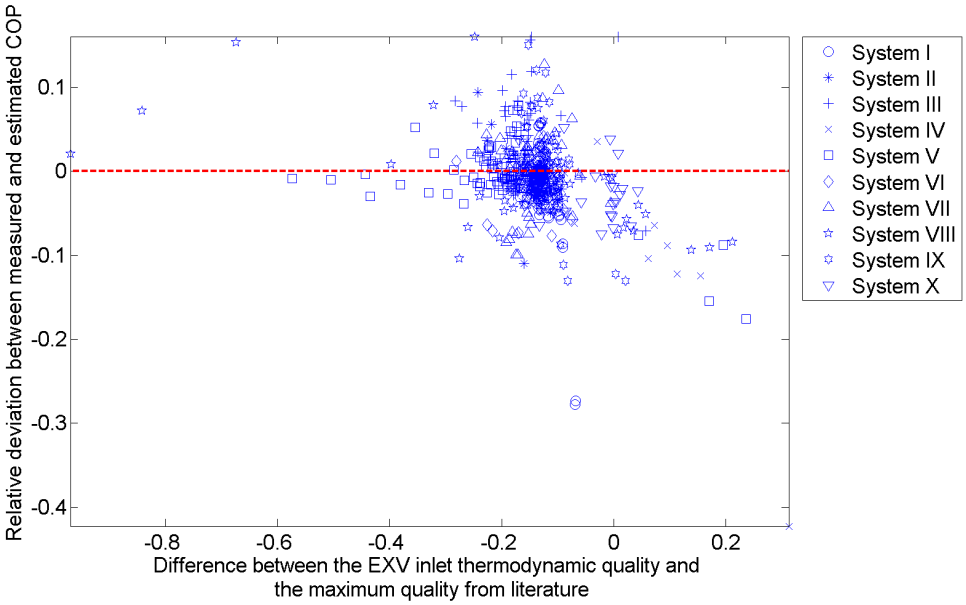

Figure 6.5. Residual plot of COP estimated by the system model with the distance of the simulation result to the applicability domain of the two-phase adjustment factor equation.

Figure 6.5 does not include system XI because the maximum opening of the expansion valve of system XI is infinitely large and the adjustment equation is never used in its system model. It can be seen that the majority of the COPs in Figure 6.5 are underestimated when the two-phase adjustment factor equation is operating outside its applicability domain. This shows that when a positive distance from the boundary of the applicability domain of the two-phase adjustment factor equation is calculated, the result becomes unreliable and should be discarded.

For FXO systems, the effect is further investigated by plotting the COP deviation with the distance of the simulation output to the applicability domain of the charge tuning equation in Figure 6.6.

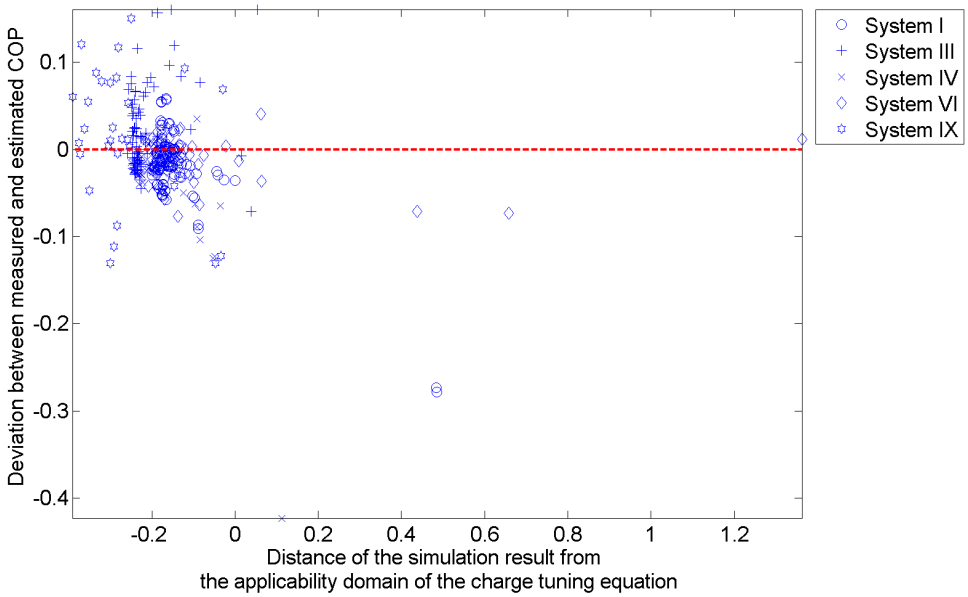

Figure 6.6. Residual plot of estimated COP with the distance of the simulation result to the applicability domain of the charge tuning equation for FXO systems.

Figure 6.6 shows that when the distance is higher than 0.11, the COP is underestimated. However, this cannot be observed for the simulation outputs from TXV systems as shown in Figure 6.7.

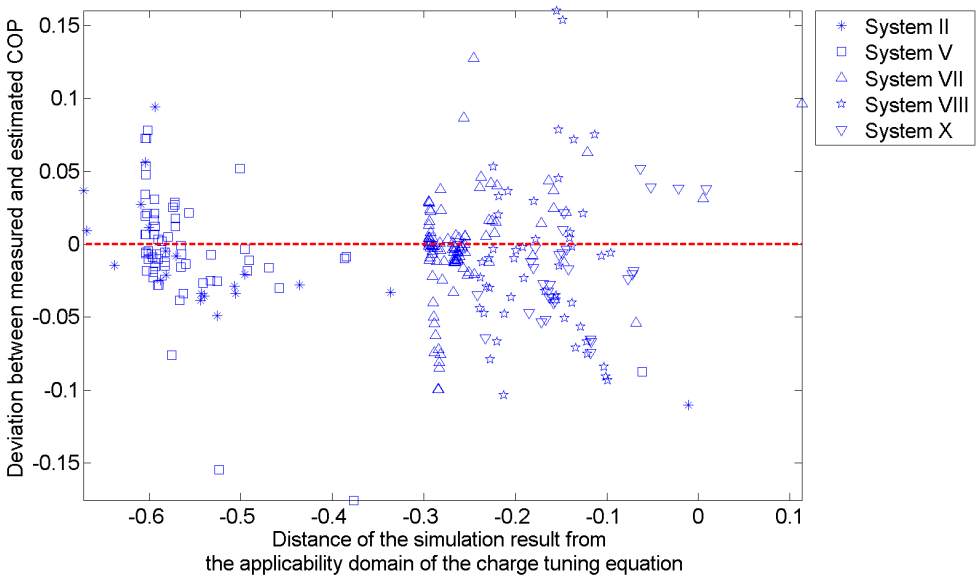

Figure 6.7. Residual plot of estimated COP with the distance of the simulation result to the applicability domain of the charge tuning equation for TXV systems.

Similar plots as Figure 6.7 were obtained when the deviations were plotted against the distances to the applicability domains of the other models. This shows that the system model is robust to the reliability of the component models. Only simulation results showing a positive distance to the applicability domain of the two-phase adjustment factor equation for all systems or a distance to the applicability domain of the charge tuning equation greater than 0.11 for FXO systems are unreliable and should be eliminated.

The number of data points eliminated for each system after checking their operation with the applicability domains is tabulated in Table 6.8.

Table 6.8. Number of data points removed from simulation results at experimental conditions due to operation outside applicability domains.

System	Number of valid experimental data points	Number of data points removed	Percentage of data points removed [%]
I	115	2	1.74
II	21	0	0.00
III	81	2	2.47
IV	41	6	14.63
V	69	4	5.80
VI	68	3	4.41
VII	104	0	0.00
VIII	47	9	19.15
IX	36	2	5.56
X	31	6	19.35
XI	40	0	0.00

Table 6.8 shows that the number of simulation result removed is independent of the number of valid experimental data points. Although the general thought is that larger number of valid experimental data points leads to larger applicability domains of the component models and hence smaller incidents to remove data points, since the number of data points removed is largely dependent on the applicability domain of the two-phase flow adjustment equation from the literature, the number of training data points available from the systems does not affect the number of unreliable simulation output.

The parameters to calculate the distance to the boundary of the applicability domain of the charge tuning equation are listed in Appendix I.

6.5 Summary

The chapter describes how the component and fault models were combined to form a system model and how charge tuning was conducted to eliminate the bias in charge estimation. The component models were combined according to the direction of refrigerant flow to form an implicit system model. Dimensionless independent variables and residuals were identified to solve the system model with quasi-Newton method or constrained optimization algorithm to maintain the speed and robustness of the solver. Refrigerant density at compressor discharge was chosen as the experimental input to pre-tuning simulations to generate simulation results for charge tuning. The final simulation model was created by modifying the pre-tuning simulation to include fault models and to have a different residual vector that allows specification of charge as an input. The convergence rate and the computational speed of the system model were also studied. The relationship between distance of the final simulation result to the boundary of the applicability domains of the component models with the accuracy of the system model was also studied to determine rules to remove unreliable simulation output. It was found that the charge tuning equation and the maximum thermodynamic quality in the training data of the expansion valve mass flow rate adjustment factor for two-phase flow were the models that are important to the reliability of the system model.

CHAPTER 7. SYSTEM MODEL VALIDATION

After constructing the system model and defining the applicability of the system model in Chapter 6, the model can be validated by comparing its simulation outputs with the experimental data. The validation was conducted in three ways as listed below.

1. Comparison of the overall system performance between the simulation outputs and the measurement:

   Evaporator heat transfer rates, compressor power consumptions and sensible heat ratios estimated by the system model were compared with the ones from the measurement to study the accuracy of the system model. This validated the accuracy of the system model to simulate the performance of the systems.

2. Comparison between the change of system performance with fault levels estimated by the simulation and the change observed in the experiments:

   In previous experiments (Breuker, 1997; Kim et al., 2009), system performance variables which change significantly with different fault levels were identified. These experimental scenarios were simulated by the system model, and the estimate change of the variables with fault levels was compared with the

experimental observations to validate the accuracy of the system model to estimate the change of system performance for various types of faults.

3. Comparison of the FDD evaluator results based on simulation outputs with the evaluation results based on experimental observations

   To validate the ability of the system model to replace experimental results as inputs to the FDD tool evaluator, it is necessary to prove that the FDD tool evaluator assesses the tools accurately with the simulation result. This is achieved by comparing the evaluation results based on valid experimental observations with the evaluation results based on the estimated performance in the system model under the same experimental conditions. The detailed evaluation is discussed in another dissertation (Yuill, 2014).

7.1 Comparing Overall System Performance

To validate the accuracy of the system model to estimate the overall system performance, the estimation of evaporator heat transfer rate, compressor power consumption and sensible heat ratio from the system model at the experimental conditions are compared with their experimental counterparts after discarding the unreliable cases according to Section 6.4.

7.1.1 Evaporator Heat Transfer Rate

Estimated evaporator heat transfer rate from the simulation outputs of all 11 systems is non-dimensionalized by their experimental averages before being plotted in Figure 7.1.

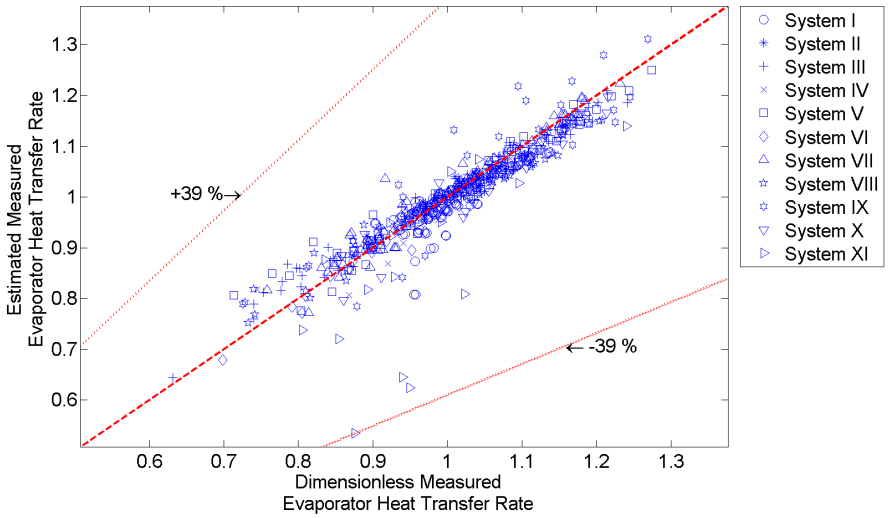

Figure 7.1. Parity plot of simulated and experimental dimensionless evaporator heat transfer rates of all 11 systems.

Figure 7.1 shows that the evaporator heat transfer rates were estimated within 39% of the experimental values which is much larger than the scattering in the parity plot for the evaporator heat transfer model as shown in Figure 5.10. Detailed analysis is conducted by tabulating the comparison result in Table 7.1.

Table 7.1 shows that the overall average abolute deviation for all systems is 2.59% only. However, multiple systems, including systems I, V, IX and XI, contain outliers with absolute maximum deviation above 12%, and these cases are studied separately.

Table 7.1. Comparison between final simulation and experimental results on evaporator heat transfer rate for the untuned model.

System	Coefficient of Determination	Average Absolute Deviation [%]	Absolute Maximum Deviation [%]	Average Deviation [%]
I	0.5728	2.40	15.69	-1.37
II	0.7888	1.41	3.67	0.19
III	0.9558	2.42	10.48	0.54
IV	0.8691	2.91	9.75	-1.86
V	0.9488	2.36	13.03	0.19
VI	0.9512	1.74	6.02	-1.12
VII	0.9700	1.45	7.69	0.58
VIII	0.9154	3.40	10.62	-1.61
IX	0.8256	5.34	12.28	2.02
X	0.8911	3.13	7.68	-2.12
XI	-0.3244	5.81	38.94	-4.57

The estimated and measured heat transfer rate of system I is compared in Figure 7.2.

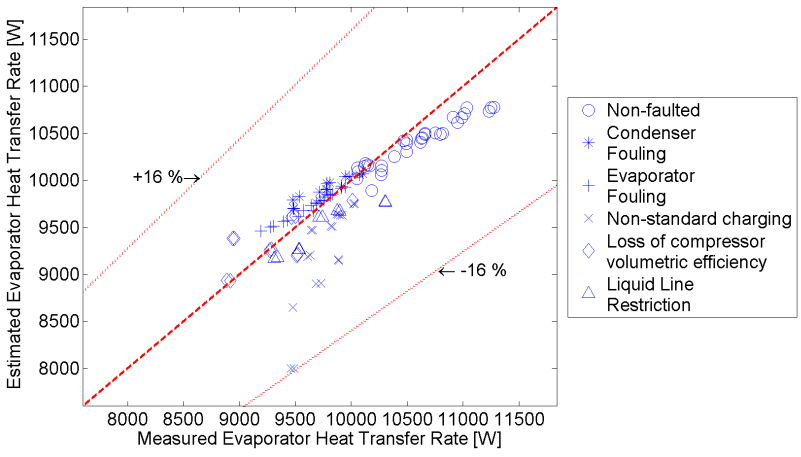

Figure 7.2. Parity plot of simulated and experimental evaporator heat transfer rates of system I.

Figure 7.2 shows that the large deviation is a result of the simulation of undercharged scenarios. The deviation is a result of bias in the charge estimation as shown in Figure 7.3

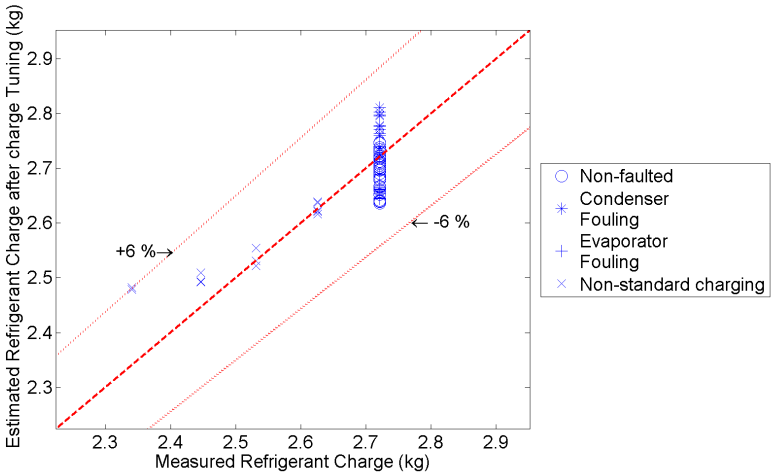

Figure 7.3. Parity plot between the estimated and measured charge level after charge tuning in the pre-tuning simulation of system I.

The two data points where charge levels are overestimated by 6% in Figure 7.3 are the same data points as the ones of which evaporator heat transfer rate is underestimated by 16%. Since evaporator heat transfer rate increases rapidly with charge levels in undercharged cases, the overestimation of charge level in Figure 7.3 implies that the evaporator heat transfer rate of the two data points in the final simulation will be lower than that of the pre-tuning simulation. This causes an underestimation of evaporator heat transfer rate of the two data points in Figure 7.2.

The estimated and measured evaporator heat transfer rate of system V is compared in Figure 7.4.

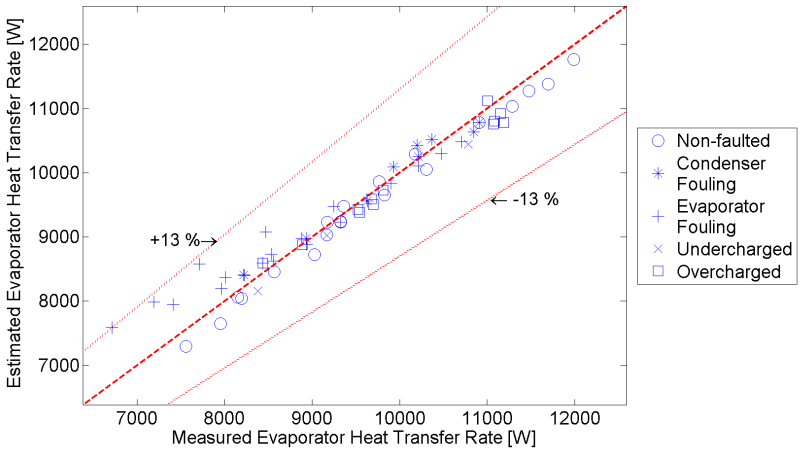

Figure 7.4. Parity plot of estimated and experimental evaporator heat transfer rates of system V.

Figure 7.4 shows that the maximum deviation occurs at three evaporator fouling cases with low evaporator heat transfer rate. To investigate why they are overestimated, the charge tuning result of system V is plotted in Figure 7.5.

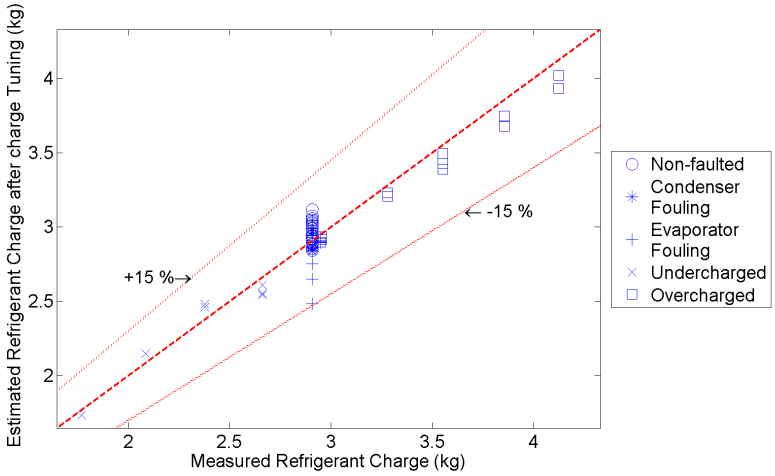

Figure 7.5. Parity plot between the estimated and measured charge level after charge tuning in the pre-tuning simulation of system V.

Figure 7.5 shows that there are three evaporator fouling cases with their charge levels underestimated after charge tuning. These data points are the data points where evaporator heat transfer rates are overestimated in Figure 7.4. Since the evaporator heat transfer rates of these data points are only overestimated by 7.4%, 6.3% and 4.2% respectively in its pre-tuning simulation, the overestimation of evaporator heat transfer rate in Figure 7.4 can be concluded as a consequence of the charge tuning inaccuracy of the system in these three cases.

The estimated and measured evaporator heat transfer rates of system VII are compared in Figure 7.6.

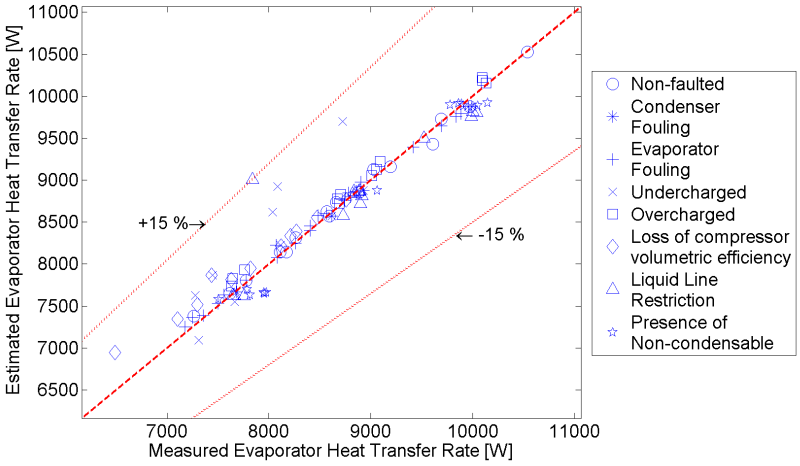

Figure 7.6. Parity plot of simulated and experimental evaporator heat transfer rates of system VII.

Several heat transfer rates in Figure 7.6 are overestimated. Further investigation is conducted by examining the charge level estimation in the pre-tuning simulation of system VII in Figure 7.7.

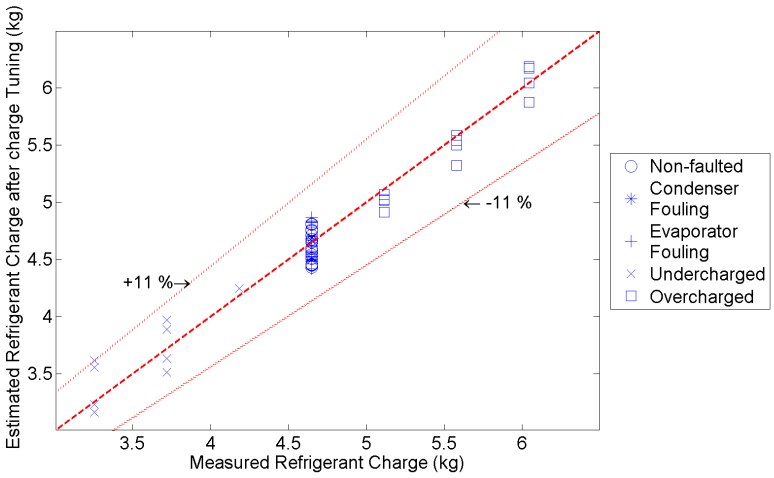

Figure 7.7. Parity plot of estimated and measured charge level of system VII in pre-tuning simulation.

The data points with overestimated charge level in the undercharged cases in Figure 7.7 are the data points with overestimated evaporator heat transfer rate in the undercharged cases in Figure 7.6. This shows that the deviation is caused by the deviation of the charge estimation in the system.

The estimated and measured evaporator heat transfer rates of system IX are plotted in Figure 7.8.

Figure 7.8 shows that there is a large scattering of estimated evaporator heat transfer rate compared to the measured values. The reason of the scattering lies in the accuracy of the evaporator heat transfer model of system IX as illustrated in Figre 7.9.

Although the scattering of evaporator heat transfer rate in Figure 7.9 is not as large as the one in Figure 7.8, it has the same pattern as the one in Figure 7.8 where the scattering appears at different values of evaporator heat transfer rate. This shows

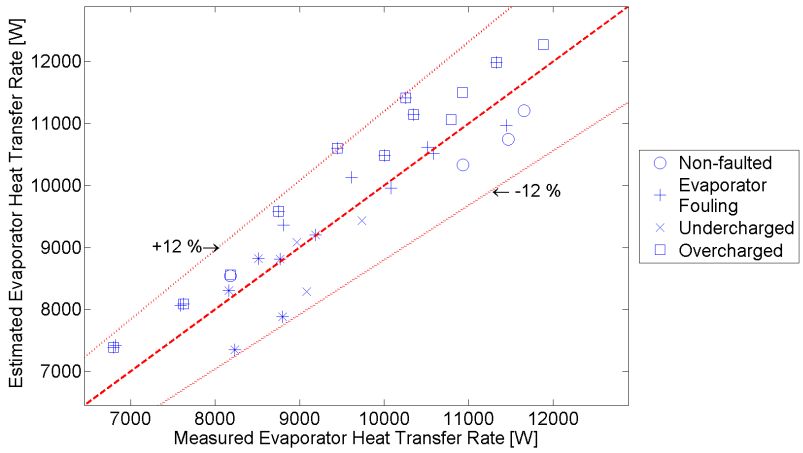

Figure 7.8. Parity plot of estimated and experimental evaporator heat transfer rates of system IX.

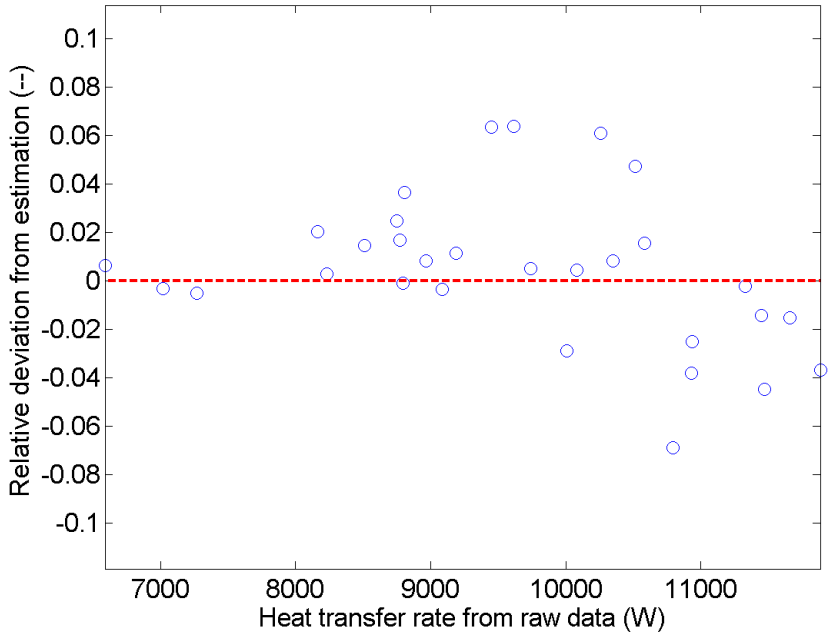

Figure 7.9. Parity plot of estimated and measured evaporator heat transfer rate after parameter estimation of the component model.

that the scattering of the estimation in the evaporator heat transfer model is the cause of the scattering of evaporator heat transfer rate in the final simulation.

The estimated and measured evaporator heat transfer rates of system XI are plotted in Figure 7.10.

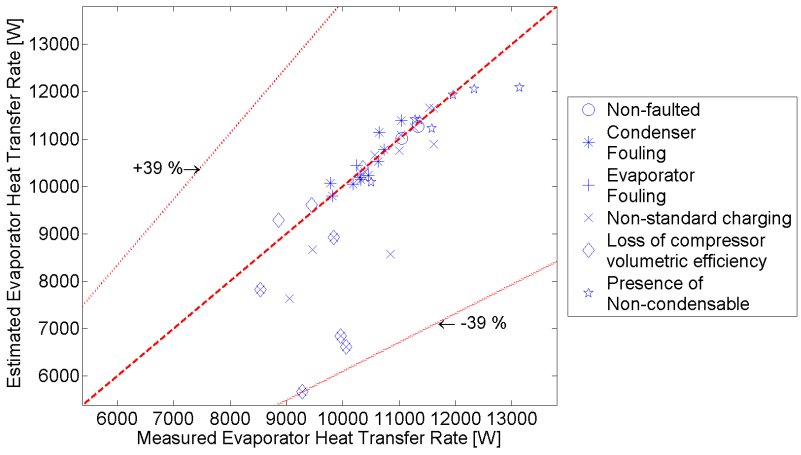

Figure 7.10. Parity plot of estimated and experimental evaporator heat transfer rates of system XI.

Figure 7.10 shows that the evaporator heat transfer rate is underestimated at low evaporator heat transfer rate values. This can be explained by plotting the change of the relative deviation between estimated and measured evaporator heat transfer rate with measured condenser outlet subcooling in Figure 7.11.

Figure 7.11 shows that the range of relative deviation increases significantly when the measured condenser outlet subcooling falls below 3K. For data points with measured condenser outlet subcooling above 3K, the evaporator heat transfer rate is calculated by the refrigerant-side measurement while the evaporator heat transfer rate is calculated by the air-side measurement adjusted according to Sections 3.4

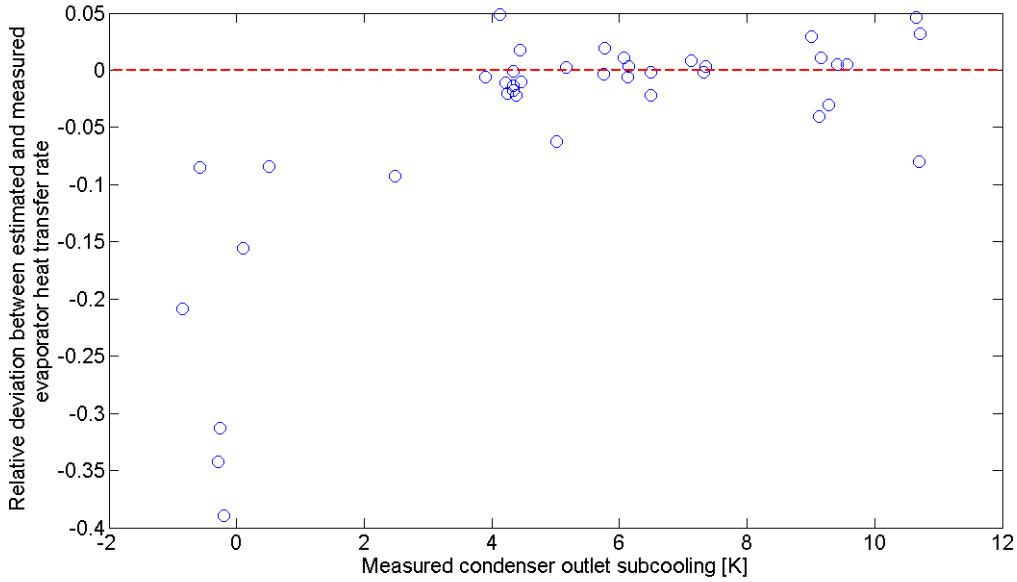

Figure 7.11. Residual plot of the relative deviation between estimated and experimental evaporator heat transfer rates with measured condenser outlet subcooling of system XI.

and 3.6.6 for data points with measured condenser outlet subcooling below 3K. The uncertainty of the measured evaporator heat transfer rates in cases with small condenser outlet subcooling is much higher than that of the cases with high condenser outlet subcooling. This causes a significant underestimation of evaporator heat transfer rate as the measured condenser subcooling falls below 3K.

7.1.2 Compressor Power Consumption

Figure 7.12 compares the experimental and simulated compressor power consumption of all simulated cooling systems.

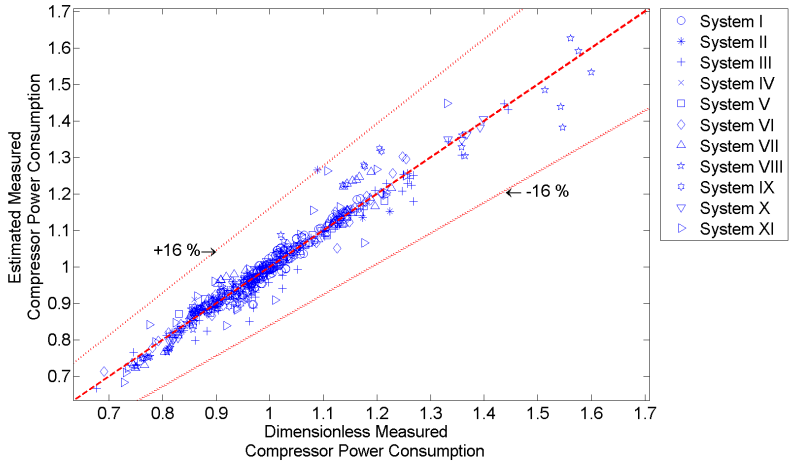

Figure 7.12. Parity plots of simulated and experimental dimensionless compressor power consumptions of all 11 systems.

Figure 7.12 shows that the compressor power consumption is estimated within 16% of the measured values and the overall average absolute deviation of all systems is 1.86%. Detailed analysis is conducted for each system, and the results are tabulated in Table 7.2.

Table 7.2 shows that systems II, III, VIII and XI have absolute maximum deviation higher than 10%. The deviations of system II, III, VIII and XI are examined in detail. The deviation in system II is studied using the parity plot in Figure 7.12 for system II only as shown in Figure 7.13.

There is only one outlier in Figure 7.13. It can be seen that the outlier is the same outlier observed in Section 5.2.3. Being the only anomaly in Section 5.2.3, the outlier is insignificant comparing to the deviation of the other data points.

The comparison of the estimated and measured compressor power consumption in system III is plotted in Figure 7.14.

Table 7.2. Comparison between final simulation and experimental results on compressor power consumption for the untuned model.

System	Coefficient of Determination	Average Absolute Deviation [%]	Absolute Maximum Deviation [%]	Average Deviation [%]
I	0.8246	1.50	7.43	-0.13
II	0.7825	2.68	16.27	1.54
III	0.9651	1.95	11.56	-1.59
IV	0.9719	0.95	6.02	-0.07
V	0.9785	0.91	4.86	-0.08
VI	0.9762	1.48	6.63	0.01
VII	0.9635	1.64	7.38	0.99
VIII	0.9752	3.08	10.65	-1.81
IX	0.7627	3.29	9.87	0.25
X	0.9956	1.22	2.85	-0.22
XI	0.8802	3.76	14.09	0.38

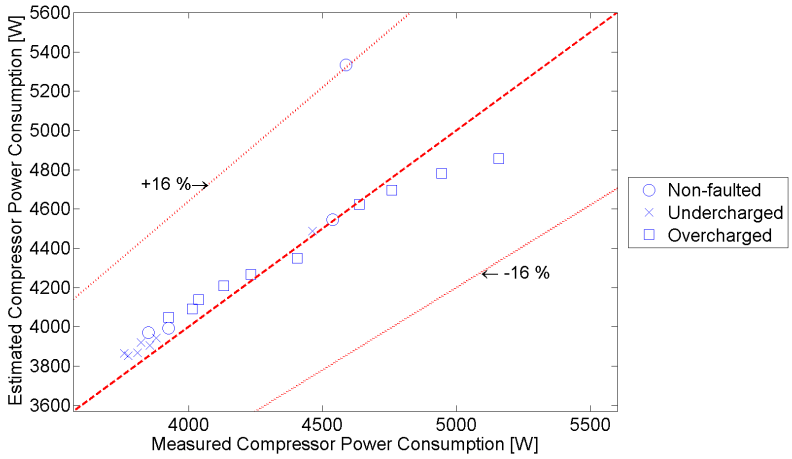

Figure 7.13. Parity plots of simulated and experimental dimensionless compressor power consumptions of system II.

To study the deviation, the estimated charge levels of the charge tuning equation in the pre-tuning simulations of system III are compared to the measured charge levels as Figure 7.15.

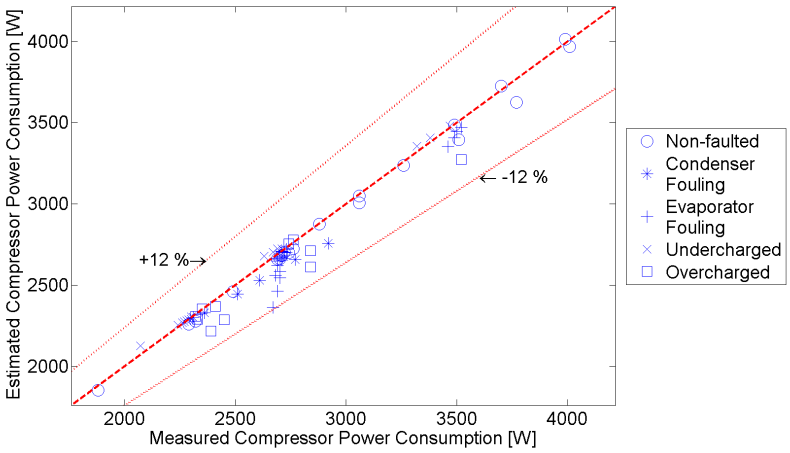

Figure 7.14. Parity plots of simulated and experimental dimensionless compressor power consumptions of system III.

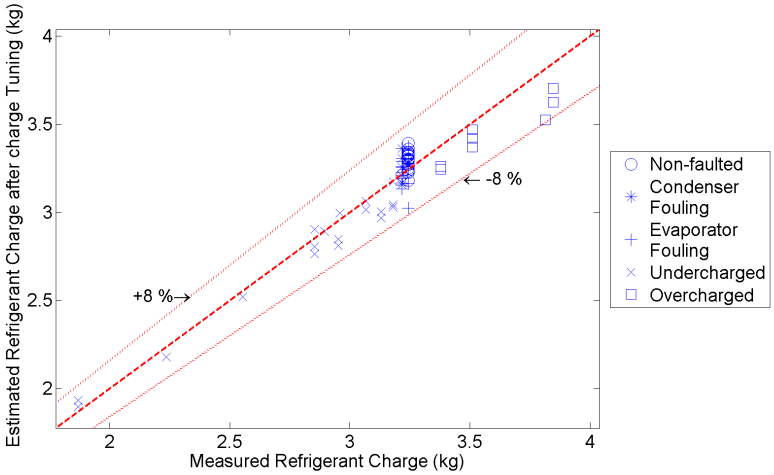

Figure 7.15. Parity plots of estimated charge levels after charge tuning and experimental charge levels in the pre-tuning simulation of system III.

The data points with the largest deviations in Figure 7.15 are the same data points as the ones with the largest deviation in Figure 7.14. This shows that the deviation is a consequence of inaccuracy in the charge tuning method.

The estimated and measured compressor power consumption of system XI is plotted in Figure 7.16.

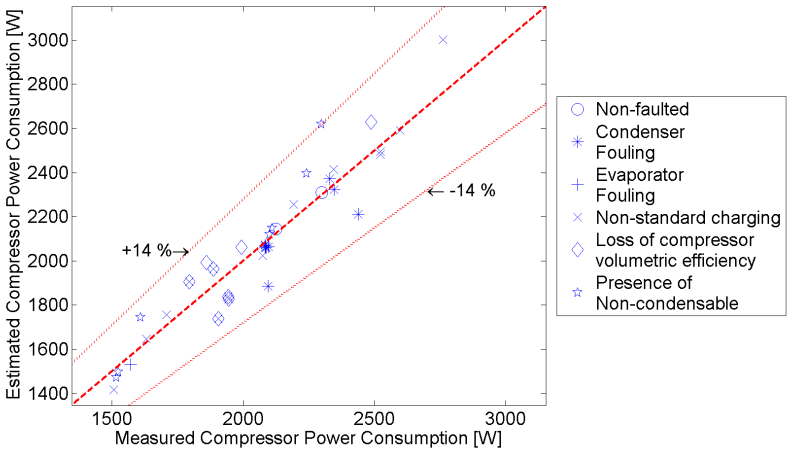

Figure 7.16. Parity plots of simulated and experimental dimenionless compressor power consumptions of system XI.

Figure 7.16 shows that the largest deviation occurs with the data point with non-condensable in the system. The data point has a non-condensable fault level at 98%, and a comparison of the estimated and experimental discharge pressure shows that the discharge pressure of the data point is overestimated by 325kPa. In the experiment, when the non-condensable fault level is higher than 60%, two-phase flow is found at the sightglass upstream of the expansion valve despite the subcooling at the same location to be higher than 3K. This shows that when the non-condensable fault level is high, the system may not trap the non-condensable between the compressor and the expansion valve, and the non-condensable circulates along the entire refrigerant circuit with the refrigerant. Since the simulation assumes that all non-condensbale is trapped in the condenser regardless of the fault

level and is different from the experimental observation at high fault level, the simulation overestimates the amount of non-condensable in the condenser and hence the partial pressure of the non-condensable. This causes an overesitmation of the compressor discharge pressure and thereafter the compressor power consumption.

7.1.3 Sensible Heat Ratio

The estimated and measured sensible heat ratio (SHR) of the systems are plotted in Figure 7.17.

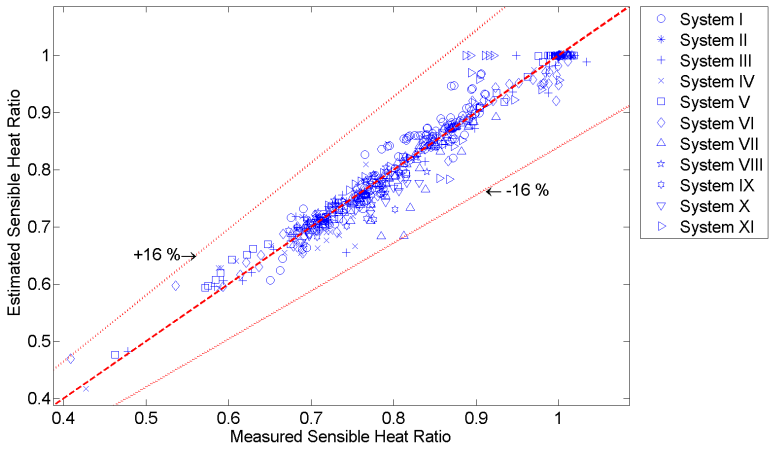

Figure 7.17. Parity plot of simulated and experimental sensible heat ratios of all cooling systems.

Figure 7.17 shows a few overestimation cases for SHR below 0.6 and some underestimated cases for SHR between 0.7 and 0.9. The scattering in Figure 7.17 is also more significant for cases with SHR above 0.9. The scattering is further studied by establishing a residual plot with the relative humidity at the evaporator air inlet in Figure 7.18.

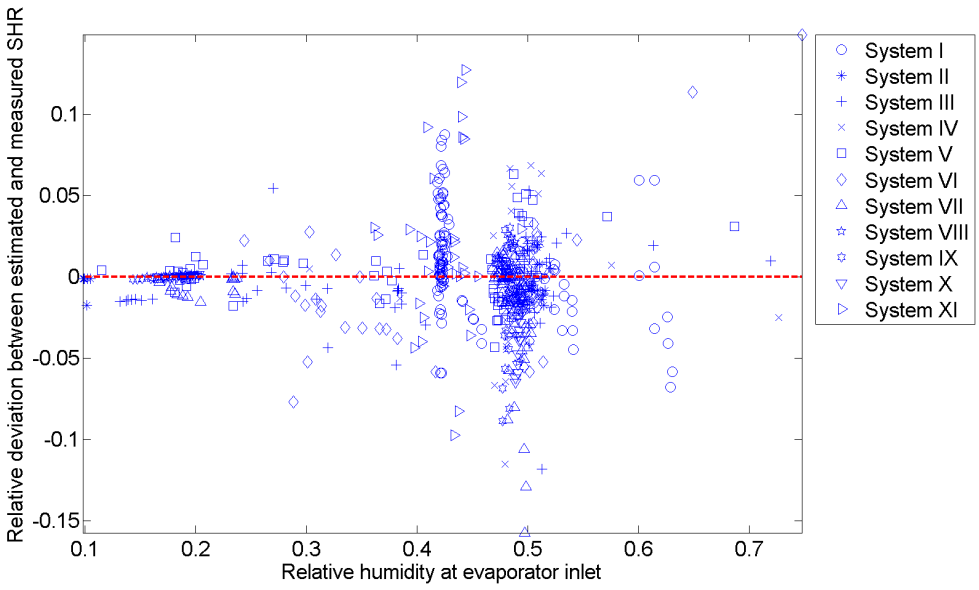

Figure 7.18. Residual plot of SHR with the relative humidity at the evaporator inlet for all cooling systems.

Figure 7.18 shows increasing scattering of deviations with higher relative humidity. Since the parameter estimation of evaporator model is weighted according to refrigerant-side data only in Eqn. (5.11), the behavior of the air side of the evaporators may not be learned appropriately. This leads to a cluster of accurately estimated cases around relative humidity 0.5 and larger scattering of deviations in cases with higher relative humidity at the inlet in Figure 7.18.

7.2 Comparing the Change of Variables of Significance with Fault Levels

In Section 2.4, the variables changing significantly with respect to various fault levels are tabulated in Table 2.1 for system I (FXO system) and Table 2.2 for system

VII (TXV system). To validate the ability of the simulation to model the change of system performance accurately, the change of the variables with respect to different fault levels are plotted and are compared with the experimental observations. The environmental conditions simulated in the study are tabulated as Tables 7.3 and 7.4.

Table 7.3. Conditions tested on system I with different fault levels.

Index	Air inlet dry-bulb temperature of evaporator [K]	Air inlet dewpoint of evaporator [K]	Air inlet temperature of condenser [K]
A	296	283	294
B	296	283	297
C	296	283	300
D	296	283	303
E	296	283	306

Table 7.4. Conditions tested on system VII with different fault levels.

Index	Air inlet dry-bulb temperature of evaporator [K]	Air inlet dewpoint of evaporator [K]	Air inlet temperature of condenser [K]
A	294	273	301
B	294	283	301
C	294	283	311
D	299	289	301
E	299	289	311
F	299	273	301

Since only systems VII and XI were tested with the presence of non-condensables, the simulation results of system XI are also compared with data collected from experiments following the test matrix in Table 7.5.

Table 7.5. Conditions tested on system XI with different fault levels.

Index	Air inlet dry-bulb temperature of evaporator [K]	Air inlet dewpoint of evaporator [K]	Air inlet temperature of condenser [K]	Valve opening control
A	300	288	297	$SH_{comp,in}$ fixed at 8.7K
B	300	288	308	$SH_{comp,in}$ fixed at 8.7K
C	300	288	308	Fixed opening with $SH_{comp,in}$ at 8.7K at the non-faulted condition

7.2.1 Non-standard Charging

FXO System

Table 2.1 indicates that at lower charge level, there will be a significant increase of superheat and compressor discharge temperature and a significant decrease of evaporating pressure. The change of superheat in simulation results and experimental results with respect to charge levels is plotted in Figure 7.19.

Parallel trends between experiments and simulations in Figure 7.19 shows that the superheat increases as the charge level decreases, and the simulation models the change of the superheat with respect to charge levels correctly. When the amount of charge reduces, the average density of refrigerant inside the system is reduced and a

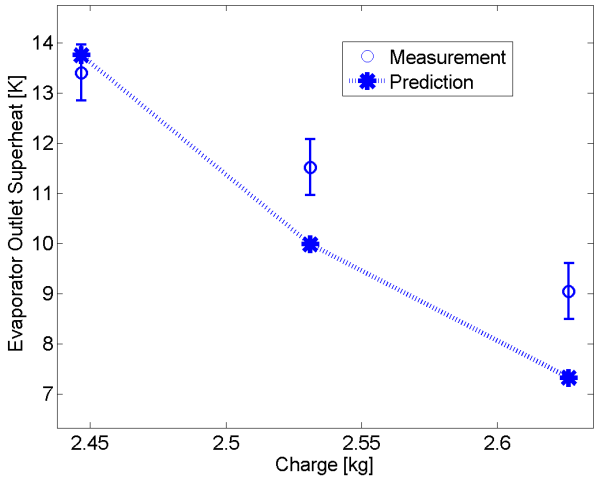

Figure 7.19. Change of superheat in system I with different charging level under condition C.

higher volume of superheated refrigerant vapor, hence the superheat, is formed during the system operation.

The change of compressor discharge temperature of system I with different charge levels is plotted in Figure 7.20.

Increasing compressor discharge temperature is found in Figure 7.20 in both experimental and simulation results for decreasing charge levels. This shows that the simulation predicts the correct change of compressor discharge temperature with respect to charge level. Similar to the increase of superheat with decreasing charge levels, the reduction of charge level inside the system induces a higher volume of superheated vapor and hence a higher temperature at the compressor discharge, where superheated vapor is found.

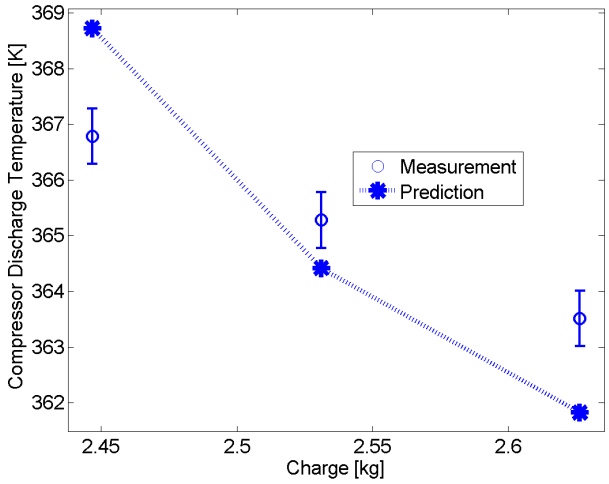

Figure 7.20. Change of compressor discharge temperature in system I with different charging level under condition C.

Figure 7.21 shows the change of compressor suction pressure with different charge levels.

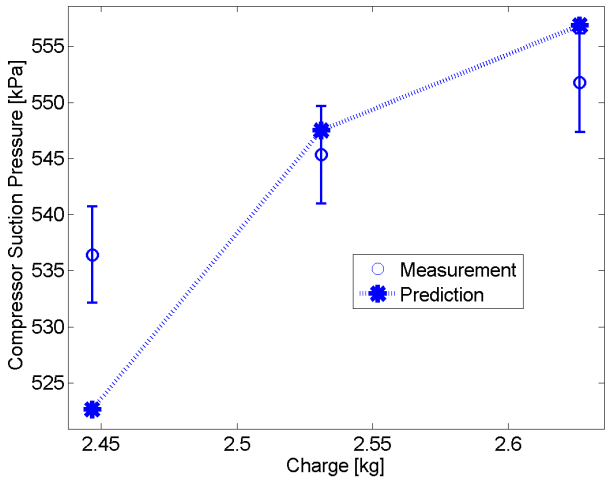

Figure 7.21. Change of compressor suction pressure in system I with different charging level under condition C.

The estimated pressure in Figure 7.21 changes in the same way as the experimental results that it decreases with decreasing charge levels. At a lower charge level, the density of refrigerant inside the system, including the evaporator, is reduced and leads to a drop of compressor suction pressure during system operation.

The change of COP of FXO systems is also inspected by examining the change of COP for system I under condition C, the change of COP for system III with evaporator air inlet temperature 300K, evaporator air inlet dewpoint at 287K, and condenser air inlet temperature at 301K, and the change of COP for system III with evaporator air inlet temperature 300L, evaporator air inlet dewpoint at 287K and condenser air inlet temperature at 309K as shown in Figures 7.22, 7.23 and 7.24.

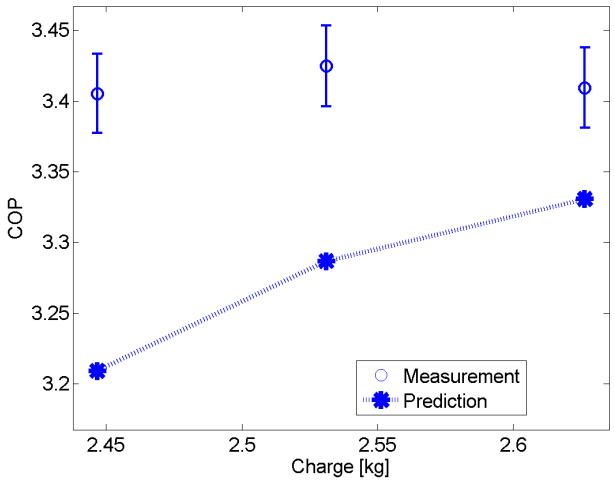

Figure 7.22. Change of COP in system I with different charging level under condition C.

Figures 7.22, 7.23 and 7.24 show that the COP of FXO systems only drops significantly when the charge level is much lower than the standard level

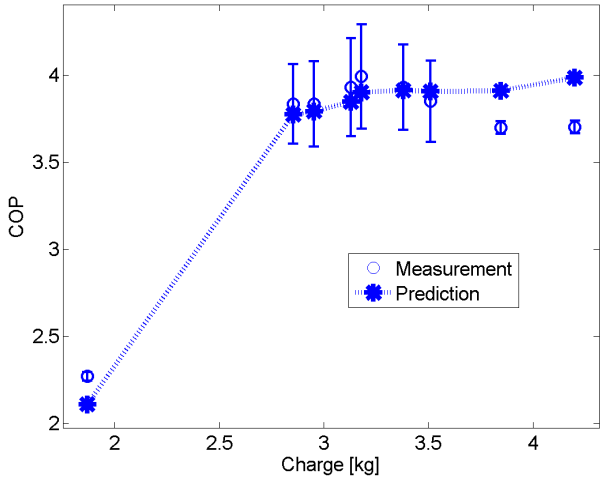

Figure 7.23. Change of COP in system III with different charging level under evaporator air inlet temperature 300K, evaporator air inlet dewpoint at 287K, and condenser air inlet temperature at 301K.

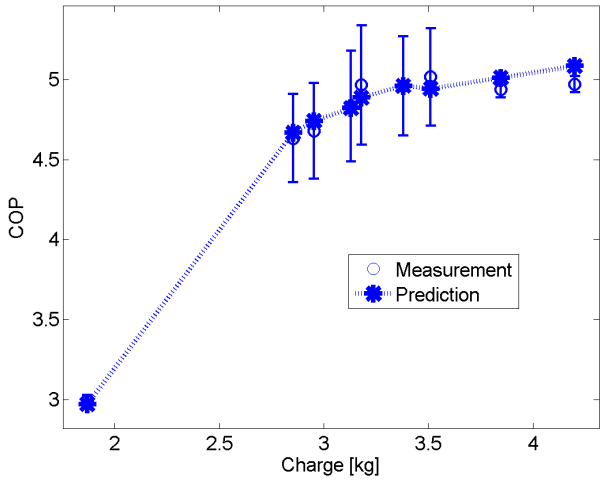

Figure 7.24. Change of COP in system III with different charging level under evaporator air inlet temperature 300L, evaporator air inlet dewpoint at 287K and condenser air inlet temperature at 309K.

significantly but the COP changes insignificantly when the charge level increases from the standard level.

TXV System

Table 2.2 shows that subcooling and evaporating temperature, hence compressor suction pressure, increase significantly with increasing charge level. The simulated subcooling is compared with experimental values under different charge levels in Figure 7.25.

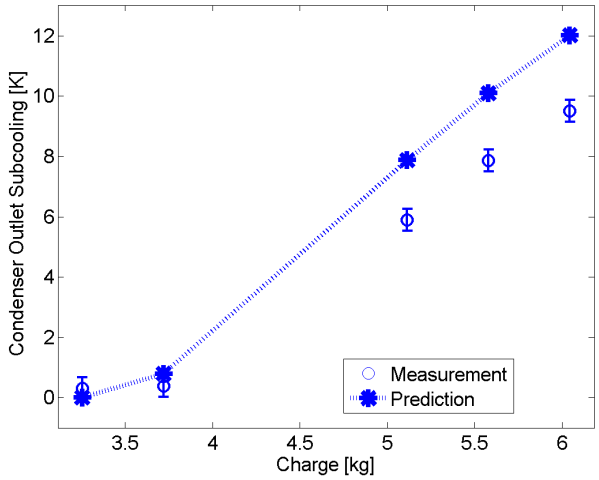

Figure 7.25. Change of condenser outlet subcooling in system VII with different charging level under condition E.

The trend of simulated condenser outlet subcooling in Figure 7.25 follows the one in the experiments that the condenser outlet subcooling decreases with charge level until the subcooling value reaches zero. Condenser is known to store most of the charge inventory in the system and is sensitive to the charge level in the system. If the system charge level decreases, the average density of refrigerant inside the condenser will decrease, and eventually the average density will be too low to maintain the

subcooling section with pure refrigerant liquid in the condenser. This leads to the drop and the eventual disappearance of subcooling with decreasing charge level.

The compressor suction pressure in these scenarios is plotted in Figure 7.26.

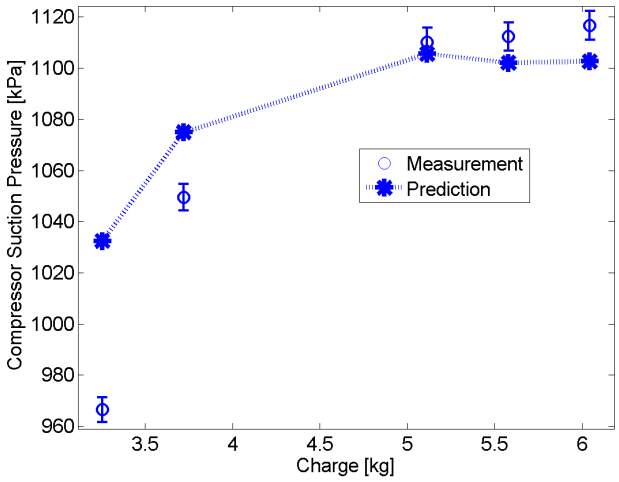

Figure 7.26. Change of compressor suction pressure in system VII with different charging level under condition E.

While the simulation estimates the same compressor suction pressure as the experiments at higher charge levels in Figure 7.26, due to the higher inaccuracy of mass flow rate estimation with two-phase flow inlet at the expansion valve model, the simulation cannot predict the pressure accurately at lower charge levels. This leads to an overestimation of the compressor suction pressure in Figure 7.26. The reduction of the pressure by a decrease of charge level, similar to an FXO system, is caused by the decrease of average refrigerant density inside the system.

The change of COP of system VII under condition E is shown in Figure 7.27.

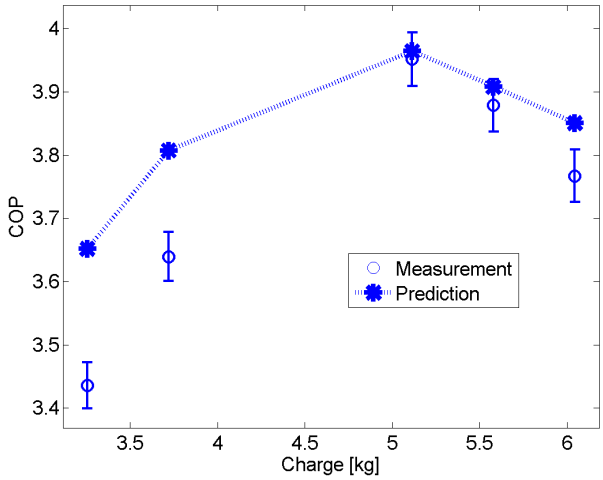

Figure 7.27. Change of compressor suction pressure in system VII with different charging level under condition E.

Figure 7.27 shows that the COP maximizes at the standard charge level and any deviation of charge level with the standard level will lower the COP of the system.

7.2.2 Evaporator Fouling

FXO System

Table 2.1 shows that a drop of compressor discharge temperature and an increase of air temperature difference across evaporator will be observed if the evaporator fouling level, represented by a reduction of evaporator airflow, increases. The change of compressor discharge temperature with respect to evaporator fouling levels is shown in Figures 7.28.

The estimated drop of compressor discharge temperature in Figure 7.28 with decreasing evaporator airflow has the same magnitude as the drop observed in the

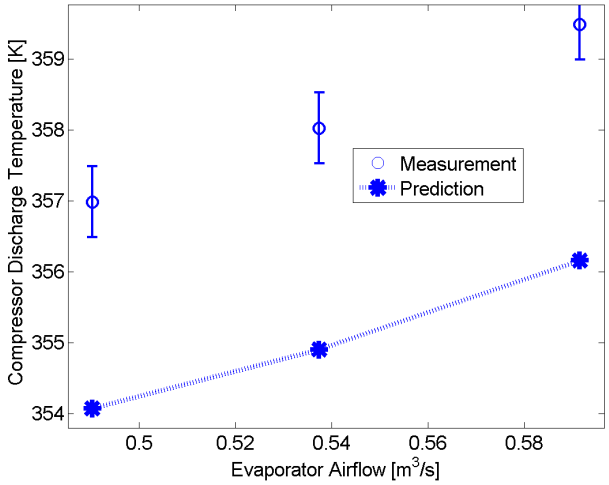

Figure 7.28. Change of compressor discharge temperature in system I with different evaporator airflow under condition B.

experiments. Although the temperature in Figure 7.28 is underestimated, it shows that the simulation is able to model the change of compressor discharge temperature with evaporator fouling levels correctly.

The air temperature difference across the evaporator in these scenarios is plotted in Figure 7.29.

The estimated air temperature difference across the evaporator in Figure 7.29 is equivalent to that of the measured values, showing that the simulation models the change of the temperature difference correctly with evaporator fouling level. The reduction of evaporator airflow reduces the saturation temperature in the evaporator and hence the air outlet temperature of the evaporator. This enhances the air temperature difference across the evaporator.

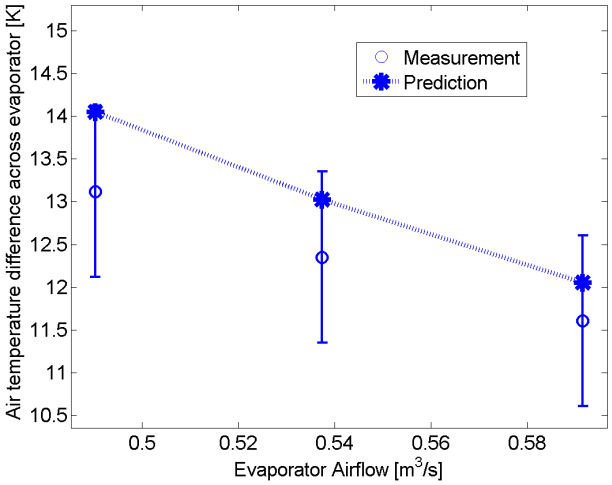

Figure 7.29. Change of air temperature difference across evaporator in system I with different evaporator airflow under condition B.

TXV System

Table 2.2 indicates that evaporator fouling leads to an increase of air temperature difference across the evaporator and a decrease of evaporating temperature, hence the compressor suction pressure. The air temperature differences across the evaporator of system VII at different evaporator fouling levels are plotted as Figure 7.30.

Although the air temperature difference is overestimated in Figure 7.30, the simulation estimates the same magnitude of the increase of temperature difference with evaporator fouling level as the experimental observations. This shows that the simulation can estimate the change of the temperature difference with evaporator fouling level correctly.

The change of compressor suction pressure with respect to evaporator fouling level is plotted in Figure 7.31.

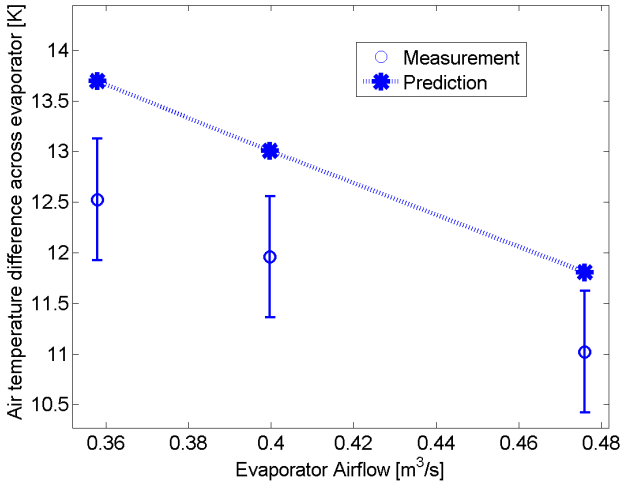

Figure 7.30. Change of air temperature difference across evaporator in system VII with different evaporator airflow under condition E.

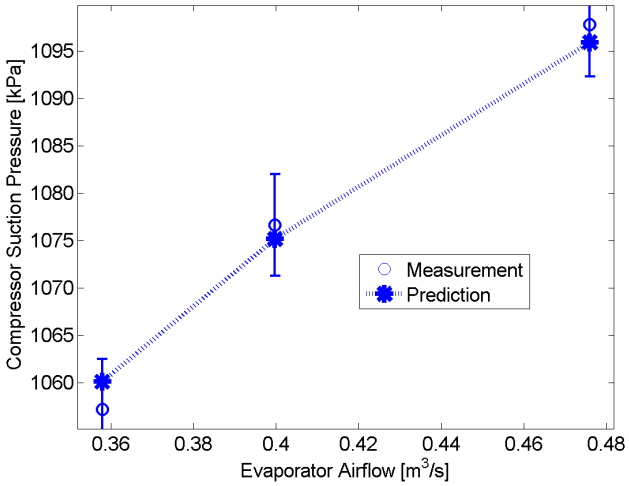

Figure 7.31. Change of compressor suction pressure in system VII with different evaporator airflow under condition E.

The simulated pressure in Figure 7.31 is equivalent to that of the measured values, showing that the simulation can predict the change of compressor suction pressure with evaporator fouling level accurately.

With increasing evaporator fouling level, the heat transfer rate of the heat exchanger drops. This reduces the evaporating temperature and hence the compressor suction pressure of the system. The drop of the evaporating temperature also causes a drop of evaporator air outlet temperature and hence an increase of air temperature difference across the evaporator.

7.2.3 Condenser Fouling

FXO system

Table 2.1 shows an increase of condensing temperature, and thereafter the compressor discharge pressure, and the air temperature across condenser as the condenser fouling level increases. The simulated air temperature across condenser is plotted in Figure 7.32.

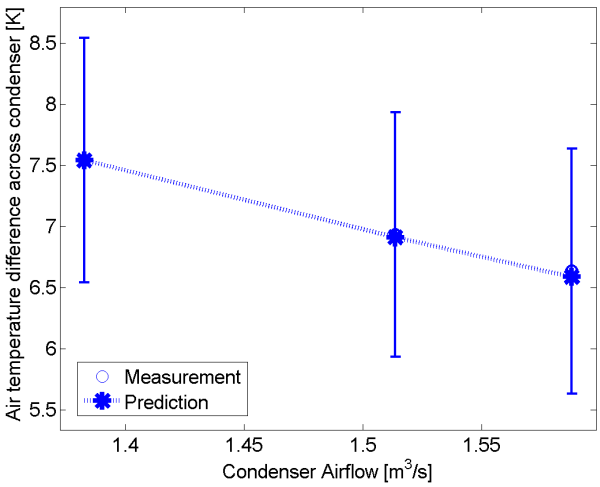

Figure 7.32. Change of air temperature across condenser in system I with different condenser airflow under condition A.

The simulation results in Figure 7.32 are equivalent to that of the experimental observations, showing that the simulation can model the change of the air temperature difference across condenser with condenser air fouling level correctly.

Figure 7.33 shows how the compressor discharge pressure changes with the condenser fouling level.

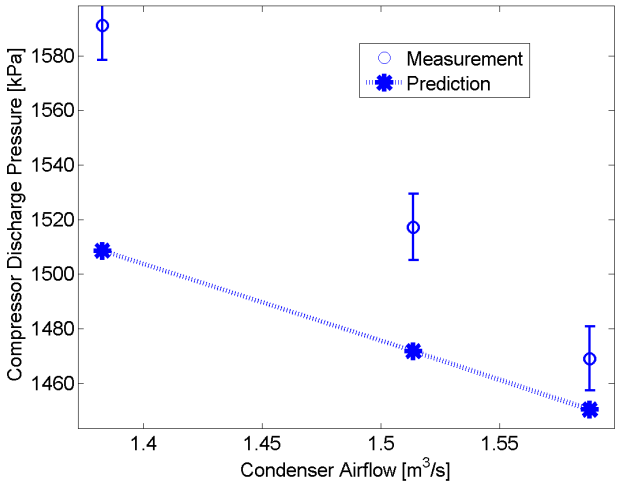

Figure 7.33. Change of compressor discharge pressure in system I with different condenser airflow under condition A.

Although the simulation outputs in Figure 7.33 are not equivalent to the experimental values, they increase with the experimental results as the condenser fouling level increases, showing the ability of the model to simulate increasing compressor discharge pressure with increasing condenser fouling level.

The effect of condenser fouling level on the condensing pressure and air temperature difference across condenser can be analyzed by considering a decrease of airflow rate across the condenser from non-faulted condition. As the condenser

fouling level increases from non-faulted condition, the condenser heat transfer rate on the air side drops while the refrigerant inlet and outlet conditions remain the same. This induces a higher temperature in the refrigerant inside the condenser and hence a higher condensing pressure. The higher saturation temperature induces a higher surface temperature on the condenser coil and a higher air outlet temperature of the condenser. This increases the air temperature difference across the condenser.

TXV System

No reliable trends with condenser fouling can be found in the experimental data of system VII and the changes of variables of system V, another TXV system, is examined for condenser fouling scenarios. Table 2.2 indicates that the condensing temperature, therefore the compressor discharge pressure, and the compressor discharge temperature increase as condenser fouling level increases. A condenser fouling case tested with air dry-bulb temperature at evaporator inlet at 300K, air dewpoint at evaporator inlet at 289K and air dry-bulb temperature at condenser inlet at 310K is examined and the change of compressor discharge pressure is plotted as Figure 7.34.

The simulated and experimental compressor discharge pressure, which are equivalent with each other, increase together in Figure 7.34 as the condenser fouling level increases. This shows that the simulation can predict the change of compressor discharge pressure with condenser fouling levels accurately.

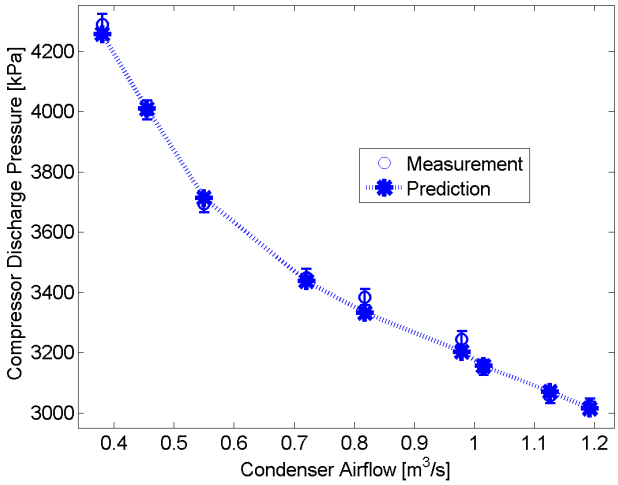

Figure 7.34. Change of compressor discharge pressure in system V with different condenser airflow.

The change of compressor discharge temperature with condenser airflow is plotted in Figure 7.35.

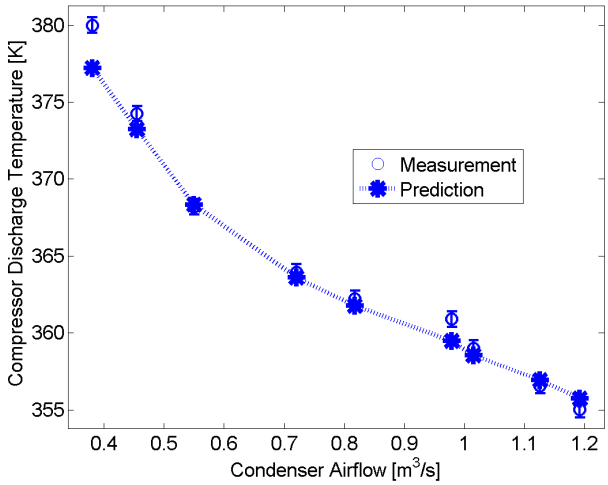

Figure 7.35. Change of compressor discharge temperature in system V with different condenser airflow.

The simulated compressor discharge temperature in Figure 7.35 underestimates the experimental values as much as 2K but both of them increase as the condenser airflow drops.

Similar to the FXO system, the increase of condenser fouling level induces a higher temperature in the refrigerant flow in the condenser and hence a higher condensing pressure. This increases the condenser inlet temperature and hence the compressor discharge temperature of the system.

7.2.4 Compressor Flow Fault

FXO System

Table 2.1 summarizes that an increase of evaporating temperature and a decrease of superheat are associated with increasing compressor flow fault levels. However, as the evaporator outlet superheat in system I was measured downstream of the refrigerant bypass return to the compressor suction in Figure 3.4, it is not a valid measurement to be used for comparison. Only the change of compressor suction pressure with increasing compressor fault level is studied, and Figure 7.36 shows the change of compressor suction pressure with increasing fault level.

An increase of simulated and experimental pressure with increasing compressor flow fault is shown in Figure 7.36. The increase of the pressure can be deduced by the response of the vapor compression system as the fault level increases from a non-faulted condition. Initially, when the fault level increases from zero, the mass

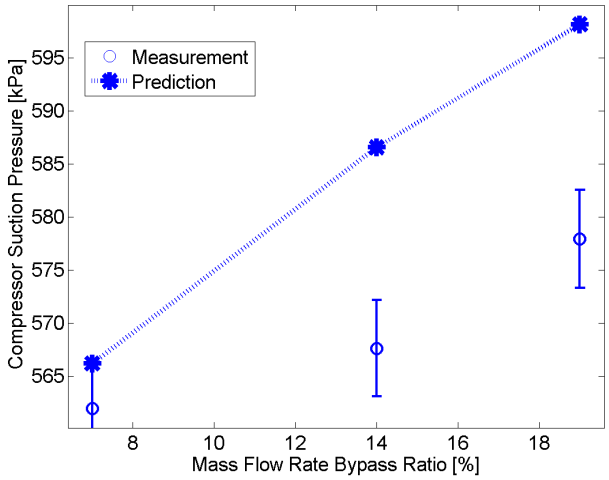

Figure 7.36. Change of compressor suction pressure in system I with different levels of compressor flow fault under condition B.

flow rate leaving the evaporator is reduced. At the same time, the refrigerant inlet and outlet condition across the expansion valve, and hence the refrigerant mass flow rate at the inlet of the evaporator remain unchanged. This increases the amount of refrigerant in the evaporator, the average refrigerant density across the evaporator and the compressor suction pressure of the system.

TXV System

Compressor flow fault reduces the condensing pressure and increases the evaporating pressure of system VII according to Table 2.2. The change of condensing pressure and evaporating pressure with increasing fault level is plotted as Figures 7.37 and 7.38.

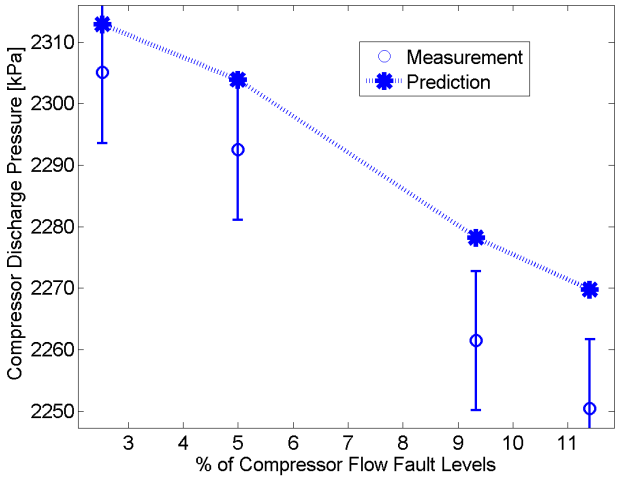

Figure 7.37. Change of compressor discharge pressure in system VII with different levels of loss of compressor volumetric efficiency under condition B.

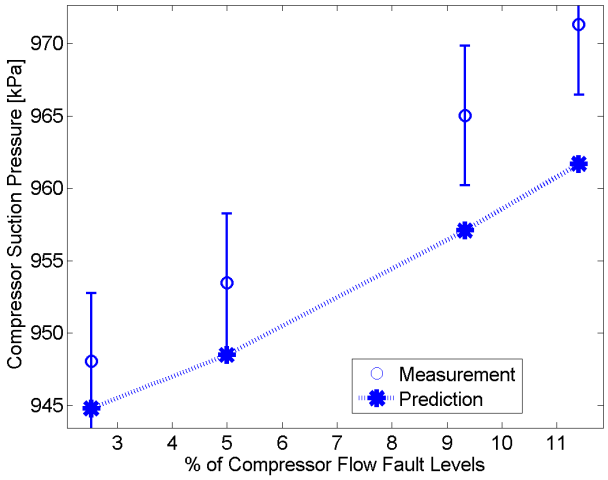

Figure 7.38. Change of compressor suction pressure in system VII with different levels of loss of compressor volumetric efficiency under condition B.

The simulation outputs in Figures 7.37 and 7.38 follow the experimental results that condenser pressure decreases and evaporating pressure increases with increasing

compressor flow fault level. When the fault level increases during operation, the refrigerant mass flow rates leaving the evaporator and entering the condenser decrease while the instantaneous refrigerant mass flow rate at the expansion valve remains unchanged. This reduces the mass of refrigerant in the condenser and increases the mass of refrigerant in the evaporator. The change of mass distribution is reflected by the decrease of condensing pressure and temperature and the increase of evaporating pressure and temperature in the system.

7.2.5 Liquid Line Restriction

FXO system

Table 2.1 shows that the changes of superheat and compressor discharge temperature are significant with increasing liquid line restriction level. Figure 7.39 shows the change of superheat with different levels of liquid line restriction and Figure 7.40 shows the change of compressor discharge temperature under the same condition.

Both Figures 7.39 and 7.40 show increasing simulated and experimental values with an increasing liquid line restriction pressure drop, and the system model can simulate the change of system performance with liquid line restriction with the correct trend.

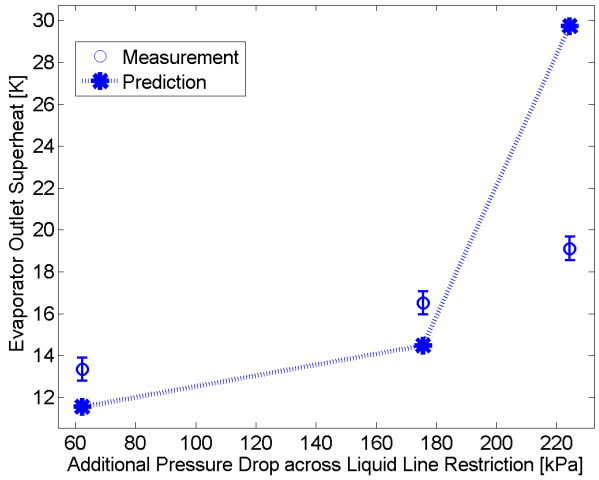

Figure 7.39. Change of evaporator outlet superheat in system I with different liquid line restriction pressure drop under condition A.

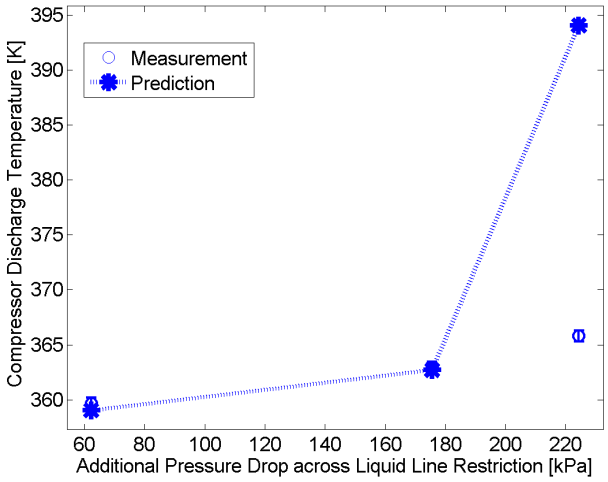

Figure 7.40. Change of compressor discharge temperature in system I with different liquid line restriction pressure drop under condition A.

TXV System

Table 2.2 shows that increasing liquid line restriction level in a TXV system increases the superheat and compressor discharge temperature. The plots of these

variables with increasing liquid line restriction level in system VII are shown in Figures 7.41 and 7.42.

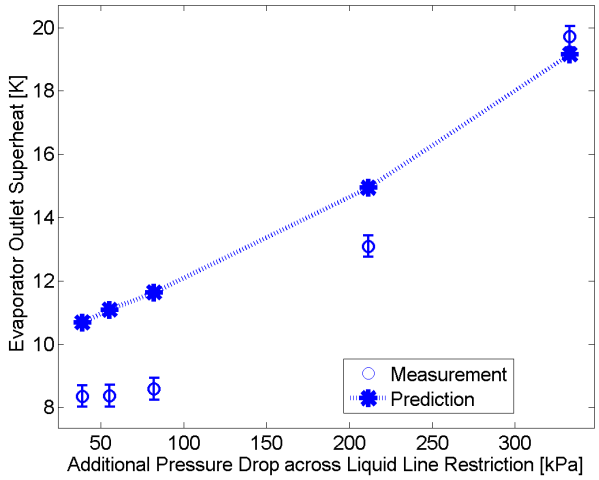

Figure 7.41. Change of evaporator outlet superheat in system VII with different liquid line restriction levels under condition D.

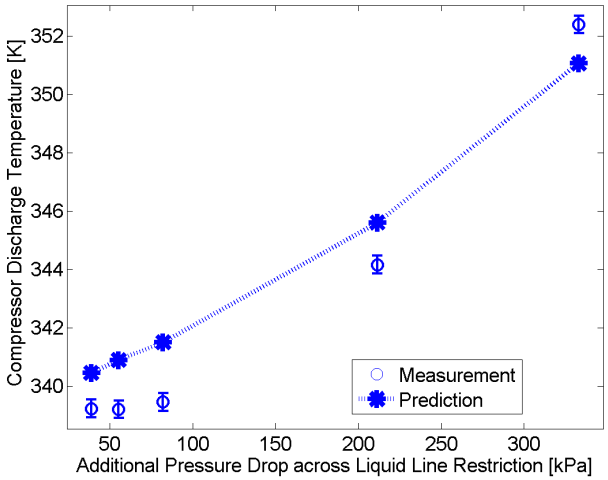

Figure 7.42. Change of compressor discharge temperature in system VII with different liquid line restriction levels under condition D.

Similar to system I which is a FXO system, the simulation and experimental results of system VII exhibit the same trend that both the superheat and compressor discharge temperature increase with increasing liquid line restriction level, and the system model can model the change of system variables with different liquid line fault levels correctly.

The change of system variables with liquid line restriction level in FXO and TXV systems can be explained by an increase of liquid line restriction during system operation. When pressure drop across the restriction of the liquid line increases, assuming negligible change of liquid line heat transfer rate, it reduces the pressure and subcooling at the inlet of the expansion valve and hence the refrigerant mass flow rate of the expansion valve. However, at this instant, the refrigerant conditions across the compressor remain unchanged, and the compressor refrigerant mass flow rate remains the same. This migrates refrigerant from the evaporator to the condenser, and the reduction of charge inventory in the evaporator is reflected by the increase of evaporator outlet superheat. Since the compressor suction superheat increases with the evaporator outlet superheat and it is common knowledge that compressor discharge superheat increases with its suction superheat for single-speed compressors, the compressor discharge temperature increases. While TXV is able to compensate the effect of liquid line restriction by controlling its system superheat, the increase of superheat in Figure 7.41 suggests that the TXV will become fully opened for high liquid line restriction level and the system

variables in TXV system, after fully opening its TXV, will respond to liquid line restriction level in the same way as an FXO system.

7.2.6 Presence of Non-condensables

FXO System

Although no FXO systems were tested with non-condensables in the system, tests under condition C of system XI in Table 7.5 were conducted with a fixed opening and they can be used to evaluate the accuracy of the system model to simulate the change of FXO system variables with increasing amount of non-condensables in the system. Since there are no literature of experimental studies on which system variables in FXO systems change with increasing amount of non-condensable in the system significantly, evaporator heat transfer rate, compressor power consumption, sensible heat ratio, condenser outlet subcooling, compressor discharge temperature, compressor discharge pressure and compressor suction pressure from the system model are compared with their experimental counterparts to validate the simulation.

Figures 7.44 and 7.43 show the comparison result of evaporator heat transfer rate and sensible heat ratio.

The change of experimental and simulation result in Figures 7.44 and 7.43 show that the evaporator heat transfer rate and sensible heat ratio do not change significantly with the fault.

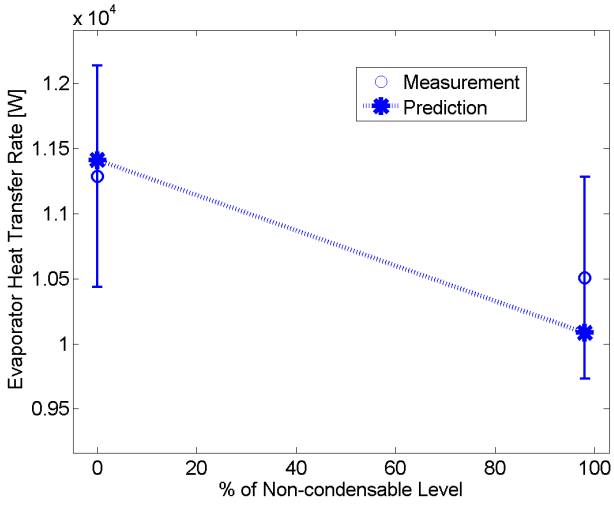

Figure 7.43. Change of evaporator heat transfer rate in system XI with different levels of presence of non-condensables under condition C.

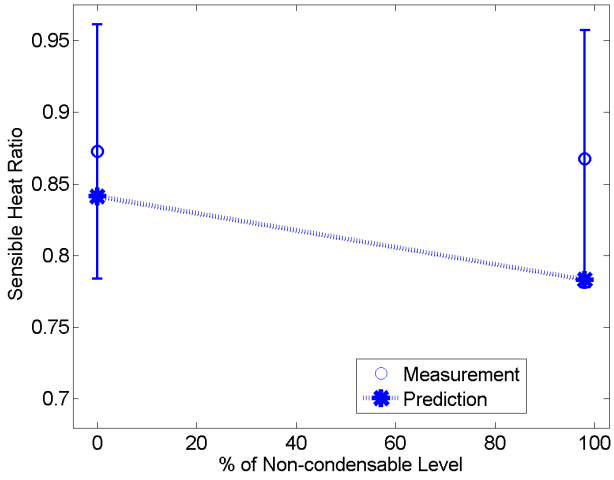

Figure 7.44. Change of sensible heat ratio in system XI with different levels of presence of non-condensables under condition C.

Figures 7.45, 7.46, 7.47 and 7.48 show the measured and estimated change of compressor power consumption, compressor discharge pressure, condenser outlet

subcooling and compressor discharge temperature with increasing levels of non-condensables under condition C.

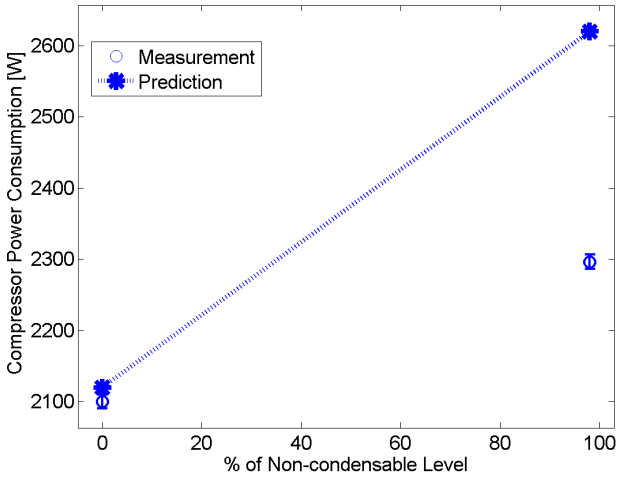

Figure 7.45. Change of compressor power consumption in system XI with different levels of presence of non-condensables under condition C.

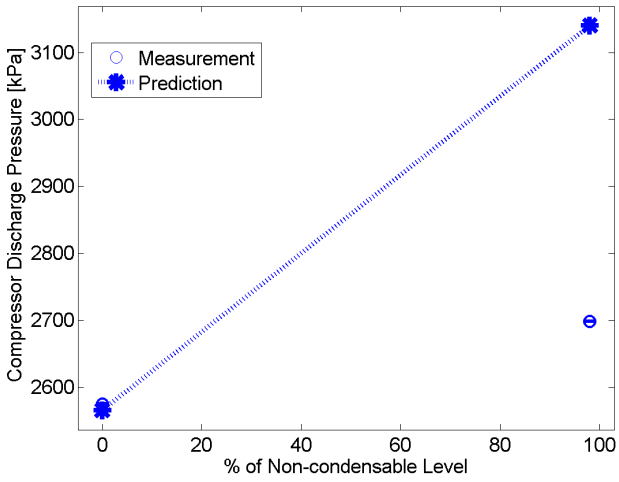

Figure 7.46. Change of compressor discharge pressure in system XI with different levels of presence of non-condensables under condition C.

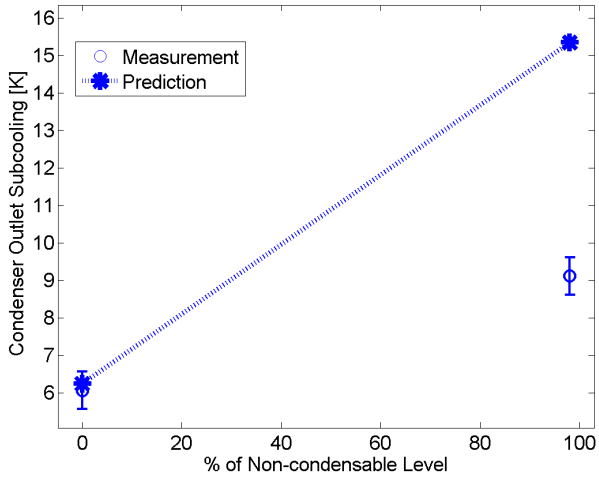

Figure 7.47. Change of compressor discharge pressure in system XI with different levels of presence of non-condensables under condition C.

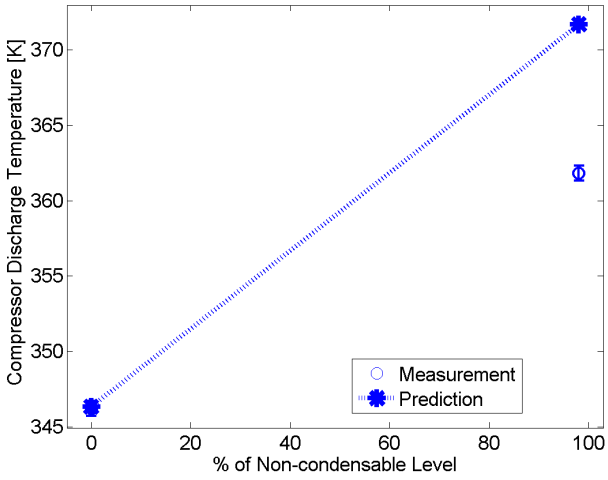

Figure 7.48. Change of compressor discharge pressure in system XI with different levels of presence of non-condensables under condition C.

While the system model can predict the increase of system variables correctly in Figures 7.45, 7.46, 7.47 and 7.48, the system variables in these figures are all

overestimated significantly. The reason of the overestimation is same as the discussion in Section 7.1. The simulation assumes that all non-condensables are trapped in the condenser, but experimental observations show that the expansion valve is taking two-phase flow when non-condensable fault level is high and the inlet subcooling to the expansion valve is higher than 3K. This shows that non-condensable in the system circulates along the refrigerant pipes with the refrigerant flow. By assuming that all non-condensables are trapped in the condenser, the simulation overestimates the partial pressure exerted by the non-condensable on the refrigerant flow in the condenser and hence the compressor discharge pressure, the compressor power consumption, the condenser outlet subcooling and the compressor discharge temperature.

Figure 7.49 shows the change of compressor suction superheat with the amount of non-condensables in the system under condition C.

Figure 7.49 shows that the compressor suction superheat increase with the amount of non-condensable in the system for a system with a fixed valve opening. Since the model of system XI takes the measured compressor suction superheat as an input, the estimated and measured compressor suction superheat in Figure 7.49 is identical.

The change of compressor suction pressure of system XI with different levels of non-condensables under condition C is plotted in Figure 7.50.

Figure 7.50 shows that the simulation can predict the drop of compressor suction pressure correctly.

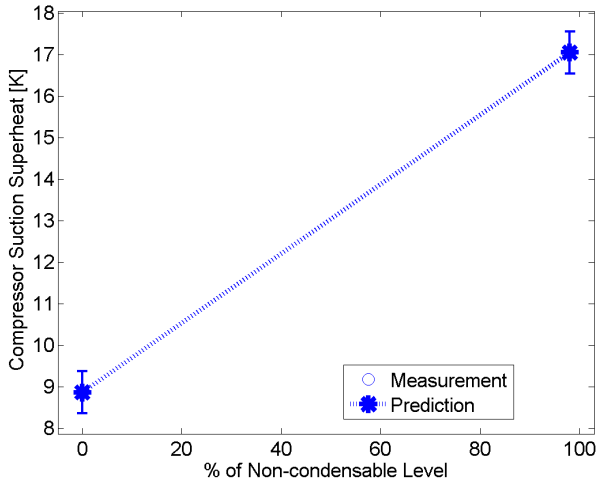

Figure 7.49. Change of compressor suction superheat in system XI with different levels of presence of non-condensables under condition C.

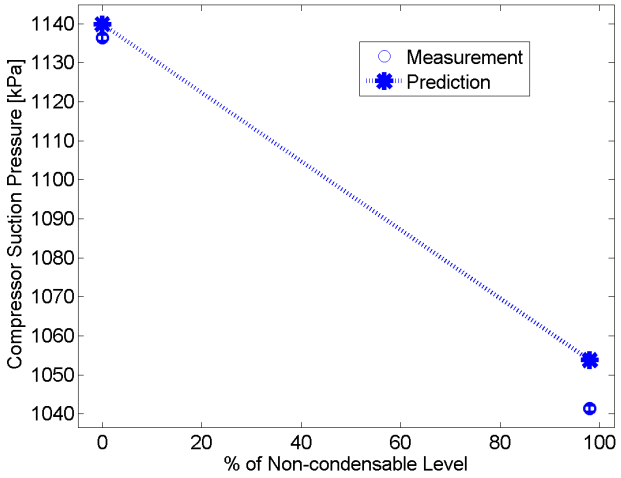

Figure 7.50. Change of compressor suction pressure in system XI with different levels of presence of non-condensables under condition C.

TXV System

Table 2.2 shows that the condenser outlet subcooling and condensing temperature increase with the level of non-condensable fault. Figures 7.51 and 7.52

plot the changes of condenser outlet subcooling and compressor discharge pressure with increasing amount of non-condensable in system VII.

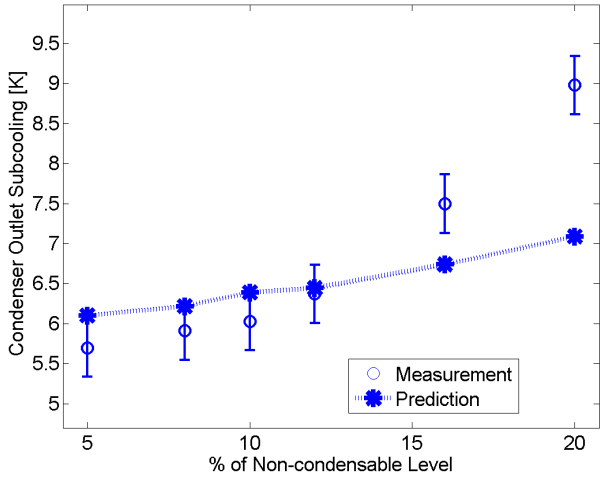

Figure 7.51. Change of condenser outlet superheat in system VII with different levels of presence of non-condensables under condition D.

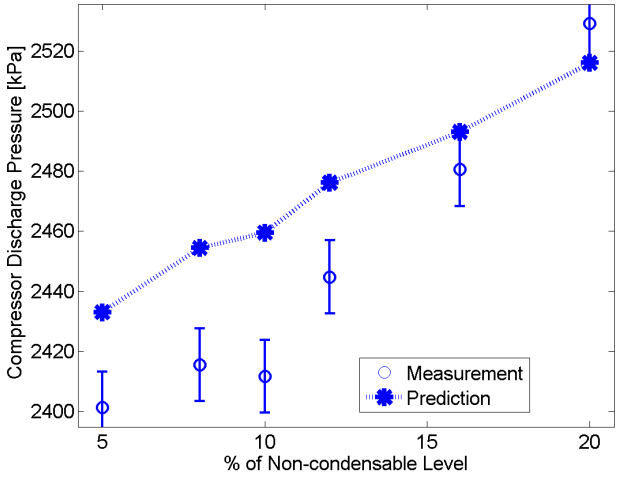

Figure 7.52. Change of compressor outlet pressure in system VII with different levels of presence of non-condensables under condition D.

Figure 7.52 shows that the simulation predicts the same compressor discharge pressure when the fault level is above 15%. Figure 7.51 also shows that the subcooling prediction is parallel with the experimental values below 12%. However, when the fault level is higher than 12%, the experimental condenser outlet subcooling increases with the fault level much more significantly than the estimated condenser outlet subcooling. To examine if the inaccuracy affects the accuracy to simulate the change of the overall performance with the fault level, the changes of compressor power consumption, COP and SHR with the fault level under condition D are plotted in Figures 7.53, 7.54 and 7.55.

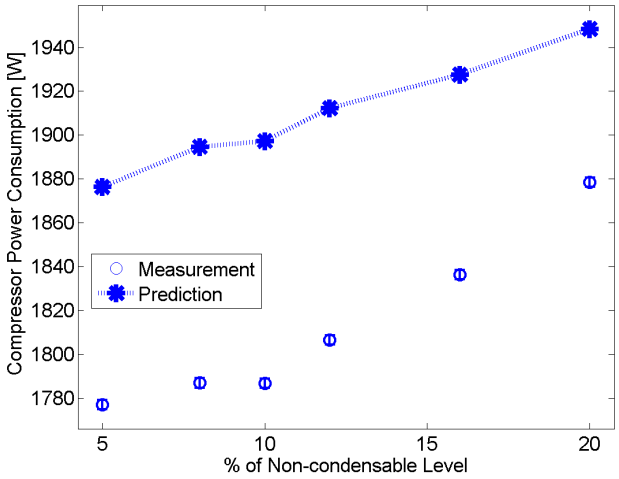

Figure 7.53. Change of compressor power consumption in system VII with different levels of presence of non-condensables under condition D.

Figures 7.53 and 7.54 show that the simulation can estimate the decrease of COP and increase of compressor power consumption with increasing amount of non-condensables in the system. However, the estimated magnitudes of the changes

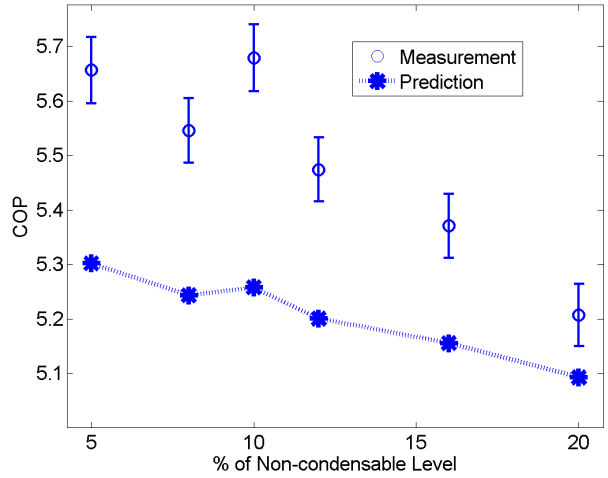

Figure 7.54. Change of COP in system VII with different levels of presence of non-condensables under condition D.

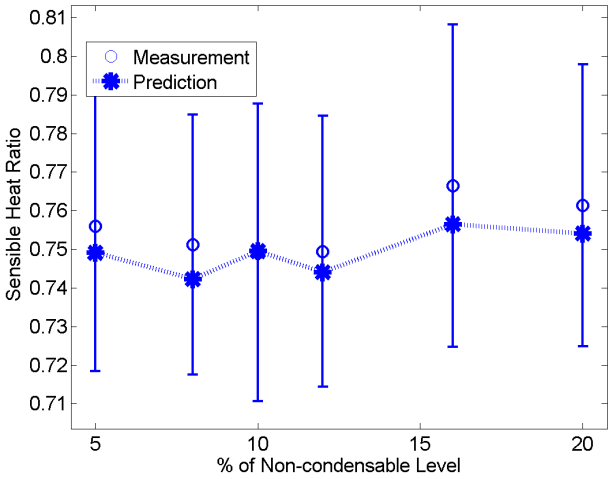

Figure 7.55. Change of SHR in system VII with different levels of presence of non-condensables under condition D.

are smaller than the experimental observations. The simulation can also predict the constant SHR with increasing fault level as shown by Figure 7.55. This shows that

the system model predicts the change of the overall system performance with respect to the fault level in the system appropriately.

To examine if the estimation inaccuracy in Figure 7.51 occurs in other systems, the simulation result of the operation of system XI under condition A, where the compressor suction superheat is maintained at 8.7K at all fault levels, is compared with the experimental observation. The estimated and measured compressor outlet pressure and condenser outlet subcooling of system XI under condition A are plotted in Figures 7.56 and 7.57.

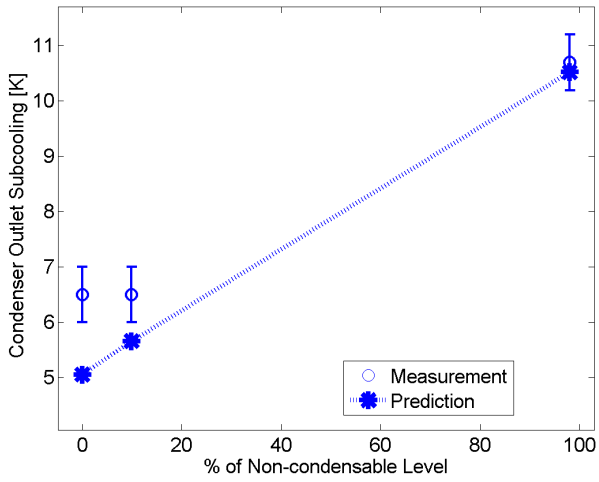

Figure 7.56. Change of condenser outlet subcooling in system XI with different levels of presence of non-condensables under condition A.

Figures 7.56 and 7.57 show that the system model can simulate the increase of condenser outlet subcooling and compressor outlet pressure of system XI correctly with increasing amount of non-condensables in the system. Since the major difference in the system model between system VII and system XI is the expansion valve model

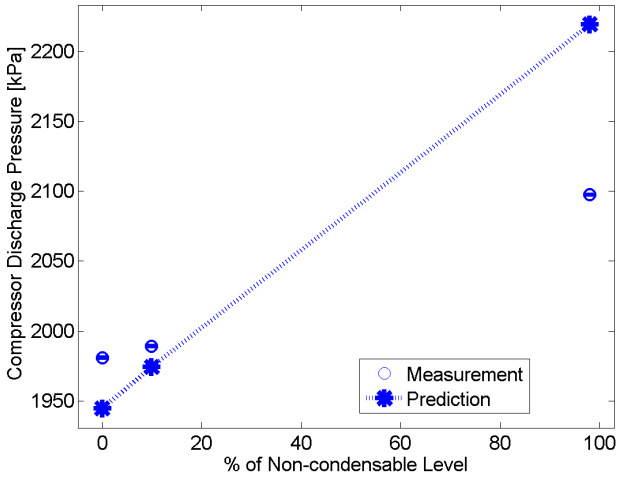

Figure 7.57. Change of compressor outlet pressure in system XI with different levels of presence of non-condensables under condition A.

and compressor suction superheat is strongly affected by the expansion valve model, the change of the compressor suction superheat in system VII under condition D and in system XI under condition A is plotted in Figures 7.58 and 7.59.

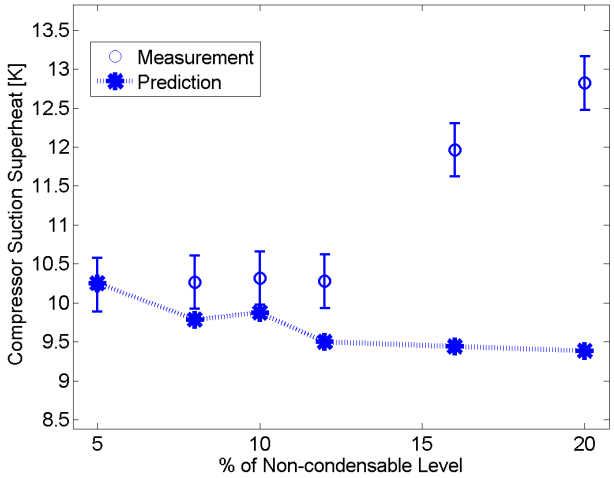

Figure 7.58. Change of compressor suction superheat in system VII with different levels of presence of non-condensables under condition D.

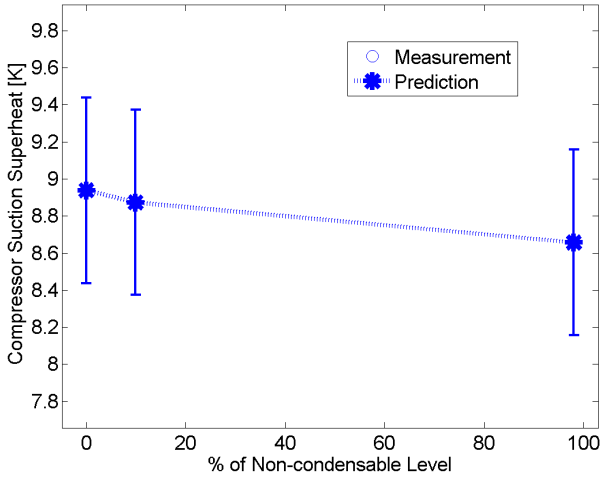

Figure 7.59. Change of compressor suction superheat in system XI with different levels of presence of non-condensables under condition A.

Figure 7.59 show a perfect match of the compressor suction superheat of the simulation result with measurement because the measured compressor suction superheat is the input to the model of system XI. However, Figure 7.58 shows that the measured compressor suction superheat increases with the amount of non-condensable in the system while the estimated compressor suction superheat decreases slightly with the amount of non-condensable in the system. This shows that the TXV in the experiments with large amount of non-condensables in the system is fully opened but the model cannot simulate the fully opened valve. When the system contains a large amount of non-condensables, non-condensables may enter the expansion valve together with refrigerant despite large subcooling at expansion valve inlet as reported in Section 7.1. This two-phase flow decreases the refrigerant mass flow rate across the expansion valve and opens the TXV

completely. As the TXV becomes fully opened, the valve works as an FXO and the compressor suction superheat increases. Although the model of system XI receives compressor suction superheat as an input and can simulate the resultant drop of refrigerant mass flow rate across the TXV, the model of system VII cannot simulate the increase of compressor suction superheat because the system model does not simulate non-condensables flow across the expansion valve and the resultant decrease of refrigerant mass flow rate in the system. Since the expansion valve models do not simulate the decrease of refrigerant flow at high non-condensable fault level, the system models without compressor suction superheat as input cannot simulate the increase of compressor suction superheat and condenser outlet subcooling with increasing amount of non-condensables in the system at high fault level correctly.

To sum up, while the change of the overall performance of the systems with the amount of non-condensables in the system can be simulated correctly, if higher accuracy is needed for the change of condenser outlet subcooling and compressor suction superheat with respect to the fault level, the expansion valve model should be modified to simulate the mass flow rate of refrigerant due to the presence of non-condensables.

7.3 Comparing Results from the FDD tool evaluator

To ensure that the model reaches its end goal: to replace experimental data for the evaluation of FDD tools of vapor compression systems, both the experimental data

and the simulation outputs at the experimental conditions were provided to the FDD tool evaluator to examine if the FDD tool evaluator yielded the same evaluation result with the simulation outputs as the experimental data. For each data point provided to the FDD tool evaluator, different FDD tools were instructed to detect if a fault exists, to identify the type of fault and to estimate the fault levels of the data point based on the refrigerant and air properties in the data point. After comparing the FDD tool prediction with the fault levels for multiple data points, multiple performance indicators of the FDD tools such as no response rate, false alarm rates, misdiagnosis rates, missed detection rate and no diagnosis rates are calculated. By examining the difference of the indicators between the ones from the simulation outputs and the ones from the experimental observations, the ability of the system model to give accurate evaluation result can be validated. The validation process would be described in another dissertation in details (Yuill, 2014).

7.4 Summary

The validation process of the system model was conducted by three different methods: overall system performance comparison, comparison of the changes of system variables with fault levels, and comparison of results from the FDD tool evaluator.

In overall system performance comparison, the simulation outputs and experimental observations were compared in terms of evaporator heat transfer rate,

compressor power consumption and sensible heat ratio. After analyzing the causes of the outliers, it can be concluded that the model estimation is accurate.

When comparing the estimated and measured changes of system variables with different fault levels, it is found that the system model can predict the changes of the system variables under non-standard charging, heat exchanger fouling, compressor flow fault and liquid line restriction correctly. However, when the system contains a large amount of non-condensables, because the system model does not simulate the reduction of refrigerant flow as a result of non-condensable passing through the expansion valve, the system model cannot simulate the correct change of compressor suction superheat and condenser outlet subcooling with increasing amount of non-condensable in the system.

The ability of the system model to replace experimental results to assess the performance of FDD tools was also examined with the help of an FDD tool evaluator, and the results would be reported in another dissertation (Yuill, 2014).

CHAPTER 8. FAULT IMPACTS ON SYSTEM PERFORMANCE

With the simulation model validated in Chapter 7, the study of fault impacts on the system can be conducted by studying the changes of the overall system performance, the pressure-enthalpy diagram, temperature-entropy diagrams of system operation with increasing fault levels. The study was conducted with the simulation outputs of system IV (an FXO system), system VII (a TXV system) and system IX (an FXO system with accumulator) under different levels of faults. Temperature-entropy (T-s) diagrams and pressure-enthalpy (P-h) diagrams were generated from the simulation outputs. The magnitudes of evaporator heat transfer rate, compressor power consumption and refrigerant mass flow rate are also non-dimensionalized with their values at the non-faulted conditions. These results under the same type of fault were compared with each other to examine how faults affect system performance.

The environmental conditions imposed in the simulation are tabulated in Table 8.1 and other standard inputs such as standard airflow rates can be found in Table 3.10.

To describe the change of system performance with change of fan power, the coefficient of system performance (COSP) is also calculated in this chapter by

Table 8.1. Conditions tested on system VII with different fault levels.

System	Air inlet dry-bulb temperature of evaporator [K]	Air inlet dewpoint of evaporator [K]	Air inlet temperature of condenser [K]
IV	300	289	308
VII	300	289	304
IX	294	289	304

$$COSP = \frac{\dot{Q}_{evap} - \dot{W}_{evap,fan}}{\dot{W}_{comp} + \dot{W}_{evap,fan} + \dot{W}_{cond,fan}}. \tag{8.1}$$

Coefficient of system performance (COSP) in Eqn. (8.1) is the ratio of total cooling capacity delivered to the system user to the total power consumption of the system. It assumes that all evaporator fan power enters the air stream across the evaporator as heat and reduces the cooling capacity delivered by the vapor compression system. It also considers the power consumption to operate the fans in its denominator. The change of COSP is similar to that of COP under most faulted conditions except heat exchanger fouling cases because fan power consumption changes with heat exchanger fouling level.

Enthalpy and entropy values shown in the T-s and P-h diagrams in this chapter were calculated from REFPROP 9.0 (Lemmon et al., 2013).

8.1 Non-standard Charging

8.1.1 FXO System

System IV was simulated with charge levels from 70% to 115% and the P-h and T-s diagrams are plotted in Figures 8.1 and 8.2.

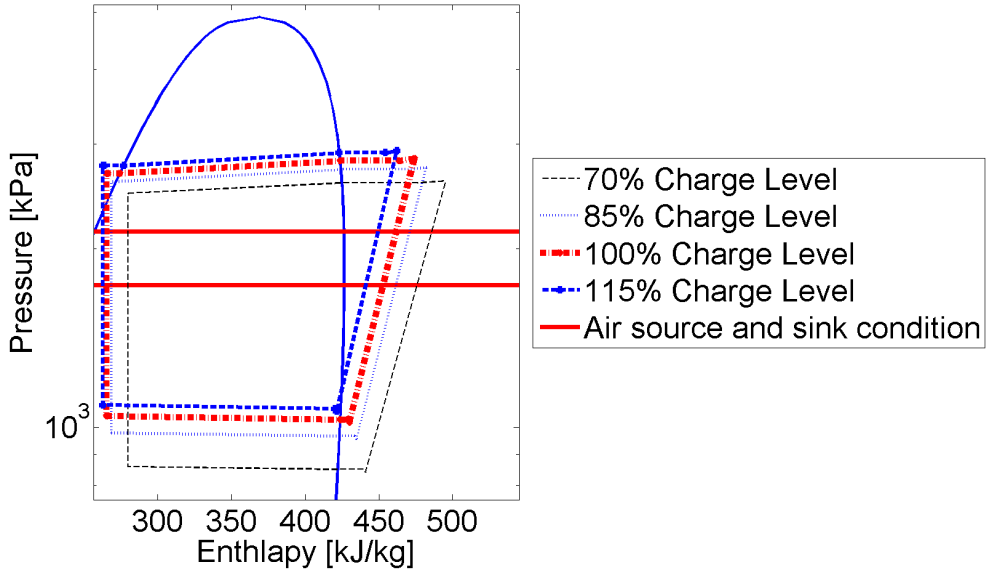

Figure 8.1. P-h diagram of system IV at different charge levels.

The subcooling at the condenser outlet and the condensing and evaporating pressure increase as the charge level increases in Figures 8.1 and 8.2. At the same time, the evaporator outlet superheat and compressor discharge superheat decrease. Upon the compressor suction superheat decreasing to zero, the thermodynamic quality of refrigerant at compressor suction decreases with increasing charge level.

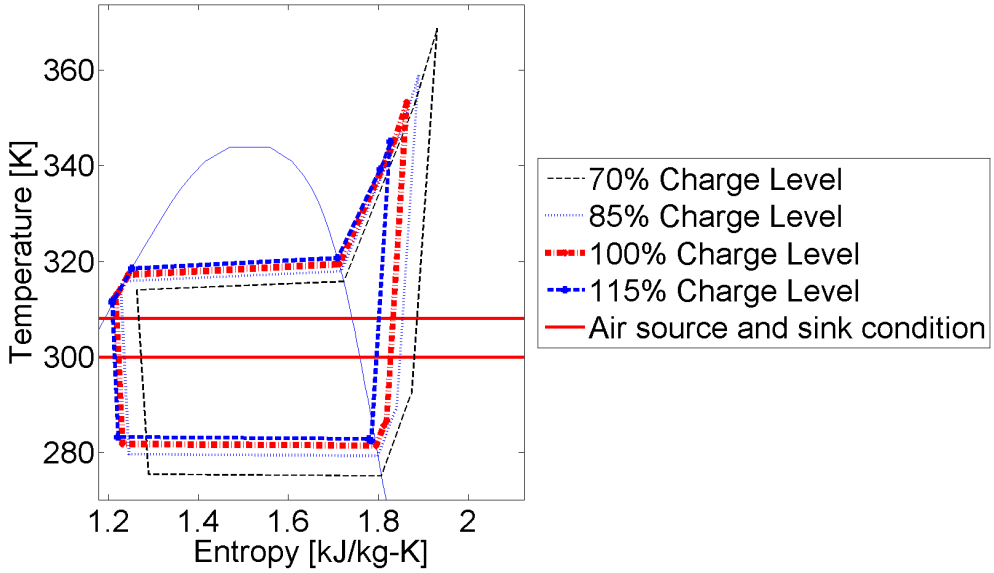

Figure 8.2. T-s diagram of system IV at different charge levels.

The overall performance associated with these changes is plotted in Figures 8.3 and 8.4, which show the change of COP, evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate, COSP and SHR with respect to different charge levels.

Figures 8.3 and 8.4 show that when charge level is increasing, the sensible heat ratio and the refrigerant mass flow rate are increasing with increasing charge level. The evaporator heat transfer rate and compressor power consumption also increase. This leads to an increasing COP with increasing charge level.

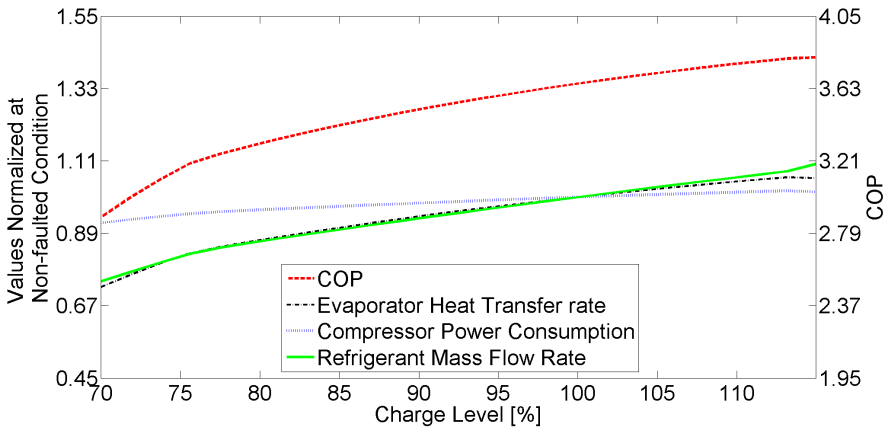

Figure 8.3. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IV at different charge levels.

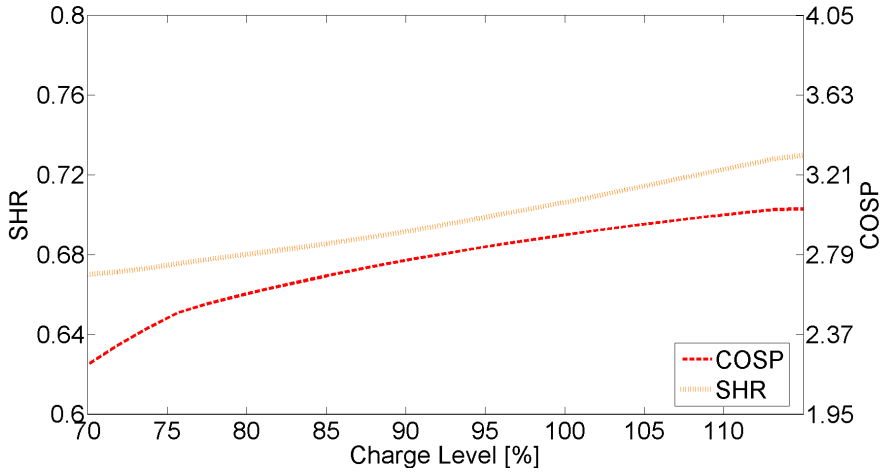

Figure 8.4. Change of SHR and COSP of system IV at different charge levels.

8.1.2 FXO System with an Accumulator

The P-h and T-s diagrams of system IX at different charge levels are plotted in Figures 8.5 and 8.6.

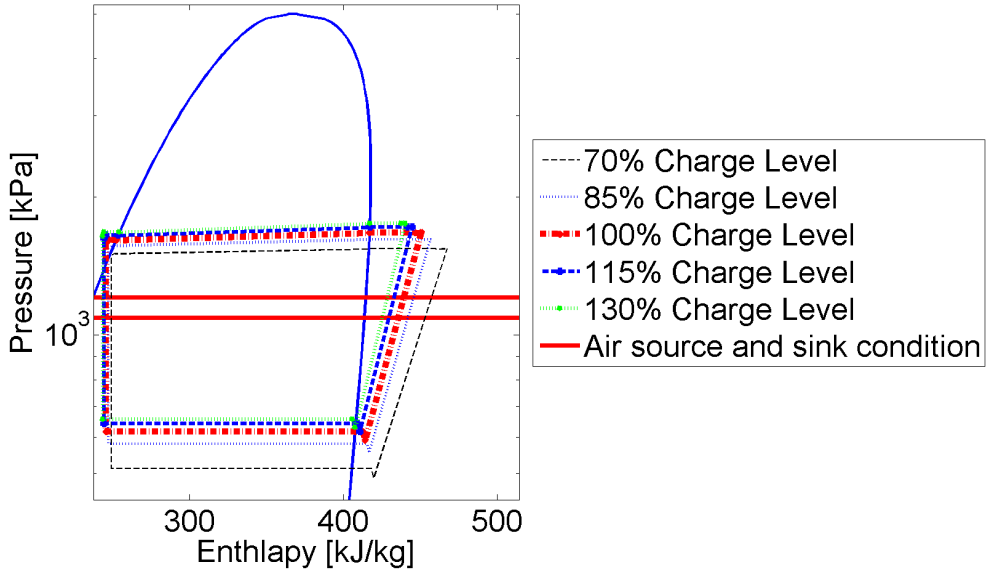

Figure 8.5. P-h diagram of system IX at different charge levels.

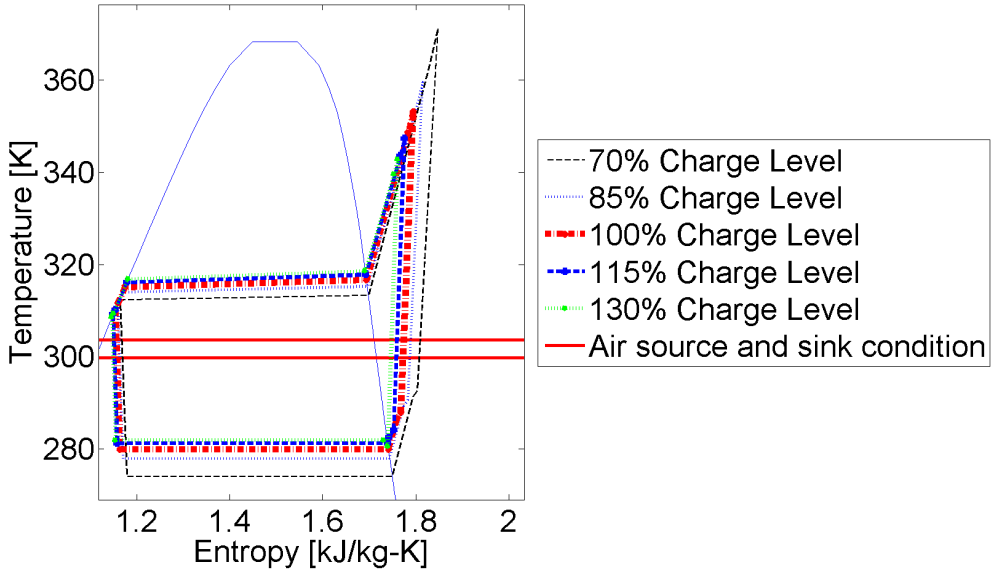

Figure 8.6. T-s diagram of system IX at different charge levels.

Figures 8.5 and 8.6 show similar change of P-h and T-s diagrams with decreasing charge level as Figures 8.1 and 8.2 when the compressor suction superheat is positive. When compressor suction superheat is positive, the accumulator does not affect the operation of the system and systems IV and IX behave in the same way with changing charge levels. However, if the change level in system IX continues to increase and the compressor suction superheat drops to zero, the accumulator will affect the system performance. This is illustrated by the change of evaporator heat transfer rate, compressor power consumption, COP, COSP and SHR of system IX with different charge levels in Figures 8.7 and 8.8.

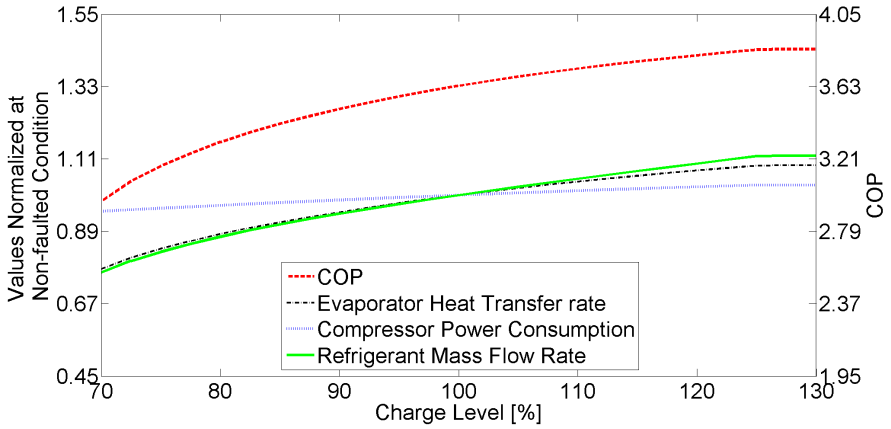

Figure 8.7. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IX at different charge levels.

Figures 8.7 and 8.8 show that COP, refrigerant mass flow rate, compressor power consumption, evaporator heat transfer rate, COSP and SHR increase with increasing charge level until the charge level reaches 128%. At this charge level, the compressor suction superheat drops to zero and the accumulator starts to hold liquid refrigerant.

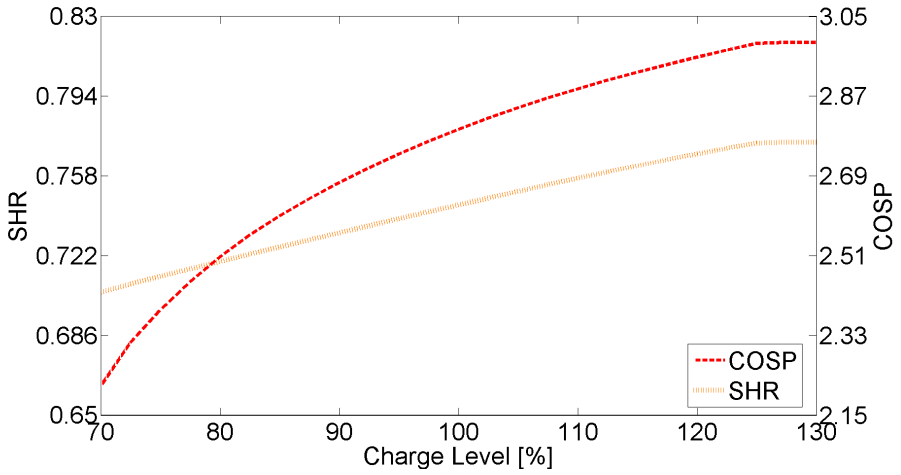

Figure 8.8. Change of SHR and COSP of system IX at different charge levels.

Any increase of refrigerant charge would only fill up the accumulator and has no effect to the system performance. This is demonstrated by the flat curves in Figures 8.7 and 8.8 beyond the charge level at 128%.

8.1.3 TXV System

The P-h and T-s diagrams of system VII at different charge levels are plotted in Figures 8.9 and 8.10.

Figures 8.9 and 8.10 show that when the charge level increases from 85% to 130%, the TXV controls the superheat at the evaporator outlet, leading to an increase of compressor discharge temperature with the increasing charge level. The P-h diagram also changes the direction from moving towards the upper left-handed corner to expanding upwards as the charge level increases. This results in a negligible change of evaporating pressure and a rapid increase of condensing

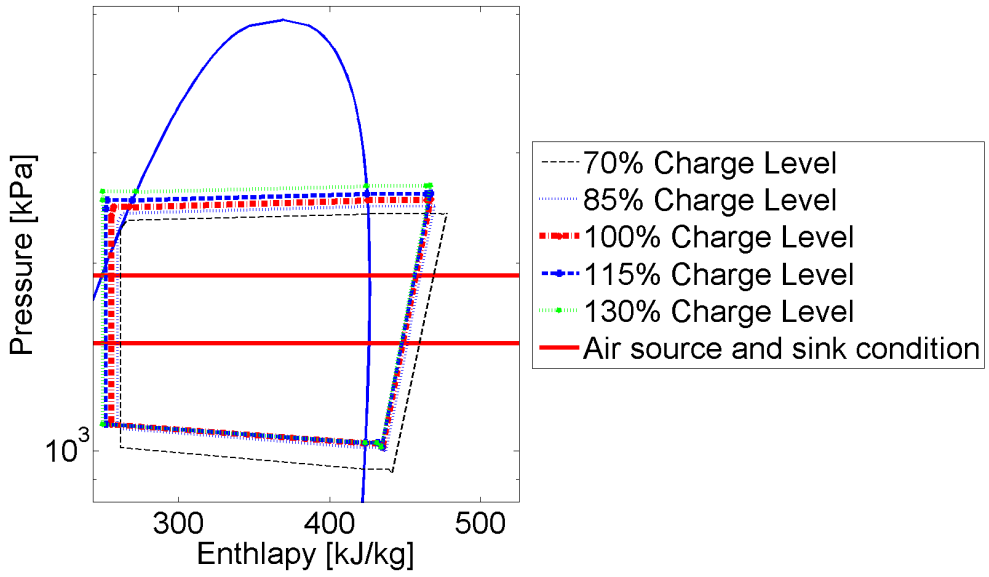

Figure 8.9. P-h diagram of system VII at different charge levels.

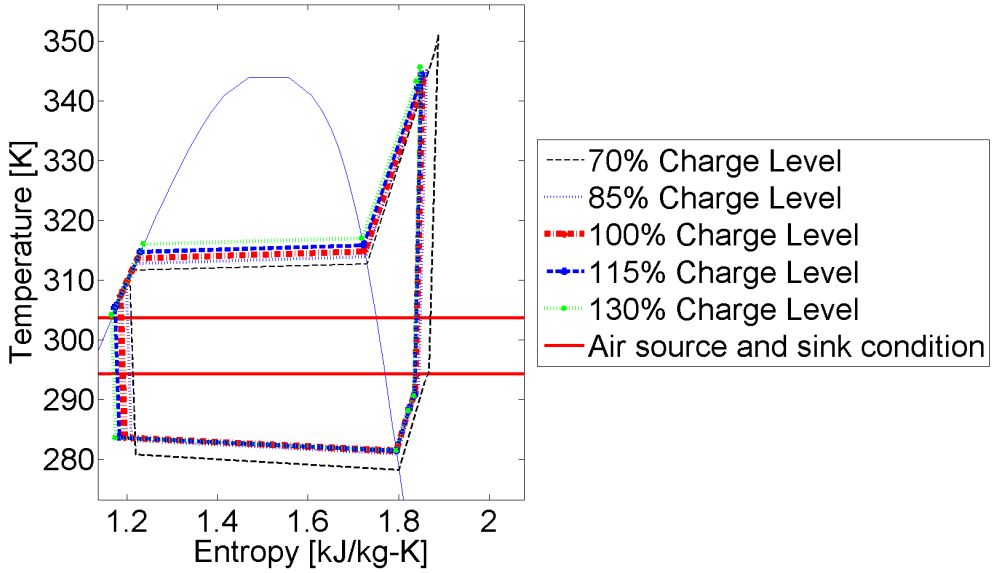

Figure 8.10. T-s diagram of system VII at different charge levels.

pressure and subcooling. However, when the charge level decreases from 85% to 70%, the TXV becomes fully opened and the evaporator outlet superheat cannot be controlled. The system reacts to the change of charge level as if it is an FXO system. Its compressor discharge temperature increases and its evaporating temperature decreases with the decrease of charge level.

Figures 8.11 and 8.12 show the change of performance with different charge levels for system VII.

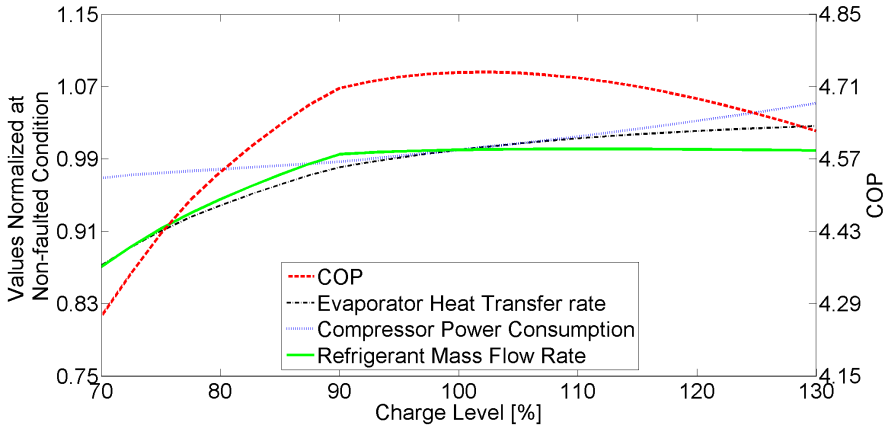

Figure 8.11. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system VII at different charge levels.

Figures 8.11 shows that the increase of refrigerant charge level from 70% to 130% increases the compressor power consumption, evaporator heat transfer rate and refrigerant mass flow rate. At charge level 90%, there is a rapid change of slope of refrigerant mass flow rate with respect to charge level because the TXV is fully opened at charge level lower than or equal to 90%. A peak COP at 4.74 can be seen in Figure 8.11 when the charge level is 102.5%. A maximum COSP can also be seen

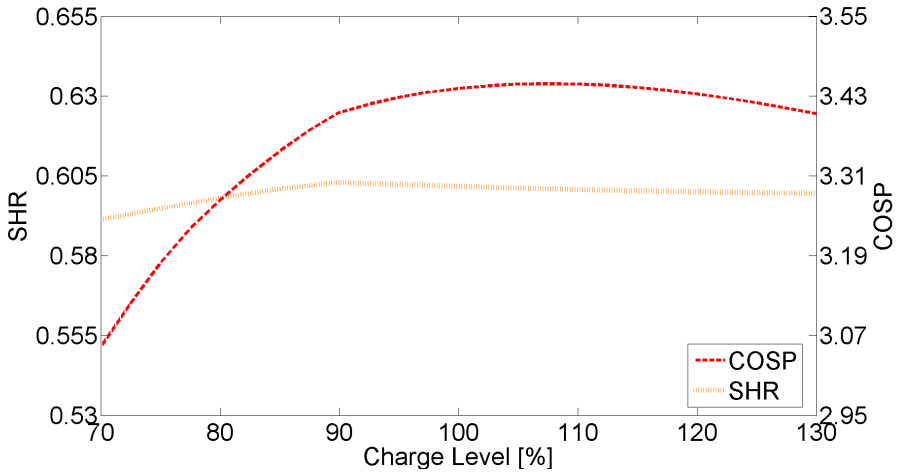

Figure 8.12. Change of SHR and COSP of system VII at different charge levels.

in Figure 8.12 at the charge level 107.5%. The SHR, as shown in Figure 8.12, changes by 1.96% when the charge level increases by 60% and the change is insignificant.

8.2 Evaporator Fouling

8.2.1 FXO System

System IV was simulated with evaporator fouling level ranging from 0% to 40% and the P-h and T-s diagrams are plotted in Figures 8.13 and 8.14.

The most significant change observed in Figures 8.13 and 8.14 with increasing fouling level is decreasing evaporating pressure.

Figures 8.15 and 8.16 show the change of performance with increasing evaporator fouling level.

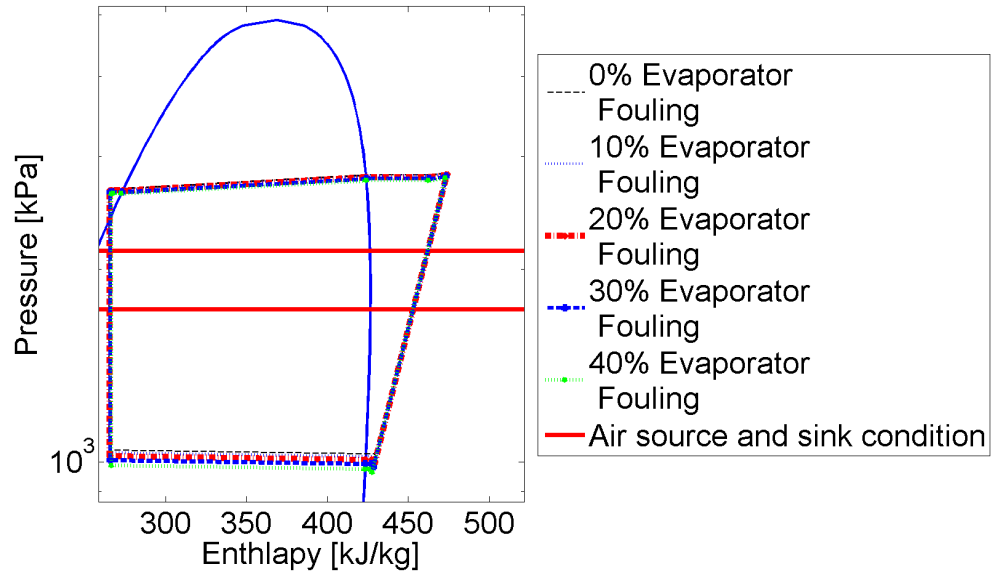

Figure 8.13. P-h diagram of system IV at different evaporator fouling levels.

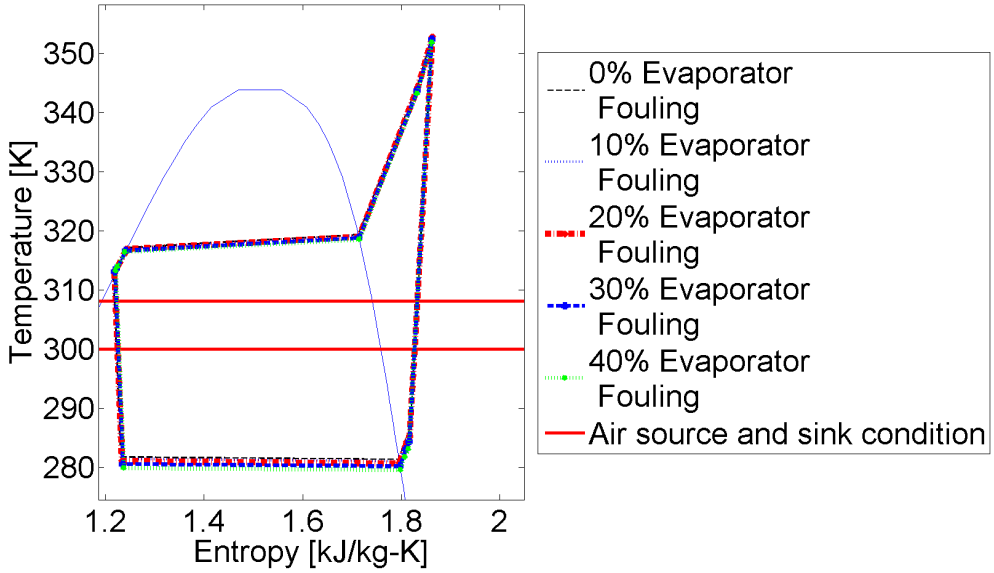

Figure 8.14. T-s diagram of system IV at different evaporator fouling levels.

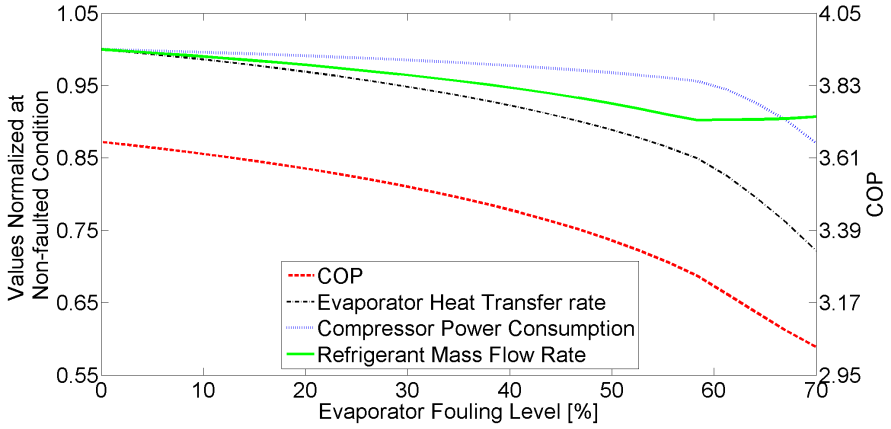

Figure 8.15. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IV at different evaporator fouling levels.

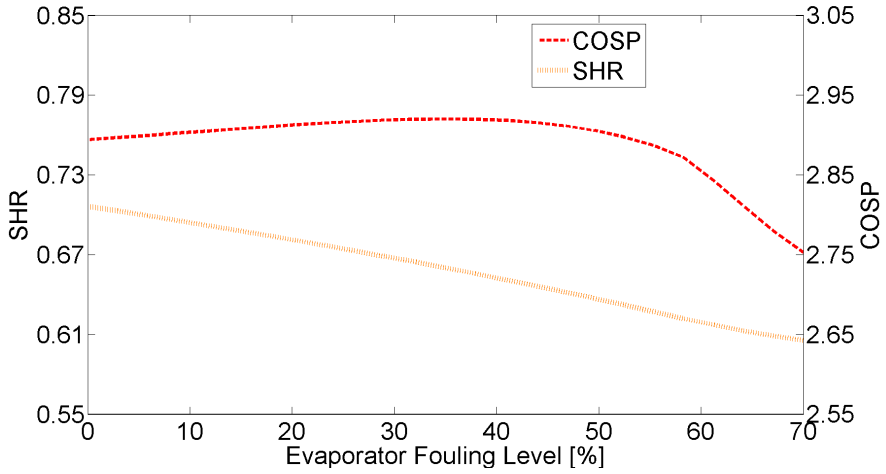

Figure 8.16. Change of SHR and COSP of system IV at different evaporator fouling levels.

Figure 8.15 shows that the refrigerant mass flow rate, evaporator heat transfer rate, compressor power consumption and COP fall with increasing evaporator fouling level. However, when the fouling level increases to 60%, the refrigerant mass flow rate starts to increase with increasing fouling level. This is caused by two-phase

refrigerant entering the compressor and refrigerant density at compressor suction increasing rapidly with increasing evaporator fouling level.

Figure 8.16 shows that SHR decreases with increasing evaporator fouling level but COSP increases slightly with increasing evaporator fouling level when the fouling level is less than 35%. When evaporator fouling level increases, evaporator fan power consumption decreases. This decreases the fan heat added to the air stream and the power consumption, leading to an increasing COSP at low evaporator fouling level.

8.2.2 FXO System with an Accumulator

System IX was simulated with evaporator fouling level from 0% to 80% and the P-h and T-s diagrams of the operation are shown in Figures 8.17 and 8.18.

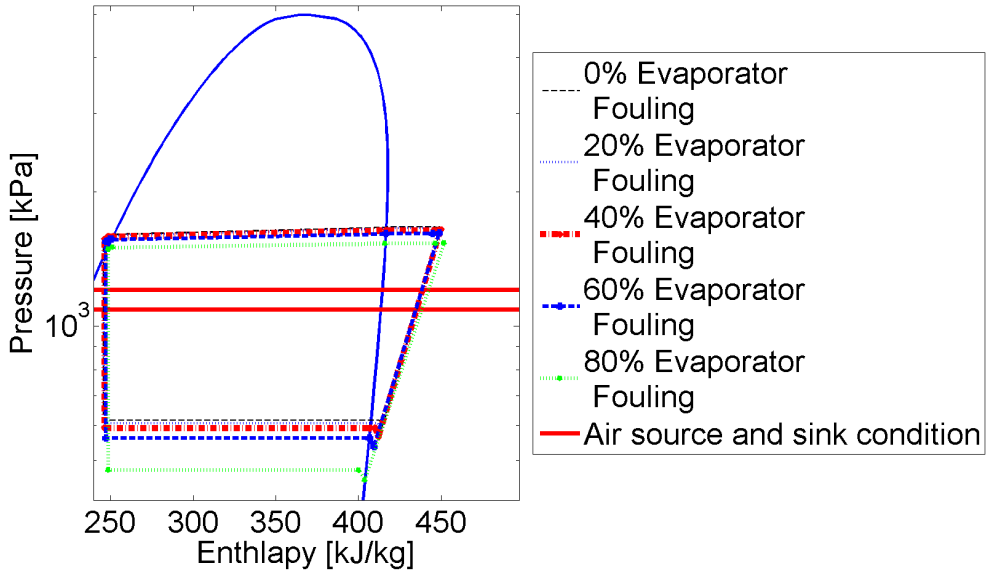

Figure 8.17. P-h diagram of system IX at different evaporator fouling levels.

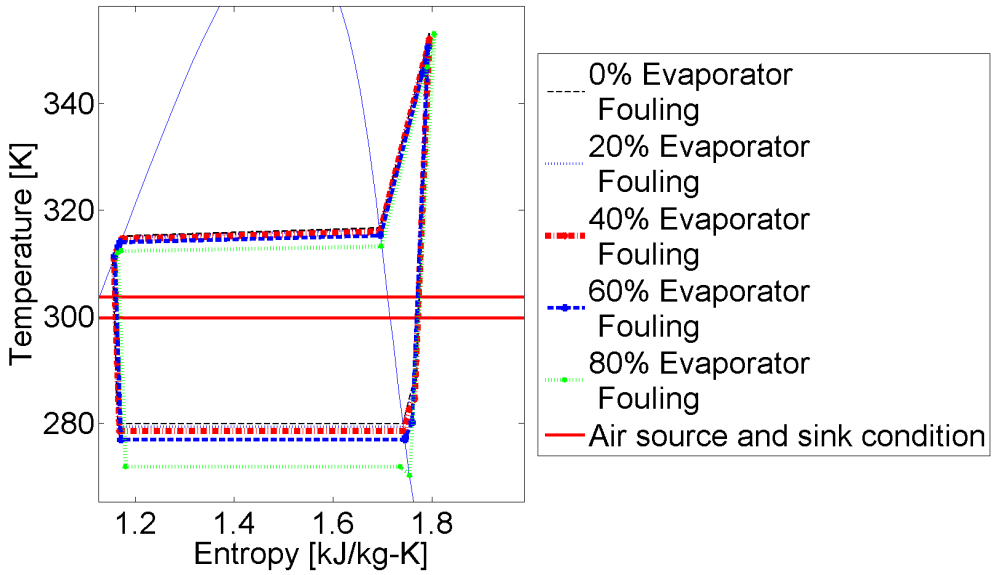

Figure 8.18. T-s diagram of system IX at different evaporator fouling levels.

In Figures 8.17 and 8.18, when evaporator fouling level increases from 0% to 60%, the change of the P-h and T-s diagrams are similar to the ones in Figures 8.13 and 8.14 because both systems IV and IX are FXO systems. However, as the compressor suction superheat of system IX drops to zero with increasing evaporator fouling level, its accumulator starts to hold liquid refrigerant and the trend changes. Although an increasing evaporator fouling level from this point onwards continues to reduce the evaporating pressure and condensing pressure, the compressor discharge temperature does not decrease with the increasing fouling level anymore. In Figure 8.18, compressor discharge temperature decreases for increasing evaporator foouling level from 0% to 60%. As the evaporator fouling level increases from 60% to 80%, the compressor discharge temperature increases. Since the compressor suction

superheat of system IX with 80% evaporator fouling level is zero, this shows that the accumulation of liquid refrigerant in the accumulator is the main reason for the increase of compressor discharge temperature with increasing fouling level.

The change of performance of system IX with increasing evaporator fouling level is shown in Figures 8.19 and 8.20.

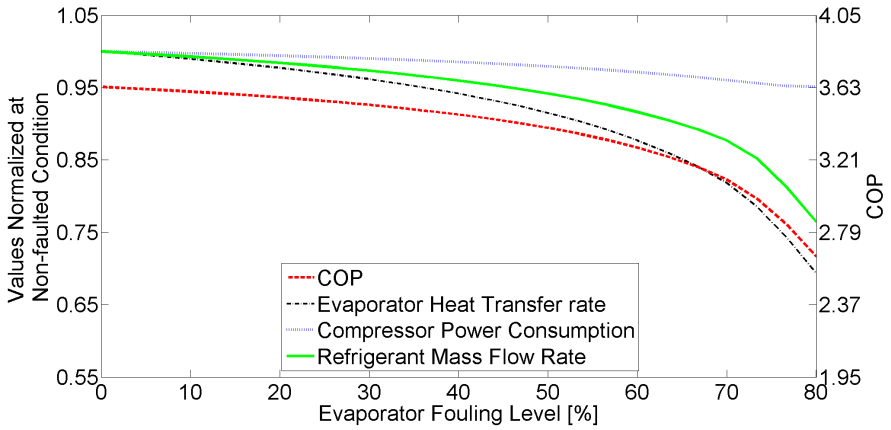

Figure 8.19. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IX at different evaporator fouling levels.

Figures 8.19 and 8.20 show that the change of performance of system IX with evaporator fouling level is similar to that of system IV in in Figures 8.15 and 8.16. The evaporator heat transfer rate, the compressor power consumption, the COP, the refrigerant mass flow rate and the SHR decrease with increasing fouling level and there is a peak of COSP in Figure 8.20 due to the decrease of evaporator fan power consumption with increasing evaporator fouling level. Although the accumulator starts to hold liquid refrigerant for evaporator fouling level above 73%, Figures 8.19

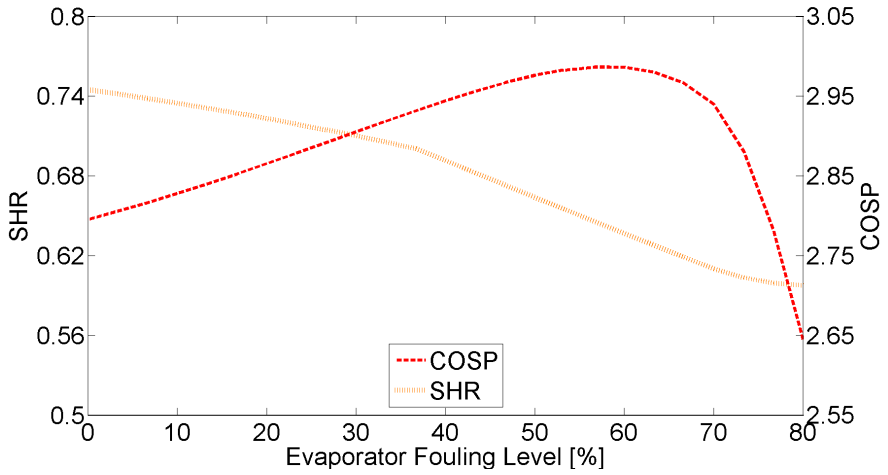

Figure 8.20. Change of SHR and COSP of system IX at different evaporator fouling levels.

and 8.20 do not show that the accumulator has any special effect on the change of system overall performance with evaporator fouling.

8.2.3 TXV System

System VII was simulated with evaporator fouling level from 0% to 80% and the P-h and T-s diagrams are shown in Figures 8.21 and 8.22.

Figures 8.21 and 8.22 show that the P-h and T-s diagrams expand towards the bottom right-handed corner as the evaporator fouling level increases. With the TXV controlling the evaporator outlet superheat, increasing evaporator fouling level increases compressor discharge superheat and diminishes the condenser outlet subcooling, the condensing pressure and evaporating pressure.

The change of the performance of system VII with increasing evaporator fouling level is plotted in Figures 8.23 and 8.24.

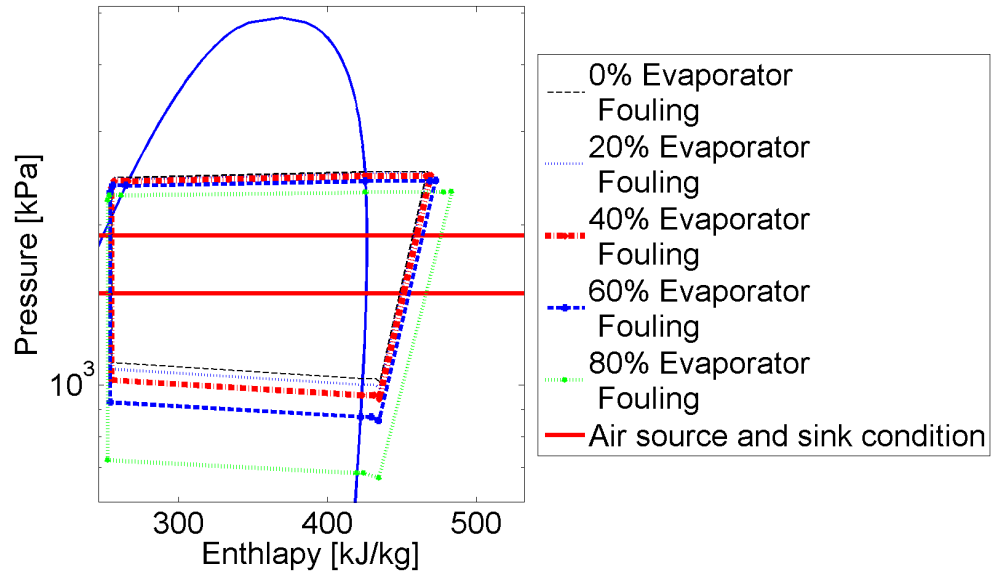

Figure 8.21. P-h diagram of system VII at different evaporator fouling levels.

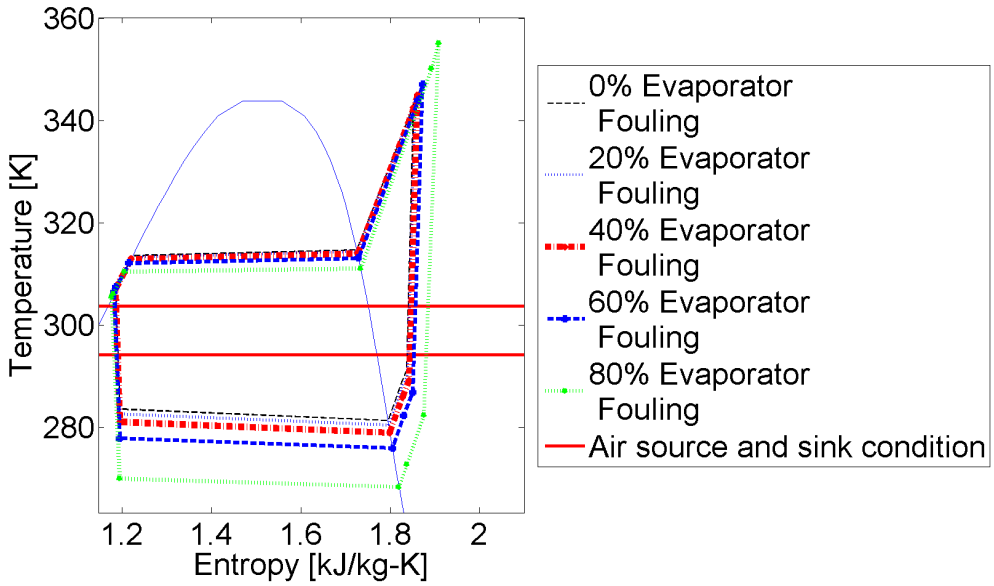

Figure 8.22. T-s diagram of system VII at different evaporator fouling levels.

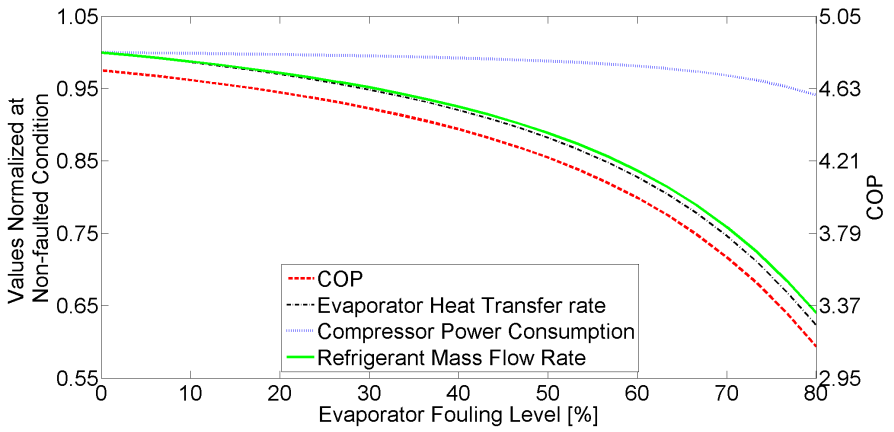

Figure 8.23. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system VII at different evaporator fouling levels.

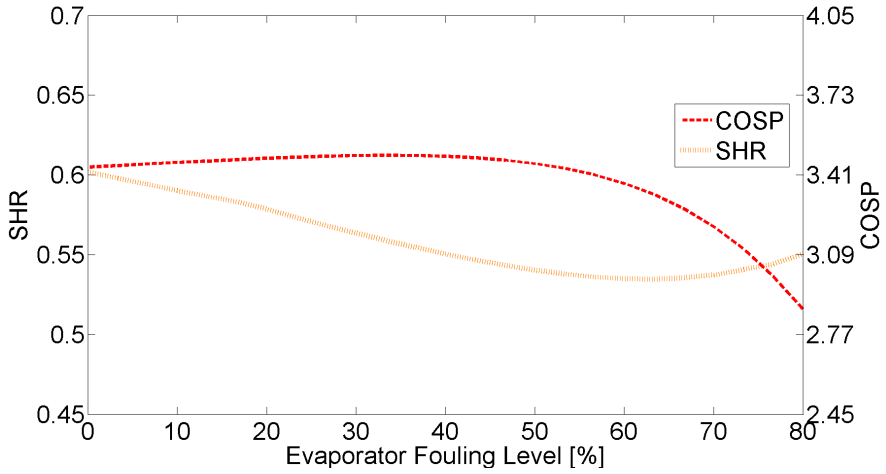

Figure 8.24. Change of SHR and COSP of system VII at different evaporator fouling levels.

Figures 8.23 and 8.24 show that evaporator fouling causes a significant drop of COP, evaporator heat transfer rate, compressor power consumption and SHR with a 34% drop of COP. Although Figure 8.24 shows that SHR does not decrease when the fault level is above 70%, the estimated evaporating temperature is below the freezing

point of water and the SHR estimation is invalid. As evaporator fouling reduces the evaporator fan power consumption, COSP in Figure 8.24 increases with the fouling level and peaks at 3.49 when the fouling level is 33%. COSP drops by 18% when the fouling level continues to increase from 33% to 80%.

8.3 Condenser fouling

8.3.1 FXO System

System IV was simulated with condenser fouling level from 0% to 40% and the resultant P-h and T-s diagrams are shown in Figures 8.25 and 8.26.

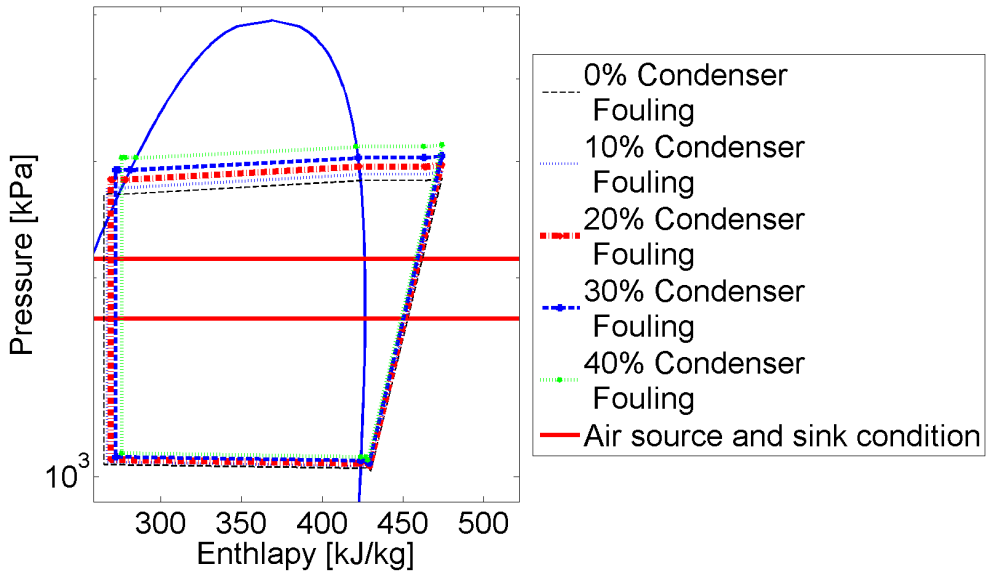

Figure 8.25. P-h diagram of system IV at different condenser fouling levels.

Figures 8.25 and 8.26 show that as the condenser fouling level increases, the condensing pressure increases while other variables change insignificantly.

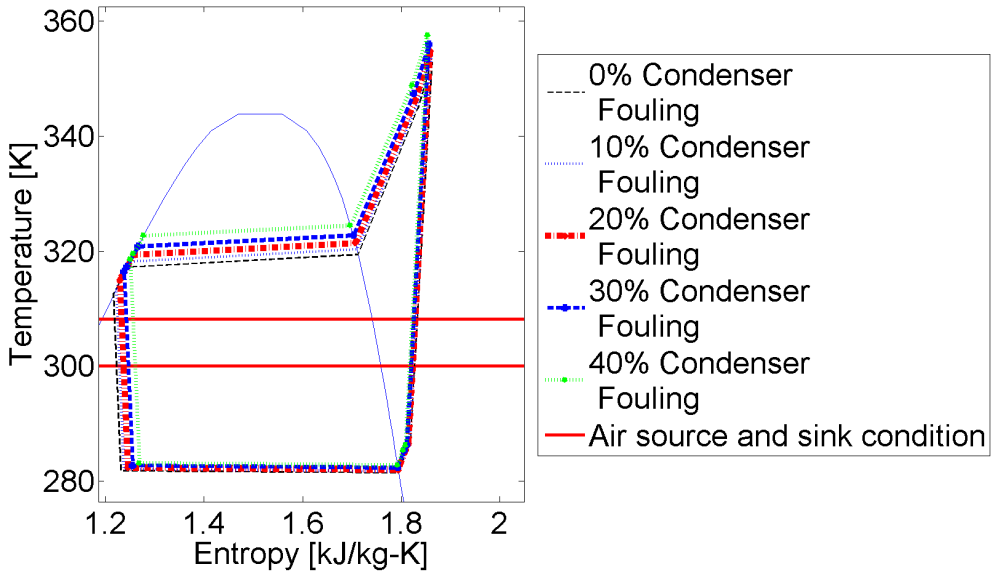

Figure 8.26. T-s diagram of system IV at different condenser fouling levels.

Figure 8.27 shows the change of refrigerant mass flow rate, evaporator heat transfer rate, compressor power consumption and COP with respect to condenser fouling level.

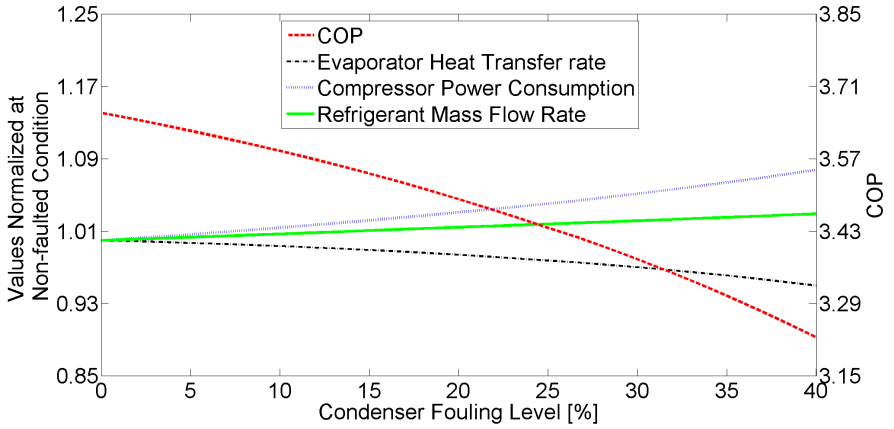

Figure 8.27. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IV at different condenser fouling levels.

Figure 8.27 shows that because of the increase of condensing pressure with condenser fouling level, the compressor power consumption increases by 7.8% when the condenser fouling level reaches 40%. This change is more significant than the drop of evaporator heat transfer rate by 4.9% and is the primary reason of the drop of COP by 11.9%.

Figure 8.28 shows how SHR and COSP change with increasing condenser fouling level.

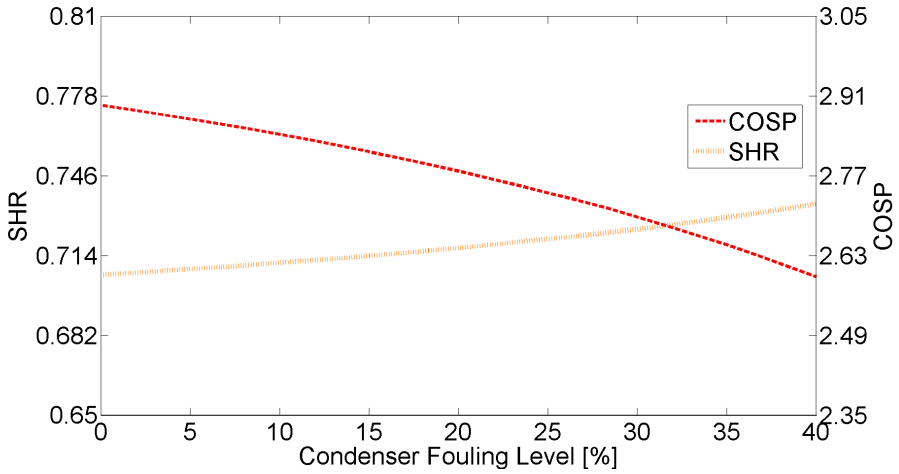

Figure 8.28. Change of SHR and COSP of system IV at different condenser fouling levels.

SHR in Figure 8.28 increases by 4% as a consequence of a 50kPa increase in the evaporator inlet pressure when condenser fouling increases by 40%, which is insignficant compared to the effects of evaporator fouling on SHR in Figure 8.24. The COSP in Figure 8.28 follows the the change of COP in Figure 8.27 that COSP drops by 10.4% when the fouling level increases from 0% to 40%.

8.3.2 FXO System with an Accumulator

The P-h and T-s diagrams plotted from the simulation result of system IX under 0% to 40% of condenser fouling are shown in Figures 8.30 and 8.30.

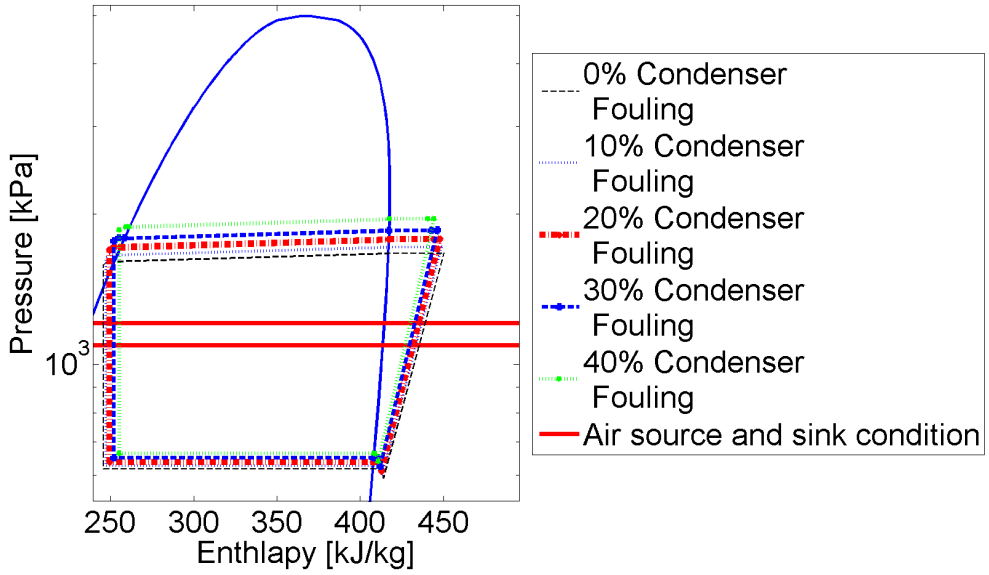

Figure 8.29. P-h diagram of system IX at different condenser fouling levels.

Since the compressor suction superheat is not zero in all simulation result in Figures 8.29 and 8.30, the accumulator in system IX has no effect on the system performance and the shift of the P-h and T-s diagrams with increasing condenser fouling level in Figures 8.29 and 8.30 is the same as the ones of an FXO system without an accumulator in Figures 8.25 and 8.26.

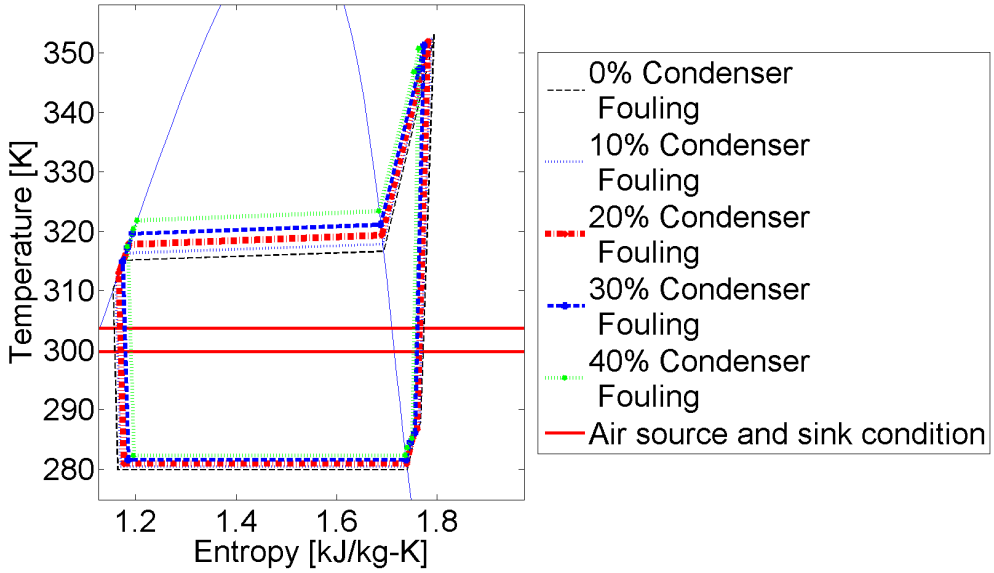

Figure 8.30. T-s diagram of system IX at different condenser fouling levels.

The change of evaporator heat transfer rate, compressor power consumption, COP, COSP and SHR in the simulation outputs of the system IX under condenser fouling are plotted in Figures 8.31 and 8.32.

The changes of the variables with condenser fouling level in Figures 8.31 and 8.32 are similar to that in Figures 8.27 and 8.28 because the accumulator does not hold liquid refrigerant under all simulated condenser fouling levels and system IX responds to condenser fouling as if an FXO system without an accumulator. The results show that an increase of condenser fouling level from 0% to 40% will cause a drop of COSP by 9.1% and an increase of SHR by 5.62%.

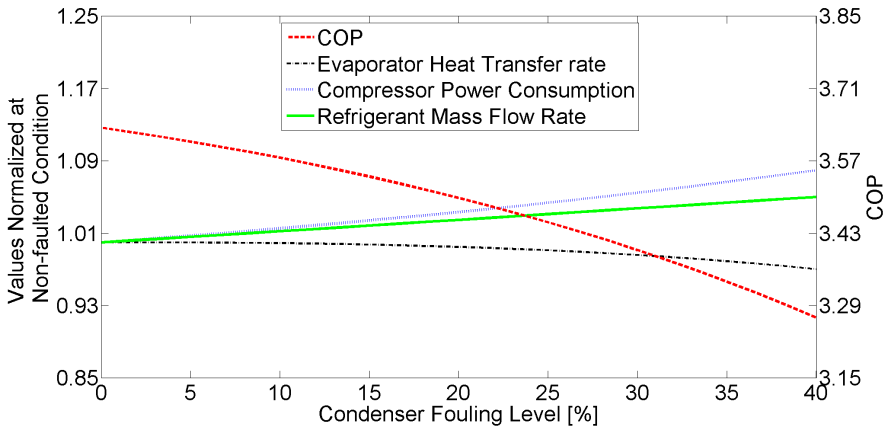

Figure 8.31. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IX at different condenser fouling levels.

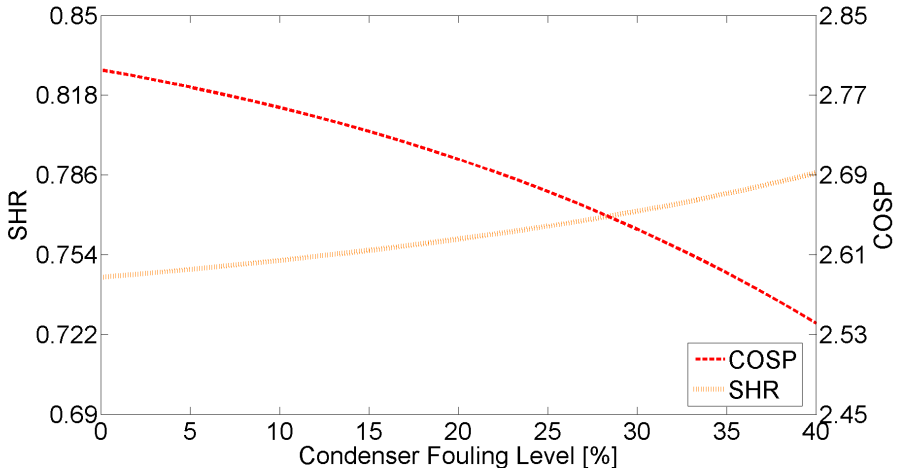

Figure 8.32. Change of SHR and COSP of system IX at different condenser fouling levels.

8.3.3 TXV System

System VII was simulated with evaporator fouling level from 0% to 40% and the P-h and T-s diagrams are plotted as Figures 8.33 and 8.34.

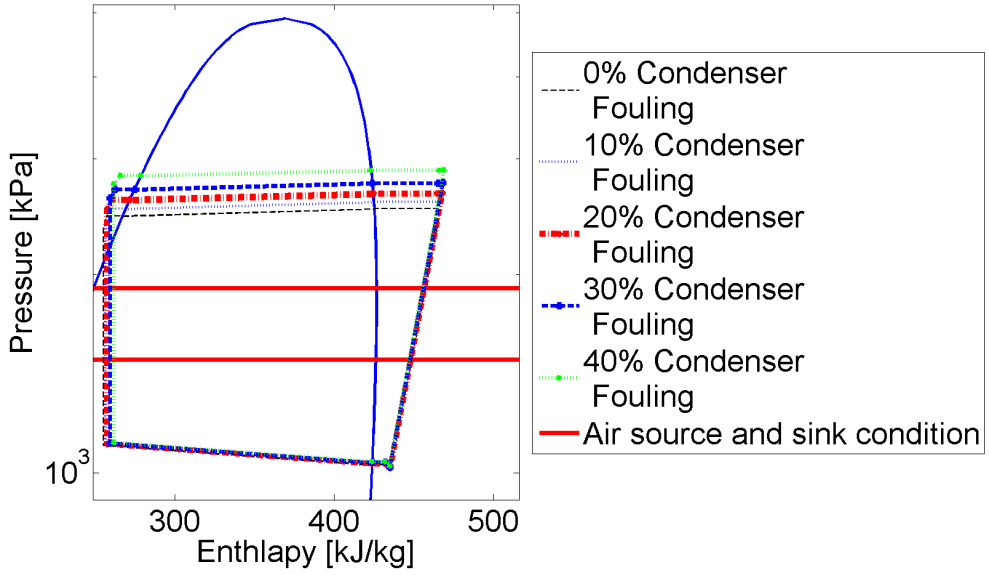

Figure 8.33. P-h diagram of system VII at different condenser fouling levels.

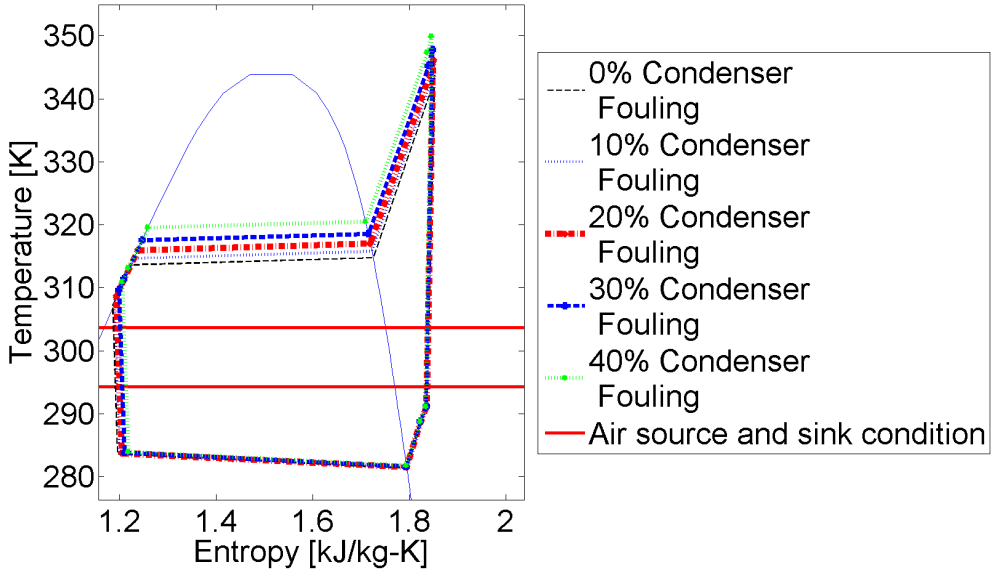

Figure 8.34. T-s diagram of system VII at different condenser fouling levels.

Figures 8.33 and 8.34 show that condenser fouling on TXV systems increases the condensing pressure but has negligible effect at the evaporating pressure, and both P-h and T-s diagrams expand upwards with increasing condenser fouling levels.

Figure 8.35 and 8.36 show the change of performance of system VII with condenser fouling levels.

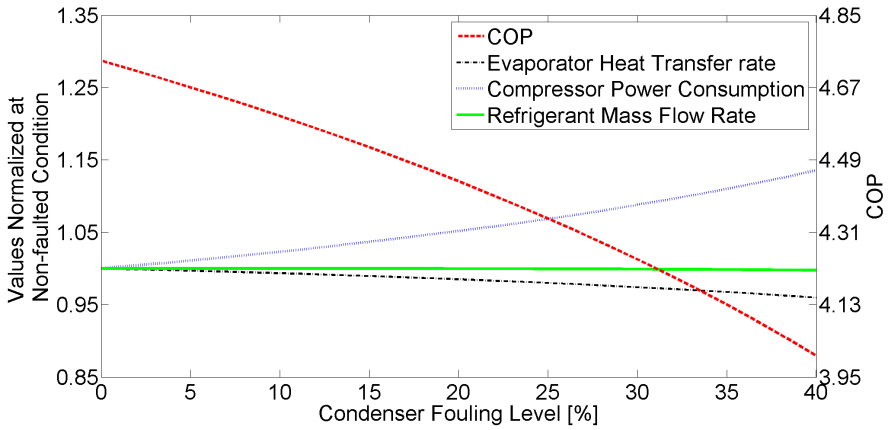

Figure 8.35. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system VII at different condenser fouling levels.

Figure 8.35 shows that condenser fouling has a negligible effect to the refrigerant mass flow rate, increases the compressor power consumption and reduces the evaporator heat transfer rate. This reduces the COP from 4.74 to 4.00 with 40% of condenser fouling. Despite the decrease of condenser fan power consumption with condenser fouling, it cannot compensate for the effect of decreasing COP on COSP, resulting in the decrease of COSP from 3.44 to 3.01 in Figure 8.36. Figure 8.36 also shows that SHR of system VII increases by 1.18% only when the condenser fouling

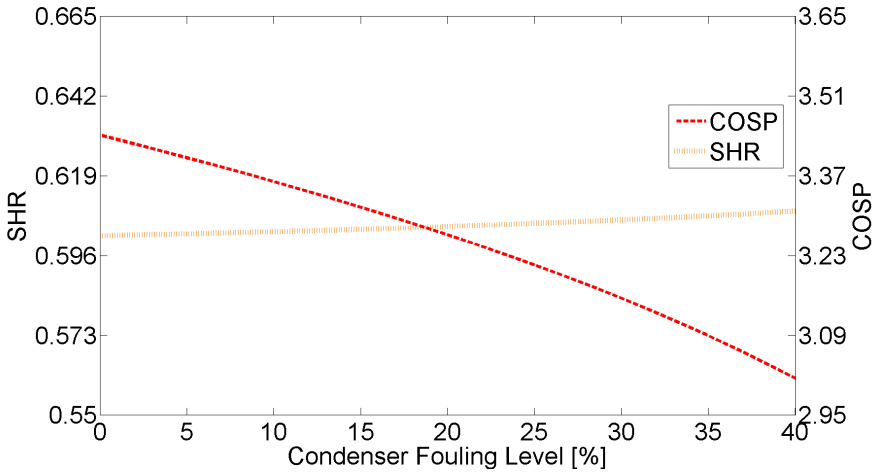

Figure 8.36. Change of SHR and COSP of system VII at different condenser fouling levels.

level of the system increases by 40%, showing that condenser fouling has a negligible effect to SHR.

8.4 Compressor Flow Fault

8.4.1 FXO System

System IV was simulated with increasing compressor flow fault level from 0% to 30% and the P-h and T-s diagrams are plotted in Figures 8.37 and 8.38.

P-h and T-s diagrams shrink towards the two-phase dome in Figures 8.37 and 8.38 as the compressor flow fault level increases. The condensing pressure, compressor discharge temperature and superheat decrease and the evaporating pressure increases when the fault level rises. The increase of the fault level causes a drop in refrigerant

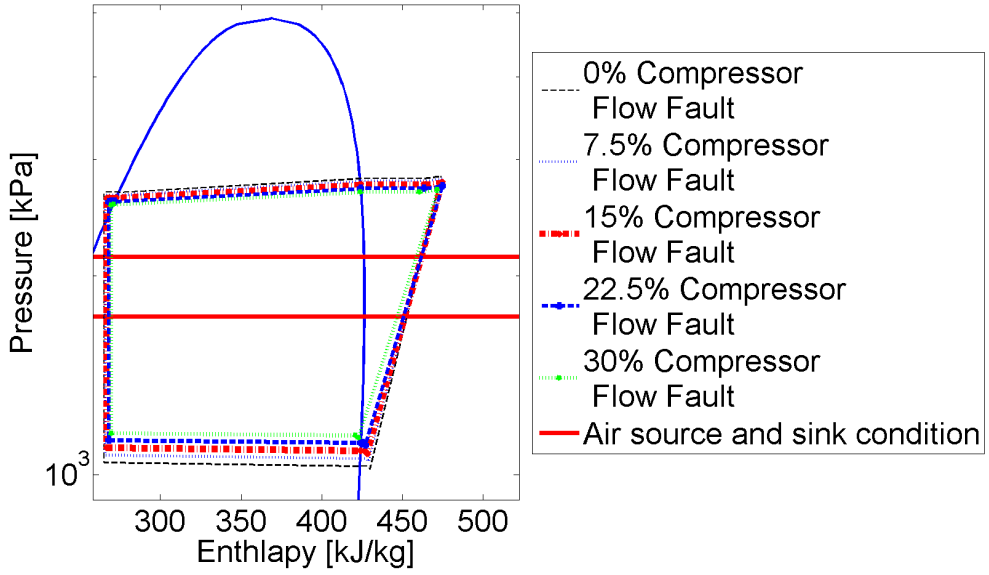

Figure 8.37. P-h diagram of system IV with increasing compressor flow fault level.

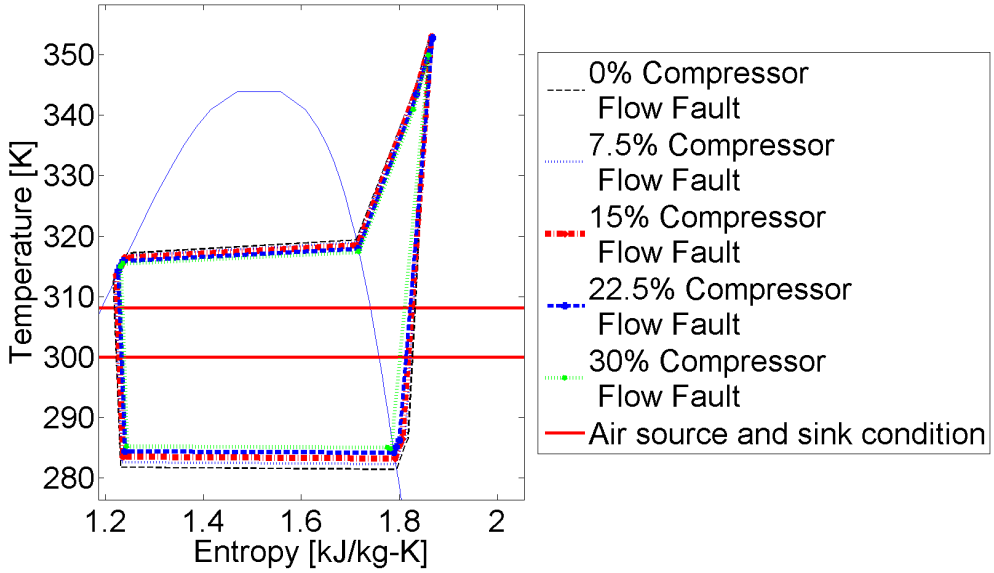

Figure 8.38. T-s diagram of system IV with increasing compressor flow fault level.

mass flow rate, the pressure ratio of the system, the evaporator heat transfer rate and hence the superheat.

The decrease of refrigerant mass flow rate and evaporator heat transfer rate, together with the change of COP and power consumption, with increasing compressor flow fault level can be observed in Figure 8.39.

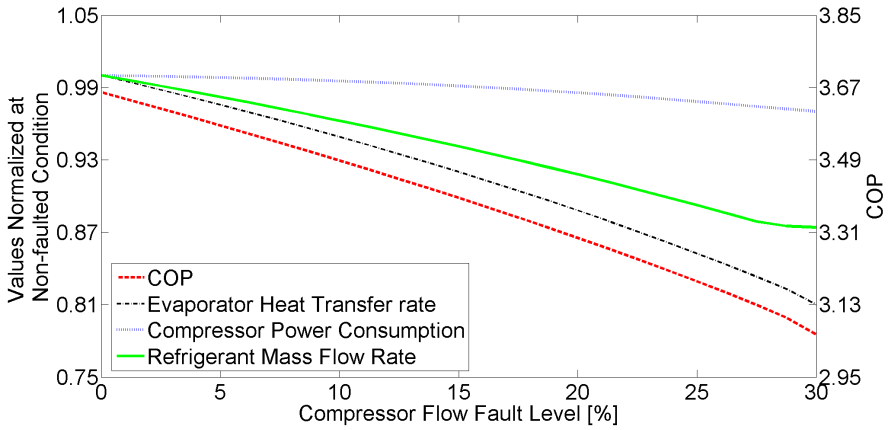

Figure 8.39. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IV at increasing compressor flow fault level.

When the compressor flow fault level increases, the evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP decrease as shown in Figure 8.39.

The change of SHR and COSP with increasing compressor flow fault level is plotted in Figure 8.40.

With the increase of evaporating pressure and temperature, the surface temperature of the evaporator increases and the SHR in Figure 8.40 increases by 18% linearly as the compressor flow fault level drops by 30%. Since there is no

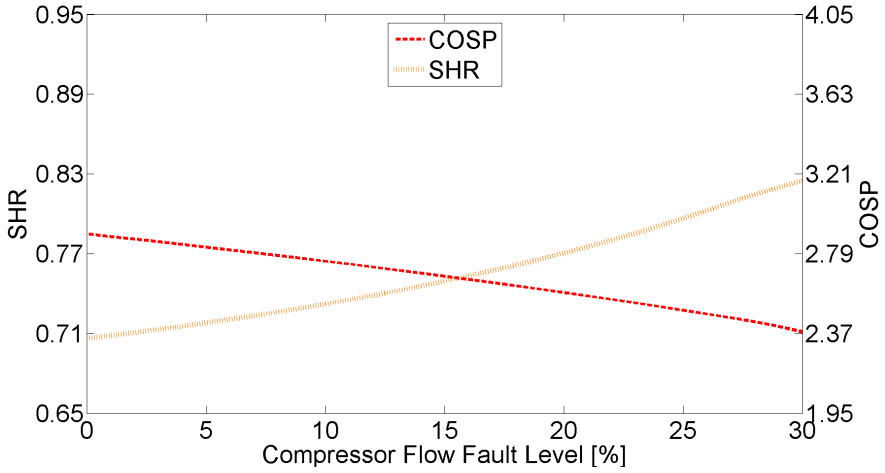

Figure 8.40. Change of SHR and COSP of system IX at increasing compressor flow fault level.

change in fan power consumption with the compressor fault, COSP in Figure 8.40 follows the decrease of COP in Figure 8.39, and it drops by 17.8% in Figure 8.40.

8.4.2 FXO System with an Accumulator

System IX was simulated with increasing compressor flow fault level from 0% to 65% and the P-h and T-s diagrams from the simulation outputs are plotted in Figures 8.41 and 8.42.

Figures 8.41 and 8.42 show that increasing compressor flow fault level reduces condenser subcooling, evaporator outlet superheat, the condensing pressure and the pressure difference across the compressor. However, unlike the other FXO system in Figures 8.37 and 8.38, the compressor discharge superheat in Figure 8.42 increases. This shows that in some FXO systems, it is possible that the effect of decreasing refrigerant mass flow rate to compressor discharge superheat outweighs that of

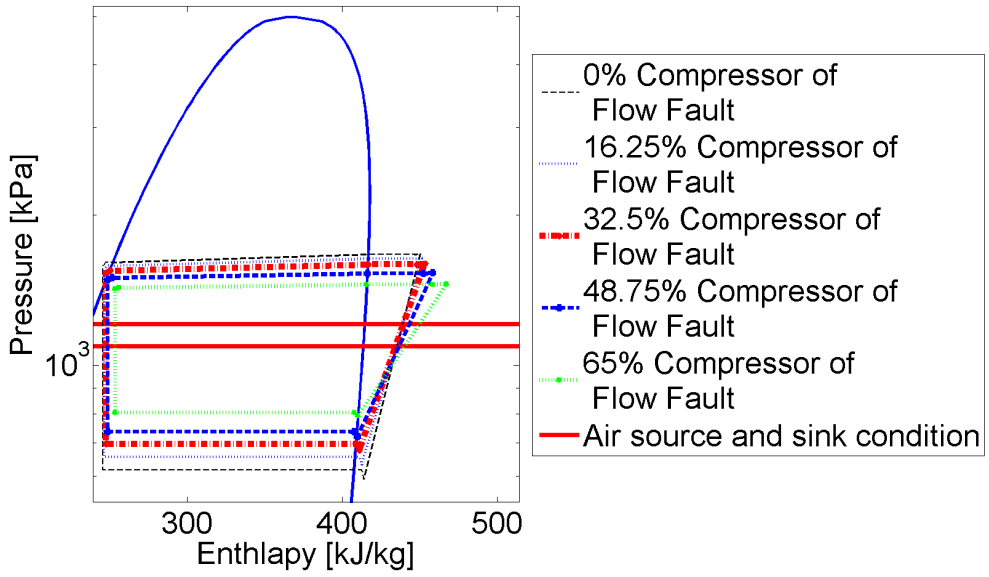

Figure 8.41. P-h diagram of system IX at increasing compressor flow fault level.

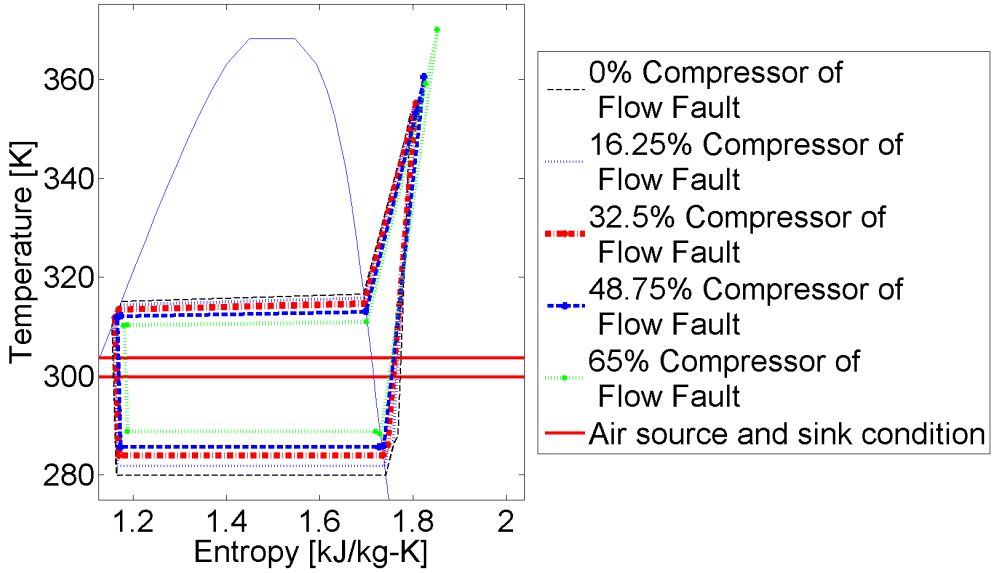

Figure 8.42. T-s diagram of system IX at increasing compressor flow fault level.

decreasing evaporator outlet superheat, and increasing compressor flow fault level can increase the compressor discharge superheat.

The change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate, COP, SHR and COSP with increasing compressor flow fault level is shown in Figures 8.43 and 8.44.

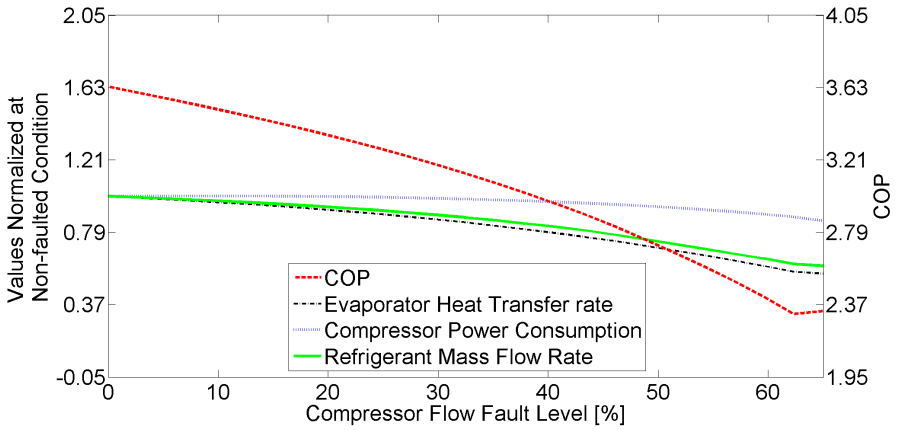

Figure 8.43. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IX at increasing compressor flow fault level.

Since system IX is an FXO system like system IV, Figures 8.43 and 8.44 show that the changes of the system variables with increasing compressor flow fault level are the same as Figures 8.39 and 8.40 when the fault level is below 60% and the compressor suction superheat is positive. However, when the fault level crosses 60%, the accumulator of system IX starts to accumulate liquid refrigerant and the effective charge level inside the system drops. Before the reduction of the effective charge level, as the increase of compressor flow fault level can be considered as the reduction of the size of the system, increasing the fault level can be considered as reducing the system

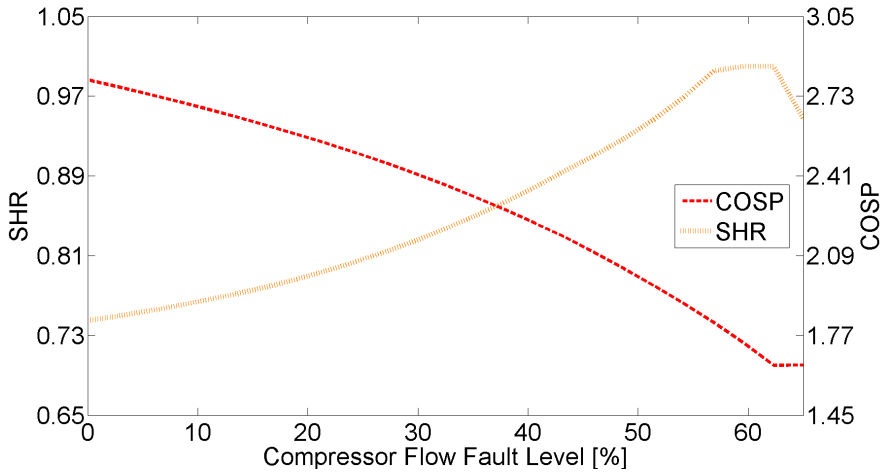

Figure 8.44. Change of SHR and COSP of system IX at increasing compressor flow fault level.

charge level for optimal COP. With a constant charge level and increasing compressor flow fault level, the charge level for optimal COP becomes lower and lower than the system charge level. However, when the accumulator reduces the effective charge level of the system, the difference between the charge levels shrink. This results in the increase of COP and COSP as the compressor flow fault level increases from 62.5% to 65% in Figures 8.43 and 8.44.

8.4.3 TXV System

System VII is simulated with increasing compressor flow fault level for 65% and the resultant P-h and T-s diagrams are attached as Figures 8.45 and 8.46.

Similar to Figure 8.37 and 8.38, the P-h and T-s diagrams in Figures 8.45 and 8.46 shrink towards the two-phase dome as the compressor flow fault level increases. However, as the TXV controls the superheat at the evaporator outlet, the compressor

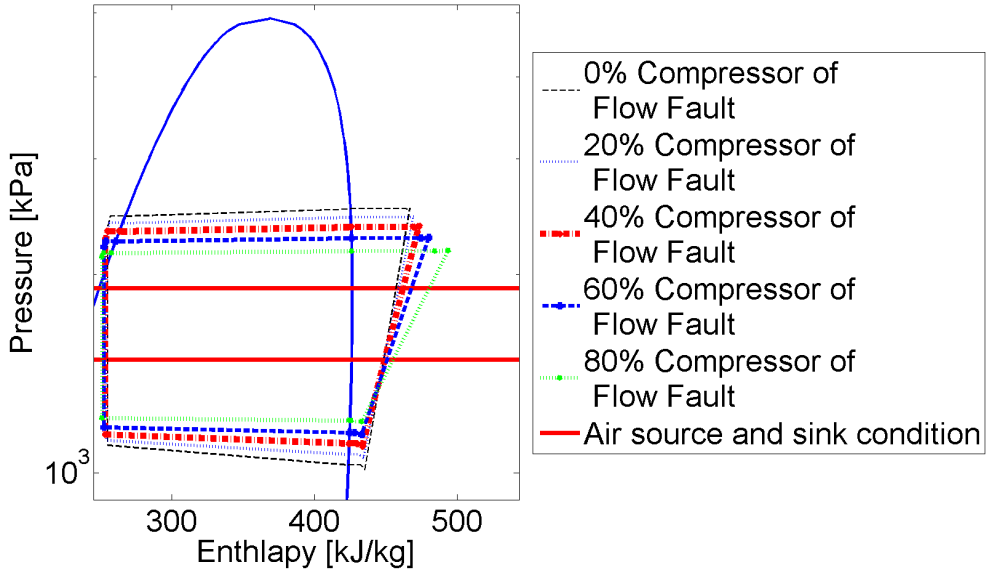

Figure 8.45. P-h diagram of system VII at different compressor flow fault levels.

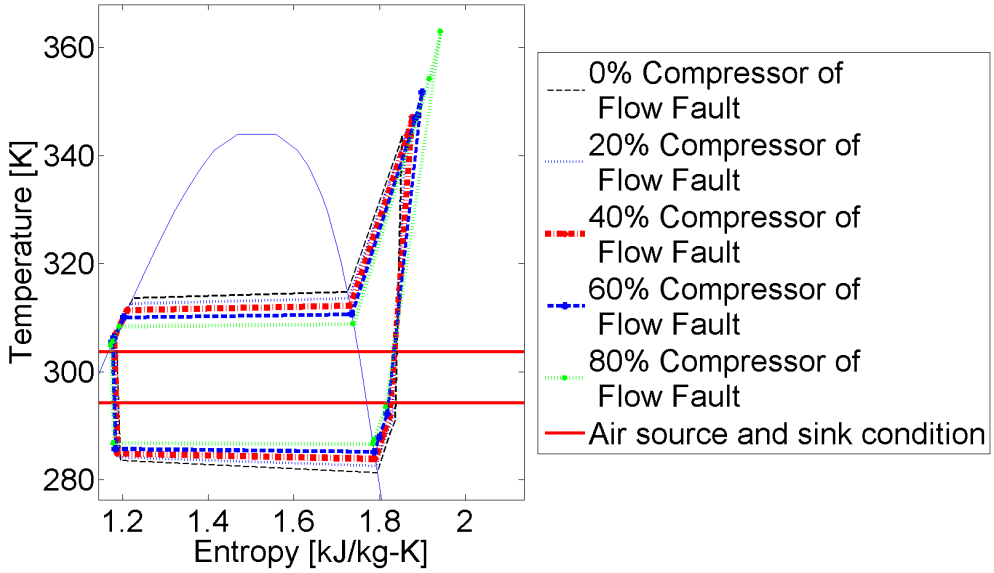

Figure 8.46. T-s diagram of system VII at different compressor flow fault levels.

discharge superheat increases with increasing fault level and the left boundaries of the P-h and T-s diagrams move to the left as the fault level increases.

Figure 8.47 shows the change of the dimensionless evaporator heat transfer rate, the compressor power consumption, the refrigerant mass flow rate and the COP of system VII with increasing flow fault level and Figure 8.48 shows the change of COSP and SHR of system VII with increasing flow fault level.

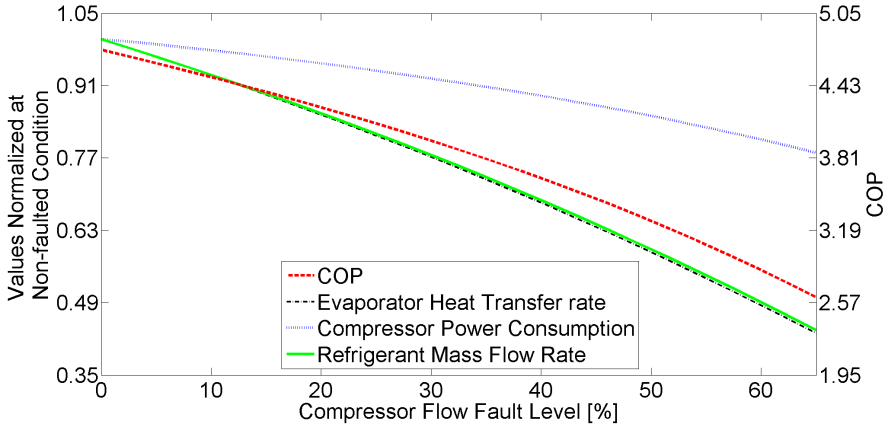

Figure 8.47. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system VII at different compressor flow fault levels.

All variables in Figures 8.47 and 8.48 except SHR decrease linearly. The drop of COP from 4.74 to 2.62 is dominated by the 57% decrease of evaporator heat transfer rate as the compressor flow fault level increases by 65%. This is also the main cause of the drop of COSP from 3.44 to 1.66. The only variable which increases with increasing compressor flow fault level is SHR which is induced by the increasing evaporating temperature with the increase of the fault level.

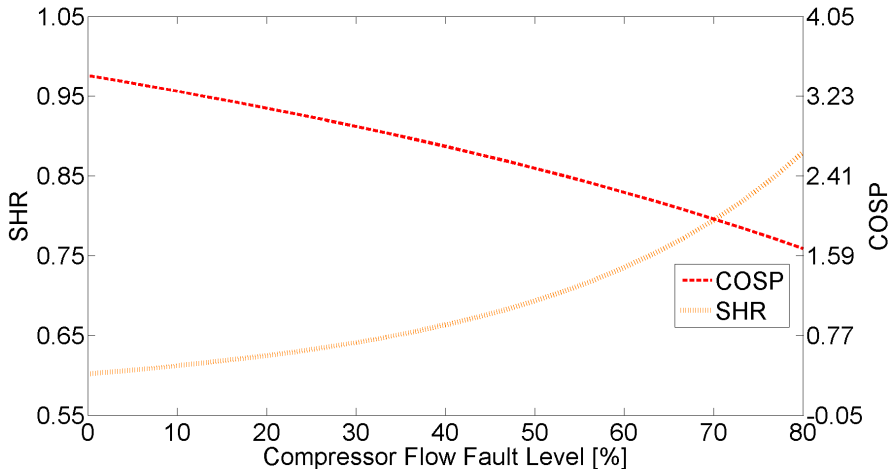

Figure 8.48. Change of SHR and COSP of system VII at different compressor flow fault levels.

8.5 Liquid Line Restriction

8.5.1 FXO System

Figures 8.49 and 8.50 show the change of P-h and T-s diagrams of system IV as the liquid line restriction level increases from 0% to 40%.

The P-h and T-s diagrams in Figures 8.49 and 8.49 expand downwards as the liquid line restriction level increases. The evaporating pressure decreases more significantly than the condensing pressure. The compressor discharge temperature also increases.

Figures 8.51 and 8.52 shows the change of performance as liquid line restriction level increases.

Figures 8.51 and 8.52 show that as the liquid line becomes more restricted, COSP, COP, SHR, evaporator heat transfer rate, compressor power consumption

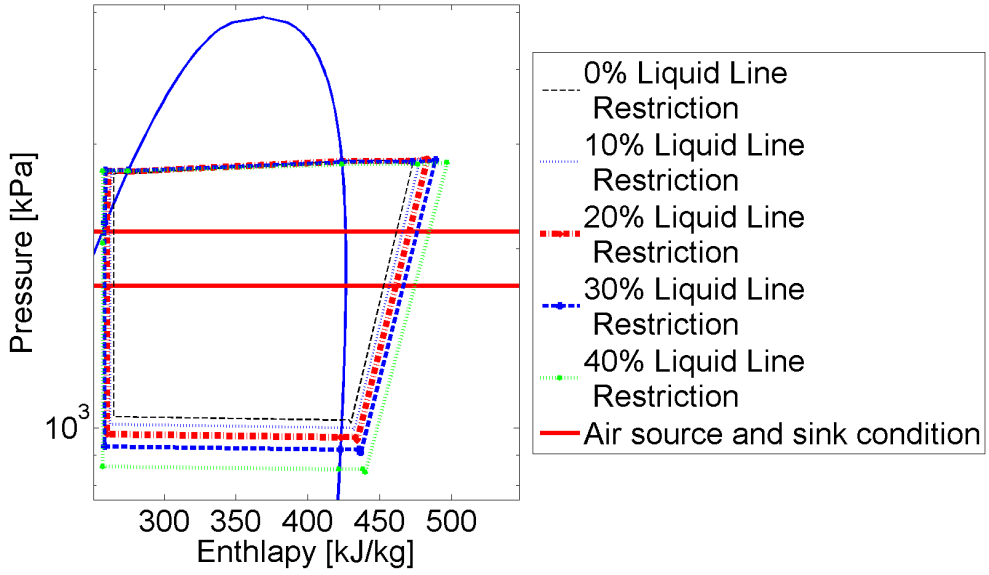

Figure 8.49. P-h diagram of system IV at different liquid line restriction levels.

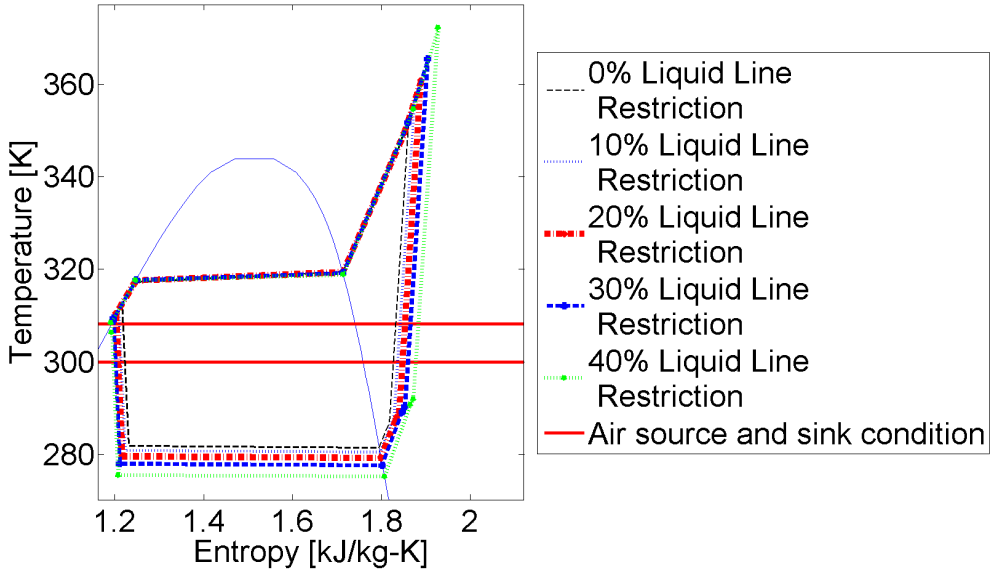

Figure 8.50. T-s diagram of system IV at different liquid line restriction levels.

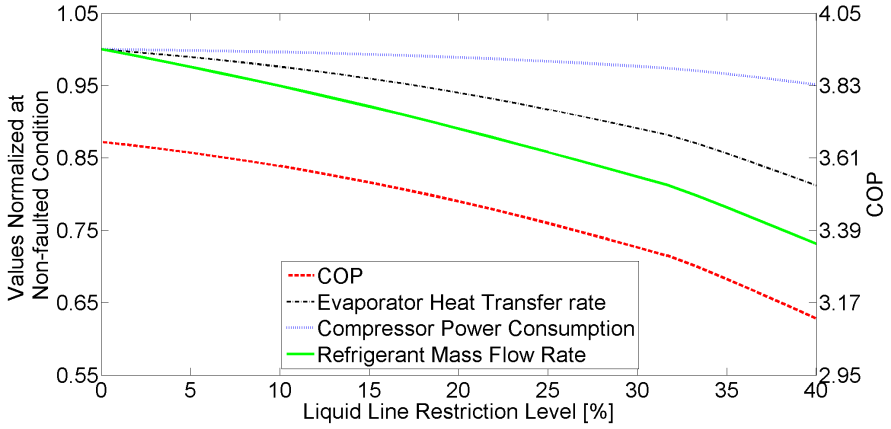

Figure 8.51. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IV at different liquid line restriction levels.

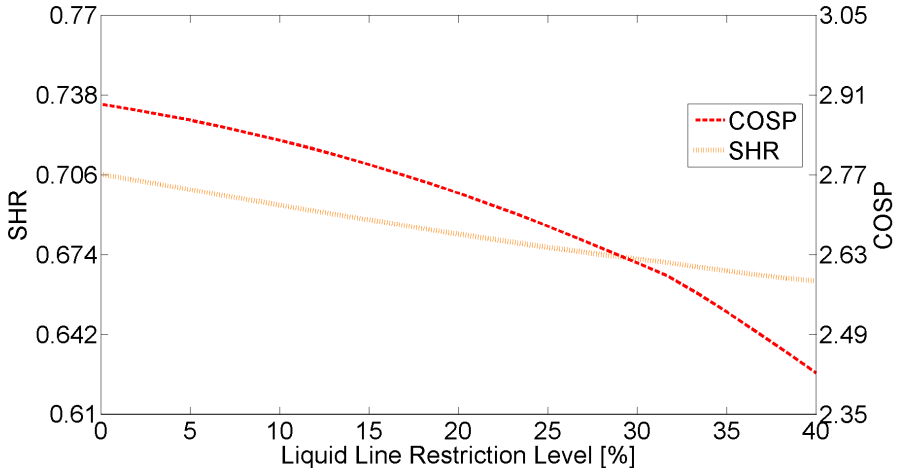

Figure 8.52. Change of SHR and COSP of system IV at different liquid line restriction levels.

and refrigerant mass flow rate decrease significantly, with COP dropped by 14.7% and COSP dropped by 16.3%.

8.5.2 FXO System with an Accumulator

System IX was simulated with a range of liquid line restriction levels from 0% to 40% and the corresponding P-h and T-s diagrams are plotted in Figures 8.53 and 8.54.

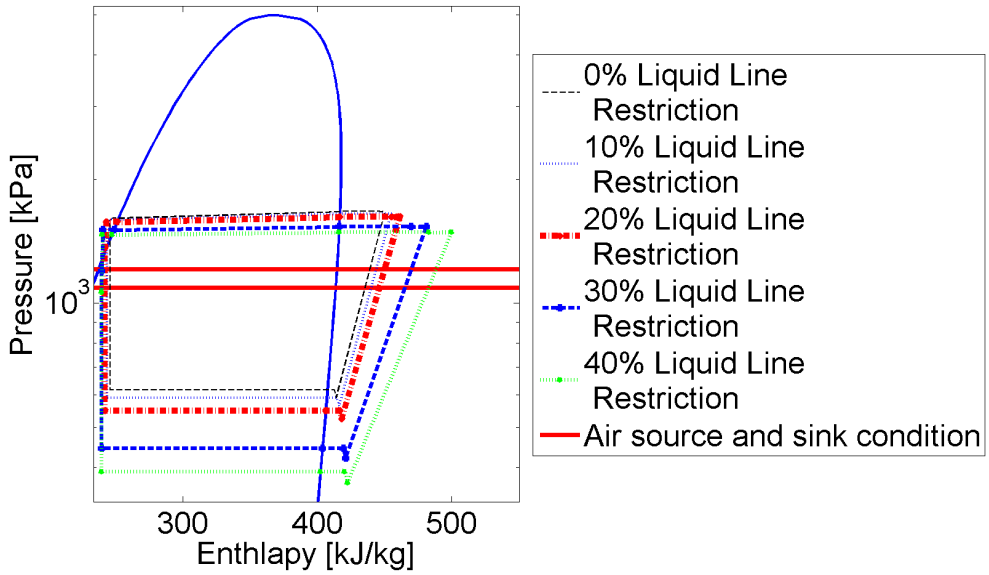

Figure 8.53. P-h diagram of system IX at different liquid line restriction levels.

Since the compressor suction superheat increases with increasing liquid line restriction level as shown in Figures 8.53 and 8.54, the accumulator of system IX does not hold any liquid refrigerant in all scenarios in 8.53 and 8.54. System IX reacts to liquid line restriction similarly as an FXO system without an accumulator like system IV in Figures 8.49 and 8.50. The evaporating pressure and condensing

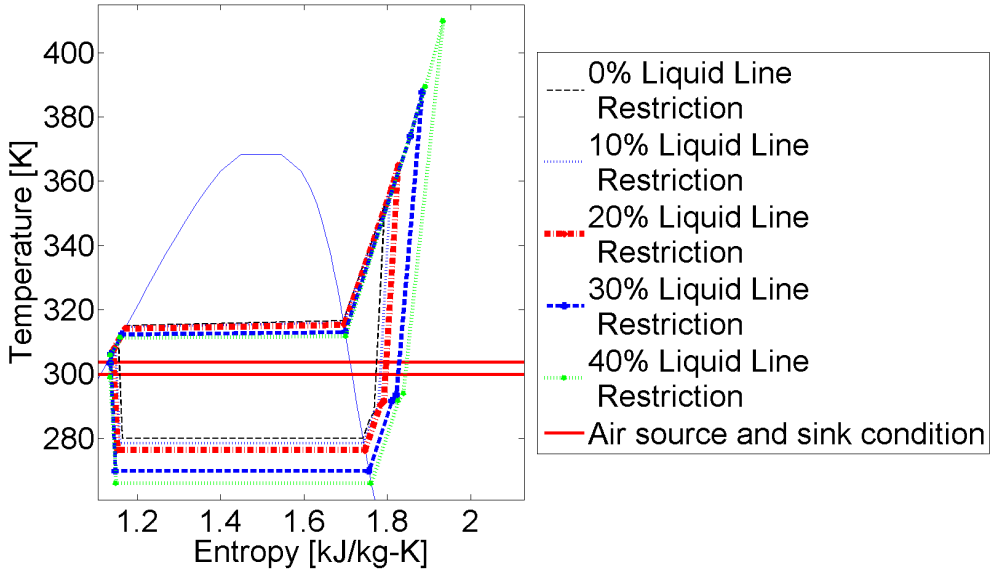

Figure 8.54. T-s diagram of system IX at different liquid line restriction levels.

pressure drop and the evaporator outlet superheat of systme IX increases with increasing liquid line restriction.

Figures 8.55 and 8.56 show the change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate, COP, SHR and COSP with increasing liquid line restriction level.

Since the accumulator does not affect the performance of an FXO system under liquid line restriction, the change of the system variables in Figures 8.55 and 8.56 are similar to those in an FXO system without an accumulator in Figures 8.51 and 8.52. Under increasing liquid line restriction level, COP, COSP, SHR, evaporator heat transfer rate and refrigerant mass flow rate decrease. Compressor power consumption also decrease with increasing liquid line restriction up to the level which causes a

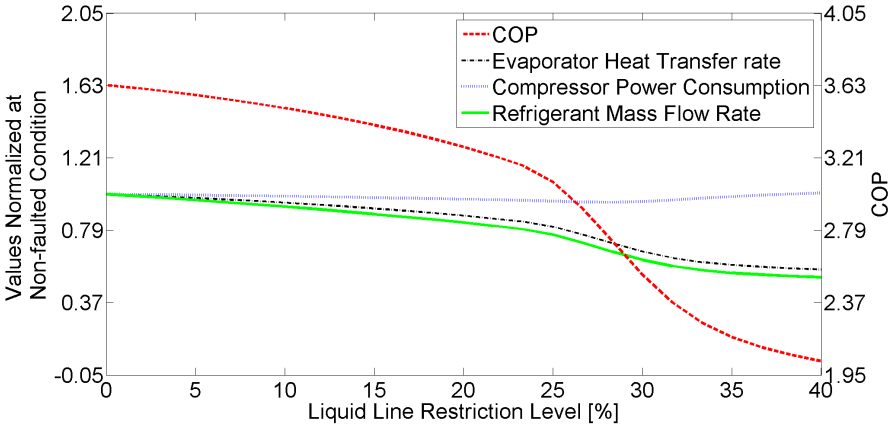

Figure 8.55. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system IX at different liquid line restriction levels.

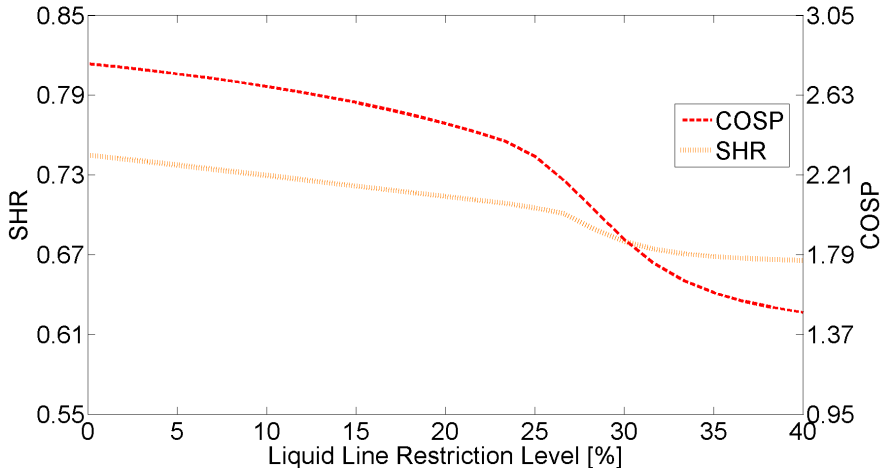

Figure 8.56. Change of SHR and COSP of system IX at different liquid line restriction levels.

zero subcooling at the expansion valve inlet. Above the level, the compressor power consumption increases with increasing liquid line restriction. The change of SHR in Figure 8.56 for liquid line restriction above 26% is unreliable because the evaporating temperature is lower than the freezing point of water.

8.5.3 TXV System

Liquid line restriction level from 0% to 29% is imposed on system VII and the resultant P-h and T-s diagrams are plotted as Figures 8.57 and 8.58.

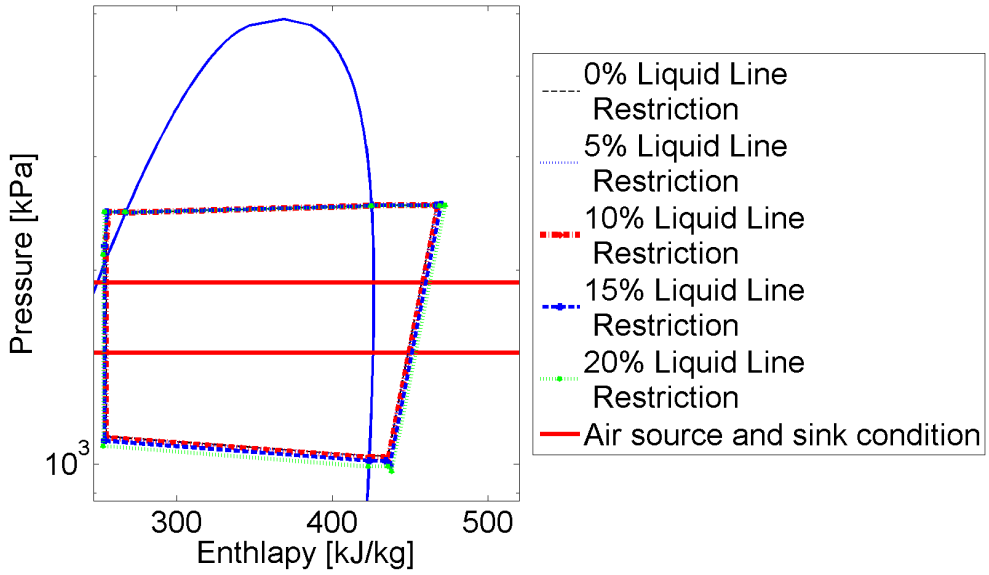

Figure 8.57. P-h diagram of system VII at different liquid line restriction levels.

With the control of superheat by TXV, no significant changes can be found except the slight drop of evaporating temperature at liquid line restriction level at 20%.

Figures 8.59 and 8.60 show the change of performance with different liquid line restriction levels.

No significant changes are observed in Figure 8.59 when the liquid line restriction level lies between 0% and 12% but at fault level above 12%, the evaporator superheat is no longer under control and the refrigerant mass flow rate starts to fall. This

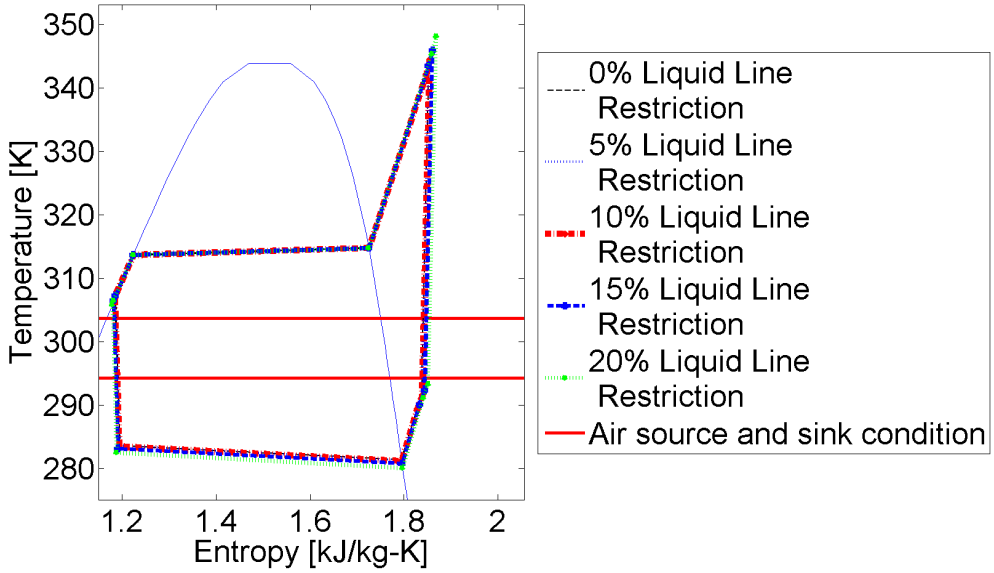

Figure 8.58. T-s diagram of system VII at different liquid line restriction levels.

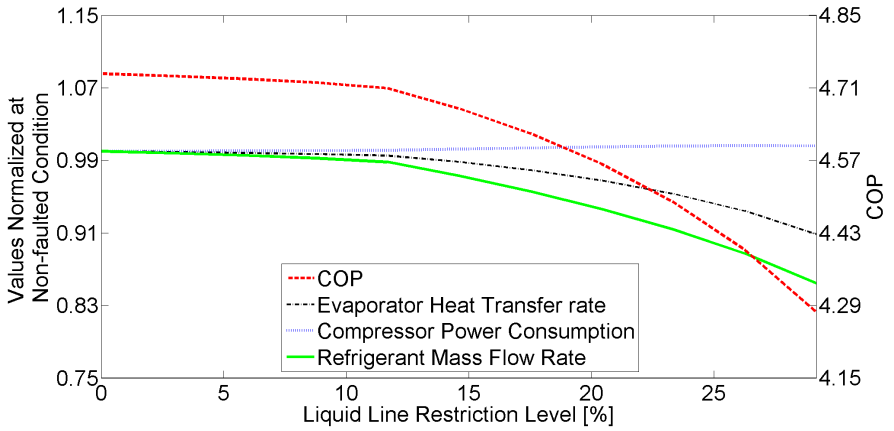

Figure 8.59. Change of evaporator heat transfer rate, compressor power consumption, refrigerant mass flow rate and COP of system VII at different liquid line restriction levels.

induces a fall of COP from 4.71 to 4.28 when the liquid line restriction level increases from 12% to 29%. The saturation of the TXV also causes a drop of COSP from 3.43 to 3.10, a decrease of evaporating temperature by 2.8K and a drop of SHR by 0.013.

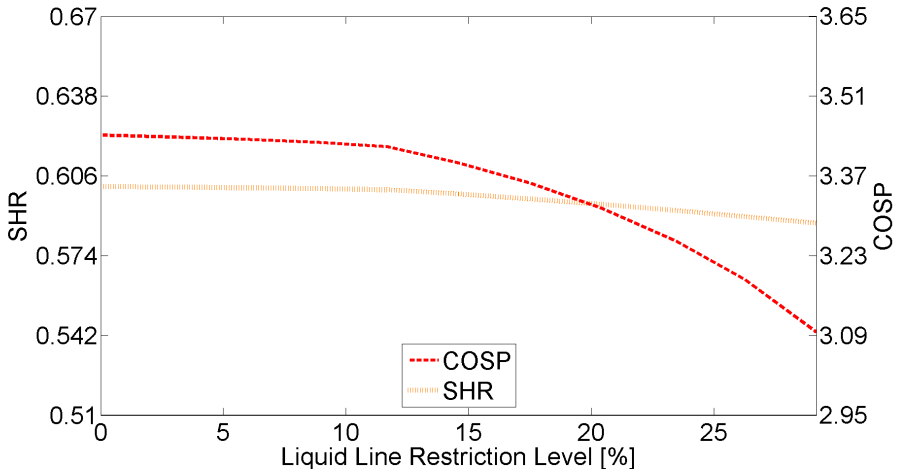

Figure 8.60. Change of SHR and COSP of system VII at different liquid line restriction levels.

8.6 Presence of non-condensable

Since Section 7.2.6 concludes that the system model can only predict the change of overall performance with increasing amount of non-condensable in the system correctly but not the condenser outlet subcooling and compressor suction superheat, only the change of overall performance is studied in this section.

8.6.1 FXO System

Figures 8.61 and 8.62 show the change of performance of the system as the non-condensable fault level increases.

Figures 8.61 and 8.62 show that increasing non-condensable in the system increases SHR by 3.7% and has negligible effect on evaporator heat transfer rate with 80%

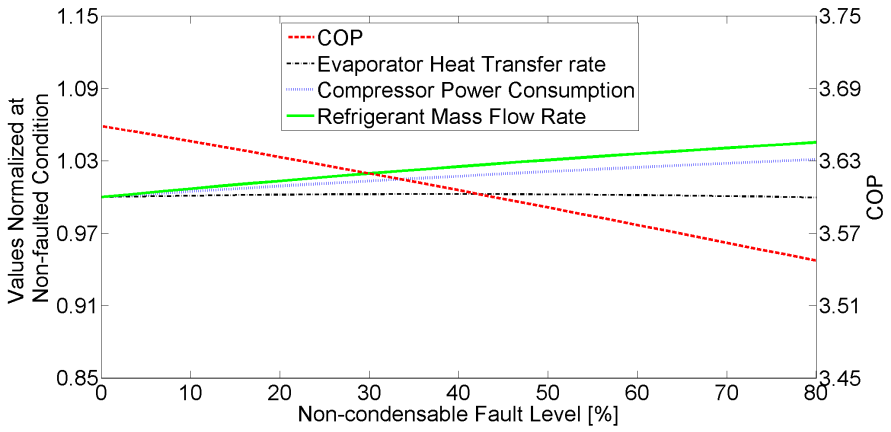

Figure 8.61. Change of evaporator heat transfer rate, compressor power consumption and COP of system IV at different non-condensable fault levels.

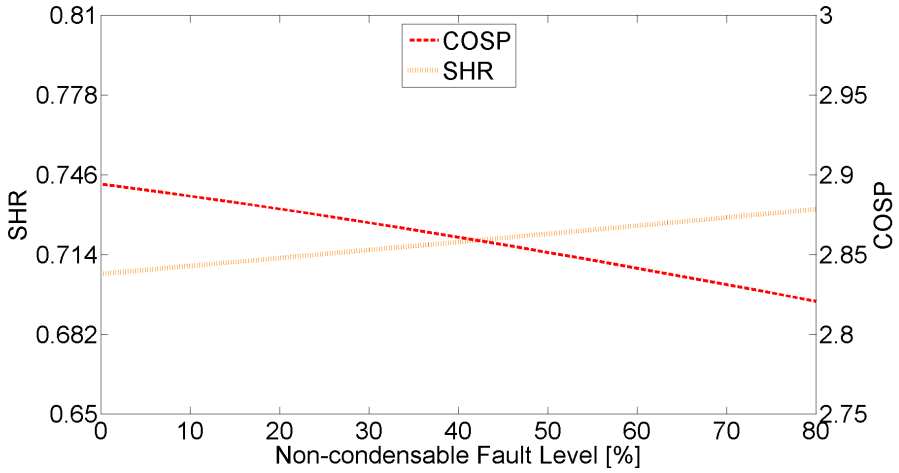

Figure 8.62. Change of SHR and COSP of system IV at different non-condensable fault levels.

change in the fault level. The drops of COP and COSP due to the fault is mainly a result of increasing compressor power consumption.

Figure 8.61 also shows an increase of refrigerant mass flow rate as a result of decreasing compressor suction superheat and hence increasing compressor suction

density. However, as Section 7.2.6 shows that the refrigerant mass flow rate of FXO systems decreases with increasing fault level, the increasing trend predicted in Figure 8.61 is a consequence of the inability of the model to simulate the decrease of refrigerant mass flow rate with increasing compressor suction superheat, and the trend is unreliable.

8.6.2 FXO System with an Accumulator

Figures 8.63 and 8.64 show the change of performance of the system as the amount of non-condensbale inside the system increases.

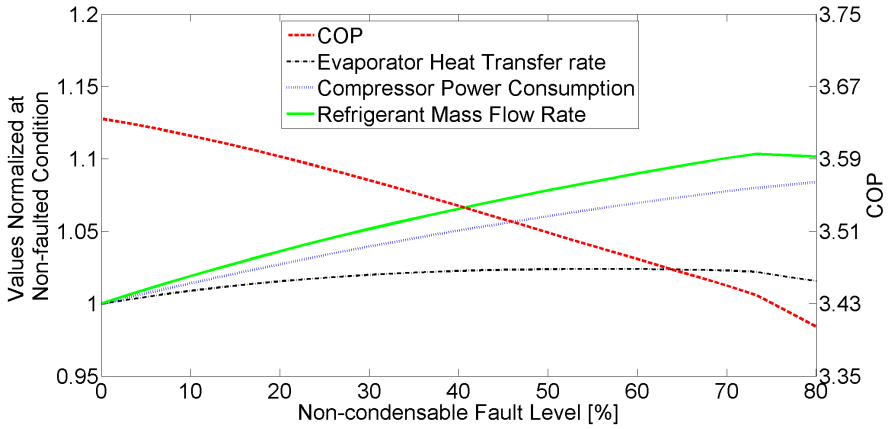

Figure 8.63. Change of evaporator heat transfer rate, compressor power consumption and COP of system IX at different non-condensable levels.

Figure 8.63 shows that increasing amount of non-condensable in system IX increases the refrigerant mass flow rate and compressor power consumption significantly. With little changes in evaporator heat transfer rate, the COP is

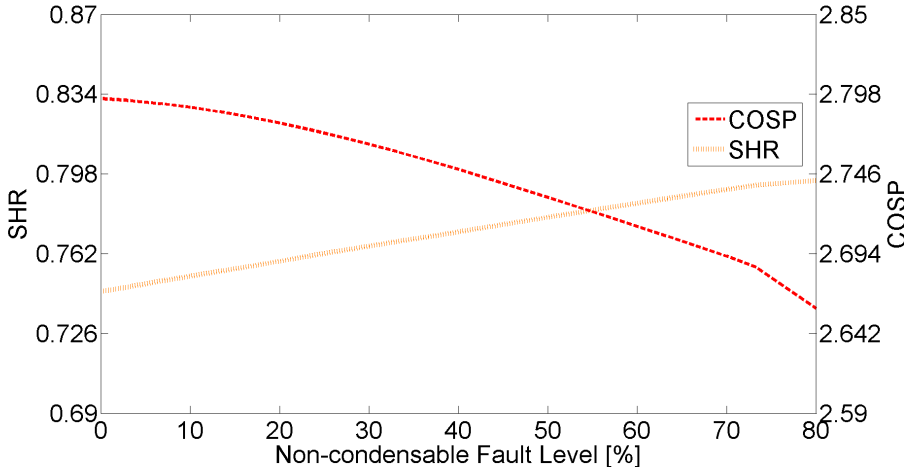

Figure 8.64. Change of SHR and COSP of system IX at different non-condensable levels.

deteriorated by 6.4%. The COSP in Figure 8.62 follows the decreasing trend and decreases by 4.9%, and the SHR is increased by 6.7%.

Figure 8.63 also shows that there is a noticeable change of slope for the change of evaporator heat transfer rate with the fault level when the fault level increases to 75%. Simulation results show that the change is related to the filling of liquid refrigerant of the accumulator as the compressor suction superheat drops to zero. However, as Section 7.2.6 shows that the compressor suction superheat of FXO systems increases with the non-condensbale fault level and will not reach zero as the fault level increases, the simulation result becomes unreliable as the compressor suction superheat reaches zero.

8.6.3 TXV System

The change of performance variables with increasing amount of non-condensables in system VII is plotted in Figures 8.65 and 8.66.

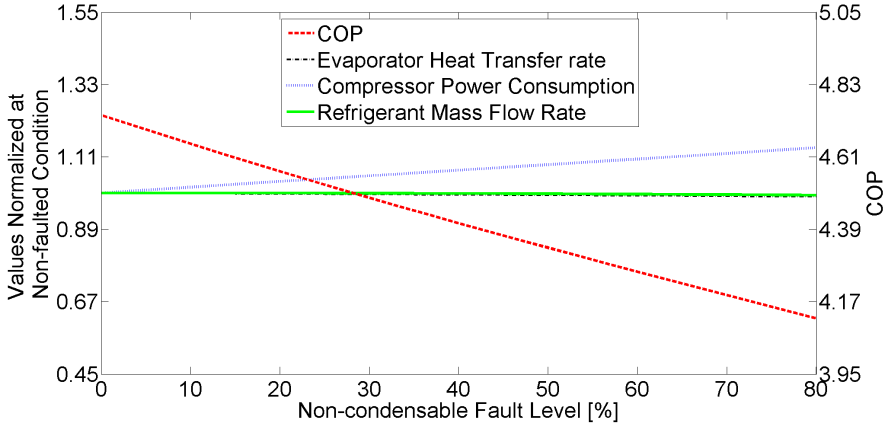

Figure 8.65. Change of evaporator heat transfer rate, compressor power consumption and COP of system VII at different non-condensable levels.

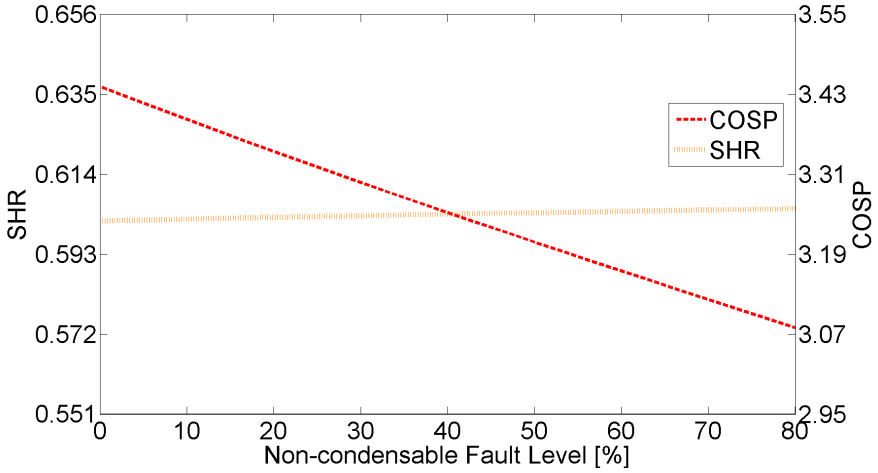

Figure 8.66. Change of SHR and COSP of system VII at different non-condensable levels.

Figure 8.65 shows an increase of compressor power consumption by 14% when the non-condensable level reaches 80% with negligible changes of refrigerant mass flow rate and evaporator heat transfer rate. This leads to a decrease of COP from 4.74 to 4.12. Figure 8.66 shows an increase of SHR less than 0.03 as a consequence of the non-condensable fault and a drop of COSP by 10.4%.

8.7 Summary

In summary, by conducting parametric studies of different types of faults with the simulation outputs of three different types of systems, including an FXO system, an FXO system with an accumulator and a TXV system, the impacts of faults on the performance of these systems, including evaporator heat transfer rate, COP and COSP, are identified.

In non-standard charging scenarios, there exists a charge level with maximum COP. When charge level decreases from the level for optimal COP, the drop of evaporator heat transfer rate is the main reason for the decrease of COP. When charge level increases, the increase of compressor power consumption is the main reason for the decrease of COP. Since compressor suction superheat decreases with increasing charge level, FXO systems with accumulator behave in the same way as ordinary FXO systems with decreasing charge levels, but their system performance does not change with charge level once the charge level is higher than what it needs to reduce the compressor suction superheat to zero.

In evaporator fouling scenarios, all systems respond with increasing evaporator fouling levels with decreasing evaporating temperature, decreasing evaporator heat transfer rate and hence decreasing COP. However, since evaporator fouling also reduces evaporator fan power consumption, COSP increases with increasing evaporator fouling level up to the level where the drop of evaporator heat transfer rate outweighs all the benefits gained by decreasing evaporator fan power consumption. Since this study is only limited to ducted cooling systems, the effect of evaporator fan power consumption on COSP may differ for systems without ducts on the evaporators, such as ductless cooling systems.

In condenser fouling scenarios, the drop of COP and COSP with increasing fouling level is mainly a result of increasing compressor power consumption. The increase of condenser fouling decreases the condenser airflow and increases the condensing pressure, the compressor pressure ratio and hence the compressor power consumption.

In scenarios with compressor flow fault, COP and COSP decrease with increasing compressor flow fault level because of the decrease of refrigerant mass flow rate and evaporator heat transfer rate. The fault also reduces compressor suction superheat of FXO systems. Hence the accumulators in FXO systems will reduce the effective charge level at high fault level and this removal of refrigerant from other components reduces the impact of the fault on COP and COSP.

In liquid line restriction scenarios, the drop of COP and COSP is a result of decreasing refrigerant mass flow rate and evaporator heat transfer rate. It can be seen that TXV systems are much more tolerant to the effect of liquid line restriction

on its performance than FXO systems because TXV systems can open the TXV to compensate the effect while FXO systems cannot compensate the fault actively. The effect of the restriction on TXV systems only becomes prominent when the TXVs are fully opened.

In cases with non-condensable in the systems, COP and COSP decrease as the fault level increases because of the increase of compressor power consumption, and the fault has negligible effect on the evaporator heat transfer rate and SHR of the systems.

CHAPTER 9. CONCLUSIONS AND FUTURE WORK

In this dissertation, an inverse modeling technique was developed and applied on eleven different cooling systems tested under different faulted conditions. Semi-empirical component and fault models were developed and combined to form a system model. The simulation models were evaluated in terms of estimation accuracy, robustness and computational speed and the effects of faults on the overall performance and the properties of the system were studied.

The size of these systems ranges from 2.5 to 5 ton with scroll or reciprocating compressors and FXO, TXV or EEV. The data collected from these systems were filtered and re-analyzed for consistency and to remove unrealistic data.

To develop the component models and to simulate effect of faults on the systems, semi-empirical component and fault models were developed from literature and fundamental physics. To avoid overemphasis on test results from certain conditions due to an imbalanced test matrix, a weighting scheme was used during the parameter estimation of the models. These models were validated individually with experimental data of all systems.

An implicit system model was generated by joining the component models together and was formulated as a multi-variable optimization problem. To speed up

the calculation, empirical equations for initial guesses of independent variables of the cycle model were constructed and quasi-Newton method was used to solve the problem with non-dimensionalized independent variables and residuals. The robustness of the solver was maintained by using constrained optimization when an error was encountered in the quasi-Newton method due to unrealistic independent variables. The two-point charge tuning method from Shen (2006) was modified to increase the accuracy of charge estimation. Unreliable simulation outputs are also identified and removed based on leverage calculation and applicability of models from literature.

To validate the model, the overall performance of the systems from the experiments and the simulations is compared. The results show that there was no significant deviation between the experimental results and simulation outputs. The accuracy of charge tuning is improved by adding new coefficients and using all valid training data to estimate the regression coefficients. The robustness of the solver was studied and only 16 out of 653 data points diverged. The evaporator heat transfer rate is estimated with an average absolute deviation at 2.59% and the compressor power consumption is estimated with an average absolute deviation at 1.86%.

The experimental and simulated changes of variables such as superheat and subcooling with respect to different fault levels are also compared to examine if the simulation can predict the trends correctly. While most changes of pressure, air temperature differences, superheat and subcooling with respect to fault levels were

predicted correctly by the simulation, the simulated changes of condenser outlet subcooling and compressor suction superheat with respect to non-condensable fault level were not the same as the experimental results. An experimental study withe non-condensable testing of system XI shows that the model does not simulate the decrease of refrigerant mass flow rate as a result of non-condensable passing through the expansion valve, and this may be the reason for the inability of the model to simulate the change of condenser outlet subcooling and compressor suction superheat correctly.

The impacts of faults on the performance of selected systems were also evaluated and the primary driver of the COP under faulted condition was identified. In the case of undercharging, evaporator fouling, compressor flow fault and liquid line restriction, the drop of evaporator heat transfer rate dominates the drop of COP. In the case of overcharging, condenser fouling and presence of non-condensables, the drop of COP was mainly due to the increase of compressor power consumption.

The analysis result in this dissertation also shows that the model has capability for improvement. This includes:

1. Modeling of the effect of non-condensable on expansion valve mass flow rate

   Section 7.2.6 shows that the system model is incapable to simulate the decrease of refrigerant mass flow rate with increasing amount of non-condensable in the system because the expansion valve model cannot simulate the effect of non-condensable on the refrigerant mass flow rate in the system. The estimation

accuracy can be improved if methods to account for the decrease of refrigerant mass flow rate of the expansion valve are developed.

2. Developing methods to combine different component models and to adjust the size of the systems to create new system models

   The component models are modular and can be combined together to form the pre-tuning simulation of new systems. They also contain normalization parameters which can be changed to adjust the size of the components. If new methods are developed, it may be capable to provide simulations of new systems with different rated capacities and seasonal energy efficiency ratios.

3. Increasing the speed of the system model

   Part of the system model, such as the expansion valve model and refrigerant property calculation, is difficult to differentiate, and the system model is solved by estimating derivatives of residual vectors with respect to the input vectors by finite difference method. The calculation speed of any system model can be greatly increased if analytical derivatives of the system model residual vector can be provided to the solver. This can be achieved by developing methods to calculate the derivatives of refrigerant properties and the derivatives of the residual vector with system model input vectors by automatic differentiation.

4. Validating the modeling methodology with multi-fault scenarios

   The system model outputs was only compared with the experimental data of one multi-fault scenario (compressor flow fault and undercharging) in the data

of system XI. More multi-fault scenarios needed to be collected in order to validate the ability of the strategy to model multi-fault scenarios.

5. Defining test matrix for future testing

   The technique regression diagnostics used to define the applicability domain of the model can be used to examine if a data point is located outside the domain of a component model. The technique may also be used to define the test matrix for future fault testing so as to understand the impact of fault on different systems with a small number of tests.

6. Studying the effect of COSP with evaporator fouling by more experiments

   While Figure 8.16 suggested that evaporator fouling may lead to an increase of COSP by modeling the fan power consumption with manufacturer data, the data from Kim (Kim, 2009) suggested that evaporator fouling decreased COSP of the system. To understand whether the case in Kim (Kim, 2009) is true, it is necessary to conduct more tests to study the effect of evaporator fouling on fan power consumption.

To conclude, a methodology which can be used to build a vapor compression system model mainly by experimental data is developed. The method covers air source cooling systems with a variety of components, including reciprocating compressors, scroll compressors, FXO, TXV and EEV. The model is able to predict the impact of different types of faults, including non-standard charging, heat exchanger fouling, compressor flow fault, liquid line restriction and presence of

non-condensable, on the refrigerant properties, air properties, cooling capacity and power consumption of the system accurately within a short time. Thereafter it can be used to assist the construction of a database to evaluate FDD tools.

LIST OF REFERENCES

LIST OF REFERENCES

Aaron, D. A. and Domanski, P. (1990). Experimentation, analysis, and correlation of refrigerant-22 flow through short tube restrictors. *ASHRAE Transactions*, 96(1):729 – 742.

AHRI (2004). *2004 Standard For Performance Rating Of Positive Displacement Refrigerant Compressors And Compressor Units*. Air-conditioning, Heating, and Refrigeration Institute, Arlington, VA.

AHRI (2008). *2008 Standard for Performance Rating of Unitary Air-Conditioning & Air-Source Heat Pump Equipment*. Air-conditioning, Heating, and Refrigeration Institute, Arlington, VA.

ASHRAE (1987). *ANSI/ASHRAE 41.2-1987 (RA92) Standard Methods for Laboratory Airflow Measurement*. American Society of Heating, Refrigerating and Air-conditioning Engineers, Inc., Atlanta, GA.

ASHRAE (2008). *2008 ASHRAE Handbook - Systems and Equipment*. American Society of Heating, Refrigerating and Air-conditioning Engineers, Inc., Atlanta, GA.

Atkinson, A. C. (1985). *Plots, Transformations, and Regression*. Oxford, Oxford.

Baroczy, C. J. (1965). Correlation of liquid fraction in two-phase flow with application to liquid metals. *Chemical Engineering Progress Symposium Series*, 61(57):179 – 191.

Bell, I. H., G. E. A. and König, H. (2012). Experimental analysis of the effects of particulate fouling on heat exchanger and air-side pressure drop for a hybrid dry cooler. *Heat Transfer Engineering*, 32(3):264 – 271.

Bell, I. (2010 (accessed September 25, 2012)). *ACHP*. [Software] `http://achp.sourceforge.net/`.

Bendapudi, S. (2002). *Development and Evaluation of Modeling Approaches for Transients in Centrifugal Chillers*. PhD thesis, Purdue University, West Lafayette, Indiana.

Berry, C. H. *Flow and Fan: Principles of Moving Air through Ducts*. Industrial Press, New York, 2nd edition.

Biegler, L. T. and Tjoa, I. (1993). A parallel implementation of parameter estimation with implicit models. *Annals of Operations Research*, 42(1):1 – 23.

Braun, J. E. (1989). *Methodologies for the Design and Control of Central Cooling Plants*. PhD thesis, University of Winsconsin - Madison, Madison, Winsconsin.

Breuker, M. S. (1997). Evaluation of a statistical, rule-based detection and diagnostics method for vapor compression air conditioner. Master's thesis, Purdue University, West Lafayette, Indiana.

Cavallini, A. and Zecchin, R. (1971). A dimensionless correlation for heat transfer in forced convection condensation. In *Proceedings of the XIIIth International Congress of Refrigeration*, volume 2, pages 193 – 200.

CEC (2008). *2008 Building Energy Efficiency Standards for residential and nonresidential buildings*. California Energy Commission, Sacramento, California. CEC-400-2008-001-CM.

Chen, J. C. (1966). Correlation for boiling heat transfer to saturated fluids in convective flow. *Industrial & engineering chemistry process design and development*, 5(3):322 – 329.

Cheung, H. and Braun, J. E. (2014). Component-based, gray-box modeling of ductless multi-split heat pump systems. *International Journal of Refrigeration*, 38:30 – 45.

Churchill, S. (1977). Friction-factor equation spans all fluid-flow regimes. *Chemical Engineering*, 84(24).

DOE (2011). *2011 Buildings Energy Data Book*. U.S. Department of Energy.

Dukler, A. E., Wicks III, M., and Cleveland, R. (1964). Frictional pressure drop in two-phase flow. *AIChE J.*, 10:38 – 51.

Fujie, N. (1964). A relation between steam quality and void fraction in two-phase flow. *AIChE Journal*, 10:227 – 232.

Gau, C. Y. and Stadtherr, M. A. (2004). Deterministic global optimization for error-in-variables parameter estimation. *AIChE Journal*, 48(6):1192 – 1197.

Hariharan, N. and Rasmussen, B. P. (2008). Parameter estimation for dynamic hvac models with limited sensor information. In *Proceedings of 2010 American Control Conference*, Baltimore, MD.

Harms, T. M. (2002). Charge inventory system modeling and validation for unitary air conditioners. Master's thesis, Purdue University, West Lafayette, Indiana.

Hjortland, A. L. (2014). Probabilistic fault detection and diagnostics for packaged air-conditioner outdoor air economizers. Master's thesis, Purdue University, West Lafayette, IN.

Hong, K. T. and Webb, R. L. (1996). Calculation of fin efficiency for wet and dry fins. *HVAC&R Research*, 2(1):27 – 41.

Hughmark, G. A. (1962). Holdup in gas-liquid flow. *Chemical Engineering Progress*, 58(4):62 – 65.

Incropera, F. P., DeWitt, D. P., Bergman, T. L., and Lavine, A. S. (2007). *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, New York.

Jähnig, D. I., Reindl, D. T., and Klein, S. A. (2000). A semi-empirical method for representing domestic refrigerator/freezer compressor calorimeter test data. *ASHRAE Transactions*, 106(2):122 – 130.

Jin, H. and D., S. J. (2002). A parameter estimation based model of water-to-water heat pumps for use in energy calculation program. *ASHRAE Transactions*, 108(1):3 – 17.

Kattan, N., Thome, J. R., and Favrat, D. (1998). Flow boiling in horizontal tubes: Part 1 development of a diabatic two-phase flow pattern map. *Journal of Heat Transfer*, 120(1):140 – 147.

Kays, W. M. and London, A. L. (1964). *Compact Heat Exchangers*. McGraw-Hill, New York.

Kern, D. and Kraus, D. (1972). *Extended Surface Heat Transfer*. McGraw-Hill, New York.

Kim, H. and Bullard, C. W. (2002). Thermal performance analysis of small hermetic refrigeration and air-conditioning compressors. *JSME International Journal, Series B*, 45(4):857 – 864.

Kim, J., Braun, J. E., Groll, E. A., and Palmiter, L. (2008). *Off-design Performance of Residential Heat Pumps*. Ray. W Herrick Laboratories, Purdue University. Internal Report.

Kim, M., Payne, W. V., Domanski, P. A., Yoon, S. H., and L., H. C. J. (2009). Performance of a residential heat pump operating in the cooling mode with single faults imposed. *Applied Engineering*, 29:770 – 778.

Kim, Y. and O'Neal, D. L. (1994). Two-phase flow of r-22 through short-tube orifices. *ASHRAE Transactions*, 100(1):323 – 334.

Kline, S. J. and McClinktock, F. A. (1953). Describing uncertatinties in single-sample experiments. *Mechanical Engineering*, 75:3 – 8.

Kuehn, T. H., Ramsey, J. W., and Threlkeld, J. L. (1998). *Thermal Environmental Engineering*. Prentice Hall.

Lemmon, E.W., H. M. M. M. (2013). Nist standard reference database 23: Reference fluid thermodynamic and transport properties-refprop, version 9.0. National Institute of Standards and Technology, Standard Reference Data Program, Gaithersburg.

Lenger, M. J. (1998). Superheat stability of an evaporator and thermostatic expansion valve. Master's thesis, Air Conditioning and Refrigeration Center, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL. ACRC TR-138.

Leroy, J. T. (1997). Capacity and power demand of unitary air conditioners and heat pumps under extreme temperature and humidity conditions. Master's thesis, Purdue University, West Lafayette, Indiana. Ind Report No. 3220-2 HL 97-25.

Li, H. and Braun, J. E. (2008). A method of modeling adjustable throat-area expansion valves using manufacturers' rating data. *HVAC&R Research*, 14(4):581 – 595.

Liang, C., Jiangping, C., Jinghui, L., and Zhijiu, C. (2009). Experimental investigation on mass flow characteristics of electronic expansion valves with r22, r410a and r407c. *Energy Conversion and Management*, 50:1033 – 1039.

MacArthur, J. W. and W., G. E. (1989). Unsteady compressible two-phase flow model for predicting cyclic heat pump performance and a comparison with experimental data. *International Journal of Refrigeration*, 12(1):29 – 41.

Martinelli, R. C. and Nelson, D. (1949). Prediction of pressure drop during forced-circulation boiling of water. *Transactions of ASME*, 70:695 – 702.

Mathworks (2011). Matlab version 2011a.

McQuiston, F. C., P. J. D. and Spitler, J. D. (1989). *Heating, Ventilating, and Air Conditioning Analysis and Design*. John Wiley & Sons, New York.

Moody, L. F. (1944). Friction factors for pipe flow. *Transactions of the ASME*, 66(8):671 – 684.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, New York, 2nd edition.

Osborne, W. C. *Fans*. Pergamon Press, London, 2nd edition.

Palmiter, L., Kim, J., Larson, B., Francisco, P. W., Groll, E. A., and Braun, J. E. (2011). Measured effect of airflow and refrigerant charge on the seasonal performance of an air-source heat pump using r-410a. *Energy and Buildings*, 43:1802 – 1810.

Park, C., Cho, H., Lee, Y., and Kim, Y. (2007). Mass flow characteristics and empirical modeling of r22 and r410a flowing through electronic expansion valves. *International Journal of Refrigeration*, 30:1401 – 1407.

Park, Y. C., Kim, Y. C., and Min, M. K. (2001). Performance analysis on a multi-type inverter air conditioner. *Energy Conversion and Management*, 42:1607 – 1621.

Payne, W. V. and O'Neal, D. L. (1999). Multiphase flow of refrigerant r410a through short tube orifices. *ASHRAE Transactions*, 105(2):66 – 74.

Payne, W. V. and O'Neal, D. L. (2004). A mass flow rate correlation for refrigerants and refrigerant mixtures flowing through short tubes. *HVAC&R Research*, 10(1):73 – 87.

Qifang, Y., Jiangping, C., and Zhijiu, C. (2007). Experimental investigation of r407c and r410a flow through electronic expansion valve. *Energy Conversion and Management*, 48:1624 – 1630.

Rabehl, R. J., Mitchell, J. W., and Beckman, W. A. (1999). Parameter estimation and the use of catalog data in modeling heat exchangers and coils. *HVAC&R Research*, 5(1):3 – 17.

Rice, C. K. (1987). The effect of void fraction correlation and heat flux assumption on refrigerant charge inventory predictions. *ASHRAE Transactions*, 93(1):341 – 367.

Rice, K. and Jackson, W. (2005). *DOE/ORNL Heat Pump Design Model on the Web, Mark VI Version*. [Web application] `http://www.ornl.gov/~wlj/hpdm/MarkVII.html`.

Rossi, T. M. (1995). *Detection, diagnosis, and evaluation of faults in vapor compression cycle equipment.* PhD thesis, Purdue University, West Lafayette, Indiana. Ind Report No. 17967-3 HL 95-13.

Rossi, T. M. and Braun, J. E. (1997). A statistical, rule-based fault detection and diagnostic method for vapor compression air conditioners. *HVAC&R Research*, 3(1):19 – 37.

Schantz, C. and Leeb, S. (2012). Non-intrusive fault detection in reciprocating compressors. In *14th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Shah, M. M. (1979). A general correlation for heat transfer during film condensation inside pipes. *International Journal of Heat and Mass Transfer*, 22(4):547 – 556.

Shanwei, M., Chuan, Z., Jiangping, C., and Zhiujiu, C. (2005). Experimental research on refrigerant mass flow coefficient of electronic expansion valve. *Applied Thermal Engineering*, 25:2351 – 2366.

Shen, B. (2006). *Improvement and validation of unitary air conditioner and heat pump simulation models at off-design conditions.* PhD thesis, Purdue University, West Lafayette, Indiana. Ind Report No. 6304-1 HL2006-1.

Singh, G. M., Hrnjak, P. S., and Bullard, C. W. (2001). Flow of refrigerant 134a through orifice tubes. *HVAC&R Research*, 7(3):245 – 262.

Taitel, Y. and Barnea, D. (1990). Two-phase slug flow. *Advances in Heat Transfer*, 20:83 – 132.

Wallis, G. B. (1969). *One-dimensional Two-phase Flow.* McGraw-Hill, New York.

Wedekind, G. L., Bhatt, B. L., and Beck, B. T. (1978). A system mean void fraction model for predicting various transient phenomena associated with two-phase evaporating and condensing flows. *International Journal of Multiphase Flow*, 4:97 – 114.

Winandy, E., S. O. C. and Lebrun, J. (2002). Experimental analysis and simplified modeling of heretic scroll refrigeration compressor. *Applied Thermal Engineering*, 22:107 – 120.

Yang, L. Braun, J. E. and Groll, E. A. (2007). The impact of fouling on the performance of filter-evaporator combinations. *International Journal of Refrigeration*, 30:489 – 498.

Yashar, D., Wilson, M., Kopke, H. R., Grahan, D. M., Chato, J. C., and Newell, T. A. (2001). An investigation of refrigerant void fraction in horizontal, microfin tubes. *HVAC&R Research*, 7:67 – 82.

Yuill, D. P. (2014). *Development of methodologies for evaluating performance of fault detection and diagnostics protocols applied to unitary air-conditioning equipment.* PhD thesis, Purdue University, West Lafayette, Indiana.

Zakula, T. Gayeski, N. T. A. P. R. and Norford, L. K. (2011). Variable-speed heat pump model for a wide range of cooling conditions and loads. *HVAC&R Research*, 17(5):670 – 691.

Zhifang, X., Lin, S., and Hongfei, O. (2008). Refrigerant flow characteristics of electronic expansion valve based on thermodynamic analysis and experiment. *Applied Thermal Engineering*, 28:238 – 243.

Zhou, X., Braun, J. E., and Zeng, Q. (2007). An improved method for determining heat transfer fin efficiencies for dehumidifying cooling coils. *HVAC &R Research*, 13(5):769 – 782.

Zivi, S. M. (1964). Estimation of steady-state steam void-fraction by means of the principle of minimum entropy production. *Journal of Heat Transfer*, 86:247 – 252.

APPENDICES

APPENDIX A. CONDENSER MATHEMATICAL MODEL

The mathematical procedure to solve the condenser model differs slightly from ACHP (Bell, 2010)) to accommodate the significant changes of refrigerant properties in faulted conditions such as the properties in highly superheated sections. As indicated in Chapter 4, the solution of the condenser model starts from the superheated section, followed by the two-phase and subcooled sections.

A.1 Superheated Section

To begin the solution for the heat transfer rate of the superheat section, the overall heat transfer conductance is computed by

$$UA_{overall,cond,sh} = (\frac{1}{UA_{a,cond}} + \frac{1}{UA_{r,cond,sh}})^{-1}. \tag{A.1}$$

Since the superheated section may be highly superheated as a result of faults such as undercharging or condenser fouling and the refrigerant properties between the inlet and outlet of the section may differ significantly, the calculation of the average

property along the section should consider both the inlet and outlet the specific heat capacity of superheated refrigerant is estimated by

$$c_{pr,sh} = \frac{h_{r,cond,in} - h_{r,cond,v}}{T_{r,cond,in} - T_{r,cond,v}}. \tag{A.2}$$

As the airflow on the superheated section depends on the ratio of the size of the superheated section to the size of the condenser, the area ratio of the superheated section is calculated, regardless of the minimum capacitance rate, by

$$w_{cond,sh} = -\frac{\ln(1 - \frac{T_{r,cond,v} - T_{r,cond,in}}{T_{a,cond,in} - T_{r,cond,in}})\dot{m}_r c_{pr,sh}}{(1 - \exp(\frac{UA_{overall,cond,sh}}{\dot{m}_{a,cond} c_{pa,cond,in}}))\dot{m}_{a,cond} c_{pa,cond,in}}. \tag{A.3}$$

The area ratio in Eqn. (A.3) is defined as the ratio between the inner surface area of the superheated section to the inner surface area of the entire condenser.

The heat transfer rate in the superheated section is given by

$$\dot{Q}_{cond,sh} = \dot{m}_r(h_{r,cond,in} - h_{r,cond,v}). \tag{A.4}$$

A.2 Two-phase Section

The overall heat transfer conductance of the two-phase section is calculated by

$$UA_{overall,cond,tp} = (\frac{1}{UA_{a,cond}} + \frac{1}{UA_{r,cond,tp}})^{-1}. \tag{A.5}$$

As the capacitance rate of the refrigerant flow in two-phase section is infinite, the conductance from Eqn. (A.5) can be used to calculate the heat exchanger effectiveness of the two-phase section by

$$\varepsilon_{cond,tp} = 1 - exp(-\frac{UA_{cond,tp}}{\dot{m}_{a,cond}c_{pa,cond,in}}). \tag{A.6}$$

To consider the temperature glide for some refrigerants in the two-phase region, the condensing temperature for the solution of the heat transfer rate of the condensing flow is defined by

$$T_{r,cond,tp} = (1 - \overline{\gamma_{cond}})T_{r,cond,l} + \overline{\gamma_{cond}}T_{r,cond,v}. \tag{A.7}$$

Assuming saturated liquid at the two-phase section outlet, the area ratio of the two-phase section is calculated by

$$w_{cond,tp} = -\frac{\dot{m}_r(h_{r,cond,v} - h_{r,cond,l})}{\varepsilon_{cond,tp}\dot{m}_{a,cond}c_{pa,cond,in}(T_{a,cond,in} - T_{r,cond,tp})}. \tag{A.8}$$

The heat transfer rate of the two-phase section is given by

$$\dot{Q}_{tp,cond} = \dot{m}_r(h_{r,cond,v} - h_{r,cond,l}). \tag{A.9}$$

If the sum of area ratios of the superheated and two-phase section is larger than one, the refrigerant exits the condenser as two-phase refrigerant instead of saturated liquid. The area ratio of the two-phase section is calculated by

$$w_{cond,tp} = 1 - w_{cond,sh} \tag{A.10}$$

and the two-phase heat transfer rate is calculated by

$$\dot{Q}_{cond,tp} = \varepsilon_{cond,tp} w_{cond,tp} \dot{m}_{a,cond} c_{pa,cond,in} (T_{r,cond,tp} - T_{a,cond,in}). \tag{A.11}$$

A.3 Subcooled Section

When the sum of area ratios of the superheated section and two-phase section is less than one, the remaining section of the condenser should be subcooled. The area ratio of the subcooled section is given by

$$w_{cond,sc} = 1 - w_{cond,tp} - w_{cond,sh}. \tag{A.12}$$

The overall heat transfer conductance of the subcooled section is given by

$$UA_{overall,cond,sc} = (\frac{1}{UA_{a,cond}} + \frac{1}{UA_{r,cond,sc}})^{-1}. \tag{A.13}$$

As the change of specific heat capacity of liquid refrigerant across the subcooled section is insignificant, the average specific heat capacity of refrigerant in the

subcooled section is estimated by the specific heat capacity of saturated liquid. This avoids iterative calculation for specific heat capacity, and the capacity ratio of the subcooled section is given by

$$Cr_{r,cond,sc} = \frac{\min(w_{cond,sc}\dot{m}_{a,cond}c_{pa,cond}, \dot{m}_r c_{pr,cond,sc})}{\max(w_{cond,sc}\dot{m}_{a,cond}c_{pa,cond}, \dot{m}_r c_{pr,cond,sc})}. \tag{A.14}$$

The number of transfer units for the ε-NTU method is given by

$$NTU_{cond,sc} = \frac{w_{cond,sc}UA_{overall,cond,sc}}{\min(w_{cond,sc}\dot{m}_{a,cond}c_{pa,cond}, \dot{m}_r c_{pr,sc})}. \tag{A.15}$$

The heat exchanger effectiveness is calculated assuming a crossflow heat exchanger by

$$\varepsilon_{cond,sc} = \begin{cases} 1 - exp(-\frac{1-exp(-Cr_{r,cond,sc}NTU_{cond,sc})}{Cr_{r,cond,sc}}) & \text{if } \frac{w_{cond,sc}\dot{m}_{a,cond}c_{pa,cond}}{\dot{m}_r c_{pr,cond,sc}} > 1 \\ \frac{1-exp(-Cr_{r,cond,sc}(1-exp(NTU_{r,cond,sc})))}{Cr_{r,cond,sc}} & \text{otherwise} \end{cases}. \tag{A.16}$$

The heat transfer rate in the subcooled section is calculated by

$$\dot{Q}_{cond,sc} = \varepsilon_{cond,sc}w_{cond,sc}\dot{m}_{a,cond}c_{pa,cond,in}(T_{r,cond,l} - T_{a,cond,in}). \tag{A.17}$$

The density of the subcooled section is calculated as the density of the saturated liquid.

A.4 Overall Performance

The total heat transfer rate of the condenser is given by

$$\dot{Q}_{cond} = \dot{Q}_{cond,sh} + \dot{Q}_{cond,tp} + \dot{Q}_{cond,sc}. \tag{A.18}$$

If a subcooled section does not exist, the heat transfer rate across the subcooled section is set to zero.

APPENDIX B. EVAPORATOR MATHEMATICAL MODEL

To solve the evaporator heat transfer model, the exit refrigerant of the evaporator is assumed to be saturated vapor first and the area ratio of the two-phase section is set to one. The evaporator is solved with the two-phase section routine in Appendix C and the heat transfer rate is compared with the maximum heat transfer rate of the two-phase section in

$$\dot{Q}_{evap,tp,max} = \dot{m}_r(h_{r,evap,v} - h_{r,evap,in}). \tag{B.1}$$

If the heat transfer rate of the evaporator model is smaller than the heat transfer rate in Eqn. (B.1), the assumption will be accepted and the result will be the solution of the evaporator model. Otherwise the area ratio of the two-phase section is iterated between 0 and 1 to find the location where the refrigerant is saturated vapor. The region between the location and the inlet of evaporator is the two-phase section, and the heat transfer rate of the two-phase section is given by Eqn. (B.1).

After estimating the properties of the two-phase section, if the area ratio of the two-phase section is less than one, the remaining section of the evaporator is considered to be the superheated section. The superheated section is solved

according to Appendix C, and the refrigerant temperature at the evaporator outlet is iterated until it reaches the temperature calculated by

$$T_{r,evap,out} = T_{r,evap,in} + \frac{\dot{Q}_{evap,sh}}{\dot{m}_r c_{pr,evap,sh}}. \tag{B.2}$$

The heat transfer rate of the evaporator is given by

$$\dot{Q}_{evap} = \dot{Q}_{evap,sh} + \dot{Q}_{evap,tp} \tag{B.3}$$

which is the sum of the heat transfer rate of the superheated section and two-phase section.

The air outlet state of the evaporator is calculated by the continuity equation of the air-water mixture and conservation of energy. This gives the enthalpy and humidity ratio at the air outlet of the evaporator by

$$h_{a,evap,out} = h_{a,evap,in} + \frac{\dot{Q}_{evap}}{\dot{m}_{a,evap}} \tag{B.4}$$

and

$$\omega_{a,evap,out} = \omega_{a,evap,sh} w_{evap,sh} + \omega_{a,evap,tp} w_{evap,tp}. \tag{B.5}$$

The sensible heat ratio of the evaporator is calculated by

$$SHR = \frac{\dot{m}_{a,evap} c_{pa,evap} (T_{a,evap,in} - T_{a,evap,out}(h_{a,evap,out}, \omega_{a,evap,out}))}{\dot{Q}_{evap}}. \tag{B.6}$$

APPENDIX C. CALCULATION PROCEDURE OF PARTIAL-WET-PARTIAL-DRY METHOD

To solve the evaporator model more accurately than the fully wet coil analysis and fully dry coil analysis, a partial-dry-partial-wet method is used (Braun, 1989). Since the refrigerant temperature profiles of the two-phase section and the superheated section are different, they are solved by different approaches. For convenience, a notation system for this appendix as shown by Figure C.1 is used.

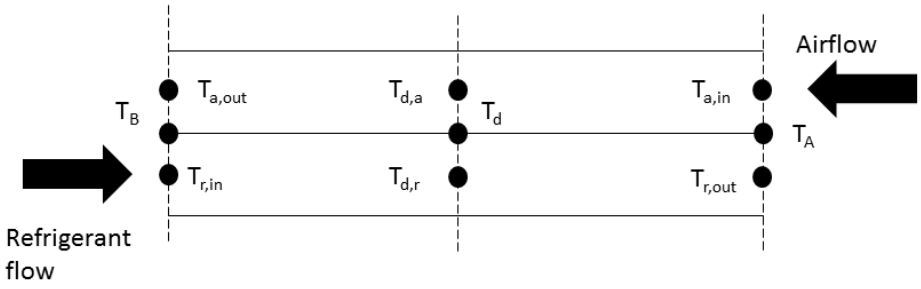

Figure C.1. Diagram showing air and refrigerant temperature in a partial-wet-partial-dry analysis.

Figure C.1 illustrates an air-to-refrigerant counterflow heat exchanger divided into two parts. At the intersection between the two parts, the surface temperature reaches T_d which is the dewpoint of air at the inlet of the section. This only occurs when

the coil is partially wet and the location does not exist for a completely dry coil or completely wet coil.

C.1 Two-phase Section

To solve the counterflow heat exchanger in Figure C.1, a set of NTUs is needed and they are calculated by

$$NTU_{overall,evap,tp,dry} = \frac{1}{\dot{m}_{a,evap} c_{pa,evap}} \left(\frac{1}{UA_{a,evap}} + \frac{1}{UA_{r,evap,tp}} \right)^{-1}, \tag{C.1}$$

$$NTU_{overall,evap,tp,wet} = \frac{1}{\dot{m}_{a,evap}} \left(\frac{c_{pa,evap}}{UA^*_{a,evap}} + \frac{c_{s,evap}}{UA_{r,evap,tp}} \right)^{-1}, \tag{C.2}$$

$$NTU_{a,evap,tp,dry} = \frac{UA_{a,evap}}{\dot{m}_{a,evap} c_{pa,evap}} \tag{C.3}$$

and

$$NTU_{a,evap,tp,wet} = \frac{UA^*_{a,evap}}{\dot{m}_{a,evap} c_{pa,evap}}. \tag{C.4}$$

In the beginning, the coil is solved assuming a completely dry coil and the heat transfer rate and other properties at the coil can be calculated by

$$\varepsilon_{overall,evap,tp,dry} = 1 - exp(-NTU_{overall,evap,tp,dry}), \tag{C.5}$$

$$\dot{Q}_{evap,tp,dry} = \varepsilon_{overall,evap,tp,dry} w_{r,evap,tp} \dot{m}_{a,evap} c_{pa,evap} (T_{a,evap,in} - T_{r,evap,sat}), \quad \text{(C.6)}$$

$$\dot{Q}_{evap,tp,wet} = 0 \text{ [kW]}, \quad \text{(C.7)}$$

$$T_{a,evap,out} = T_{a,evap,in} - \frac{\dot{Q}_{evap,tp,dry}}{w_{r,evap,tp} \dot{m}_{a,evap} c_{pa,evap}}, \quad \text{(C.8)}$$

$$T_A = \frac{UA_{r,evap,tp} T_{r,evap,sat} + w_{r,evap,tp} UA_{a,evap} T_{a,evap,in}}{UA_{r,evap,tp} + w_{r,evap,tp} UA_{a,evap}} \quad \text{(C.9)}$$

and

$$T_B = \frac{UA_{r,evap,tp} T_{r,evap,sat} + w_{r,evap,tp} UA_{a,evap} T_{a,evap,out}}{UA_{r,evap,tp} + w_{r,evap,tp} UA_{a,evap}}. \quad \text{(C.10)}$$

If T_B in Eqn. (C.10) is higher than T_d, the coil surface is not covered with condensate and the dry coil assumption is accepted. The answers from Eqns. (C.5) to (C.9) become the solution of the two-phase section.

If T_A in Eqn. (C.9) is lower than T_d, the entire two-phase section is covered with condensate and the coil should be completely wet. The dry portion of the coil $f_{dry,tp}$ becomes 0, $\dot{Q}_{evap,tp,dry}$ is set to zero and $T_{d,a}$ becomes equivalent to $T_{a,in}$. If both conditions are not achieved, the two-phase section should be partially wet.

The solution of the dry portion of the coil is given by

$$T_{d,a} = T_d + \frac{UA_{r,evap,tp}}{UA_{a,evap}}(T_d - T_{r,evap,sat}), \tag{C.11}$$

$$\varepsilon_{overall,evap,tp,dry} = \frac{T_{a,in} - T_{d,a}}{T_{a,in} - T_{r,evap,sat}}, \tag{C.12}$$

$$f_{dry,tp} = -\frac{1}{NTU_{overall,evap,tp,dry}} ln(1 - \varepsilon_{overall,evap,tp,dry}) \tag{C.13}$$

and

$$\dot{Q}_{evap,tp,dry} = w_{r,evap,tp}\dot{m}_{a,evap}c_{pa,evap}(T_{a,in} - T_{d,a}). \tag{C.14}$$

For both partially wet and fully wet cases, the solution of the wet coil is given by

$$\varepsilon_{overall,evap,tp,wet} = 1 - exp(-(1 - f_{dry,tp})NTU_{overall,evap,tp,wet}) \tag{C.15}$$

and

$$\dot{Q}_{evap,tp,wet} = \varepsilon_{overall,evap,tp,wet} w_{evap,tp}\dot{m}_{a,evap}(h_a(T_{a,in}, T_d) - h_{a,sat}(T_{r,evap,sat})). \tag{C.16}$$

The total heat transfer rate of the two-phase section is given by the sum of $\dot{Q}_{evap,tp,dry}$ and $\dot{Q}_{evap,tp,wet}$, and the enthalpy at the outlet of the two-phase section is obtained by

$$h_{a,out} = h_{a,in} - \frac{\dot{Q}_{evap,tp,dry} + \dot{Q}_{evap,tp,wet}}{w_{evap,tp}\dot{m}_{a,evap}}. \tag{C.17}$$

The outlet temperature on the air side in the presence of the wet portion is given by

$$h_{a,sat}(T_{ss,evap,tp}) = h_{a,in} - \frac{h_a(T_{a,in}, T_d) - h_{a,out}}{1 - exp(-(1 - f_{dry,tp}))NTU_{a,evap,tp,wet}} \tag{C.18}$$

and

$$T_{a,out} = T_{ss,evap,tp} + (T_{a,d} - T_{ss,evap,tp})exp(-(1 - f_{dry,tp})NTU_{a,evap,tp,dry}). \tag{C.19}$$

C.2 Superheated Section

In the superheated section, the characteristics of the dry and wet portion (capacitance rate, capacity ratio and NTU) are defined by

$$Capr_{evap,sh} = min(\dot{m}_r c_{pr,sh}, w_{evap,sh}\dot{m}_a c_{pa,evap}), \tag{C.20}$$

$$Capr_{wet,evap,sh} = min(\dot{m}_r \frac{c_{pr,evap,sh}}{c_{s,evap,sh}}, w_{evap,sh}\dot{m}_a), \tag{C.21}$$

$$Cr_{evap,sh} = \frac{Capr_{evap,sh}}{max(\dot{m}_r c_{pr,sh}, w_{evap,sh}\dot{m}_a c_{pa,evap})}, \tag{C.22}$$

$$Cr_{wet,evap,sh} = \frac{Capr_{wet,evap,sh}}{max(\dot{m}_r \frac{c_{pr,evap,sh}}{c_{s,evap,sh}}, w_{evap,sh}\dot{m}_a)}, \tag{C.23}$$

$$c_{s,evap,sh} = \frac{dh_{a,sat}(0.5(T_{r,in} + T_{r,out}))}{dT}, \tag{C.24}$$

$$NTU_{a,evap,sh,dry} = \frac{UA_{a,evap}}{\dot{m}_{a,evap} c_{pa,evap}}, \tag{C.25}$$

$$NTU_{a,evap,sh,wet} = \frac{UA^*_{a,evap}}{\dot{m}_{a,evap} c_{pa,evap}}, \tag{C.26}$$

$$NTU_{r,evap,sh} = \frac{UA_{r,evap,sh}}{\dot{m}_r c_{pr,evap,sh}}, \tag{C.27}$$

$$NTU_{overall,evap,sh,dry} = \frac{1}{Capr_{evap,sh}}(\frac{1}{UA_{a,evap}} + \frac{1}{UA_{r,evap,sh}})^{-1} \tag{C.28}$$

and

$$NTU_{overall,evap,sh,wet} = \begin{cases} \frac{NTU_{a,evap,sh,dry}}{1+Cr_{wet,evap,sh}\frac{NTU_{a,evap,sh,wet}}{NTU_{r,evap,sh}}} & \text{if } \frac{\dot{m}_r c_{pr,evap,sh}}{w_{evap,sh}\dot{m}_a c_{s,evap,sh}} > 1 \\ \frac{NTU_{r,evap,sh}}{1+Cr_{wet,evap,sh}\frac{NTU_{r,evap,sh}}{NTU_{a,evap,sh,wet}}} & \text{otherwise} \end{cases}. \tag{C.29}$$

In the beginning, the superheated section is assumed to be completely dry. The dry coil analysis is conducted by solving

$$\varepsilon_{overall,evap,sh,dry} = \frac{1 - exp(-NTU_{overall,evap,sh,dry}(1 - Cr_{evap,sh,dry}))}{1 - Cr_{evap,sh,dry} exp(-NTU_{overall,evap,sh,dry}(1 - Cr_{evap,sh,dry}))}, \tag{C.30}$$

$$\dot{Q}_{evap,sh,dry} = \varepsilon_{overall,evap,sh,dry} Capr_{evap,sh}(T_{a,in} - T_{r,in}), \tag{C.31}$$

$$T_{a,out} = T_{a,in} - \frac{\dot{Q}_{evap,sh,dry}}{w_{evap,sh}\dot{m}_{a,evap}c_{pa,evap}}, \tag{C.32}$$

$$h_{a,out} = h_{a,in} - \frac{\dot{Q}_{evap,sh,dry}}{w_{evap,sh}\dot{m}_{a,evap}} \tag{C.33}$$

and

$$T_{r,out} = T_{r,in} + w_{evap,sh}\frac{\dot{m}_{a,evap}c_{pa,evap}}{\dot{m}_r c_{pr,sh}}(T_{a,in} - T_{a,out}). \tag{C.34}$$

The surface temperature at the air outlet of the dry coil analysis is computed by

$$T_B = \frac{UA_{r,evap,sh}T_{r,in} + w_{evap,sh}UA_{a,evap}T_{a,out}}{UA_{r,evap,sh} + w_{evap,sh}UA_{a,evap}}. \tag{C.35}$$

If T_B from Eqn. (C.35) is higher than T_d, the dry-coil assumption is accepted and the solution of the dry-coil analysis is output as the solution of the superheated

section with $Q_{evap,sh} = Q_{evap,sh,dry}$. Otherwise, the superheated section is assumed to be completely wet and the wet coil analysis is listed in

$$\varepsilon_{overall,evap,sh,wet} = \frac{1 - exp(-NTU_{overall,evap,sh,wet}(1 - Cr_{evap,sh,wet}))}{1 - Cr_{evap,sh,wet} exp(-NTU_{overall,evap,sh,wet}(1 - Cr_{evap,sh,wet}))}, \tag{C.36}$$

$$\dot{Q}_{evap,sh,wet} = \varepsilon_{overall,evap,sh,wet} Capr_{wet,evap,sh}(h_{a,in} - h_{a,sat}(T_{r,in})), \tag{C.37}$$

$$h_{a,out} = h_{a,in} - \frac{\dot{Q}_{evap,sh,wet}}{w_{evap,sh}\dot{m}_{a,evap}} \tag{C.38}$$

and

$$T_{r,out} = T_{r,in} + \frac{w_{evap,sh}\dot{m}_{a,evap}}{\dot{m}_r c_{pr,sh}}(h_{a,in} - h_{a,out}). \tag{C.39}$$

As the wet-coil analysis requires $c_{s,evap,sh}$ and hence $T_{r,out}$ as inputs, the solution is implicit and the equations are iterated by Newton's method until the independent variable $T_{r,out}$ matches the result of Eqn. (C.39). To examine if the wet-coil assumption is acceptable, T_A is obtained by solving

$$c_{s,evap,sh,local} = \frac{dh_{a,sat}(0.5(T_{r,out} + T_{a,in}))}{dT}, \tag{C.40}$$

$$UA_{evap,sh,local} = (\frac{c_{pa,evap}}{UA^*_{a,evap}} + \frac{c_{s,evap,sh,local}}{UA_{r,evap,sh}})^{-1} \tag{C.41}$$

and

$$T_A = T_{r,out} + \frac{UA_{evap,sh,local}}{UA_{r,evap,sh}}(h_{a,in} - h_{a,sat}(T_{r,out})). \tag{C.42}$$

If T_A from Eqn. (C.42) is lower than T_d, condensate covers the entire superheated section and the solution of the wet-coil analysis should be the solution of the superheated section. Otherwise, the superheated section should be partially wet, and the partial-wet-partial-dry analysis should be conducted.

In the partial-wet-partial-dry analysis of the superheated section, the $f_{dry,sh}$ is iterated between 0 to 1 to solve

$$K_{dry} = NTU_{overall,evap,sh,dry}(1 - Cr_{evap,sh}), \tag{C.43}$$

$$expk = exp(-K_{dry} f_{dry,sh}), \tag{C.44}$$

$$T_{r,out,1} = \begin{cases} \frac{T_d + Cr_{evap,sh}(T_{a,in} - T_d) - (expk)(1 - \frac{K_{dry}}{NTU_{a,evap,sh,dry}})T_{a,in}}{1 - (expk)(1 - \frac{K_{dry}}{NTU_{a,evap,sh,dry}})} & \text{if } \frac{\dot{m}_r c_{pr,evap,sh}}{\dot{m}_a c_{pa,evap} w_{evap,sh}} > 1 \\ \frac{((expk)(T_{a,in} + (Cr_{evap,sh} - 1)T_d) - Cr_{evap,sh}(1 + \frac{K_{dry}}{NTU_{a,evap,sh,dry}})T_{a,in})}{(expk Cr_{evap,sh} - Cr_{evap,sh}(1 + \frac{K_{dry}}{NTU_{a,evap,sh,dry}}))} & \text{otherwise} \end{cases}, \tag{C.45}$$

$$\varepsilon_{overall,evap,sh,dry} = \frac{1 - exp(-f_{dry,sh} NTU_{overall,evap,sh,dry}(1 - Cr_{evap,sh}))}{1 - Cr_{evap,sh} exp(-f_{dry,sh} NTU_{overall,evap,sh,dry}(1 - Cr_{evap,sh}))}, \tag{C.46}$$

$$\begin{aligned} & \varepsilon_{overall,evap,sh,wet} \\ = & \frac{1 - exp(-(1 - f_{dry,sh}) NTU_{overall,evap,sh,wet}(1 - Cr_{wet,evap,sh}))}{1 - Cr_{wet,evap,sh} exp(-(1 - f_{dry,sh}) NTU_{overall,evap,sh,wet}(1 - Cr_{wet,evap,sh}))} \end{aligned}, \tag{C.47}$$

$$\begin{aligned} & T_{d,r} \\ = & \frac{T_{r,in} + \frac{Capr_{wet,evap,sh}}{\dot{m}_r c_{pr,evap,sh}} \varepsilon_{overall,evap,sh,wet}(h_{a,in} - h_{a,sat})(T_{r,in}) - \varepsilon_{overall,evap,sh,dry} \frac{Capr_{evap,sh}}{\dot{m}_{a,evap}} T_{a,in}}{1 - \frac{(Capr_{evap,sh} Capr_{wet,evap,sh})}{\dot{m}_r cp_{r,evap,sh} \dot{m}_{a,evap}} \varepsilon_{overall,evap,sh,dry} \varepsilon_{overall,evap,sh,wet}}, \end{aligned} \tag{C.48}$$

$$T_{d,a} = T_{a,in} - \varepsilon_{overall,evap,sh,dry} Capr_{evap,sh} \frac{T_{a,in} - T_{d,r}}{\dot{m}_{a,evap} c_{pa,evap}}, \tag{C.49}$$

and

$$T_{r,out,2} = \frac{Capr_{evap,sh}}{\dot{m}_r c_{pr,evap,sh}} \varepsilon_{overall,evap,sh,dry} T_{a,in} + (1 - \frac{Capr_{evap,sh}}{\dot{m}_r c_{pr,evap,sh}} \varepsilon_{overall,evap,sh,dry}) T_{d,r}, \tag{C.50}$$

where their derivation can be found in the ACHP program website (Bell, 2010).

The iteration of $f_{dry,sh}$ continues until the results from Eqns. (C.45) and (C.50) match each other. The heat transfer rate of the superheated section and the temperature at the air outlet of the phase section is calculated by solving

$$\dot{Q}_{evap,sh} = \dot{m}_r c_{pr,evap,sh} (T_{r,out} - T_{r,in}), \tag{C.51}$$

$$h_{a,out} = h_{a,in} - \frac{\dot{Q}_{evap,sh}}{w_{evap,sh} \dot{m}_{a,evap}}, \tag{C.52}$$

$$h_{d,a} = h_{a,in} - c_{pa,evap} (T_{a,in} - T_{d,a}), \tag{C.53}$$

$$h_{a,sat}(T_{ss,evap,sh}) = h_{d,a} + \frac{h_{a,out} - h_{d,a}}{1 - exp(-(1 - f_{dry,sh}) NTU_{a,evap,sh,wet})} \tag{C.54}$$

and

$$T_{a,out} = T_{ss,evap,sh} + (T_{d,a} - T_{ss,evap,sh}) exp(-(1 - f_{dry,sh}) NTU_{a,evap,sh,dry}). \tag{C.55}$$

APPENDIX D. INTEGRATION OF MULTIPLIER FOR HEAT TRANSFER COEFFICIENT IN EVAPORATING FLOW FROM SHAH (1982)

The calculation method of variable Φ(x), which helps solving the heat transfer conductance of evaporating flow, is given by

$$\phi_i(x) = max(\phi_{cb,i}(x), \phi_{nb,i}(x), \phi_{bs,i}(x)), \tag{D.1}$$

$$\phi_{cb,i}(x) = 1.8Co_i^{-0.8}, \tag{D.2}$$

$$\phi_{nb,i}(x) = \begin{cases} 230Bo_i^{\frac{1}{2}} & \text{if } Co_i(x) > 1 \text{ and } Bo_i > .3 \text{ x } 10^{-4} \\ 1 + 46Bo_i^{\frac{1}{2}} & \text{if } Co_i(x) > 1 \text{ and } Bo_i \leq .3 \text{ x } 10^{-4} \\ 0 & \text{otherwise} \end{cases}, \tag{D.3}$$

$$\phi_{bs,i}(x) = \begin{cases} Bo_i^{\frac{1}{2}} F_i exp(2.74Co_i^{-0.1}) & \text{if } 0.1 < Co_i(x) \leq 1 \\ Bo_i^{\frac{1}{2}} F_i exp(2.47Co_i^{-0.15}) & \text{if } Co_i(x) \leq 0.1 \\ 0 & \text{otherwise} \end{cases}, \tag{D.4}$$

$$Co_i(x) = (\frac{1}{x} - 1)^{0.8} (\frac{\rho_{v,i}}{\rho_{f,i}})^{\frac{1}{2}}, \tag{D.5}$$

$$F_i = \begin{cases} 14.7 & \text{if } Bo_i \geq 11 \text{ x } 10^{-4} \\ 15.43 & \text{otherwise} \end{cases} \tag{D.6}$$

and

$$Bo_i = \frac{h_{v,i} - h_{r,in,i}}{h_{v,i} - h_{l,i}}. \tag{D.7}$$

APPENDIX E. DEFINITION OF Π GROUPS, VALUES OF EMPIRICAL COEFFICIENTS AND THE TWO-PHASE FLOW MULTIPLIER (PAYNE AND O'NEAL, 2004)

The values of coefficients and definitions of the Π groups are shown in Table E.1 and Table E.2.

Table E.1. Coefficients in the mass flow rate model in Payne and O'Neal (2004).

Coefficients	Values
$a_{EXV,1}$	0.38811
$a_{EXV,2}$	11.427
$a_{EXV,3}$	-14.194
$a_{EXV,4}$	1.0703
$a_{EXV,5}$	-0.09193
$a_{EXV,6}$	21.425
$a_{EXV,7}$	-581.95

Table E.2. Definition of Π groups.

Pi groups	Definition
Π_3	$\frac{P_{r,EXV,in}-P_{sat}(T_{r,EXV,in})}{P_{r,cric}}$
Π_6	$\frac{\rho_{r,v}(P_{r,EXV,in})}{\rho_{r,l}(P_{r,EXV,in})}$
Π_9	$\frac{T_{r,sat}(P_{r,EXV,in})-T_{r,EXV,in}}{T_{r,cric}}$

To model the expansion valve mass flow rate with two-phase flow entering the valve, a multiplier is added to the original model. The final mass flow rate is a product of the multiplier and the mass flow rate modeled with saturated liquid at the expansion valve inlet as shown in

$$\dot{m}_{r,EXV}(x_{r,EXV,in} < 0) = C_{EXV,tp,in}(x_{r,EXV,in})\dot{m}_{r,EXV,in}(x_{r,EXV,in} = 0). \tag{E.1}$$

The adjustment coefficient in Eqn. (E.1) is given by

$$\rho_{EXV,tp} = (\frac{1 - x_{r,EXV,in}}{\rho_{l,EXV,in}} + \frac{x_{r,EXV,in}}{\rho_{v,EXV,in}})^{-1}, \tag{E.2}$$

$$\Pi_{tp,6} = \frac{\rho_{EXV,tp}}{\rho_{l,EXV,tp}}, \tag{E.3}$$

$$\Pi_{tp,28} = \frac{P_{EXV,in}}{P_{r,cric}}, \tag{E.4}$$

$$\Pi_{tp,32} = 1 - \frac{P_{EXV,in}}{P_{r,cric}}, \tag{E.5}$$

$$\Pi_{tp,34} = \frac{x_{r,EXV,in}}{1 - x_{r,EXV,in}}(\frac{\rho_{r,EXV,l}}{\rho_{r,EXV,v}})^{0.5}, \tag{E.6}$$

$$\Pi_{tp,35} = 1 - \frac{P_{r,sat}(T_{r,EXV,in})}{P_{r,cric}} \tag{E.7}$$

and

$$C_{EXV,tp,in} = \frac{\left(\begin{array}{c} b_{EXV,1}\Pi_{tp,6} + b_{EXV,2}\Pi_{tp,6}^2 + b_{EXV,3}(ln(\Pi_{tp,6}))^2 \\ +b_{EXV,4}(ln(\Pi_{tp,35}))^2 + b_{EXV,5}(\ln(\Pi_{tp,32}))^2 + b_{EXV,6}(\ln(\frac{L_{EXV}}{D_{EXV}}))^2 \end{array} \right)}{1 + b_{EXV,7}\Pi_{tp,6} + b_{EXV,8}\Pi_{tp,34} + b_{EXV,9}\Pi_{tp,28}^3}. \tag{E.8}$$

The empirical coefficients in Eqn. (E.8) can be found from Payne and O'Neal (2004) as listed in Table E.3.

Table E.3. Coefficients for two-phase flow adjustment factor in the mass flow rate model in Payne and O'Neal (2004).

Coefficients	Values
$b_{EXV,1}$	1.18E+00
$b_{EXV,2}$	-1.47E+00
$b_{EXV,3}$	-1.53E-01
$b_{EXV,4}$	-1.46E+01
$b_{EXV,5}$	9.84E+00
$b_{EXV,6}$	-1.98E-02
$b_{EXV,7}$	-1.53E+00

APPENDIX F. COMPONENT MODEL PARAMETERS

This appendix shows the regression parameters and the unit-specific normalization parameters of various component models of different systems.

Table F.1. Parameters of compressor mass flow rate model.

System	$C_{comp,0}$	$C_{comp,1}$	$C_{comp,2}$
I	4.693e-03	-9.233e-04	0.000e+00
II	4.474e-03	-1.104e-04	0.000e+00
III	1.967e-03	-1.229e-04	0.000e+00
IV	2.245e-03	-2.259e-04	0.000e+00
V	2.245e-03	-2.259e-04	0.000e+00
VI	5.313e-03	-1.124e-04	-5.898e-08
VII	1.516e-03	-5.646e-06	-6.651e-08
VIII	1.934e-03	-7.203e-05	-5.021e-08
IX	4.139e-03	-5.547e-04	0.000e+00
X	2.950e-03	-7.700e-05	-1.149e-07
XI	1.740e-03	-9.404e-05	0.000e+00

Table F.2. Parameters of compressor power consumption model.

System	$C_{comp,3}$	$C_{comp,4}$	$C_{comp,5}$	$C_{comp,6}$	$C_{comp,7}$	$P_{r,comp,in,rated}$
I	1.767e+00	-1.161e+00	1.468e-02	-7.489e+01	-3.290e+00	4.944e+02
II	1.724e+01	1.402e+01	-2.809e-03	-3.073e+01	-2.377e-03	6.550e+02
III	2.809e+00	-2.252e+00	-1.025e-01	-1.102e-02	6.396e-01	9.657e+02
IV	7.622e-01	-1.092e-01	-3.207e-01	-1.253e+00	-1.489e+00	9.778e+02
V	7.622e-01	-1.092e-01	-3.207e-01	-1.253e+00	-1.489e+00	9.778e+02
VI	1.287e+00	-5.113e-01	-5.431e-01	-1.305e-01	1.590e-01	5.391e+02
VII	7.474e-01	-1.117e-01	-1.193e+00	-4.458e-08	3.880e+00	9.250e+02
VIII	1.342e+00	-7.306e-01	-3.579e-01	-3.085e-02	4.458e-01	1.005e+03
IX	6.266e-01	-3.575e+00	-5.252e+00	-1.323e-06	2.304e+00	5.877e+02
X	7.973e-01	-3.338e-01	-2.115e+00	-1.277e-02	4.838e-01	6.024e+02
XI	7.183e-01	2.220e-01	-1.071e+01	-9.116e-10	6.399e+00	1.195e+03

Table F.3. Parameters of compressor heat loss model.

System	$C_{comp,8}$	$C_{comp,9}$
I	1.699e-03	1.218e+01
II	2.561e+01	3.314e+01
III	9.157e-04	8.047e+00
IV	1.387e-06	3.850e+00
V	1.387e-06	3.850e+00
VI	5.911e+00	9.021e+00
VII	2.076e-04	2.549e+01
VIII	5.052e-03	2.774e+00
IX	2.054e-04	4.846e+01
X	1.591e-04	1.122e+01
XI	6.750e+00	1.471e-02

Table F.4. Parameters of hot gas line presusre drop model.

System	$C_{pipeline,\Delta P,1}$	$C_{pipeline,\Delta P,2}$	$C_{pipeline,\Delta P,3}$
I	1.033e+02	2.000e+00	0.000e+00
II	0.000e+00	1.000e+00	0.000e+00
III	2.801e+01	1.000e+00	1.000e+00
IV	2.304e+01	1.000e+00	1.000e+00
V	2.304e+01	1.000e+00	1.000e+00
VI	8.910e+01	2.000e+00	1.000e+00
VII	5.864e+00	1.000e+00	6.464e-01
VIII	2.209e+01	1.000e+00	1.000e+00
IX	3.495e+00	1.000e+00	0.000e+00
X	2.147e+01	2.000e+00	3.413e-01
XI	0.000e+00	1.000e+00	0.000e+00

Table F.5. Parameters of hot gas line heat loss model.

System	$C_{pipeline,\Delta h,1}$	$C_{pipeline,\Delta h,2}$
III	8.057e-01	5.800e-02
IV	1.013e+00	5.800e-02
V	1.013e+00	5.800e-02
VI	5.456e-03	5.800e-02
VII	1.287e+00	9.090e-02
VIII	1.118e+00	5.800e-02
IX	1.356e+00	5.800e-02
X	3.125e+00	2.955e-01

Table F.6. Miscellaneous parameters of hot gas line model.

System	$\dot{m}_{r,pipeline,rated}$	$\Delta T_{pipeline,rated}$	$\Delta h_{r,pipeline,rated}$	$\rho_{r,pipeline,rated}$	$\mu_{r,pipeline,rated}$
I	5.381e-02	0.000e+00	0.000e+00	4.995e+01	1.558e-05
II	1.124e-01	0.000e+00	0.000e+00	6.027e+01	1.548e-05
III	6.171e-02	4.024e+01	6.313e+03	9.954e+01	1.672e-05
IV	6.401e-02	4.636e+01	1.202e+04	9.819e+01	1.711e-05
V	6.401e-02	4.636e+01	1.202e+04	9.819e+01	1.711e-05
VI	1.057e-01	5.492e+01	1.705e-03	9.134e+01	1.574e-05
VII	4.771e-02	3.546e+01	2.460e+03	8.345e+01	1.627e-05
VIII	6.380e-02	4.319e+01	3.731e+03	9.275e+01	1.693e-05
IX	7.385e-02	6.645e+01	3.985e+03	6.184e+01	1.560e-05
X	7.095e-02	4.407e+01	1.015e+03	6.053e+01	1.550e-05
XI	6.461e-02	0.000e+00	0.000e+00	8.262e+01	1.634e-05

Table F.7. Regression parameters of condenser heat transfer rate model.

System	$U_{a,cond,rated}$	$n_{a,cond,rated}$	$U_{r,cond,sh,rated}$	$U_{r,cond,tp,rated}$	$U_{r,cond,sc,rated}$
I	6.283e+02	8.400e-01	1.580e+03	2.182e+04	8.776e+02
II	1.876e+03	4.000e-01	2.920e+03	3.042e+03	3.681e+02
III	2.970e+03	8.400e-01	3.328e+02	5.255e+03	6.403e+02
IV	1.595e+03	7.406e-01	4.569e+02	9.424e+03	2.223e+03
V	1.595e+03	7.406e-01	4.569e+02	9.424e+03	2.223e+03
VI	1.197e+03	4.000e-01	1.406e+03	2.821e+03	3.683e+02
VII	1.549e+03	8.400e-01	5.462e+02	7.803e+04	8.172e+03
VIII	9.791e+02	8.400e-01	1.125e+03	2.656e+04	1.627e+02
IX	7.941e+02	8.400e-01	2.104e+03	2.522e+04	2.504e+03
X	1.002e+03	8.400e-01	2.135e+03	2.281e+04	6.599e+02
XI	2.625e+03	8.400e-01	1.252e+03	3.545e+03	3.545e+03

Table F.8. Unit-specific normalization parameters of condenser heat transfer rate model.

System	$\dot{m}_{a,cond,rated}$	$A_{r,cond,rated}$	$\dot{m}_{r,cond,rated}$	$K_{r,cond,sh}$	$K_{r,cond,sc}$	D_{cond}
I	1.937e+00	1.925e+00	5.381e-02	7.903e+02	1.518e+02	8.490e-03
II	2.250e+00	2.912e+00	1.124e-01	1.380e+03	3.260e+02	8.890e-03
III	1.266e+00	1.791e+00	6.171e-02	9.014e+02	2.500e+02	8.530e-03
IV	1.438e+00	1.925e+00	6.401e-02	9.294e+02	2.537e+02	8.490e-03
V	1.438e+00	1.925e+00	6.401e-02	9.294e+02	2.537e+02	8.490e-03
VI	2.038e+00	2.786e+00	1.057e-01	1.326e+03	3.399e+02	6.939e-03
VII	1.264e+00	1.501e+00	4.771e-02	7.447e+02	1.801e+02	4.200e-03
VIII	1.966e+00	1.925e+00	6.372e-02	9.223e+02	2.654e+02	8.490e-03
IX	1.653e+00	1.925e+00	7.373e-02	9.900e+02	2.257e+02	8.490e-03
X	1.680e+00	2.778e+00	7.074e-02	9.592e+02	2.163e+02	8.490e-03
XI	2.045e+00	3.560e+00	6.461e-02	9.313e+02	2.752e+02	6.400e-03

Table F.9. Parameters of condenser pressure drop model.

System	$C_{r,cond,\Delta P,sh}$	$C_{r,cond,\Delta P,tp,1}$	$C_{r,cond,\Delta P,tp,2}$	$C_{r,cond,\Delta P,sc}$	$C_{r,cond,\Delta P,acc}$
I	4.400e+02	7.971e+02	-2.947e+00	1.176e+02	1.191e+02
II	2.082e+03	6.883e+01	-3.711e+01	9.347e+01	4.805e+01
III	1.691e+02	1.276e+02	-6.267e+00	3.560e+01	0.000e+00
IV	5.490e+02	9.781e+01	-1.086e+01	2.850e+01	0.000e+00
V	5.490e+02	9.781e+01	-1.086e+01	2.850e+01	0.000e+00
VI	1.582e+02	7.418e+02	2.422e-01	6.234e+01	0.000e+00
VII	5.550e+02	1.509e+02	-3.902e+00	6.389e+01	6.751e+01
VIII	4.975e+02	4.072e+01	1.134e+01	4.035e+01	0.000e+00
IX	5.181e+02	1.811e+02	1.111e+01	1.590e+02	0.000e+00
X	2.444e+03	2.363e+03	-4.885e+00	3.639e+02	5.803e+02
XI	6.945e+02	1.540e+02	-1.828e+00	4.108e+01	1.384e+01

Table F.10. Unit-specific normalization parameters of condenser pressure drop model.

System	$\rho_{r,cond,rated}$	$\rho_{r,cond,l}$	$\mu_{r,cond,v,rated}$
I	4.591e+01	1.166e+03	1.314e-05
II	6.027e+01	1.146e+03	1.374e-05
III	1.028e+02	9.487e+02	1.601e-05
IV	1.046e+02	9.080e+02	1.627e-05
V	1.046e+02	9.080e+02	1.627e-05
VI	8.718e+01	1.069e+03	1.463e-05
VII	8.461e+01	9.923e+02	1.525e-05
VIII	9.450e+01	9.075e+02	1.632e-05
IX	6.335e+01	7.675e+02	1.384e-05
X	6.047e+01	1.035e+03	1.376e-05
XI	8.262e+01	9.611e+02	1.527e-05

Table F.11. Parameters of condenser fan power consumption model.

System	$C_{cond,fan,1}$	$C_{cond,fan,2}$
II	9.725e+02	2.473e+02
III	2.876e+02	1.308e+02
IV	2.461e+02	1.009e+02
V	2.461e+02	1.009e+02
VI	3.650e+02	1.001e+02
VII	3.151e+02	1.550e+02
VIII	2.265e+02	6.662e+01
X	2.800e+02	9.561e+01
XI	2.723e+02	7.624e+01

Table F.12. Parameters of liquid line presusre drop model.

System	$C_{pipeline,\Delta P,1}$	$C_{pipeline,\Delta P,2}$	$C_{pipeline,\Delta P,3}$
I	8.988e+00	2.000e+00	0.000e+00
II	1.561e+01	2.000e+00	0.000e+00
III	1.246e+01	1.269e+00	0.000e+00
IV	2.742e+00	1.038e+00	0.000e+00
V	2.742e+00	1.038e+00	0.000e+00
VI	3.212e+01	2.000e+00	0.000e+00
VII	7.583e+01	1.238e+00	0.000e+00
VIII	8.389e+00	1.000e+00	0.000e+00
IX	2.264e+01	2.000e+00	0.000e+00
X	5.743e+01	1.000e+00	0.000e+00
XI	1.082e+01	2.000e+00	0.000e+00

Table F.13. Parameters of liquid line heat loss model.

System	$C_{pipeline,\Delta h,1}$	$C_{pipeline,\Delta h,2}$
II	1.406e+00	5.800e-02
III	6.528e+00	5.800e-02
IV	1.121e+00	5.800e-02
V	1.121e+00	5.800e-02
VI	1.199e+00	5.800e-02
VII	3.775e+00	2.838e-01
VIII	1.263e+00	5.800e-02
IX	1.968e+00	1.897e-01
X	2.008e+00	2.003e-01

Table F.14. Miscellaneous parameters of liquid line model.

System	$\dot{m}_{r,pipeline,rated}$	$\Delta T_{pipeline,rated}$	$\Delta h_{r,pipeline,rated}$	$\rho_{r,pipeline,rated}$	$\mu_{r,pipeline,rated}$
I	5.381e-02	0.000e+00	0.000e+00	1.176e+03	1.585e-04
II	1.124e-01	6.512e+00	2.964e+02	1.149e+03	1.467e-04
III	6.171e-02	5.194e+00	3.328e+01	9.728e+02	9.589e-05
IV	6.401e-02	4.378e+00	9.065e+02	9.652e+02	9.401e-05
V	6.401e-02	4.378e+00	9.065e+02	9.652e+02	9.401e-05
VI	1.057e-01	4.083e+00	1.901e+01	1.067e+03	1.273e-04
VII	4.771e-02	4.538e+00	1.535e+03	1.003e+03	1.027e-04
VIII	6.020e-02	4.317e+00	1.040e+03	9.815e+02	9.832e-05
IX	6.640e-02	8.633e+00	1.959e+03	1.144e+03	1.448e-04
X	6.633e-02	5.927e+00	3.467e+02	1.130e+03	1.404e-04
XI	6.461e-02	0.000e+00	0.000e+00	1.012e+03	1.052e-04

Table F.15. Parameters of FXO mass flow rate model.

System	D_{EXV}	L_{EXV}	$C_{EXV,1}$	$C_{EXV,2}$
I	1.869e-03	1.176e-02	-1.182e+03	-1.675e+01
III	1.835e-03	4.041e-02	-6.022e+02	-1.388e+01
IV	1.786e-03	1.390e-02	-1.158e+03	-1.682e+01
VI	2.560e-03	3.555e-02	-7.295e+02	-1.736e+01
IX	2.037e-03	2.464e-02	-5.488e+02	-1.285e+01

Table F.16. Parameters of TXV mass flow rate model.

System	L_{EXV}	$C_{TXV,1}$	$C_{TXV,2}$	$C_{TXV,3}$	$C_{TXV,4}$
I	4.664e-09	1.064e-06	1.891e-06	-2.465e-24	1.710e+02
II	7.801e-03	2.435e-05	1.386e-05	-0.000e+00	1.358e+02
III	1.133e-02	1.726e-07	2.180e-07	-0.000e+00	2.748e+02
IV	4.988e-06	7.889e-07	1.146e-06	-2.067e-09	1.697e+02
V	6.362e-13	7.613e-07	7.938e-07	1.101e-28	3.781e+01
VI	5.406e-13	1.313e-05	-1.520e-05	6.169e-06	3.263e+02

Table F.17. Unit-specific normalization parameters of TXV mass flow rate model.

System	$C_{EXV,1}$	$C_{EXV,2}$	$C_{EXV,3}$	$C_{EXV,4}$	$C_{EXV,5}$
I	-1.321e+02	-1.505e+01	-6.054e-03	1.215e+01	1.562e+00
II	-4.231e+03	-1.993e+01	-1.216e+00	2.790e+02	-2.863e-01
III	-2.394e+01	6.501e+01	1.317e-01	1.460e+01	6.325e+00
IV	-4.363e+02	-1.728e+01	-2.997e-02	1.771e+01	1.109e+00
V	-8.177e+02	-1.245e+01	-2.715e-02	1.100e+01	1.103e+00
VI	-2.278e+02	6.320e+01	3.959e-02	1.398e+02	7.111e+00

Table F.18. Parameters of evaporator heat transfer rate model.

System	$U_{a,evap,rated}$	$U_{r,evap,sh,rated}$	$C_{a,evap,1}$	$\frac{U_{r,evap,tp,rated}}{K_{r,evap,tp}}$	$U_{r,evap,sh,rated}$
I	1.466e+03	3.360e-01	4.454e-02	1.214e+02	3.443e+02
II	1.585e+03	1.600e-01	3.178e-02	1.329e+03	3.451e+02
III	1.302e+03	3.360e-01	1.141e-02	2.305e+01	4.722e+02
IV	1.214e+03	3.360e-01	1.746e-02	3.170e+01	1.406e+02
V	1.214e+03	3.360e-01	1.746e-02	3.170e+01	1.406e+02
VI	7.036e+02	3.360e-01	1.767e-02	1.106e+01	2.079e+02
VII	2.747e+03	1.668e-01	1.202e-02	2.691e+01	4.886e+02
VIII	1.107e+03	3.360e-01	5.782e-02	1.393e+02	1.181e+02
IX	6.027e+02	3.360e-01	5.002e-02	2.642e+02	1.290e+02
X	1.165e+03	3.360e-01	4.438e-02	1.670e+02	7.775e+01
XI	1.175e+03	3.360e-01	2.840e-02	2.008e+02	2.216e+02

Table F.19. Unit-specific normalization parameters of evaporator heat transfer model model.

System	$\dot{m}_{a,evap,rated}$	$A_{r,evap,rated}$	$K_{r,evap,sh}$	D_{evap}
I	7.362e-01	9.744e-01	8.332e+02	7.747e-03
II	1.278e+00	1.922e+00	1.474e+03	7.747e-03
III	7.125e-01	9.744e-01	9.571e+02	7.747e-03
IV	6.877e-01	9.241e-01	9.848e+02	7.747e-03
V	6.877e-01	9.241e-01	9.848e+02	7.747e-03
VI	1.284e+00	2.638e+00	1.428e+03	7.747e-03
VII	6.133e-01	7.418e-01	7.808e+02	7.747e-03
VIII	5.385e-01	1.756e+00	9.795e+02	7.747e-03
IX	5.256e-01	1.756e+00	1.066e+03	7.747e-03
X	6.732e-01	1.756e+00	1.017e+03	7.747e-03
XI	8.168e-01	1.958e+00	9.827e+02	7.747e-03

Table F.20. Parameters of evaporator pressure drop model.

System	$C_{r,evap,deltaP,tp,1}$	$C_{r,evap,\Delta P,sh}$	$C_{r,cond,\Delta P,acc}$	$C_{r,evap,\Delta P,2}$
I	0.000e+00	2.684e+01	1.585e+00	-1.000e+01
II	6.617e-04	9.246e+00	4.578e+00	-1.000e+01
III	0.000e+00	0.000e+00	4.735e+00	-1.000e+01
IV	0.000e+00	0.000e+00	2.275e+00	-1.000e+01
V	0.000e+00	0.000e+00	2.275e+00	-1.000e+01
VII	3.653e-04	5.712e+01	6.750e+00	-1.000e+01
XI	7.643e-04	1.426e+02	9.898e+00	-1.000e+01

Table F.21. Unit specific normalizaiton parameters of evaporator pressure drop model.

System	$\dot{m}_{r,evap,rated}$	$\rho_{r,evap,out,rated}$	$\rho_{r,evap,rated}$	$\mu_{r,evap,rated}$
I	5.381e-02	2.191e+01	3.391e+02	1.148e-05
II	1.124e-01	2.579e+01	3.585e+02	1.170e-05
III	6.171e-02	4.012e+01	2.987e+02	1.267e-05
IV	6.401e-02	3.768e+01	2.682e+02	1.261e-05
V	6.401e-02	3.768e+01	2.682e+02	1.261e-05
VI	1.057e-01	2.414e+01	2.192e+02	1.154e-05
VII	4.771e-02	3.643e+01	3.384e+02	1.261e-05
VIII	6.372e-02	3.875e+01	3.045e+02	1.267e-05
IX	7.373e-02	2.574e+01	2.561e+02	1.161e-05
X	7.074e-02	2.715e+01	3.124e+02	1.172e-05
XI	6.461e-02	3.924e+01	3.721e+02	1.277e-05

Table F.22. Parameters of evaporator fan power consumption model.

System	$\dot{W}_{evap,fan,NF}C_{evap,fan,1}$	$\frac{\dot{W}_{evap,fan,NF}C_{evap,fan,2}}{\dot{V}_{a,evap,NF}}$	$\frac{\dot{W}_{evap,fan,NF}C_{evap,fan,3}}{\dot{V}^2_{a,evap,NF}}$	$\dot{W}_{evap,fan,NF}$
II	-1.769e+02	9.071e+02	-2.607e+02	4.881e+02
III	-1.812e+02	1.659e+03	-8.514e+02	5.000e+02
IV	-1.631e+02	1.552e+03	-8.282e+02	4.500e+02
V	-1.631e+02	1.552e+03	-8.282e+02	4.500e+02
VI	-3.134e+02	1.605e+03	-4.610e+02	8.649e+02
VII	-1.546e+02	1.655e+03	-9.926e+02	4.267e+02
VIII	-1.808e+02	1.627e+03	-8.208e+02	4.990e+02
IX	-2.432e+02	1.789e+03	-7.378e+02	6.711e+02
X	-1.442e+02	1.175e+03	-5.366e+02	3.979e+02
XI	-1.428e+02	1.156e+03	-5.245e+02	3.942e+02

Table F.23. Parameters of suction line presusre drop model.

System	$C_{pipeline,\Delta P,1}$	$C_{pipeline,\Delta P,2}$	$C_{pipeline,\Delta P,3}$
II	2.160e+00	2.000e+00	0.000e+00
III	1.345e+01	1.133e+00	1.000e+00
IV	1.137e+01	1.327e+00	8.056e-01
V	1.137e+01	1.327e+00	8.056e-01
VII	1.501e+01	1.229e+00	0.000e+00
VIII	2.669e+01	1.202e+00	1.000e+00
IX	3.169e+01	1.000e+00	0.000e+00
X	4.113e+01	2.000e+00	0.000e+00

Table F.24. Parameters of suction line heat loss model.

System	$C_{pipeline,\Delta h,1}$	$C_{pipeline,\Delta h,2}$
II	2.859e-01	3.330e-01
III	7.539e-01	5.800e-02
IV	1.026e+00	2.240e-01
V	1.026e+00	2.240e-01
VI	7.256e-01	5.800e-02
VII	9.440e-01	5.800e-02
VIII	1.415e+00	5.800e-02
IX	1.510e+00	5.800e-02
X	1.511e+00	5.800e-02

Table F.25. Miscellaneous parameters of suction line model.

System	$\dot{m}_{r,pipeline,rated}$	$\Delta T_{pipeline,rated}$	$\Delta h_{r,pipeline,rated}$	$\rho_{r,pipeline,rated}$	$\mu_{r,pipeline,rated}$
I	5.381e-02	0.000e+00	0.000e+00	2.191e+01	1.199e-05
II	1.124e-01	-1.785e+01	-8.898e+02	2.579e+01	1.214e-05
III	6.171e-02	-2.018e+01	-1.620e+03	3.750e+01	1.301e-05
IV	6.401e-02	-1.427e+01	-1.562e+03	3.758e+01	1.277e-05
V	6.401e-02	-1.427e+01	-1.562e+03	3.758e+01	1.277e-05
VI	1.057e-01	-2.575e+01	-9.465e+02	2.382e+01	1.188e-05
VII	4.771e-02	-7.861e+00	-2.096e+03	3.643e+01	1.295e-05
VIII	6.372e-02	-2.175e+01	-1.569e+03	3.875e+01	1.285e-05
IX	7.373e-02	-2.157e+01	-1.356e+03	2.277e+01	1.203e-05
X	7.074e-02	-2.169e+01	-1.414e+03	2.803e+01	1.189e-05
XI	6.461e-02	0.000e+00	0.000e+00	3.924e+01	1.319e-05

Table F.26. Inner volume of different components.

System	Volume of hot gas line	Volume of condenser	Volume of liquid line	Volume of evaporator	Volume of suction line
I	4.971e-05	4.085e-03	8.694e-05	1.887e-03	1.167e-04
II	0.000e+00	6.471e-03	1.520e-04	3.723e-03	0.000e+00
III	4.971e-05	3.819e-03	8.694e-05	1.887e-03	2.736e-04
IV	1.446e-04	4.085e-03	2.611e-04	1.790e-03	1.269e-03
V	1.446e-04	4.085e-03	2.611e-04	1.790e-03	1.269e-03
VI	0.000e+00	4.833e-03	0.000e+00	5.108e-03	0.000e+00
VII	0.000e+00	1.577e-03	2.510e-04	1.437e-03	1.569e-03
VIII	0.000e+00	4.085e-03	2.510e-04	3.401e-03	1.269e-03
IX	0.000e+00	4.085e-03	2.611e-04	3.401e-03	1.269e-03
X	0.000e+00	5.897e-03	2.611e-04	3.401e-03	1.269e-03
XI	0.000e+00	5.695e-03	0.000e+00	3.793e-03	1.167e-04

APPENDIX G. COEFFICIENTS IN REGRESSION EQUATION FOR INITIAL GUESSES OF SYSTEM MODEL

This appendix shows the coefficients of the regression equation Eqn. (6.13) to estimate initial guesses of the cycle solver of different systems. As the variables in the equation are dimensional, the enginnering units of different coefficients in $\mathbf{C}_{5x7}$ are different from each other. For simplicity, the engineering units of the coefficients are omitted in this appendix.

Table G.1. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system I.

-3.206e+03	-1.044e+04	-9.190e+01	-2.390e+01	-2.245e-01
4.724e+00	7.747e+00	4.835e-01	5.391e-01	3.993e-04
3.466e+00	1.492e+00	4.099e-01	2.792e-01	2.956e-04
1.187e+02	2.200e+02	1.992e+01	2.134e+01	1.036e-02
4.127e+00	3.287e+01	-5.535e-01	5.648e-01	1.848e-04
-8.661e+01	-6.267e+02	1.949e+01	-4.770e+00	-4.483e-03
7.580e+01	1.457e+02	-1.437e+01	-1.049e+01	9.304e-03

Table G.2. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system II.

1.609e+04	-7.637e+04	-5.263e+03	-8.057e+03	3.805e+00
-4.711e+01	1.878e+02	1.595e+01	2.358e+01	-1.140e-02
8.432e-01	4.309e+00	3.222e-01	4.195e-01	2.026e-04
-1.230e+02	4.267e+02	3.214e+01	4.755e+01	1.324e-02
2.502e+00	4.343e+01	-2.895e-01	1.239e+00	1.484e-04
-1.123e+03	3.308e+03	2.350e+02	4.025e+02	-2.026e-01
4.096e+00	7.774e+01	-1.626e+00	9.852e-01	7.129e-04

Table G.3. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system III.

-5.849e+03	-1.533e+04	-1.609e+01	-3.652e+01	-2.989e-01
1.064e+01	-1.122e+01	8.011e-02	3.330e-02	4.890e-04
4.539e+00	3.320e+00	4.957e-01	4.783e-01	1.455e-04
2.986e+02	2.106e+02	2.683e+01	3.065e+01	1.165e-02
5.489e+00	6.769e+01	-3.900e-01	9.762e-01	4.093e-04
-1.200e+02	-1.078e+03	9.977e+00	-9.061e+00	-7.539e-03
2.117e+02	2.854e+02	-1.580e+01	-2.066e+01	1.582e-02

Table G.4. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system IV.

-7.247e+02	-2.214e+03	-3.575e+02	1.347e+02	-8.940e-01
-1.148e+01	-5.626e+01	1.128e+00	-5.452e-01	2.785e-03
6.725e+00	6.080e+00	4.019e-01	4.608e-01	-3.466e-04
2.189e+02	3.193e+02	1.383e+01	2.028e+01	1.144e-03
7.930e+00	6.443e+01	-1.822e-01	9.813e-01	6.292e-04
-1.006e+02	-6.266e+02	2.613e+00	-8.867e+00	-6.960e-03
2.849e+02	3.292e+02	-1.659e+01	-1.863e+01	1.334e-02

Table G.5. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system V.

-5.273e+03	-2.113e+04	6.725e+01	5.307e+01	-3.533e-01
7.725e+00	6.794e+00	-2.206e-01	-1.193e-01	8.863e-04
8.290e+00	5.121e+00	9.568e-02	-1.378e-01	6.099e-04
5.532e+02	4.728e+02	7.768e+00	-6.633e+00	4.482e-02
4.229e+00	6.631e+01	-6.756e-02	1.276e+00	-1.265e-04
-1.069e+02	-1.130e+03	1.810e+00	-1.871e+01	3.052e-03
4.021e+01	4.187e+02	-3.457e+00	3.845e+00	-4.172e-03

Table G.6. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system VI.

-4.417e+03	-1.868e+04	-6.536e+01	-2.634e+02	-6.816e-01
6.379e+00	3.726e+00	3.476e-01	6.728e-01	9.339e-04
6.100e+00	1.142e+01	6.623e-01	7.438e-01	8.285e-04
1.120e+02	1.895e+02	1.029e+01	1.217e+01	1.626e-02
2.961e+00	5.161e+01	-5.526e-01	8.113e-01	5.339e-04
-5.245e+00	-2.358e+02	1.942e+00	-2.968e+00	8.079e-04
1.280e+02	4.196e+02	-2.505e+01	-1.717e+01	3.730e-02

Table G.7. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system VII.

-4.003e+03	-1.985e+04	-1.704e+02	-2.898e+02	-1.276e-01
1.495e+01	9.637e+00	2.629e-01	4.045e-02	7.353e-04
3.880e+00	3.660e+00	1.373e-01	9.219e-02	1.778e-04
3.828e+02	2.919e+02	4.278e+00	-1.080e+00	2.051e-02
2.830e+00	5.970e+01	-3.610e-02	1.630e+00	-1.270e-05
-1.784e+03	-2.414e+02	8.327e+01	1.091e+02	-1.116e-01
4.997e+01	9.383e+01	-3.020e+00	-1.812e+00	3.266e-03

Table G.8. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system VIII.

-1.206e+04	-5.153e+04	-7.392e+02	-3.254e+02	-8.767e-01
3.406e+01	9.924e+01	2.490e+00	5.075e-01	2.747e-03
4.242e+00	-3.362e+00	3.061e-02	-2.811e-01	3.457e-04
3.192e+02	1.115e+02	3.106e+00	-1.151e+01	2.204e-02
5.423e+00	7.751e+01	-3.706e-02	1.844e+00	1.247e-04
-1.347e+02	1.675e+02	2.865e+00	1.129e+01	-1.477e-02
1.687e+01	4.272e+02	-2.017e+00	8.598e+00	-2.680e-03

Table G.9. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system IX.

-1.077e+04	1.941e+04	-1.102e+03	-2.581e+03	3.334e+00
3.525e+01	-9.467e+01	4.297e+00	8.585e+00	-1.151e-02
1.670e+00	4.831e-01	1.125e-01	2.590e-01	-2.155e-05
1.074e+02	9.997e+01	1.147e+01	9.929e+00	1.996e-03
3.994e+00	4.039e+01	-1.397e-01	1.180e+00	7.897e-04
-8.057e+02	-1.487e+03	-9.483e+01	-1.193e+01	-3.431e-02
6.228e+01	8.567e+01	-1.383e+01	-1.776e+01	5.387e-04

Table G.10. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system X.

3.791e+03	-7.775e+03	-1.857e+02	-3.923e+02	3.095e+00
-1.859e+01	-1.443e+01	5.941e-01	8.921e-01	-1.068e-02
5.890e+00	1.256e+00	7.672e-02	-3.187e-01	6.270e-04
2.406e+02	1.686e+02	2.183e+00	-9.557e+00	3.066e-02
4.160e+00	4.231e+01	5.516e-03	1.664e+00	2.006e-04
-4.525e+02	1.415e+02	-7.370e+00	4.222e+01	-5.446e-02
2.317e+00	3.029e+01	-8.904e-01	2.310e-01	-9.103e-04

Table G.11. Coefficient matrix $\mathbf{C}_{5x7}^{T}$ for system XI.

-6.367e+03	-9.456e+03	1.488e+01	8.069e+01	-2.877e-01
1.287e+01	-8.224e+00	-4.440e-01	-1.146e+00	9.676e-04
6.331e+00	-1.805e+01	4.540e-01	4.905e-01	-2.120e-04
3.137e+02	2.829e+01	2.309e+00	-7.154e+00	2.304e-02
5.251e+00	6.620e+01	-2.502e-03	1.616e+00	3.519e-04
-5.867e+01	-6.494e+02	2.227e+00	-1.431e+01	-2.190e-03
1.400e+01	8.358e+01	-1.146e+00	7.477e-01	1.568e-04

APPENDIX H. CHARGE TUNING COEFFICIENTS

This appendix contains the charge tuning equation coefficients of different systems. Table H.1 shows the coefficients in charge tuning equation Eqn. (6.19) for the two-point tuning method, as solved by minimizing the objective function Eqn. (6.24).

Table H.1. Coefficients of old charge tuning equation.

System	$C_{M,0}$	$C_{M,1}$
I	2.38e+00	3.86e+00
II	3.39e+00	8.98e+00
III	2.16e+00	7.89e+00
IV	2.03e+00	9.75e+00
V	2.27e+00	3.62e+00
VI	1.53e+00	3.78e+00
VII	3.72e+00	1.04e+01
VIII	3.03e+00	2.00e+00
IX	2.63e+00	1.37e+01
X	4.33e+00	1.32e+01
XI	3.53e+00	6.21e+00

Table H.2 shows the coefficients in charge tuning equation Eqn. (6.22).

Table H.2. Coefficients of new charge tuning equation.

System	$C_{M,0}$	$C_{M,1}$	$C_{M,2}$	$C_{M,3}$	$w_{cond,sc,rated}$	$w_{cond,sh,rated}$	$w_{evap,sh,rated}$
I	1.00e+00	-3.48e-01	5.60e+00	-2.52e-01	3.00e-02	1.04e-01	2.52e-01
II	2.51e+00	3.92e+00	1.11e+01	8.78e-02	4.52e-01	5.63e-02	1.62e-01
III	1.49e+00	1.85e+00	1.29e+00	-7.79e-01	5.31e-02	3.63e-01	1.04e-01
IV	9.17e-01	5.57e-01	1.21e+00	-8.78e-01	1.10e-01	2.18e-01	2.31e-01
V	8.60e-01	-3.85e-02	1.51e+00	-1.03e+00	1.43e-01	2.16e-01	7.92e-02
VI	-1.73e-01	-1.45e+00	1.37e+00	-1.01e-01	2.31e-01	1.13e-01	4.55e-01
VII	3.67e+00	7.23e+00	-2.49e+00	-1.79e+00	9.26e-02	2.31e-01	1.41e-01
VIII	1.06e+00	-1.18e+00	-4.87e-01	-2.46e+00	3.55e-01	1.17e-01	1.28e-01
IX	1.37e+00	4.30e+00	1.40e+00	-1.09e+00	3.13e-02	9.48e-02	1.28e-01
X	1.94e+00	-2.07e+00	-7.88e+00	-1.39e+01	1.15e-01	6.41e-02	2.36e-01
XI	2.19e+00	6.47e-01	-1.17e+01	-2.87e+00	2.34e-01	1.02e-01	2.13e-01

APPENDIX I. PARAMETERS TO CALCULATE THE DISTANCE OF A SIMULATION OUTPUT TO THE APPLICABILITY DOMAIN OF THE CHARGE TUNING EQUATION

Section 6.4 concludes that the distance to the applicability domain of the charge tuning equation is needed for simulation outputs from FXO systems to determine if the simulation output is reliable. To calculate the distance according to Eqn. (5.47), the maximum leverage obtained at the training data point and the covariance $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ in Eqn. (5.42) are needed. The maximum leverages of different systems are tabulated in Table I.1.

Table I.1. Maximum leverage of charge tuning equation in different systems..

System	Maximum leverage in charge tuning equation
I	1.99e-01
II	6.72e-01
III	2.59e-01
IV	2.50e-01
V	6.04e-01
VI	2.09e-01
VII	2.95e-01
VIII	2.39e-01
IX	3.91e-01
X	2.42e-01
XI	8.36e-01

By considering

$$\vec{x}_M = [w_{cond,sc} w_{cond,sh} w_{evap,sh}]^T, \tag{I.1}$$

the covariance $\mathbf{X}_{train}^T \mathbf{X}_{train}$ of the charge tuning equation is defined and their values are tabulated from Table I.2 to Table I.12

Table I.2. Covariance matrix $\mathbf{X}_{train}^T \mathbf{X}_{train}$ for system I.

8.586e+00	5.411e+01	-8.604e-01
5.411e+01	7.348e+02	-7.957e+00
-8.604e-01	-7.957e+00	1.422e+00

Table I.3. Covariance matrix $\mathbf{X}_{train}^T \mathbf{X}_{train}$ for system II.

5.559e+00	5.023e+01	4.851e+00
5.023e+01	8.198e+02	1.087e-01
4.851e+00	1.087e-01	1.361e+01

Table I.4. Covariance matrix $\mathbf{X}_{train}^T \mathbf{X}_{train}$ for system III.

1.285e+01	8.486e+00	1.368e+00
8.486e+00	6.607e+00	8.250e-01
1.368e+00	8.250e-01	6.910e-01

Table I.5. Covariance matrix $\mathbf{X}_{train}^T \mathbf{X}_{train}$ for system IV.

2.280e+01	1.919e+01	1.824e+00
1.919e+01	2.320e+01	1.513e+00
1.824e+00	1.513e+00	8.391e-01

Table I.6. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system V.

1.210e+00	1.077e+00	-1.675e-01
1.077e+00	9.382e+00	-2.172e+00
-1.675e-01	-2.172e+00	2.094e+00

Table I.7. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system VI.

1.487e+01	-2.392e+00	2.079e+00
-2.392e+00	6.824e+01	2.071e-01
2.079e+00	2.071e-01	4.525e-01

Table I.8. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system VII.

5.437e+00	5.058e-01	1.571e+00
5.058e-01	9.233e+00	5.176e-01
1.571e+00	5.176e-01	2.532e+00

Table I.9. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system VIII.

6.221e-01	6.330e-01	1.075e+00
6.330e-01	6.823e+00	-1.301e+00
1.075e+00	-1.301e+00	5.964e+00

Table I.10. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system IX.

9.457e+01	1.014e+02	4.460e+00
1.014e+02	2.373e+02	8.567e-01
4.460e+00	8.567e-01	1.083e+00

Table I.11. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system X.

7.507e+00	1.902e+01	1.210e+00
1.902e+01	1.047e+02	1.016e+01
1.210e+00	1.016e+01	9.125e+00

Table I.12. Covariance matrix $\mathbf{X}_{train}^{T}\mathbf{X}_{train}$ for system XI.

1.830e+00	8.240e+00	2.087e-01
8.240e+00	2.200e+02	-1.255e+01
2.087e-01	-1.255e+01	5.301e+00

INVERSE MODELING OF VAPOR COMPRESSION EQUIPMENT TO ENABLE SIMULATION OF FAULT IMPACTS

VOLUME 2

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Howard Cheung

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2014

Purdue University

West Lafayette, Indiana

APPENDIX J. DATA COLLECTED FROM SYSTEM XI WITH DIFFERENT AMOUNT OF NON-CONDENSABLE IN THE SYSTEM

The data collected from testing system XI under the conditions listed in Table 3.6 are listed in this appendix.

Table J.1. : Return and supply duct measurement in tests 1 to 5.

Test	1	2	3	4	5
$T_{a,ret,1}$ [K]	300.32	300.33	300.51	300.37	300.06
$T_{a,ret,2}$ [K]	300.29	300.30	300.44	300.34	300.06
$T_{a,ret,3}$ [K]	299.74	299.79	299.63	299.83	299.63
$T_{a,ret,4}$ [K]	299.78	299.83	299.69	299.89	299.67
$T_{a,ret,5}$ [K]	299.78	299.83	299.73	299.89	299.70
$T_{a,ret,6}$ [K]	299.77	299.83	299.76	299.87	299.70
$T_{a,ret,7}$ [K]	299.97	299.98	299.79	299.99	299.81
$T_{a,ret,8}$ [K]	299.93	299.94	299.81	300.00	299.84
$T_{a,ret,9}$ [K]	299.95	299.96	299.86	300.03	299.89
$Dew_{a,ret}$ [K]	287.62	287.62	287.84	288.05	287.83
$T_{a,evap,in,1}$ [K]	298.48	298.43	299.00	298.86	299.13

Table J.1 Continued.

Test	1	2	3	4	5
$T_{a,evap,in,2}$ [K]	298.91	298.87	299.32	299.18	299.44
$T_{a,evap,in,3}$ [K]	298.31	298.30	298.66	298.62	298.81
$T_{a,evap,in,4}$ [K]	297.99	297.98	298.49	298.40	298.70
$T_{a,evap,in,5}$ [K]	298.68	298.70	298.98	298.90	299.00
$T_{a,evap,in,6}$ [K]	299.08	299.12	299.26	299.16	299.21
$T_{a,evap,in,7}$ [K]	298.94	298.92	299.06	299.05	299.20
$T_{a,evap,in,8}$ [K]	298.21	298.12	298.71	298.58	298.92
$T_{a,evap,in,9}$ [K]	299.30	299.28	299.39	299.39	299.39
$T_{a,evap,in,10}$ [K]	299.51	299.49	299.53	299.54	299.55
$T_{a,evap,in,11}$ [K]	299.57	299.60	299.80	299.67	299.74
$T_{a,evap,in,12}$ [K]	299.10	299.12	299.47	299.33	299.54
$T_{a,evap,in,13}$ [K]	299.70	299.74	299.86	299.77	299.75
$T_{a,evap,in,14}$ [K]	299.66	299.70	299.83	299.69	299.73
$T_{a,evap,in,15}$ [K]	299.39	299.38	299.70	299.58	299.80
$T_{a,evap,in,16}$ [K]	299.91	299.90	300.09	300.00	300.11
$T_{a,sup,1}$ [K]	288.27	288.41	288.78	288.65	288.51
$T_{a,sup,2}$ [K]	287.24	287.62	287.77	287.70	287.78
$T_{a,sup,3}$ [K]	285.69	285.54	286.05	285.94	285.97
$T_{a,sup,4}$ [K]	285.07	285.17	285.49	285.42	284.96

Table J.1 Continued.

Test	1	2	3	4	5
$T_{a,sup,5}$ [K]	285.30	285.22	285.66	285.64	285.22
$T_{a,sup,6}$ [K]	287.22	286.76	287.72	287.25	287.53
$Dew_{a,sup}$ [K]	284.94	285.44	285.88	285.95	285.20

Table J.2. : Return and supply duct measurement in tests 6 to 11.

Test	6	7	8	9	10	11
$T_{a,ret,1}$ [K]	300.43	300.40	300.44	300.41	300.08	300.18
$T_{a,ret,2}$ [K]	300.39	300.39	300.40	300.39	300.10	300.20
$T_{a,ret,3}$ [K]	299.83	299.94	299.72	299.71	299.69	299.71
$T_{a,ret,4}$ [K]	299.85	299.97	299.77	299.74	299.72	299.75
$T_{a,ret,5}$ [K]	299.87	299.96	299.80	299.80	299.76	299.79
$T_{a,ret,6}$ [K]	299.87	299.92	299.82	299.82	299.76	299.80
$T_{a,ret,7}$ [K]	299.98	300.16	299.88	299.86	299.86	299.87
$T_{a,ret,8}$ [K]	299.99	300.16	299.91	299.92	299.92	299.94
$T_{a,ret,9}$ [K]	300.02	300.16	299.96	299.98	299.98	299.99
$Dew_{a,ret}$ [K]	288.01	288.05	287.93	288.07	287.81	287.79
$T_{a,evap,in,1}$ [K]	302.06	302.00	302.00	302.93	302.52	302.65
$T_{a,evap,in,2}$ [K]	301.51	301.44	301.39	302.06	301.80	301.88
$T_{a,evap,in,3}$ [K]	301.36	301.46	301.21	301.87	301.67	301.65

Table J.2 Continued.

Test	6	7	8	9	10	11
$T_{a,evap,in,4}$ [K]	301.99	302.07	301.63	303.01	302.44	302.45
$T_{a,evap,in,5}$ [K]	300.97	300.92	300.91	301.29	301.09	301.12
$T_{a,evap,in,6}$ [K]	300.31	300.30	300.23	300.51	300.31	300.34
$T_{a,evap,in,7}$ [K]	300.79	300.86	300.71	301.12	300.88	300.87
$T_{a,evap,in,8}$ [K]	302.18	302.21	301.68	303.25	302.63	302.75
$T_{a,evap,in,9}$ [K]	300.39	300.45	300.33	300.55	300.45	300.48
$T_{a,evap,in,10}$ [K]	300.01	300.06	299.95	300.08	299.97	299.99
$T_{a,evap,in,11}$ [K]	300.72	300.63	300.64	300.96	300.74	300.88
$T_{a,evap,in,12}$ [K]	301.48	301.43	301.21	301.90	301.62	301.81
$T_{a,evap,in,13}$ [K]	300.55	300.50	300.52	300.63	300.49	300.58
$T_{a,evap,in,14}$ [K]	300.31	300.22	300.27	300.42	300.25	300.32
$T_{a,evap,in,15}$ [K]	301.31	301.21	301.16	301.63	301.39	301.49
$T_{a,evap,in,16}$ [K]	300.74	300.68	300.68	300.94	300.81	300.89
$T_{a,sup,1}$ [K]	290.79	290.19	290.87	290.91	290.53	292.29
$T_{a,sup,2}$ [K]	289.87	290.67	289.99	290.28	290.37	291.63
$T_{a,sup,3}$ [K]	287.80	289.35	287.97	288.54	288.10	290.36
$T_{a,sup,4}$ [K]	286.79	285.91	286.81	286.82	286.42	289.24
$T_{a,sup,5}$ [K]	286.54	285.98	286.64	286.48	286.07	287.14
$T_{a,sup,6}$ [K]	289.86	290.51	290.23	290.60	290.18	288.12

Table J.2 Continued.

Test	6	7	8	9	10	11
$Dew_{a,sup}$ [K]	286.98	286.85	287.22	287.32	286.58	286.70

Table J.3. : Outdoor chamber air-side measurement in tests 1 to 5.

Test	1	2	3	4	5
$T_{a,cond,in,1}$ [K]	297.79	297.79	296.39	297.30	294.63
$T_{a,cond,in,2}$ [K]	297.64	297.65	296.28	297.12	294.54
$T_{a,cond,in,3}$ [K]	297.70	297.68	296.35	297.22	294.66
$T_{a,cond,in,4}$ [K]	297.97	297.98	296.52	297.43	294.79
$T_{a,cond,in,5}$ [K]	297.91	297.95	296.48	297.27	294.80
$T_{a,cond,in,6}$ [K]	297.75	297.77	296.45	297.20	294.73
$T_{a,cond,in,7}$ [K]	296.94	296.78	297.67	297.18	297.33
$T_{a,cond,in,8}$ [K]	297.50	297.57	297.12	297.15	295.22
$T_{a,cond,in,9}$ [K]	297.55	297.35	296.72	296.94	294.71
$T_{a,cond,in,10}$ [K]	297.86	297.82	297.71	297.54	296.05
$T_{a,cond,in,11}$ [K]	297.95	297.84	297.37	297.43	295.34
$T_{a,cond,in,12}$ [K]	297.92	297.83	296.94	297.27	295.04
$T_{a,cond,in,13}$ [K]	297.35	297.12	297.54	297.41	295.78
$T_{a,cond,in,14}$ [K]	297.62	297.66	296.53	296.92	294.61
$T_{a,cond,in,15}$ [K]	297.67	297.71	296.40	297.02	294.59

Table J.3 Continued.

Test	1	2	3	4	5
$T_{a,cond,in,16}$ [K]	296.28	296.29	297.43	297.05	320.67
$T_{a,cond,in,17}$ [K]	296.42	296.45	297.42	296.95	299.07
$T_{a,cond,in,18}$ [K]	296.09	296.16	297.32	296.83	298.86
$T_{a,cond,in,19}$ [K]	296.46	296.58	297.51	296.81	298.64
$T_{a,cond,in,20}$ [K]	296.25	296.22	297.92	297.11	298.90
$Dew_{a,cond,in}$ [K]	286.51	287.43	288.78	288.22	287.58
$T_{a,cond,out,1}$ [K]	303.96	303.92	304.04	304.17	303.00
$T_{a,cond,out,2}$ [K]	302.97	302.91	302.80	303.70	301.88
$T_{a,cond,out,3}$ [K]	302.50	302.45	302.83	303.31	301.92
$T_{a,cond,out,4}$ [K]	303.12	303.05	303.42	303.13	302.76
$T_{a,cond,out,5}$ [K]	304.06	303.90	304.01	304.23	303.19
$T_{a,cond,out,6}$ [K]	303.03	302.86	302.57	302.72	301.25
$T_{a,cond,out,7}$ [K]	303.31	303.24	303.09	303.51	302.32
$T_{a,cond,out,8}$ [K]	304.76	304.63	304.34	304.65	302.73
$T_{a,cond,out,9}$ [K]	303.57	303.49	303.37	303.75	302.62
$T_{a,cond,out,10}$ [K]	304.05	304.00	303.76	304.15	302.70
$T_{a,cond,out,11}$ [K]	304.05	304.00	303.76	304.15	302.70
$T_{a,cond,out,12}$ [K]	304.59	304.50	304.23	304.81	303.49

Table J.4. : Outdoor chamber air-side measurement in tests 6 to 11.

Test	6	7	8	9	10	11
$T_{a,cond,in,1}$ [K]	308.36	308.40	308.77	307.22	306.85	306.86
$T_{a,cond,in,2}$ [K]	308.25	308.22	308.68	306.99	306.81	306.79
$T_{a,cond,in,3}$ [K]	308.24	308.33	308.79	306.90	306.94	306.97
$T_{a,cond,in,4}$ [K]	308.49	308.59	308.92	307.31	307.01	307.05
$T_{a,cond,in,5}$ [K]	308.52	308.54	308.84	307.25	307.05	307.08
$T_{a,cond,in,6}$ [K]	308.35	308.39	308.75	307.20	307.03	307.07
$T_{a,cond,in,7}$ [K]	307.71	307.71	307.70	308.44	308.55	308.63
$T_{a,cond,in,8}$ [K]	308.20	308.21	308.89	308.78	307.57	307.58
$T_{a,cond,in,9}$ [K]	308.33	308.28	309.04	306.84	307.25	307.21
$T_{a,cond,in,10}$ [K]	308.49	308.44	309.14	308.67	308.17	308.20
$T_{a,cond,in,11}$ [K]	308.49	308.45	309.47	307.74	307.80	307.82
$T_{a,cond,in,12}$ [K]	308.51	308.51	309.29	307.44	307.55	307.57
$T_{a,cond,in,13}$ [K]	308.07	308.06	309.15	307.64	308.00	307.99
$T_{a,cond,in,14}$ [K]	308.11	308.23	308.83	307.54	307.04	307.00
$T_{a,cond,in,15}$ [K]	308.18	308.27	308.78	307.11	306.96	306.93
$T_{a,cond,in,16}$ [K]	307.38	307.37	306.60	310.53	320.78	320.79
$T_{a,cond,in,17}$ [K]	307.39	307.44	306.23	310.34	308.83	309.10
$T_{a,cond,in,18}$ [K]	307.31	307.35	305.83	310.07	308.76	309.02
$T_{a,cond,in,19}$ [K]	307.48	307.50	306.21	310.14	308.67	308.93

Table J.4 Continued.

Test	6	7	8	9	10	11
$T_{a,cond,in,20}$ [K]	307.51	307.52	306.78	309.93	308.87	309.16
$Dew_{a,cond,in}$ [K]	288.15	287.31	287.54	287.20	287.26	287.38
$T_{a,cond,out,1}$ [K]	314.88	314.76	315.22	315.63	314.79	313.97
$T_{a,cond,out,2}$ [K]	313.74	313.71	314.10	314.03	313.47	313.34
$T_{a,cond,out,3}$ [K]	313.44	313.38	314.03	314.74	313.67	313.05
$T_{a,cond,out,4}$ [K]	314.02	313.90	314.32	314.77	314.25	313.67
$T_{a,cond,out,5}$ [K]	314.97	314.85	315.15	315.53	314.91	314.33
$T_{a,cond,out,6}$ [K]	313.98	313.86	313.97	313.87	313.27	312.73
$T_{a,cond,out,7}$ [K]	314.45	314.44	314.79	314.67	314.32	314.01
$T_{a,cond,out,8}$ [K]	315.67	315.55	315.73	315.64	314.81	313.75
$T_{a,cond,out,9}$ [K]	314.45	314.48	314.60	314.99	314.31	313.52
$T_{a,cond,out,10}$ [K]	315.01	314.99	315.22	315.33	314.63	313.82
$T_{a,cond,out,11}$ [K]	315.01	314.99	315.22	315.33	314.63	313.82
$T_{a,cond,out,12}$ [K]	315.68	315.74	316.00	316.11	315.53	314.92

Table J.5. : Measurement on the refrigerant side, on the power consumption and at nozzle in tests 1 to 5.

Test	1	2	3	4	5
$P_{r,comp,out}$ [kPa]	1072	1080	1094	1104	1115

Table J.5 Continued.

Test	1	2	3	4	5
$P_{r,cond,out}$ [kPa]	1981	1989	2001	2069	2097
$P_{r,EXV,in}$ [kPa]	1911	1914	1926	1999	2006
$P_{r,evap,in}$ [kPa]	1899	1911	1923	1995	1986
$P_{r,evap,out}$ [kPa]	1150	1156	1176	1193	1213
$T_{r,comp,in}$ [K]	291.67	291.89	292.44	292.46	292.74
$T_{r,comp,out}$ [K]	330.27	330.19	330.35	331.94	332.21
$T_{r,cond,out}$ [K]	296.95	297.00	296.39	296.65	294.95
$T_{r,EXV,in}$ [K]	297.09	297.16	296.53	296.79	294.81
$T_{r,evap,in,1}$ [K]	284.63	284.52	284.44	283.61	282.69
$T_{r,evap,in,2}$ [K]	284.24	284.09	284.11	283.38	282.06
$T_{r,evap,in,3}$ [K]	282.49	282.29	282.28	281.44	280.26
$T_{r,evap,in,4}$ [K]	283.93	283.80	283.74	282.92	281.79
$T_{r,evap,in,5}$ [K]	282.94	282.78	282.72	281.88	280.72
$T_{r,evap,out,1}$ [K]	289.55	289.56	290.32	290.43	291.70
$T_{r,evap,out,2}$ [K]	292.10	292.28	292.71	292.56	292.83
$T_{r,evap,out,3}$ [K]	293.14	293.51	293.84	293.59	293.85
$T_{r,evap,out,4}$ [K]	288.19	288.32	289.98	290.39	290.09
$T_{r,evap,out,5}$ [K]	292.63	292.75	292.97	292.82	292.71
$T_{r,evap,out,6}$ [K]	293.64	293.86	293.95	293.80	293.92

Table J.5 Continued.

Test	1	2	3	4	5
$T_{r,evap,out,7}$ [K]	296.09	296.28	296.40	296.17	296.41
$\dot{m}_r$ [kg/s]	0.0611	0.0614	0.0623	0.0629	0.0480
$\dot{W}_{comp}$ [W]	1516	1524	1522	1590	1608
$\dot{W}_{evap,fan}$ [W]	398	396	401	395	401
$\dot{W}_{cond,fan}$ [W]	254	257	254	248	251
$\dot{V}_{a,evap}$ [m^3/s]	0.6598	0.6582	0.6611	0.6588	0.6622
T_{nozzle} [K]	290.30	290.49	290.56	290.53	290.48

Table J.6. : Measurement on the refrigerant side, on the power consumption and at nozzle in tests 6 to 11.

Test	6	7	8	9	10	11
$P_{r,comp,out}$ [kPa]	1136	1142	1144	1164	1172	1041
$P_{r,cond,out}$ [kPa]	2575	2582	2619	2689	2725	2698
$P_{r,EXV,in}$ [kPa]	2502	2508	2546	2619	2645	2644
$P_{r,evap,in}$ [kPa]	2492	2503	2542	2613	2636	2647
$P_{r,evap,out}$ [kPa]	1250	1253	1261	1287	1302	1142
$T_{r,comp,in}$ [K]	293.59	293.80	293.86	294.49	294.46	298.82
$T_{r,comp,out}$ [K]	346.22	346.45	347.30	348.56	348.81	361.83
$T_{r,cond,out}$ [K]	308.24	308.26	308.31	308.09	307.53	307.56

Table J.6 Continued.

Test	6	7	8	9	10	11
$T_{r,EXV,in}$ [K]	308.36	308.39	308.45	308.21	307.48	307.61
$T_{r,evap,in,1}$ [K]	287.93	288.74	287.72	287.53	286.74	283.76
$T_{r,evap,in,2}$ [K]	287.15	287.09	286.66	286.76	285.72	282.62
$T_{r,evap,in,3}$ [K]	284.72	284.59	284.32	284.15	283.25	279.69
$T_{r,evap,in,4}$ [K]	286.81	286.74	286.43	286.33	285.30	281.82
$T_{r,evap,in,5}$ [K]	285.32	285.25	284.90	284.76	283.83	280.20
$T_{r,evap,out,1}$ [K]	292.79	295.84	292.93	294.37	295.28	299.53
$T_{r,evap,out,2}$ [K]	295.62	297.31	295.73	296.54	296.26	299.09
$T_{r,evap,out,3}$ [K]	296.44	297.02	296.50	297.23	296.80	300.17
$T_{r,evap,out,4}$ [K]	285.84	285.45	287.12	288.33	289.06	297.77
$T_{r,evap,out,5}$ [K]	294.56	292.70	294.78	294.60	293.97	298.46
$T_{r,evap,out,6}$ [K]	296.02	295.48	296.15	296.14	295.83	298.65
$T_{r,evap,out,7}$ [K]	298.79	298.03	298.81	298.86	298.41	300.63
$\dot{m}_r$ [kg/s]	0.0638	0.0637	0.0640	0.0650	0.0530	0.0508
$\dot{W}_{comp}$ [W]	2100	2109	2145	2206	2240	2297
$\dot{W}_{evap,fan}$ [W]	393	399	401	395	392	387
$\dot{W}_{cond,fan}$ [W]	245	251	244	239	243	240
$\dot{V}_{a,evap}$ [m^3/s]	0.6684	0.6705	0.6733	0.6722	0.6675	0.6685
T_{nozzle} [K]	292.50	292.54	292.23	292.50	292.14	293.53

APPENDIX K. DATA COLLECTED FROM SYSTEM I

Since the data from system I (Breuker, 1997) were only stored in the form of dynamic data but not steady state data, in order to use the data for training, steady state data was extracted from the dynamic data and were used to train and validate the system model and to evaluate the FDD tools (Yuill, 2014). The data are tabulated as follows.

Table K.1. : Air-side measurement across evaporator in tests in system I.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
1	296.06	289.07	288.27	287.23	0.5671
2	297.63	290.58	290.08	289.01	0.5688
3	294.32	286.94	286.22	285.10	0.5673
4	297.55	289.85	289.95	288.38	0.5693
5	295.94	288.12	288.14	286.38	0.5667
6	299.27	290.18	289.25	287.89	0.5688
7	294.32	286.32	286.28	284.62	0.5675
8	297.47	289.28	289.89	287.90	0.5693
9	297.66	288.43	287.27	285.81	0.5654
10	298.49	289.17	288.31	286.69	0.5673

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
11	294.32	285.88	286.29	284.21	0.5675
12	295.86	287.52	288.06	285.83	0.5676
13	299.20	289.66	289.19	287.33	0.5692
14	297.63	289.00	290.15	287.78	0.5693
15	298.48	288.64	288.29	286.24	0.5678
16	297.61	287.88	287.21	285.33	0.5671
17	294.32	285.37	286.26	283.75	0.5676
18	295.76	286.94	287.98	285.36	0.5670
19	296.68	286.59	286.11	284.13	0.5657
20	297.59	288.53	290.06	287.39	0.5695
21	299.03	289.05	289.01	286.75	0.5699
22	295.20	284.95	284.16	282.32	0.5641
23	298.49	288.13	288.27	285.96	0.5693
24	296.87	286.40	286.29	283.99	0.5657
25	295.72	286.52	287.95	285.06	0.5674
26	300.15	288.91	287.22	285.64	0.5683
27	294.41	285.01	286.36	283.47	0.5678
28	295.83	285.35	284.98	282.98	0.5673
29	297.62	287.42	287.26	284.95	0.5672

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
30	299.42	289.10	289.41	286.84	0.5697
31	301.05	289.63	288.32	286.46	0.5687
32	294.27	283.26	282.95	280.72	0.5678
33	297.59	286.91	287.21	284.54	0.5675
34	295.27	284.48	284.15	281.79	0.5640
35	300.05	288.31	287.17	285.16	0.5681
36	299.23	287.54	286.08	284.26	0.5670
37	295.96	285.01	285.08	282.65	0.5688
38	300.95	289.06	288.23	286.02	0.5696
39	299.28	287.18	286.09	283.85	0.5673
40	296.49	285.66	283.04	281.49	0.5705
41	296.42	285.57	282.96	281.30	0.5698
42	296.44	285.61	283.02	281.36	0.5698
43	296.44	285.61	282.91	281.40	0.5700
44	296.46	285.92	283.02	281.91	0.5683
45	296.38	285.89	282.91	281.79	0.5685
46	296.33	285.59	282.81	281.50	0.5687
47	296.44	285.68	282.93	281.72	0.5692
48	296.33	285.58	282.78	281.51	0.5693

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
49	296.66	285.87	283.03	281.77	0.5699
50	296.66	285.93	283.09	281.96	0.5697
51	296.65	286.07	283.08	282.13	0.5692
52	296.61	286.38	283.04	282.31	0.5692
53	296.34	285.67	282.88	281.86	0.5696
54	296.38	285.84	282.86	282.00	0.5686
55	296.90	286.39	283.48	282.44	0.5694
56	296.83	286.63	283.20	282.51	0.5693
57	296.39	286.19	282.76	282.11	0.5685
58	296.73	286.56	283.14	282.45	0.5692
59	296.76	286.22	283.20	282.26	0.5693
60	296.73	286.34	283.18	282.35	0.5697
61	296.82	286.39	283.18	282.36	0.5684
62	297.01	287.03	283.33	282.65	0.5693
63	296.99	286.88	283.49	282.70	0.5697
64	296.98	287.04	283.40	282.75	0.5687
65	296.94	286.83	283.32	282.62	0.5694
66	296.94	287.28	283.33	282.76	0.5697
67	296.91	287.27	283.31	282.80	0.5696

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
68	296.98	287.35	283.71	283.18	0.5689
69	296.97	287.53	283.32	282.93	0.5691
70	296.31	284.52	282.67	280.39	0.4958
71	296.33	284.57	282.70	280.49	0.4983
72	296.50	284.89	283.18	281.33	0.4945
73	296.60	285.01	283.25	281.36	0.4946
74	296.65	285.23	283.03	281.68	0.4943
75	296.86	285.47	283.35	281.92	0.4952
76	296.45	283.94	282.82	280.06	0.4254
77	296.37	283.94	282.93	280.08	0.4258
78	296.62	285.67	283.14	282.17	0.4941
79	296.53	284.17	283.06	280.78	0.4253
80	296.84	285.95	283.45	282.39	0.4954
81	296.65	284.30	283.15	280.89	0.4248
82	296.79	284.77	283.22	281.42	0.4301
83	296.90	284.87	283.29	281.49	0.4303
84	296.56	283.40	282.98	279.53	0.3714
85	296.99	286.42	283.51	282.80	0.4939
86	296.87	283.75	283.15	280.36	0.3713

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
87	296.75	285.22	283.22	281.87	0.4297
88	296.96	285.40	283.36	282.03	0.4303
89	296.46	283.74	282.90	280.60	0.3708
90	296.96	284.26	283.33	281.07	0.3724
91	296.81	284.15	283.20	281.05	0.3713
92	297.08	285.84	283.37	282.45	0.4278
93	297.02	284.84	283.36	281.70	0.3708
94	296.72	284.60	283.13	281.52	0.3703
95	296.98	285.13	283.28	281.99	0.3695
96	296.38	285.60	282.74	281.17	0.5716
97	296.33	285.53	282.71	281.14	0.5710
98	296.42	287.57	282.71	281.40	0.5682
99	296.31	286.24	282.64	281.18	0.5692
100	296.32	286.23	282.69	281.26	0.5688
101	296.32	287.54	282.65	281.32	0.5674
102	296.37	285.64	282.77	281.56	0.5708
103	296.40	286.26	282.82	281.62	0.5680
104	296.38	285.68	282.78	281.57	0.5704
105	296.41	285.89	282.81	281.59	0.5707

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
106	296.39	287.39	282.77	281.66	0.5671
107	296.37	286.22	282.76	281.60	0.5678
108	296.39	285.92	282.80	281.57	0.5711
109	296.38	287.42	282.81	281.72	0.5669
110	296.47	286.04	282.87	281.90	0.5710
111	296.42	285.99	282.85	281.89	0.5709
112	296.51	286.53	282.90	281.91	0.5669
113	296.42	286.46	282.86	281.89	0.5673
114	296.44	287.44	282.89	281.94	0.5656
115	296.50	287.47	282.90	281.99	0.5660
116	296.45	286.19	282.91	281.95	0.5677
117	296.50	286.21	282.87	281.95	0.5683
118	296.59	286.53	283.03	282.34	0.5713
119	296.62	287.00	283.04	282.30	0.5673
120	296.62	286.56	283.07	282.39	0.5714
121	296.64	287.03	283.06	282.32	0.5674
122	296.61	287.77	283.06	282.32	0.5656
123	296.60	286.66	283.00	282.32	0.5684
124	296.64	287.80	283.05	282.32	0.5655

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
125	296.62	286.68	283.02	282.34	0.5679
126	296.91	287.54	283.27	282.73	0.5663
127	296.91	287.17	283.28	282.86	0.5705
128	296.94	288.26	283.28	282.75	0.5652
129	296.93	287.30	283.31	282.84	0.5665
130	296.39	285.51	282.77	281.16	0.5709
131	296.34	285.68	282.82	281.17	0.5680
132	296.30	285.46	282.71	281.18	0.5713
133	296.26	285.71	282.83	281.18	0.5693
134	296.37	285.90	282.90	281.27	0.5683
135	296.34	285.83	282.78	281.22	0.5678
136	296.39	285.62	282.84	281.61	0.5714
137	296.41	285.73	282.85	281.54	0.5689
138	296.45	285.77	282.85	281.59	0.5681
139	296.36	285.61	282.84	281.60	0.5703
140	296.44	286.21	282.85	281.66	0.5697
141	296.43	285.99	282.88	281.64	0.5679
142	296.39	286.21	282.84	281.67	0.5697
143	296.44	286.00	282.89	281.65	0.5685

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
144	296.50	285.95	282.97	282.05	0.5696
145	296.53	285.99	282.99	282.08	0.5700
146	296.56	286.63	282.94	282.01	0.5690
147	296.56	286.16	283.01	282.00	0.5678
148	296.55	286.15	283.01	282.03	0.5677
149	296.54	286.38	282.99	281.95	0.5660
150	296.54	286.64	282.97	282.02	0.5689
151	296.54	286.38	283.00	281.98	0.5656
152	296.84	286.76	283.21	282.46	0.5678
153	296.80	287.26	283.17	282.42	0.5695
154	296.81	286.74	283.26	282.50	0.5674
155	296.81	286.99	283.18	282.37	0.5662
156	296.80	286.61	283.26	282.61	0.5700
157	296.81	287.27	283.12	282.42	0.5691
158	296.76	286.59	283.18	282.53	0.5701
159	296.80	287.01	283.23	282.47	0.5666
160	296.96	287.12	283.30	282.90	0.5668
161	296.95	287.25	283.34	282.86	0.5665
162	296.90	287.73	283.26	282.79	0.5677

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
163	296.96	287.50	283.28	282.80	0.5647
164	296.43	286.06	282.90	281.72	0.5667
165	296.30	285.98	282.86	281.64	0.5672
166	296.36	285.71	282.84	281.42	0.5688
167	296.36	286.81	283.01	282.12	0.5658
168	296.38	286.82	282.89	282.05	0.5657
169	296.55	286.06	283.02	282.00	0.5691
170	296.48	285.99	282.94	281.95	0.5689
171	296.52	286.41	282.95	282.15	0.5666
172	296.50	286.14	282.98	282.05	0.5674
173	296.57	286.21	283.06	282.12	0.5672
174	296.50	286.42	282.99	282.19	0.5661
175	296.55	287.48	283.21	282.54	0.5654
176	296.56	287.49	283.24	282.55	0.5661
177	296.75	286.61	283.11	282.36	0.5698
178	296.76	286.78	283.13	282.42	0.5670
179	296.69	286.55	283.10	282.36	0.5695
180	296.73	286.79	283.16	282.46	0.5674
181	296.73	287.10	283.10	282.47	0.5664

Table K.1 Continued.

Test	$T_{a,evap,in}[K]$	$T_{a,evap,out}[K]$	$Dew_{a,evap,in}[K]$	$Dew_{a,evap,in}[K]$	$\dot{V}_{a,evap}[m^3/s]$
182	296.71	287.09	283.13	282.51	0.5658
183	296.61	287.06	282.97	282.41	0.5696
184	296.70	288.17	283.02	282.62	0.5682
185	296.70	288.20	283.23	282.82	0.5694
186	296.60	287.08	283.02	282.49	0.5699
187	296.60	287.28	283.06	282.54	0.5673
188	296.62	287.30	283.06	282.59	0.5666
189	296.61	287.65	283.00	282.55	0.5664
190	296.58	287.63	282.99	282.53	0.5661
191	296.98	287.88	283.33	282.92	0.5695
192	296.61	288.82	283.07	282.75	0.5691
193	296.61	288.82	283.06	282.75	0.5689
194	296.96	288.08	283.31	282.98	0.5668
195	296.99	288.49	283.33	283.00	0.5657
196	296.96	289.65	283.32	283.21	0.5688

Table K.2. : Air-side measurement across condenser in tests in system I.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
1	310.91	317.39	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
2	310.84	317.94	0
3	310.77	317.01	0
4	305.19	312.29	0
5	305.53	312.31	0
6	305.56	312.63	0
7	305.20	311.65	0
8	299.65	306.72	0
9	305.26	312.10	0
10	302.39	309.42	0
11	299.94	306.45	0
12	300.00	306.76	0
13	299.98	307.04	0
14	294.04	301.07	0
15	296.84	303.82	0
16	299.60	306.40	0
17	294.42	300.92	0
18	294.43	301.16	0
19	297.00	303.69	0
20	288.70	295.61	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
21	294.48	301.52	0
22	297.26	303.64	0
23	291.28	298.31	0
24	291.30	298.08	0
25	288.69	295.38	0
26	297.16	304.12	0
27	288.54	295.07	0
28	294.10	300.64	0
29	294.23	300.98	0
30	288.62	295.53	0
31	294.14	301.17	0
32	288.51	294.84	0
33	288.51	295.20	0
34	291.49	297.87	0
35	291.40	298.28	0
36	294.43	301.15	0
37	288.49	294.94	0
38	288.50	295.47	0
39	288.66	295.39	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
40	293.56	300.47	29.2
41	293.47	300.11	14.6
42	293.47	300.08	14.6
43	293.57	300.50	29.2
44	293.60	301.90	56.1
45	293.63	301.83	56.1
46	293.60	301.14	41.4
47	296.58	303.11	14.6
48	293.60	301.08	41.4
49	296.55	303.07	14.6
50	296.69	303.49	29.2
51	296.77	304.09	41.4
52	296.78	304.84	56.1
53	296.64	303.46	29.2
54	296.76	304.09	41.4
55	299.79	306.35	14.6
56	300.02	307.29	41.4
57	296.76	304.78	56.1
58	299.96	307.21	41.4

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
59	299.73	306.25	14.6
60	299.88	306.64	29.2
61	299.89	306.70	29.2
62	303.34	310.13	29.2
63	303.27	309.74	14.6
64	303.35	310.16	29.2
65	303.25	309.75	14.6
66	303.43	310.66	41.4
67	303.41	310.65	41.4
68	306.39	312.78	14.6
69	306.55	313.19	29.2
70	293.47	299.88	0
71	293.47	299.92	0
72	296.58	303.04	0
73	296.58	303.04	0
74	299.81	306.26	0
75	299.80	306.26	0
76	293.59	299.90	0
77	293.58	299.86	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
78	303.25	309.60	0
79	296.68	302.95	0
80	303.31	309.68	0
81	296.66	302.97	0
82	299.89	306.20	0
83	299.92	306.22	0
84	293.58	299.78	0
85	306.42	312.77	0
86	296.62	302.82	0
87	303.28	309.50	0
88	303.29	309.56	0
89	299.30	305.41	0
90	299.76	305.93	0
91	299.70	305.88	0
92	306.38	312.60	0
93	303.18	309.28	0
94	303.16	309.24	0
95	306.19	312.18	0
96	293.56	299.90	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
97	293.55	299.88	0
98	293.55	298.98	0
99	293.52	299.43	0
100	293.55	299.54	0
101	293.55	298.95	0
102	296.66	303.07	0
103	296.62	302.66	0
104	296.66	303.05	0
105	296.65	302.97	0
106	296.65	302.24	0
107	296.63	302.72	0
108	296.63	302.92	0
109	296.68	302.26	0
110	299.87	306.29	0
111	299.89	306.31	0
112	299.87	306.09	0
113	299.85	306.03	0
114	299.89	305.68	0
115	299.85	305.65	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
116	299.86	306.20	0
117	299.84	306.20	0
118	303.28	309.69	0
119	303.28	309.52	0
120	303.29	309.69	0
121	303.28	309.51	0
122	303.29	309.17	0
123	303.28	309.62	0
124	303.27	309.17	0
125	303.27	309.62	0
126	306.41	312.68	0
127	306.38	312.79	0
128	306.43	312.43	0
129	306.42	312.77	0
130	293.54	299.90	0
131	293.55	299.87	0
132	293.58	299.94	0
133	293.56	299.83	0
134	293.55	299.73	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
135	293.54	299.78	0
136	296.65	303.05	0
137	296.64	302.99	0
138	296.64	302.98	0
139	296.62	303.02	0
140	296.64	302.70	0
141	296.66	302.88	0
142	296.62	302.73	0
143	296.66	302.92	0
144	299.89	306.33	0
145	299.90	306.37	0
146	299.88	306.03	0
147	299.90	306.27	0
148	299.90	306.31	0
149	299.88	306.16	0
150	299.89	306.03	0
151	299.89	306.15	0
152	303.31	309.71	0
153	303.30	309.46	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
154	303.30	309.69	0
155	303.29	309.57	0
156	303.30	309.74	0
157	303.25	309.41	0
158	303.32	309.77	0
159	303.29	309.58	0
160	306.42	312.87	0
161	306.44	312.82	0
162	306.39	312.58	0
163	306.41	312.69	0
164	293.52	299.41	0
165	293.52	299.39	0
166	293.56	299.63	0
167	293.53	298.98	0
168	293.50	298.98	0
169	296.64	302.78	0
170	296.64	302.77	0
171	296.64	302.55	0
172	296.62	302.66	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
173	296.63	302.70	0
174	296.61	302.53	0
175	296.68	302.12	0
176	296.64	302.12	0
177	299.87	306.02	0
178	299.80	305.85	0
179	299.79	305.94	0
180	299.85	305.91	0
181	299.88	305.79	0
182	299.85	305.78	0
183	303.25	309.35	0
184	299.56	304.97	0
185	299.62	305.06	0
186	303.23	309.33	0
187	303.21	309.22	0
188	303.24	309.24	0
189	303.24	309.11	0
190	303.25	309.11	0
191	306.39	312.53	0

Table K.2 Continued.

Test	$T_{a,cond,in}[K]$	$T_{a,cond,out}[K]$	Area blockage ratio [%]
192	303.06	308.42	0
193	303.05	308.41	0
194	306.37	312.42	0
195	306.40	312.34	0
196	306.11	311.48	0

Table K.3. : Refrigerant pressure in tests in system I.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
1	530.26	1990.64	668.72
2	556.05	2029.52	698.44
3	498.07	1945.20	634.59
4	525.39	1794.11	669.83
5	504.14	1772.74	646.52
6	527.69	1808.14	672.10
7	482.52	1733.52	621.70
8	498.66	1590.75	643.28
9	503.90	1762.88	644.32
10	504.23	1676.52	647.62
11	460.33	1547.17	601.77

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
12	477.30	1571.81	620.67
13	499.14	1603.43	645.42
14	473.29	1407.37	618.53
15	478.52	1488.15	622.80
16	476.36	1550.37	618.53
17	437.73	1365.34	580.26
18	450.98	1385.34	594.53
19	458.61	1455.88	600.67
20	444.97	1240.24	590.40
21	471.71	1418.03	617.49
22	440.63	1442.08	580.26
23	453.58	1303.85	596.53
24	437.99	1281.20	581.23
25	423.11	1211.42	567.23
26	477.98	1496.38	622.80
27	412.43	1192.81	555.37
28	432.00	1351.19	568.20
29	451.51	1378.98	594.53
30	443.36	1237.58	588.33

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
31	472.13	1408.33	618.53
32	394.39	1168.35	535.17
33	424.59	1206.83	568.20
34	415.03	1259.97	555.37
35	450.15	1310.64	594.53
36	452.75	1383.76	596.53
37	407.23	1184.64	546.62
38	442.83	1227.63	586.33
39	425.64	1210.85	570.20
40	442.05	1416.65	583.30
41	433.47	1369.14	573.23
42	433.74	1368.29	573.23
43	441.22	1417.31	580.26
44	465.00	1609.49	602.81
45	465.09	1607.81	601.77
46	452.01	1491.15	589.36
47	447.73	1475.33	589.36
48	451.98	1488.32	589.36
49	446.73	1466.42	586.33

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
50	456.78	1526.11	592.47
51	465.85	1607.07	602.81
52	477.84	1724.74	614.32
53	455.18	1523.26	592.47
54	462.51	1601.19	599.64
55	464.46	1586.26	602.81
56	479.60	1731.69	616.39
57	475.40	1720.06	612.19
58	478.64	1727.43	615.36
59	462.88	1581.77	599.64
60	470.06	1641.75	606.95
61	470.54	1644.54	607.98
62	484.99	1775.47	621.70
63	478.02	1712.19	615.36
64	485.05	1776.31	622.80
65	477.37	1713.05	615.36
66	493.95	1856.97	630.25
67	493.61	1852.45	630.25
68	489.28	1822.76	627.01

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
69	496.32	1886.63	631.35
70	419.57	1323.76	557.37
71	421.28	1326.08	559.30
72	435.27	1426.57	573.23
73	437.26	1428.07	575.23
74	449.72	1538.28	587.36
75	453.19	1541.80	591.43
76	414.99	1320.28	553.44
77	416.59	1319.74	554.41
78	463.34	1660.75	601.77
79	429.59	1419.43	567.23
80	466.30	1666.22	603.84
81	430.76	1420.97	568.20
82	446.60	1536.09	585.30
83	447.42	1534.73	584.26
84	411.22	1313.81	547.65
85	477.24	1776.30	614.32
86	425.41	1413.30	562.27
87	457.25	1652.34	594.53

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
88	459.52	1657.31	596.53
89	432.86	1499.93	568.20
90	439.63	1522.15	575.23
91	438.45	1518.98	574.20
92	469.19	1765.90	606.95
93	451.36	1641.21	587.36
94	448.87	1637.96	584.26
95	459.80	1746.29	595.57
96	417.92	1322.86	557.37
97	420.12	1322.31	560.27
98	345.18	1252.15	479.46
99	383.98	1291.30	520.07
100	386.37	1297.28	522.90
101	345.19	1252.03	479.46
102	432.43	1424.86	573.23
103	401.79	1399.32	538.96
104	432.25	1422.41	573.23
105	423.24	1417.92	563.23
106	365.79	1364.23	501.73

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
107	402.40	1402.06	539.93
108	422.14	1416.37	562.27
109	365.89	1364.94	501.73
110	445.79	1535.27	586.33
111	445.17	1536.43	584.26
112	420.86	1521.81	559.30
113	420.07	1519.86	557.37
114	386.75	1488.83	522.90
115	387.26	1489.09	523.86
116	436.88	1530.29	577.23
117	437.02	1530.28	577.23
118	459.96	1658.46	598.60
119	435.43	1645.44	574.20
120	460.23	1659.81	598.60
121	435.47	1645.32	574.20
122	405.31	1619.55	541.86
123	451.43	1653.92	591.43
124	405.35	1618.95	541.86
125	451.87	1654.63	591.43

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
126	449.81	1763.90	585.30
127	473.07	1766.97	610.12
128	421.51	1744.19	556.41
129	464.77	1770.37	602.81
130	422.65	1322.57	562.27
131	415.77	1312.43	556.41
132	423.10	1323.64	561.30
133	414.97	1310.44	555.37
134	409.75	1300.11	549.58
135	410.64	1300.89	549.58
136	436.84	1422.09	576.19
137	432.35	1414.70	572.20
138	430.86	1414.23	572.20
139	436.22	1421.70	576.19
140	411.81	1382.63	549.58
141	422.73	1401.58	563.23
142	410.61	1381.52	549.58
143	424.11	1403.65	564.27
144	451.78	1538.09	591.43

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
145	453.02	1541.21	593.50
146	425.41	1497.61	564.27
147	445.36	1528.36	585.30
148	445.49	1530.27	585.30
149	436.30	1513.86	577.23
150	425.54	1498.17	564.27
151	436.44	1514.73	576.19
152	460.72	1655.17	598.60
153	440.21	1621.55	575.23
154	461.08	1654.99	599.64
155	451.15	1637.53	591.43
156	468.46	1665.08	606.95
157	439.19	1618.83	576.19
158	467.87	1664.71	605.91
159	451.13	1638.81	590.40
160	480.34	1776.67	617.49
161	472.65	1768.85	609.01
162	450.88	1736.41	587.36
163	461.85	1751.27	598.60

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
164	462.70	1270.08	594.53
165	460.42	1268.96	593.50
166	447.11	1293.32	581.23
167	491.97	1226.65	618.53
168	491.67	1226.59	617.49
169	462.68	1395.26	598.60
170	461.97	1394.84	596.53
171	478.08	1369.28	609.01
172	467.64	1384.92	600.67
173	468.60	1386.80	601.77
174	477.97	1369.11	609.01
175	510.82	1323.81	636.73
176	510.93	1324.32	636.73
177	477.27	1506.26	612.19
178	482.88	1494.52	615.36
179	477.10	1502.50	611.15
180	482.92	1496.35	615.36
181	494.34	1480.83	624.87
182	494.14	1480.25	625.97

Table K.3 Continued.

Test	$P_{r,comp,in}[kPa]$	$P_{r,comp,out}[kPa]$	$P_{r,evap,in}[kPa]$
183	490.19	1624.43	622.80
184	529.71	1418.83	655.35
185	530.26	1422.37	657.55
186	489.92	1624.12	622.80
187	496.81	1614.20	628.11
188	497.05	1615.04	628.11
189	510.44	1597.89	642.18
190	510.34	1598.35	641.07
191	507.46	1748.35	638.87
192	551.40	1535.94	673.20
193	551.24	1535.93	675.48
194	515.35	1735.68	646.52
195	530.83	1720.86	658.66
196	574.54	1646.69	697.06

Table K.4. : Refrigerant temperature in tests in system I.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
1	286.57	372.15	320.51	282.54	286.57
2	290.83	375.15	321.67	284.00	290.83

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
3	280.20	367.64	320.53	280.83	280.20
4	291.37	370.92	314.87	282.59	291.37
5	288.37	369.06	315.63	281.41	288.37
6	291.46	371.21	315.43	282.70	291.46
7	283.13	365.26	314.97	280.16	283.13
8	291.34	366.31	308.02	281.27	291.34
9	288.56	369.16	315.34	281.32	288.56
10	290.55	368.32	311.58	281.47	290.55
11	286.21	363.31	309.06	279.08	286.21
12	289.01	365.03	309.09	280.08	289.01
13	291.83	366.90	308.49	281.36	291.83
14	291.25	361.91	301.07	280.00	291.25
15	290.67	363.84	304.77	280.22	290.67
16	289.51	365.39	308.63	279.99	289.51
17	286.70	359.30	302.43	277.94	286.70
18	288.92	360.56	302.21	278.70	288.92
19	286.99	361.26	305.59	279.04	286.99
20	290.65	357.52	294.49	278.46	290.65
21	291.41	362.21	301.79	279.94	291.41

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
22	283.79	359.37	306.21	277.94	283.79
23	290.29	359.36	298.04	278.81	290.29
24	287.92	357.83	298.57	277.99	287.92
25	288.69	356.46	295.05	277.18	288.69
26	290.97	364.30	305.19	280.20	290.97
27	286.86	355.21	295.07	276.53	286.86
28	286.06	359.28	301.63	277.28	286.06
29	289.58	361.07	301.97	278.73	289.58
30	291.26	357.99	294.40	278.35	291.26
31	292.20	362.81	301.26	279.96	292.20
32	284.51	354.08	295.25	275.38	284.51
33	289.28	356.82	294.91	277.24	289.28
34	285.43	356.43	299.12	276.56	285.43
35	290.72	359.66	298.14	278.72	290.72
36	289.49	361.03	302.22	278.84	289.49
37	286.68	355.45	294.72	276.03	286.68
38	291.50	358.12	294.36	278.25	291.50
39	289.56	357.18	295.08	277.38	289.56
40	285.52	359.57	305.08	278.08	285.52

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
41	285.91	359.31	302.55	277.53	285.91
42	286.11	358.99	302.59	277.55	286.11
43	285.03	359.27	304.86	277.90	285.03
44	280.85	360.51	311.20	279.13	280.85
45	280.74	360.51	311.19	279.11	280.74
46	282.74	359.82	307.54	278.43	282.74
47	285.38	361.63	306.56	278.41	285.38
48	283.19	359.78	307.43	278.41	283.19
49	285.38	361.01	306.29	278.24	285.38
50	283.86	361.19	308.18	278.61	283.86
51	281.88	361.61	310.87	279.16	281.88
52	278.56	361.90	314.46	279.76	278.56
53	282.78	360.90	308.28	278.62	282.78
54	281.16	361.32	310.73	278.99	281.16
55	284.07	362.62	310.00	279.17	284.07
56	278.91	362.79	314.40	279.87	278.91
57	278.36	361.21	314.36	279.66	278.36
58	278.84	362.65	314.31	279.82	278.84
59	283.74	362.45	309.89	279.00	283.74

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
60	282.10	362.86	311.81	279.36	282.10
61	282.08	362.96	311.88	279.41	282.08
62	279.45	364.36	315.55	280.15	279.45
63	281.27	363.99	313.75	279.84	281.27
64	279.52	364.52	315.80	280.23	279.52
65	280.90	363.86	313.76	279.80	280.90
66	279.36	364.82	317.96	280.60	279.36
67	279.30	364.77	317.93	280.58	279.30
68	279.67	365.55	317.04	280.42	279.67
69	279.33	366.13	318.69	280.67	279.33
70	284.80	357.05	301.13	276.67	284.80
71	284.76	357.06	301.22	276.74	284.76
72	284.32	359.49	304.89	277.52	284.32
73	284.19	359.14	304.98	277.66	284.19
74	281.81	360.28	308.71	278.33	281.81
75	282.42	360.09	308.72	278.52	282.42
76	283.67	356.55	301.40	276.41	283.67
77	283.16	356.13	301.55	276.50	283.16
78	278.55	360.89	312.63	279.07	278.55

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
79	282.04	358.03	305.06	277.22	282.04
80	279.23	361.33	312.61	279.22	279.23
81	282.31	357.82	305.02	277.29	282.31
82	281.02	359.72	308.90	278.20	281.02
83	280.91	359.49	308.76	278.14	280.91
84	282.66	355.32	301.50	276.12	282.66
85	278.80	362.54	315.92	279.77	278.80
86	281.18	356.99	305.06	276.91	281.18
87	278.23	360.11	312.37	278.69	278.23
88	278.38	360.56	312.46	278.82	278.38
89	277.55	356.92	308.04	277.28	277.55
90	279.38	358.54	308.54	277.67	279.38
91	278.69	358.38	308.47	277.61	278.69
92	278.62	361.69	315.79	279.35	278.62
93	277.58	359.46	312.25	278.32	277.58
94	277.34	359.12	312.17	278.18	277.34
95	277.84	360.46	315.40	278.77	277.84
96	287.71	359.69	299.85	276.67	287.71
97	287.29	359.44	300.07	276.80	287.29

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
98	288.87	365.81	295.58	271.97	288.87
99	288.92	362.65	297.22	274.49	288.92
100	288.94	362.96	297.40	274.63	288.94
101	288.81	365.29	295.67	271.96	288.81
102	287.08	362.71	303.89	277.56	287.08
103	289.24	366.36	300.91	275.60	289.24
104	287.14	362.73	303.78	277.54	287.14
105	288.17	364.19	302.67	276.96	288.17
106	289.35	368.98	299.26	273.37	289.35
107	289.20	366.50	301.00	275.66	289.20
108	288.39	364.43	302.56	276.90	288.39
109	289.34	368.95	299.29	273.38	289.34
110	286.70	365.37	307.68	278.23	286.70
111	286.69	365.33	307.62	278.17	286.69
112	289.52	369.73	304.96	276.76	289.52
113	289.44	369.61	304.85	276.67	289.44
114	289.87	372.39	303.00	274.64	289.87
115	289.93	372.42	303.01	274.68	289.93
116	288.18	367.13	306.56	277.75	288.18

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
117	288.20	367.21	306.57	277.76	288.20
118	286.28	368.04	311.49	278.91	286.28
119	289.33	372.76	308.96	277.58	289.33
120	286.37	368.05	311.50	278.93	286.37
121	289.25	372.72	309.01	277.60	289.25
122	290.28	375.88	306.93	275.75	290.28
123	287.99	370.18	310.72	278.53	287.99
124	290.28	375.83	306.98	275.75	290.28
125	288.01	370.18	310.65	278.52	288.01
126	288.59	375.02	312.29	278.20	288.59
127	285.10	369.48	314.91	279.56	285.10
128	290.56	378.97	310.65	276.60	290.56
129	286.97	372.04	314.12	279.13	286.97
130	287.16	358.79	301.12	276.91	287.16
131	288.27	360.24	301.29	276.59	288.27
132	287.08	358.86	301.28	276.86	287.08
133	288.15	359.94	301.28	276.52	288.15
134	288.88	360.56	301.44	276.21	288.88
135	288.65	360.84	301.36	276.19	288.65

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
136	286.43	361.54	305.00	277.73	286.43
137	287.43	362.62	305.09	277.50	287.43
138	287.71	362.69	305.07	277.48	287.71
139	286.45	361.60	304.98	277.69	286.45
140	289.22	364.77	304.74	276.22	289.22
141	288.84	363.74	304.85	276.97	288.84
142	289.22	364.94	304.74	276.21	289.22
143	288.82	363.84	304.96	277.04	288.82
144	285.39	363.51	308.81	278.52	285.39
145	285.71	363.73	308.94	278.64	285.71
146	289.44	368.09	308.44	277.03	289.44
147	287.49	365.28	308.70	278.18	287.49
148	287.46	365.32	308.73	278.19	287.46
149	289.05	367.02	308.78	277.79	289.05
150	289.44	368.27	308.45	277.05	289.44
151	288.86	366.79	308.56	277.71	288.86
152	287.37	368.13	312.43	278.92	287.37
153	289.16	371.23	311.84	277.67	289.16
154	287.38	368.14	312.50	278.98	287.38

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
155	289.15	370.15	312.51	278.52	289.15
156	284.67	365.54	312.57	279.35	284.67
157	289.11	371.23	311.85	277.69	289.11
158	284.63	365.54	312.60	279.32	284.63
159	289.05	370.14	312.43	278.48	289.05
160	282.61	366.72	315.95	279.94	282.61
161	285.90	369.64	315.75	279.50	285.90
162	288.40	373.76	315.35	278.29	288.40
163	287.42	371.65	315.69	278.95	287.42
164	284.61	357.66	300.72	278.70	284.61
165	284.69	357.70	300.71	278.63	284.69
166	285.79	357.35	301.08	278.01	285.79
167	279.97	357.48	300.17	279.96	279.97
168	279.98	357.45	300.15	279.94	279.98
169	283.95	359.39	304.78	278.91	283.95
170	283.26	359.24	304.64	278.84	283.26
171	281.47	359.61	304.27	279.48	281.47
172	283.50	359.75	304.38	279.04	283.50
173	283.50	359.78	304.49	279.10	283.50

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
174	281.60	359.67	304.26	279.49	281.60
175	280.33	359.55	303.60	280.92	280.33
176	280.34	359.61	303.60	280.94	280.34
177	281.81	361.30	308.35	279.64	281.81
178	280.75	361.00	307.96	279.82	280.75
179	282.17	361.22	308.20	279.60	282.17
180	280.91	361.16	308.05	279.83	280.91
181	280.06	360.89	307.82	280.34	280.06
182	280.07	360.92	307.82	280.35	280.07
183	280.24	363.12	311.78	280.23	280.24
184	281.30	361.31	306.73	281.87	281.30
185	281.59	361.59	307.07	282.00	281.59
186	279.97	362.46	311.72	280.20	279.97
187	280.14	362.19	311.51	280.47	280.14
188	280.14	362.25	311.58	280.50	280.14
189	280.74	362.81	311.65	281.22	280.74
190	280.58	362.65	311.46	281.12	280.58
191	280.72	364.51	315.09	281.05	280.72
192	282.25	364.12	310.29	282.79	282.25

Table K.4 Continued.

Test	$T_{r,comp,in}[K]$	$T_{r,cond,in}[K]$	$T_{r,cond,out}[K]$	$T_{r,evap,in}[K]$	$T_{r,evap,out}[K]$
193	282.41	364.24	310.48	282.88	282.41
194	281.07	364.69	315.09	281.43	281.07
195	281.53	365.72	314.87	282.06	281.53
196	283.46	368.19	313.60	283.91	283.46

Table K.5. : Other measurement data in tests in system I.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
1	3278	0.06178	0	0	2.72
2	3384	0.06418	0	0	2.72
3	3127	0.05868	0	0	2.72
4	3168	0.06257	0	0	2.72
5	3088	0.06079	0	0	2.72
6	3185	0.06283	0	0	2.72
7	2998	0.05878	0	0	2.72
8	2973	0.06141	0	0	2.72
9	3083	0.06062	0	0	2.72
10	3043	0.06104	0	0	2.72
11	2840	0.05787	0	0	2.72
12	2900	0.05921	0	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
13	2984	0.06137	0	0	2.72
14	2790	0.06032	0	0	2.72
15	2857	0.05998	0	0	2.72
16	2889	0.05904	0	0	2.72
17	2681	0.05673	0	0	2.72
18	2721	0.05811	0	0	2.72
19	2785	0.05833	0	0	2.72
20	2613	0.05895	0	0	2.72
21	2796	0.05989	0	0	2.72
22	2720	0.05692	0	0	2.72
23	2686	0.05897	0	0	2.72
24	2638	0.05763	0	0	2.72
25	2552	0.05674	0	0	2.72
26	2864	0.05991	0	0	2.72
27	2522	0.05573	0	0	2.72
28	2663	0.05612	0	0	2.72
29	2721	0.05800	0	0	2.72
30	2608	0.05864	0	0	2.72
31	2792	0.06003	0	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
32	2469	0.05381	0	0	2.72
33	2550	0.05690	0	0	2.72
34	2560	0.05520	0	0	2.72
35	2678	0.05866	0	0	2.72
36	2725	0.05811	0	0	2.72
37	2505	0.05502	0	0	2.72
38	2607	0.05848	0	0	2.72
39	2559	0.05681	0	0	2.72
40	2738	0.05710	0	0	2.72
41	2692	0.05664	0	0	2.72
42	2694	0.05678	0	0	2.72
43	2737	0.05732	0	0	2.72
44	2897	0.05823	0	0	2.72
45	2897	0.05814	0	0	2.72
46	2807	0.05798	0	0	2.72
47	2776	0.05684	0	0	2.72
48	2807	0.05797	0	0	2.72
49	2777	0.05751	0	0	2.72
50	2829	0.05802	0	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
51	2898	0.05834	0	0	2.72
52	2981	0.05825	0	0	2.72
53	2823	0.05792	0	0	2.72
54	2883	0.05809	0	0	2.72
55	2881	0.05846	0	0	2.72
56	2994	0.05868	0	0	2.72
57	2969	0.05833	0	0	2.72
58	2986	0.05859	0	0	2.72
59	2874	0.05831	0	0	2.72
60	2922	0.05843	0	0	2.72
61	2929	0.05849	0	0	2.72
62	3029	0.05889	0	0	2.72
63	2977	0.05874	0	0	2.72
64	3030	0.05885	0	0	2.72
65	2975	0.05881	0	0	2.72
66	3090	0.05900	0	0	2.72
67	3091	0.05884	0	0	2.72
68	3064	0.05898	0	0	2.72
69	3115	0.05898	0	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
70	2625	0.05532	0	0	2.72
71	2629	0.05557	0	0	2.72
72	2719	0.05623	0	0	2.72
73	2723	0.05659	0	0	2.72
74	2807	0.05720	0	0	2.72
75	2816	0.05788	0	0	2.72
76	2611	0.05499	0	0	2.72
77	2614	0.05541	0	0	2.72
78	2899	0.05764	0	0	2.72
79	2699	0.05610	0	0	2.72
80	2913	0.05812	0	0	2.72
81	2703	0.05631	0	0	2.72
82	2797	0.05665	0	0	2.72
83	2801	0.05685	0	0	2.72
84	2598	0.05504	0	0	2.72
85	2998	0.05815	0	0	2.72
86	2686	0.05586	0	0	2.72
87	2880	0.05704	0	0	2.72
88	2892	0.05730	0	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
89	2738	0.05584	0	0	2.72
90	2772	0.05633	0	0	2.72
91	2764	0.05631	0	0	2.72
92	2969	0.05738	0	0	2.72
93	2851	0.05650	0	0	2.72
94	2840	0.05629	0	0	2.72
95	2916	0.05611	0	0	2.72
96	2630	0.05469	0	62.23	2.72
97	2635	0.05489	0	62.34	2.72
98	2353	0.04431	0	224.37	2.72
99	2513	0.05006	0	175.23	2.72
100	2521	0.05019	0	175.56	2.72
101	2355	0.04442	0	226.12	2.72
102	2724	0.05541	0	63.78	2.72
103	2624	0.05112	0	180.65	2.72
104	2722	0.05535	0	63.09	2.72
105	2700	0.05406	0	114.00	2.72
106	2475	0.04577	0	238.48	2.72
107	2628	0.05111	0	180.77	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
108	2695	0.05393	0	113.75	2.72
109	2476	0.04592	0	237.82	2.72
110	2823	0.05595	0	62.45	2.72
111	2821	0.05586	0	62.68	2.72
112	2747	0.05217	0	184.27	2.72
113	2742	0.05210	0	184.58	2.72
114	2603	0.04725	0	251.47	2.72
115	2606	0.04731	0	250.51	2.72
116	2799	0.05467	0	114.32	2.72
117	2798	0.05470	0	114.12	2.72
118	2926	0.05652	0	61.66	2.72
119	2853	0.05267	0	186.88	2.72
120	2927	0.05647	0	61.11	2.72
121	2854	0.05277	0	186.02	2.72
122	2732	0.04827	0	263.54	2.72
123	2904	0.05505	0	113.22	2.72
124	2732	0.04832	0	262.51	2.72
125	2905	0.05513	0	112.87	2.72
126	2955	0.05312	0	189.79	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
127	3016	0.05691	0	60.22	2.72
128	2843	0.04900	0	273.64	2.72
129	2997	0.05554	0	113.32	2.72
130	2633	0.05548	0	0	2.63
131	2612	0.05464	0	0	2.53
132	2634	0.05551	0	0	2.63
133	2609	0.05457	0	0	2.53
134	2586	0.05433	0	0	2.45
135	2591	0.05439	0	0	2.45
136	2727	0.05638	0	0	2.63
137	2713	0.05594	0	0	2.53
138	2709	0.05578	0	0	2.53
139	2724	0.05644	0	0	2.63
140	2641	0.05319	0	0	2.34
141	2681	0.05464	0	0	2.45
142	2637	0.05333	0	0	2.34
143	2688	0.05493	0	0	2.45
144	2828	0.05703	0	0	2.63
145	2832	0.05708	0	0	2.63

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	bp[%]	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
146	2745	0.05332	0	0	2.34
147	2813	0.05638	0	0	2.53
148	2813	0.05627	0	0	2.53
149	2781	0.05518	0	0	2.45
150	2744	0.05360	0	0	2.34
151	2784	0.05504	0	0	2.45
152	2920	0.05680	0	0	2.53
153	2853	0.05386	0	0	2.34
154	2921	0.05682	0	0	2.53
155	2890	0.05552	0	0	2.45
156	2937	0.05783	0	0	2.63
157	2849	0.05379	0	0	2.34
158	2933	0.05790	0	0	2.63
159	2890	0.05568	0	0	2.45
160	3023	0.05794	0	0	2.63
161	3010	0.05696	0	0	2.53
162	2942	0.05375	0	0	2.34
163	2977	0.05560	0	0	2.45
164	2674	0.05202	19	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
165	2669	0.05192	19	0	2.72
166	2659	0.05431	7	0	2.72
167	2699	0.04819	28	0	2.72
168	2698	0.04806	28	0	2.72
169	2755	0.05482	7	0	2.72
170	2755	0.05479	7	0	2.72
171	2779	0.05238	19	0	2.72
172	2762	0.05380	14	0	2.72
173	2764	0.05394	14	0	2.72
174	2780	0.05253	19	0	2.72
175	2820	0.04794	28	0	2.72
176	2822	0.04810	28	0	2.72
177	2861	0.05502	7	0	2.72
178	2869	0.05410	14	0	2.72
179	2859	0.05501	7	0	2.72
180	2870	0.05403	14	0	2.72
181	2896	0.05264	19	0	2.72
182	2897	0.05257	19	0	2.72
183	2966	0.05506	7	0	2.72

Table K.5 Continued.

Test	$\dot{W}_{comp}[W]$	$\dot{m}_r[kg/s]$	$bp[\%]$	$\Delta P_{liquidline}$[kPa]	M_{mea}[kg]
184	2939	0.04764	28	0	2.72
185	2942	0.04763	28	0	2.72
186	2965	0.05519	7	0	2.72
187	2979	0.05437	14	0	2.72
188	2979	0.05429	14	0	2.72
189	3014	0.05280	19	0	2.72
190	3016	0.05276	19	0	2.72
191	3084	0.05580	7	0	2.72
192	3081	0.04683	28	0	2.72
193	3080	0.04675	28	0	2.72
194	3102	0.05483	14	0	2.72
195	3147	0.05303	19	0	2.72
196	3219	0.04693	28	0	2.72

APPENDIX L. INSTRUCTIONS TO USE THE GRAPHICAL USER INTERFACE TO SIMULATE THE FAULT IMPACTS ON MULTIPLE SYSTEMS

L.1 Files

The program should come with the following files:

- *executable.exe* for the graphical user interface (GUI) of the simulation
- *MCRInstaller.exe* for the standard libraries from MATLAB
- *ParforProgressClient2.java* and *ParforProgressServer2.java* for progress timeline during the simulation
- *Matlab_DLL.dll* for the heat exchanger models in dynamic link libraries
- MATLAB binary files with extensions *.mat* in multiple folders to store the parameters of the system models

L.2 Requirement

The program uses the following external components:

- MATLAB library from 64-bit MATLAB version 7.12 for standard libraries (Mathworks, 2011)

- REFPROP from National Institute of Standards and Technology (NIST) for refrigerant property calculation (Lemmon et al., 2013)
- 64-bit dynamic link libraries for heat exchanger models

To prepare the computer to use these external components during the simulation, the MATLAB libraries should be installed by running *MCRInstaller.exe* and REFPROP fluid files (*R410A.ppf, R407C.ppf, R22.fld, R32.fld, R125.fld*, etc.) should be stored in the path *C:\Program Files (x86) \REFPROP \fluids*. The machine should also be a 64-bit Windows machine in order to use the dynamic link library *Matlab_DLL.dll*.

L.3 Procedures to Simulate the Systems under Different Faulted Conditions

1. Run the *executable.exe* to see the screen in Figure L.1.

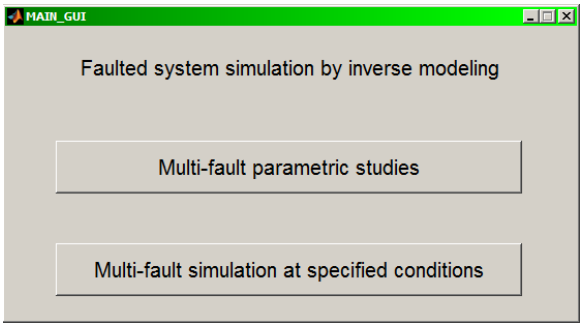

Figure L.1. Main screen of GUI of the simulation.

2. There are two options which allow the users to enter the input conditions by two different methods. The first option, *Multi-fault parameter studies*, conducts

parametric studies of the performance of the system under different input conditions, including fault levels. The screen in Figure L.2 will appear.

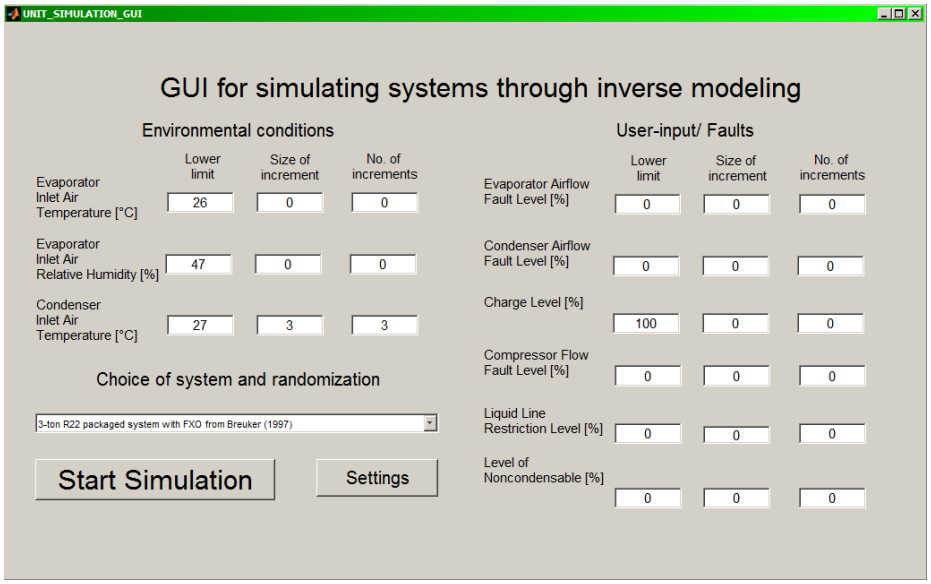

Figure L.2. Screen for parameter studies with different input conditions by simulation.

3. In this screen, the input conditions to be simulated are defined by a lower limit, a size of increment and a number of increments of the environmental conditions and fault levels. The choice of the systems is done under Choice of system and randomization. For instance, under the inputs in Figure L.2, the simulation will operate with the input conditions in Table L.1.

4. When *Settings* is chosen, the users can choose the zero threshold for the implicit solver and the number of central processing units (CPUs) for parallel computing.

5. Press *Start Simulation* to begin the simulation with the input conditions.

Table L.1. Input conditions to the simulation according to Figure L.2.

Evaporator inlet air temperature [° C]	Evaporator inlet air relative humidity [%]	Condenser inlet air temperature [° C]	Evaporator fouling level [%]	Condenser fouling level [%]	Charge level [%]	Compressor Flow Fault Level [%]	Liquid Line Restriction Level [%]
26	47	27	0	0	100	0	0
26	47	30	0	0	100	0	0
26	47	33	0	0	100	0	0
26	47	36	0	0	100	0	0

6. Choose a location to save a simplified output file in the format of Microsoft Excel Binary file (.xlsb) and a detailed output file in the format of MATLAB binary file (.mat). If the file name is the same as other files in the same directory, the MATLAB binary file will be replaced and the Excel binary file will be modified.
7. If the Excel binary file is locked by other programs, a warning dialog box will appear. The file should be unlocked before the simulation can continue.
8. Wait for the simulation to complete.
9. After simulation, the Excel binary file will be opened automatically. It contains simulation outputs in SI unit in the worksheet *Data_SI*, simulation outputs in IP units in the worksheet *Data_IP* and a nomenclature list in the worksheet *Nomenclature*.
10. The other option on the main screen Figure L.1, *Multi-fault simulation at specified conditions*, is used to simulate the system operation at specific conditions. The screen to enter the input conditions under this option is shown in Figure L.3.
11. The screen in Figure L.3 allows the user to specify the input conditions and do not need to specify the size of increment and the number of increments of input conditions under the option *Multi-fault parametric studies*. For instance, the input conditions specified in Figure L.3 are shown in Table L.2.

Table L.2. Input conditions to the simulation according to Figure L.2.

Evaporator inlet air temperature [° C]	Evaporator inlet air relative humidity [%]	Condenser inlet air temperature [° C]	Evaporator fouling level [%]	Condenser fouling level [%]	Charge level [%]	Compressor Flow Fault Level [%]	Liquid Line Restriction Level [%]
26.67	45	30	0	0	95	0	0
26.67	45	30	0	0	105	0	0
26.67	45	35	0	0	95	0	0
26.67	45	35	0	0	105	0	0

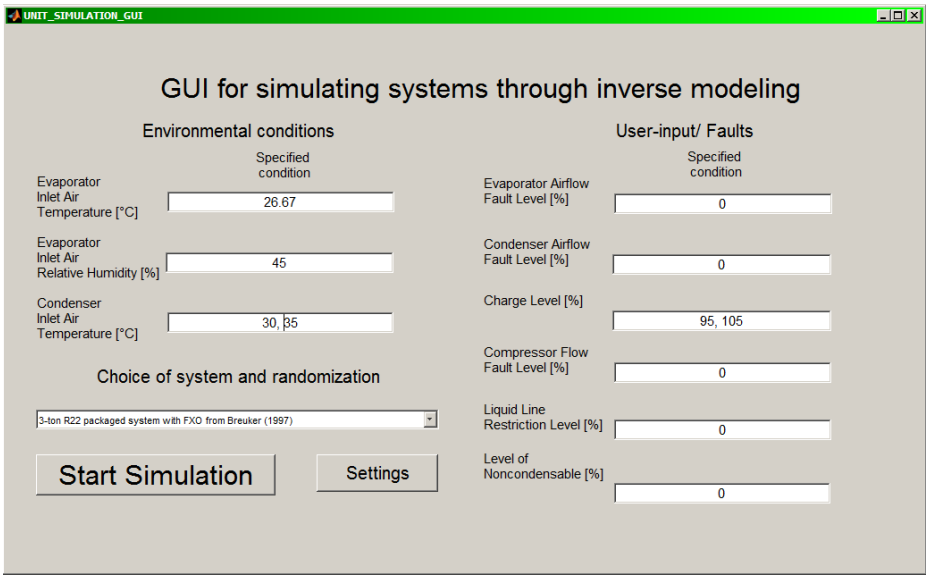

Figure L.3. Screen to enter specific input conditttions to the simulation.

12. The other settings of the option *Multi-fault simulation at specified conditions* are the same as the option *Multi-fault parametric studies*, and the same output files are created at the end of the simulation.

APPENDIX M. MATLAB PROGRAM CODE OF THE SYSTEM MODEL

M.1 Solver Initialization

```
function filename = cycle_solver_for_GUI(...
    xlsFileName, xlsPathName, matFileName, matPathName, ...
    data_air_in_temp, data_air_out_temp, vol_air_in, vol_air_out, ...
    charge_level, vl_level, ll_level, nc_level, fault_case, ...
    refri, Pc, Tc, Standard_inflow, Standard_outflow, ...
    Standard_charge, Comp, Cond, EXV, Evap, HotGas, LiquidLine,...
    SuctionLine, Accumulator_stat, ig_coeff, system_choice_string, ...
    num_core, xtol, ftol, UnitSystem, cusp_cal...
    )

% cycle_solver_for_GUI receives arrays of inputs of various
% conditions and the file inoformation required to write an Excel
% Binary file .xslb and a Matlab Data file .mat to generate simulation
% results according to the inverse modeling technique. The list of
% input variables is as follows:

% xlsFilename: string of the name of the output .xlsb file
% xlsPathName: string of the path of the output .xlsb file
```

```
19 % matFileName: string of the name of the output .mat file
20 % matPathName: string of the path of the output .mat file
21 % data_air_in_temp: dry-bulb air temp. at the inlet of the evaporator in
22 % degree Celcius in the first column, dewpoint in degree Celcius/
23 % Fahrenheit in the second column, wet-bulb temp. in the third column
24 % in degree Celcius/ Fahrenheit and relative humidity in the forth
25 % column. Note that if data in humidity are redundant, the program
26 % will prioritize to do the calculation in the order
27 % of dewpoint, wet-bulb temp. and relative humidity
28 % data_air_out_temp: dry-bulb air temp. at the inlet of the condenser in
29 % degree Celcius/ Fahrenheit
30 % vol_air_in: ratio of airflow across the evaporator to the standard
31 % airflow
32 % vol_air_out: ratio of airflow across the condenser to the standard
33 % airflow
34 % charge_level: ratio of charge inside the system to the standard
35 % charging amount
36 % fault_case: an array showing the fault types to be simulated
37 % vl_level: compressor valve leakage fault level, between 0 to 1
38 % ll_level: liquid line restriction fault level, between 0 to 1
39 % nc_level: non-condensable fault level, between 0 to 1
40 % refri: name of refrigerant
41 % Pc: critical pressure in kPa
42 % Tc: critical temperature in K
43 % Standard_inflow: Standard indoor air flow in m3/s
44 % Standard_outflow: Standard outdoor air flow in m3/s
```

```
45 % Standard_charge: Standard charge level in kg
46 % Comp, Cond, EXV, Evap, HotGas, LiquidLine, SuctionLine,
47 % Accumulator_stat: Structures storing parameters
48 % ig_coeff: coefficients for initial guess
49 % system_choice_string: name of system simulated
50 % num_core: number of computer cores for parallel computing
51 % xtol: tolerance on input for numerical solver
52 % ftol: tolerance on function values in numerical solver
53 % UnitSystem: Unit system at input
54 % cusp_cal: check if cusp calculation is needed
55 % and it outputs a filename of the output file after simulation
56
57 %% Initialization
58 % Do unit conversion if the UnitSystem is English
59 if strcmp(UnitSystem, 'English')
60     % from F to C
61     data_air_out_temp = Unit_Conversion(data_air_out_temp,'F','C');
62     for i = 1:3
63         data_air_in_temp(:,i) = Unit_Conversion(...
64             data_air_in_temp(:,i),'F','C'...
65         );
66     end
67     UnitSystem = 'SI';
68 end
69
70 % loading libraries
```

```
71 if num_core > 1
72     try
73         matlabpool(num_core); % try setting up multiple loops
74     catch err
75         try
76             matlabpool close; % if catch an error, close and reset it
77             matlabpool(num_core);
78         catch err
79             pll_computing = false;
80         end
81     end
82 end
83 if ¬isunix()
84     addpath('c:\windows\system32\');
85 end
86 try
87     if num_core > 1
88         if isunix()
89 pctRunOnAll loadlibrary('Matlab_DLL.so', @Matlab_def,'alias','lib')
90 pctRunOnAll loadlibrary('./refprop.so', @rp_proto,'alias','refprop')
91         else
92 pctRunOnAll loadlibrary('Matlab_DLL.dll', @Matlab_def,'alias','lib')
93         end
94     else
95         if isunix()
96             loadlibrary('Matlab_DLL.so', @Matlab_def,'alias','lib');
```

```
97              loadlibrary('refprop.so', @rp_proto,'alias','refprop')
98          else
99              loadlibrary('Matlab_DLL.dll', @Matlab_def,'alias','lib');
100         end
101     end
102 catch err
103     msgbox(err.message);
104     for i = 1:length(err.stack)
105         msgbox(...
106             [...
107                 err.stack(i).file,' ',err.stack(i).name,' ',...
108                 num2str(err.stack(i).line)...
109             ]...
110         );
111     end
112     if isunix()
113         loadlibrary('Matlab_DLL.so', @Matlab_def,'alias','lib');
114         loadlibrary('refprop.so', @rp_proto,'alias','refprop')
115     else
116         loadlibrary('Matlab_DLL.dll', @Matlab_def,'alias','lib');
117     end
118 end
119
120 % make conditions the same if they are slightly different
121 data_air_in_temp(:,1) = consistent(data_air_in_temp(:,1));
122 data_air_in_temp(:,3) = consistent(data_air_in_temp(:,3));
```

```
123 data_air_out_temp = consistent(data_air_out_temp);
124 vol_air_in = consistent(vol_air_in, 1.e-4);
125 vol_air_out = consistent(vol_air_out, 1.e-4);
126
127 % Initialization of parameters
128 savefilename = [matFileName(1:strfind(matFileName,'.')-1)];
129 m = length(data_air_in_temp(:,1));
130 m_ori = m;
131 filename = xlsFileName;
132
133 % Total Volume and Max. mass of NC Calculation
134 HighVolume = pi*Cond.para.Din^2/4*Cond.para.Lt;
135 HighVolume = HighVolume + pi*HotGas.dia^2/4*HotGas.line;
136 TotalVolume = HighVolume + pi*Evap.para.Din^2/4*Evap.para.Lt;
137 TotalVolume = TotalVolume + pi*LiquidLine.dia^2/4*LiquidLine.line;
138 TotalVolume = TotalVolume + pi*SuctionLine.dia^2/4*SuctionLine.line;
139 NC_Total_mass = TotalVolume*101.325/0.2968/(26.7+273.15);
140
141 % coefficient definition
142 P_sat_comp_out_coeff = ig_coeff.P_sat_comp_out_coeff;
143 P_sat_comp_in_coeff = ig_coeff.P_sat_comp_in_coeff;
144 SH_evap_coeff = ig_coeff.SH_evap_coeff;
145 T_out_coeff = ig_coeff.T_out_coeff;
146 mdot_r_coeff = ig_coeff.mdot_r_coeff;
147
148 % declaration of variables to be output after parfor
```

```
149 func_count_total = zeros(m,1);
150 dev = zeros(m,1);
151 flag = zeros(m,1);
152 dev2 = zeros(m,1);
153 Comp_flag = zeros(m,1);
154 EXV_flag = zeros(m,1);
155 Evap_flag = zeros(m,1);
156 Cond_flag = zeros(m,1);
157 for i = 1:m
158     System(i) = System_definition();
159 end
160 Real_length = 6;
161 residual = zeros(m,Real_length);
162 Real = zeros(m,Real_length);
163 Real_temp = ones(m,Real_length)*NaN;
164 cusp = zeros(m,1);
165
166 airinlet = zeros(m,5);
167 % Start calculation
168 try
169     parfor qq = 1:raw.n
170         System_main(qq) = 1; % a statement leads to error
171     end
172 catch err
173 end
174 for i = 1:m
```

```
175     error_statement(i) = err;
176 end
177 err_indicator = zeros(m,1);
178 time_CPU = zeros(m,1);
179
180 addpath(pwd);
181 libname ='lib'; % Name of library
182
183 %% Calculation
184 display(['Running ',Comp.NBI_sys_info{1}]);
185 beg_index = 1;
186 end_index = 1;
187 new_cusp = 1;
188 cusp_checker = zeros(m,1);
189 trial = 1;
190 trial_limit = 3;
191 nf_calculation = false;
192 m_ori = m;
193 beg_index = 1;
194 end_index = length(data_air_out_temp);
195 nf_index = beg_index:end_index;
196 System_nf = [];
197 Real_nf = [];
198 residual_nf = [];
199 fig = msgbox(['Simulation begins after 5s']);
200 try
```

```
201     pause(5);
202     close(fig);
203 end
204 try % skip if the package is not installed
205     ppm = ParforProgressStarter2('Simulation in Progress', m, 0.1);
206 end
207 parfor i = beg_index:end_index
208     display([...
209         strcat('Starting case SIM',int2str(i)),' in ',...
210         Comp.NBI_sys_info{1}...
211     ]);
212     try
213         % allocating inputs
214         Pain = 101.325;
215         Tain_cond = data_air_out_temp(i,1)+273.15;
216         air_cond_inlet = calllib(libname, 'HumAir_DLL', Tain_cond, ...
217             Pain, 4, 0.2, zeros(1,5));
218         wain_cond = air_cond_inlet(2);
219         Tamb = Tain_cond;
220 
221         Tain_evap = data_air_in_temp(i,1)+273.15;
222         if isreal(...
223                 data_air_in_temp(i,2)) && ...
224                 ¬isnan(data_air_in_temp(i,2)) && ...
225                 ¬isempty(data_air_in_temp(i,2)...
226             )
```

```
227             airinlet_temp = calllib(libname, 'HumAir_DLL', ...
228                 Tain_evap, Pain, 1, data_air_in_temp(i,2)+273.15, ...
229                 zeros(1,5));
230             WB = Wetbulb(Tain_evap, data_air_in_temp(i,2)+273.15, ...
231                 Pain, libname);
232             Dain_evap = data_air_in_temp(i,2)+273.15;
233         elseif isreal(...
234                 data_air_in_temp(i,3)) && ...
235                 ¬isnan(data_air_in_temp(i,3)) && ...
236                 ¬isempty(data_air_in_temp(i,3)...
237             )
238             % solver of wet-bulb is unstable: resolve for air-water
239             % properties
240             WB = data_air_in_temp(i,3)+273.15;
241             plus = Wetbulb_residual(...
242                 Tain_evap, Tain_evap, WB, Pain, libname...
243             );
244             TD = Tain_evap-1;
245             minus = Wetbulb_residual(...
246                 Tain_evap, TD, WB, Pain, libname ...
247             );
248             if abs(plus) < 1.e-8 || plus > 0
249                 Dain_evap = Tain_evap;
250             else
251                 iter_WB = 0;
252                 while plus*minus > 0 && iter_WB < 60
```

```
253                     TD = TD-1;
254                     minus = Wetbulb_residual(...
255                         Tain_evap, TD, WB, Pain, libname...
256                     );
257                     iter_WB = iter_WB + 1;
258                 end
259                 try
260                     Dain_evap = fzero(...
261                         @(TD) Wetbulb_residual(Tain_evap, TD, WB, ...
262                         Pain, libname), [TD, Tain_evap]...
263                     );
264                 catch err_WB  % wet-bulb too low: use 230K dewpoint
265                     Dain_evap = 230;
266                 end
267             end
268             airinlet_temp = calllib(...
269                 libname, 'HumAir_DLL', Tain_evap, Pain, 1, ...
270                 Dain_evap, zeros(1,5)...
271             );
272         else
273             airinlet_temp = calllib(...
274                 libname, 'HumAir_DLL', Tain_evap, Pain, 4, ...
275                 data_air_in_temp(i,4), zeros(1,5)...
276             );
277             Dain_evap = airinlet_temp(1);
278             WB = Wetbulb(Tain_evap, Dain_evap, Pain, libname);
```

```
279         end
280
281         airinlet(i,:) = airinlet_temp;
282         wain_evap = airinlet_temp(2);
283         RH_evap = airinlet_temp(4);
284
285         Vdot_evap = vol_air_in(i,1)*Standard_inflow;
286
287         Vdot_cond = vol_air_out(i,1)*Standard_outflow;
288
289         ValveLeakage = vl_level(i,1);
290         NC_level = nc_level(i,1);
291         DP_cond = [];
292         DP_evap = [];
293         hf = [];
294         hv = [];
295         rhov = [];
296         nf_set = 0;
297         timeout=0;
298         time_CPU(i,1) = cputime;
299         if (...
300                 vl_level(i,1) > 0 || ll_level(i,1) > 0 || ...
301                 nc_level(i,1) > 0 || vol_air_in(i,1) < 1 || ...
302                 vol_air_out(i,1) < 1 || charge_level(i,1) > 1 || ...
303                 charge_level(i,1) < 1 ...
304             )
```

```
305             if nf_calculation & cusp_cal
306                 % try to find non-faulted cases first if cusp points
307                 % are needed
308                 nf_cur_index = find(...
309                     data_air_in_temp(i,1)==...
310                         data_air_in_temp(nf_index,1)...
311                     & data_air_in_temp(i,3)==...
312                         data_air_in_temp(nf_index,3)&...
313                     data_air_out_temp(i,1)==...
314                         data_air_out_temp(nf_index,1));
315                 if ¬isempty(nf_cur_index)
316                     nf_set = 1;
317                     System_norm = System_nf(nf_cur_index);
318                     Real_norm = Real_nf(nf_cur_index,:);
319                     residual_norm = residual_nf(nf_cur_index,:);
320                 end
321             end
322             if nf_set == 0
323                 SCSH = struct(...
324                     'HighVolume',HighVolume,'NC_Total_mass',...
325                     NC_Total_mass,'Charge',Standard_charge,'VL',0,...
326                     'LLpercent',0,'NC_level',0, ...
327                     'LL_restriction',0, 'xtol', xtol, 'ftol', ftol, ...
328                     'Accumulator', 0,'Scaling', 0.1 ...
329                 );
330                 % redefine guess values
```

```
331                 Guess = initial_Guess(...
332                     Tain_evap, Dain_evap, Standard_inflow, ...
333                     Tain_cond, Standard_outflow, SCSH,
334                     P_sat_comp_in_coeff, P_sat_comp_out_coeff, ...
335                     SH_evap_coeff, T_out_coeff, mdot_r_coeff, Cond, ...
336                     refri, Tc, Pc ...
337                 );
338                 Real_norm = Guess;
339                 residual_norm = ones(length(Guess),1)*NaN;
340                 System_norm = System_definition();
341                 k = 0;
342                 while ((sqrt(max(residual_norm.^2))>ftol) || ...
343                         isnan(max(residual_norm))) && k < 5
344                     if Guess(3,1) > 1 & k < 2
345                         Guess(3,1) = Guess(3,1)*.5;
346                     elseif Guess(2,1) < .5 & Guess(4,1) < 0.75
347                         Guess(2,1) = Guess(2,1) + 0.1;
348                         Guess(4,1) = Guess(4,1) + 0.05;
349                     else
350                         Guess(4,1) = Guess(4,1)*0.25;
351                     end
352                     if (strcmp(EXV.Type,'TXV') | ...
353                             charge_level(i,1) < 1 | ...
354                             vol_air_in(i,1) < 0.9) & ...
355                             Guess(3,1) > 0
356                         Guess(3,1) = Guess(3,1)*2;
```

```
357                     elseif (strcmp(EXV.Type,'TXV') | ...
358                             charge_level(i,1) < 1 | ...
359                             vol_air_in(i,1) < 1) & Guess(3,1) < 0
360                         Guess(3,1) = 1;
361                     end
362                     try
363                         if Accumulator_stat == 1
364                             if strcmp(EXV.Type,'TXV')
365                                 SCSH.Accumulator = 0;
366                                 [...
367                                     System_norm,Real_norm,dum,...
368                                     residual_norm ...
369                                 ] = cycle_output(...
370                                     Guess, Tain_cond, wain_cond, ...
371                                     Standard_outflow, Tain_evap, ...
372                                     wain_evap, Standard_inflow, ...
373                                     Tamb, Pain, SCSH, Comp, ...
374                                     HotGas, ...
375                                     Cond, LiquidLine, EXV, Evap, ...
376                                     SuctionLine, refri, Tc, Pc, ...
377                                     hf, hv, rhov, DP_cond, DP_evap, ...
378                                     libname ...
379                                 );
380                                 if (System_norm.CompressorSH ≤ 1.e-3 ...
381                                         & sqrt(...
382                                             mean(residual_norm.^2) ...
```

```
383                                     )≤ftol)
384                                 SCSH.Accumulator = 1;
385                                 [...
386                                     System_norm,Real_norm,dum,...
387                                     residual_norm...
388                                 ] = cycle_output(...
389                                     Guess, Tain_cond, ...
390                                     wain_cond, Standard_outflow, ...
391                                     Tain_evap, wain_evap, ...
392                                     Vdot_evap, Tamb, Pain, ...
393                                     SCSH, Comp, HotGas, Cond, ...
394                                     LiquidLine, EXV, Evap, ...
395                                     SuctionLine, refri, ...
396                                     Tc, Pc, hf, hv, rhov, ...
397                                     DP_cond, DP_evap, libname ...
398                                 );
399                             else
400                                 SCSH.Accumulator = 1;
401                                 [System_zeroSH,Real_zeroSH,...
402                                     dev2_zeroSH,residual_zeroSH...
403                                 ]  = cycle_output(Guess, ...
404                                     Tain_cond, wain_cond, ...
405                                     Vdot_cond, Tain_evap, ...
406                                     wain_evap, Vdot_evap, Tamb, ...
407                                     Pain, SCSH, Comp, HotGas, ...
408                                     Cond, LiquidLine, EXV, ...
```

```
409                                     Evap, SuctionLine, refri, ...
410                                     Tc, Pc, hf, hv, rhov, ...
411                                     DP_cond, DP_evap, libname);
412                                 if (System_zeroSH.Charge_tuned ...
413                                         ≤ SCSH.Charge & ...
414                                         sqrt(...
415                                             mean(...
416                                                 residual_zeroSH.^2 ...
417                                             )...
418                                         )≤ftol)
419                                     System_norm = System_zeroSH;
420                                     Real_norm = Real_zeroSH;
421                                     residual_norm = residual_zeroSH;
422                                 end
423                             end
424                         else
425                             SCSH.Accumulator = 1;
426                             [System_zeroSH,Real_zeroSH,...
427                                 dev2_zeroSH,residual_zeroSH] = ...
428                                 cycle_output(...
429                                     Guess, Tain_cond, wain_cond, ...
430                                     Vdot_cond, Tain_evap, ...
431                                     wain_evap, Vdot_evap, Tamb, ...
432                                     Pain, SCSH, Comp, HotGas, ...
433                                     Cond, LiquidLine, EXV, Evap, ...
434                                     SuctionLine, refri, Tc, Pc, ...
```

```
435                     hf, hv, rhov, DP_cond, ...
436                     DP_evap, libname ...
437                 );
438             if (System_zeroSH.Charge_tuned ≤ ...
439                     SCSH.Charge & ...
440                     (sqrt(mean(...
441                     residual_zeroSH.^2 ...
442                     ))≤ftol))
443                 System_norm = System_zeroSH;
444                 Real_norm = Real_zeroSH;
445                 residual_norm = residual_zeroSH;
446             else
447                 SCSH.Accumulator = 0;
448                 [System_norm,Real_norm,dum,...
449                     residual_norm] = ...
450                     cycle_output(...
451                         Guess, Tain_cond, ...
452                         wain_cond, ...
453                         Standard_outflow, ...
454                         Tain_evap, wain_evap, ...
455                         Standard_inflow, Tamb, ...
456                         Pain, SCSH, Comp, HotGas, ...
457                         Cond, LiquidLine, EXV, ...
458                         Evap, SuctionLine, refri, ...
459                         Tc, Pc, hf, hv, rhov, ...
460                         DP_cond, DP_evap, libname ...
```

```
461                                         );
462                                     end
463                                 end
464                             else
465                                 SCSH.Accumulator = 0;
466                                 [System_norm,Real_norm,dum,...
467                                     residual_norm] = cycle_output(...
468                                         Guess, Tain_cond, wain_cond, ...
469                                         Standard_outflow, Tain_evap, ...
470                                         wain_evap, Standard_inflow, ...
471                                         Tamb, Pain, SCSH, Comp, HotGas, ...
472                                         Cond, LiquidLine, EXV, Evap, ...
473                                         SuctionLine, refri, Tc, Pc, hf, ...
474                                         hv, rhov, DP_cond, DP_evap, ...
475                                         libname ...
476                                     );
477                                 end
478                             catch err
479                                 residual_norm = ones(length(Guess),1)*NaN;
480                             end
481                             k = k + 1;
482                         end
483                         while length(Real_norm)<length(Real_length)
484                             Real_norm = [Real_norm; 0;];
485                         end
486                     end
```

```
487             FIR_capacity = NaN;
488             FIR_COP = NaN;
489         else
490             residual_norm = zeros(Real_length,1);
491             FIR_capacity = 1;
492             FIR_COP = 1;
493         end
494
495         SCSH = struct('Charge',charge_level(i,1)*Standard_charge,...
496             'VL',ValveLeakage,'LLpercent',ll_level(i,1),...
497             'NC_level',NC_level,'LL_restriction',0,...
498             'HighVolume',HighVolume,'NC_Total_mass',NC_Total_mass,...
499             'xtol', xtol, 'ftol', ftol, 'Accumulator', 0);
500
501         % guesses for data known cases
502         % use non-faulted case as initial guess
503         Guess = zeros(5,1);
504         if ¬isnan(sum(Real_temp(i,:)))
505             Guess(1:5) = Real_temp(i,1:5);
506         elseif end_index > length(nf_index)
507             Guess(1:5) = Real_norm(1:5);
508         else
509             Guess = initial_Guess(Tain_evap, Dain_evap, Vdot_evap, ...
510                 Tain_cond, Vdot_cond, SCSH, P_sat_comp_in_coeff, ...
511                 P_sat_comp_out_coeff, SH_evap_coeff, T_out_coeff, ...
512                 mdot_r_coeff, Cond, refri, Tc, Pc);
```

```
513         end
514         Real_II = Guess;
515         System_II = System_definition();
516         residual_II = ones(length(Guess),1)*Inf;
517
518         % setting up the solver
519         % solve for a good temperature difference
520         func_count = 0;
521         k = 0;
522         while ((sqrt(mean(residual_II.^2))>ftol) || ...
523                 isnan(max(residual_II))) && k < 4
524             if end_index ≤ length(nf_index) & k == 1
525                 Guess = initial_Guess(Tain_evap, Dain_evap, ...
526                     Vdot_evap, Tain_cond, Vdot_cond, SCSH, ...
527                     P_sat_comp_in_coeff, P_sat_comp_out_coeff, ...
528                     SH_evap_coeff, T_out_coeff, mdot_r_coeff, Cond, ...
529                     refri, Tc, Pc);
530             else
531                 if Guess(3,1) > 1 & k < 2
532                     Guess(3,1) = Guess(3,1)*.5;
533                 elseif Guess(2,1) < .5 & Guess(4,1) < 0.75
534                     Guess(2,1) = Guess(2,1) + 0.1;
535                     Guess(4,1) = Guess(4,1) + 0.1;
536                 else
537                     Guess(4,1) = Guess(4,1)*0.25;
538                 end
```

```
539                 if (strcmp(EXV.Type,'TXV') | ...
540                         charge_level(i,1) < 1 | ...
541                         vol_air_in(i,1) < 0.9) & Guess(3,1) > 0
542                     Guess(3,1) = Guess(3,1)*2;
543                 elseif (strcmp(EXV.Type,'TXV') | ...
544                         charge_level(i,1) < 1 | ...
545                         vol_air_in(i,1) < 1) & Guess(3,1) < 0
546                     Guess(3,1) = 1;
547                 end
548             end
549             try
550                 if Accumulator_stat == 1
551                     if strcmp(EXV.Type,'TXV')
552                         SCSH.Accumulator = 0;
553                         [System_II,Real_II,dev2(i,1),residual_II,...
554                             flag(i,1),func_output,err] = ...
555                             cycle_output(...
556                                 Guess, Tain_cond, wain_cond, ...
557                                 Vdot_cond, Tain_evap, wain_evap, ...
558                                 Vdot_evap, Tamb, Pain, SCSH, Comp, ...
559                                 HotGas, Cond, LiquidLine, EXV, Evap, ...
560                                 SuctionLine, refri, Tc, Pc, hf, hv, ...
561                                 rhov, DP_cond, DP_evap, libname ...
562                             );
563                         func_count = func_count + func_output.funcCount;
564                         if (System_II.CompressorSH ≤ 1.e-3 & ...
```

```
565                     (sqrt(mean(residual_II.^2))≤ftol))
566                 SCSH.Accumulator = 1;
567                 [System_II,Real_II,dev2(i,1),residual_II,...
568                     flag(i,1),func_output,err] = ...
569                     cycle_output(...
570                         Guess, Tain_cond, wain_cond, ...
571                         Vdot_cond, Tain_evap, wain_evap, ...
572                         Vdot_evap, Tamb, Pain, SCSH, ...
573                         Comp, ...
574                         HotGas, Cond, LiquidLine, EXV, ...
575                         Evap, SuctionLine, refri, Tc, Pc, ...
576                         hf, hv, rhov, DP_cond, DP_evap, ...
577                         libname ...
578                     );
579             elseif (sqrt(mean(residual_II.^2))>ftol)
580                 SCSH.Accumulator = 1;
581                 [System_zeroSH,Real_zeroSH, ...
582                     dev2_zeroSH,residual_zeroSH,flag_zero,...
583                     func_output_zeroSH,err_zeroSH] = ...
584                     cycle_output(...
585                         Guess, Tain_cond, wain_cond, ...
586                         Vdot_cond, Tain_evap, wain_evap, ...
587                         Vdot_evap, Tamb, Pain, SCSH, ...
588                         Comp, HotGas, Cond, LiquidLine, ...
589                         EXV, Evap, SuctionLine, refri, ...
590                         Tc, Pc, hf, hv, rhov, DP_cond, ...
```

```
591                                     DP_evap, libname ...
592                                 );
593                             if (System_zeroSH.Charge_tuned ≤ ...
594                                     SCSH.Charge & ...
595                                     (sqrt(mean(...
596                                         residual_zeroSH.^2...
597                                     ))≤ftol))
598                                 System_II = System_zeroSH;
599                                 Real_II = Real_zeroSH;
600                                 dev2(i,1) = dev2_zeroSH;
601                                 residual_II = residual_zeroSH;
602                                 flag(i,1) = flag_zero;
603                                 func_output = func_output_zeroSH;
604                                 err = err_zeroSH;
605                             end
606                         end
607                     else
608                         SCSH.Accumulator = 1;
609                         [System_zeroSH,Real_zeroSH,dev2_zeroSH,...
610                             residual_zeroSH,flag_zero,...
611                             func_output_zeroSH,err_zeroSH] = ...
612                             cycle_output(...
613                                 Guess, Tain_cond, wain_cond, ...
614                                 Vdot_cond, Tain_evap, wain_evap, ...
615                                 Vdot_evap, Tamb, Pain, SCSH, Comp, ...
616                                 HotGas, Cond, LiquidLine, EXV, Evap, ...
```

```
617                                 SuctionLine, refri, Tc, Pc, hf, hv, ...
618                                 rhov, DP_cond, DP_evap, libname ...
619                             );
620                         if (System_zeroSH.Charge_tuned ≤ SCSH.Charge ...
621                                 & ...
622                                 (sqrt(mean(residual_zeroSH.^2))≤ftol))
623                             System_II = System_zeroSH;
624                             Real_II = Real_zeroSH;
625                             dev2(i,1) = dev2_zeroSH;
626                             residual_II = residual_zeroSH;
627                             flag(i,1) = flag_zero;
628                             func_output = func_output_zeroSH;
629                             err = err_zeroSH;
630                         else
631                             SCSH.Accumulator = 0;
632                             [System_II,Real_II,dev2(i,1),...
633                                 residual_II,flag(i,1),func_output,...
634                                 err] = cycle_output(...
635                                     Guess, Tain_cond, wain_cond, ...
636                                     Vdot_cond, Tain_evap, wain_evap, ...
637                                     Vdot_evap, Tamb, Pain, SCSH, ...
638                                     Comp, ...
639                                     HotGas, Cond, LiquidLine, EXV, ...
640                                     Evap, SuctionLine, refri, Tc, ...
641                                     Pc, hf, hv, rhov, DP_cond, ...
642                                     DP_evap, libname ...
```

```
643                                 );
644                             func_count = func_count + ...
645                                 func_output.funcCount;
646                         end
647                     end
648                 else
649                     SCSH.Accumulator = 0;
650                     [System_II,Real_II,dev2(i,1),residual_II,...
651                         flag(i,1),func_output,err] = cycle_output(...
652                             Guess, Tain_cond, wain_cond, Vdot_cond, ...
653                             Tain_evap, wain_evap, Vdot_evap, Tamb, ...
654                             Pain, SCSH, Comp, HotGas, Cond, ...
655                             LiquidLine, EXV, Evap, SuctionLine, ...
656                             refri, Tc, Pc, hf, hv, rhov, DP_cond, ...
657                             DP_evap, libname ...
658                         );
659                     func_count = func_count + func_output.funcCount;
660                 end
661             catch err
662                 residual_II = ones(length(Guess),1)*NaN;
663             end
664             k = k + 1;
665             % timeout
666             if cputime - time_CPU(i,1) > 400
667                 timeout=1;
668                 break;
```

```
669             end
670         end
671         SCSH.LL_restriction = (1+SCSH.LLpercent)*(...
672             System_II.LiquidLine.Pin - System_II.EXV.Pin ...
673         );
674         System_II.LL_extra_ΔP = SCSH.LLpercent*(...
675             System_II.LiquidLine.Pin - System_II.EXV.Pin ...
676         );
677         if SCSH.LLpercent > 0 & timeout==0
678             SCSH.Accumulator = 0;
679             k_old = k;
680             k = 0;
681             residual_II = ones(length(Guess),1)*NaN;
682             while ((sqrt(mean(residual_II.^2))>xtol) || ...
683                     isnan(max(residual_II))) && k < 3 && timeout==0
684                 if k ≥ 1
685                     Guess(4,1) = Guess(4,1)*0.25;
686                     if strcmp(EXV.Type,'TXV') | ...
687                             charge_level(i,1) < 1 | ...
688                             vol_air_in(i,1) < 1
689                         Guess(3,1) = Guess(3,1)*2;
690                     end
691                 end
692                 try
693                     if Accumulator_stat == 1
694                         if strcmp(EXV.Type,'TXV')
```

```
695                         SCSH.Accumulator = 0;
696                         [System_II,Real_II,dev2(i,1),...
697                             residual_II,flag(i,1),func_output,...
698                             err] = cycle_output(...
699                                 Guess, Tain_cond, ...
700                                 wain_cond, Vdot_cond, ...
701                                 Tain_evap, ...
702                                 wain_evap, Vdot_evap, Tamb, ...
703                                 Pain, ...
704                                 SCSH, Comp, HotGas, Cond, ...
705                                 LiquidLine, EXV, Evap, ...
706                                 SuctionLine, refri, Tc, Pc, hf, ...
707                                 hv, rhov, DP_cond, DP_evap, ...
708                                 libname ...
709                             );
710                         func_count = func_count + ...
711                             func_output.funcCount;
712                         if (System_II.CompressorSH ≤ 1.e-3 & ...
713                                 (sqrt(mean(residual_II.^2))≤ftol))
714                             SCSH.Accumulator = 1;
715                             [System_II,Real_II,dev2(i,1),...
716                                 residual_II,flag(i,1),...
717                                 func_output,err] = ...
718                                 cycle_output(...
719                                     Guess, Tain_cond, wain_cond, ...
720                                     Vdot_cond, Tain_evap, ...
```

```
721                                     wain_evap, Vdot_evap, Tamb, ...
722                                     Pain, SCSH, Comp, HotGas, ...
723                                     Cond, LiquidLine, EXV, Evap, ...
724                                     SuctionLine, refri, Tc, Pc, ...
725                                     hf, hv, rhov, DP_cond, ...
726                                     DP_evap, libname ...
727                                 );
728                             elseif (sqrt(mean(residual_II.^2))>ftol)
729                                 SCSH.Accumulator = 1;
730                                 [System_zeroSH,Real_zeroSH,...
731                                     dev2_zeroSH,residual_zeroSH,...
732                                     flag_zero,...
733                                     func_output_zeroSH,err_zeroSH] = ...
734                                     cycle_output(...
735                                     Guess, Tain_cond, wain_cond, ...
736                                     Vdot_cond, Tain_evap, ...
737                                     wain_evap, Vdot_evap, Tamb, ...
738                                     Pain, SCSH, Comp, HotGas, ...
739                                     Cond, LiquidLine, EXV, Evap, ...
740                                     SuctionLine, refri, Tc, Pc, ...
741                                     hf, hv, rhov, DP_cond, ...
742                                     DP_evap, libname ...
743                                 );
744                                 if (System_zeroSH.Charge_tuned ≤ ...
745                                     SCSH.Charge & ...
746                                     (sqrt(mean(...
```

```
747                                         residual_zeroSH.^2 ...
748                                     ))≤ftol))
749                                     System_II = System_zeroSH;
750                                     Real_II = Real_zeroSH;
751                                     dev2(i,1) = dev2_zeroSH;
752                                     residual_II = residual_zeroSH;
753                                     flag(i,1) = flag_zero;
754                                     func_output = func_output_zeroSH;
755                                     err = err_zeroSH;
756                                 end
757                             end
758                         else
759                             SCSH.Accumulator = 1;
760                             [System_zeroSH,Real_zeroSH,dev2_zeroSH,...
761                                 residual_zeroSH,flag_zero,...
762                                 func_output_zeroSH,err_zeroSH] = ...
763                                 cycle_output(...
764                                     Guess, Tain_cond, wain_cond, ...
765                                     Vdot_cond, Tain_evap, wain_evap, ...
766                                     Vdot_evap, Tamb, Pain, SCSH, ...
767                                     Comp, ...
768                                     HotGas, Cond, LiquidLine, EXV, ...
769                                     Evap, SuctionLine, refri, Tc, Pc, ...
770                                     hf, hv, rhov, DP_cond, DP_evap, ...
771                                     libname ...
772                                 );
```

```
773                         if (System_zeroSH.Charge_tuned ≤ ...
774                                 SCSH.Charge & ...
775                                 (sqrt(mean(...
776                                     residual_zeroSH.^2 ...
777                                 ))≤ftol))
778                             System_II = System_zeroSH;
779                             Real_II = Real_zeroSH;
780                             dev2(i,1) = dev2_zeroSH;
781                             residual_II = residual_zeroSH;
782                             flag(i,1) = flag_zero;
783                             func_output = func_output_zeroSH;
784                             err = err_zeroSH;
785                         else
786                             SCSH.Accumulator = 0;
787                             [System_II,Real_II,dev2(i,1),...
788                                 residual_II,flag(i,1),func_output,...
789                                 err]  = cycle_output(...
790                                     Guess, Tain_cond, wain_cond, ...
791                                     Vdot_cond, Tain_evap, ...
792                                     wain_evap, Vdot_evap, Tamb, ...
793                                     Pain, SCSH, Comp, HotGas, ...
794                                     Cond, LiquidLine, EXV, Evap, ...
795                                     SuctionLine, refri, Tc, Pc, ...
796                                     hf, hv, rhov, DP_cond, ...
797                                     DP_evap, libname ...
798                                 );
```

```
799                                 func_count = func_count + ...
800                                     func_output.funcCount;
801                             end
802                         end
803                     else
804                         SCSH.Accumulator = 0;
805                         [System_II,Real_II,dev2(i,1),...
806                             residual_II,flag(i,1),func_output,err...
807                             ]  = cycle_output(...
808                                 Guess, Tain_cond, wain_cond, ...
809                                 Vdot_cond, Tain_evap, wain_evap, ...
810                                 Vdot_evap, Tamb, Pain, SCSH, Comp, ...
811                                 HotGas, Cond, LiquidLine, EXV, ...
812                                 Evap, SuctionLine, refri, Tc, Pc, ...
813                                 hf, hv, rhov, DP_cond, DP_evap, ...
814                                 libname ...
815                             );
816                         func_count = func_count + func_output.funcCount;
817                     end
818                 catch err
819                     residual_II = ones(length(Real_II),1)*NaN;
820                 end
821                 k = k + 1;
822                 % timeout
823                 if cputime - time_CPU(i,1) > 750
824                     timeout=1;
```

```
825                     break;
826                 end
827             end
828             k = k + k_old;
829         end
830         COP = System_II.Evap.Q/System_II.Comp.W;
831 
832         % calculate FIR
833         if isnan(FIR_capacity)
834             try
835                 FIR_capacity = System_II.Evap.Q/System_norm.Evap.Q;
836                 FIR_COP = COP/(System_norm.Evap.Q/System_norm.Comp.W);
837             end
838         end
839         time_CPU(i,1) = cputime - time_CPU(i,1);
840         func_count_total(i,1) = func_count;
841         System(i) = System_II;
842         System(i).SOLVED = 1;
843 
844         % dimensionalize the residual to a temperature for checking
845         dev(i) = (System(i).SuctionLine.hin-System(i).Evap.hout) /...
846             propertyRH('C','P',System(i).Comp.Pin,'Q',1,refri);
847         residual(i,:) = residual_II;
848         while length(Real_II)<Real_length
849             Real_II = [Real_II; 0;];
850         end
```

```
851         Real(i,:) = Real_II';
852
853         % checking model status
854         [System_II.EXV.status dum System_II.EXV.status_x] = ...
855             EXV_status(System_II, EXV, Pc, Tc, refri);
856         [status_Q status_ΔP] = Cond_status(...
857             System_II, Cond, Vdot_cond, Pain, Pc, Tc, refri, ...
858             libname ...
859         );
860         System_II.Cond.status_Q = status_Q;
861         System_II.Cond.status_ΔP = status_ΔP;
862         [status_Q status_SHR status_ΔP] = Evap_status(...
863             System_II, Evap, Vdot_evap, Pain, Pc, Tc, refri, ...
864             libname ...
865         );
866         System_II.Evap.status_Q = status_Q;
867         System_II.Evap.status_SHR = status_SHR;
868         System_II.Evap.status_ΔP = status_ΔP;
869         System(i) = System_II;
870         System(i).Accumulator_stat = Accumulator_stat;
871
872         % offset fan heat
873         if strcmp(Evap.para.type,'Split')
874             Tain_evap_offset = Tain_evap + ...
875                 System_II.Evap.FanPower/System_II.Evap.mdot_a/(...
876                     1006+1860*System_II.Evap.wain ...
```

```
877                 );
878             Taout_evap_offset = System_II.Evap.Taout;
879             Taout_evap = System_II.Evap.Taout;
880         else
881             Tain_evap_offset = Tain_evap;
882             Taout_evap_offset = System_II.Evap.Taout;
883             Taout_evap = System_II.Evap.Taout + ...
884                 System_II.Evap.FanPower/System_II.Evap.mdot_a/(...
885                     1006+1860*System_II.Evap.waout ...
886                 );
887         end
888
889         % allocate data for writing in SI units
890         Tain_evap = Unit_Conversion(Tain_evap,'K','C');
891         Dain_evap = Unit_Conversion(Dain_evap,'K','C');
892         WB = Unit_Conversion(WB,'K','C');
893         Tain_evap_offset = Unit_Conversion(Tain_evap_offset,'K','C');
894
895         waout_evap = System_II.Evap.waout;
896         airoutlet = calllib(...
897             libname,'HumAir_DLL',Taout_evap, Pain, 2, waout_evap, ...
898             zeros(1,5) ...
899         );
900         Daout_evap = airoutlet(1);
901         WB_out = Wetbulb(Taout_evap, Daout_evap, Pain, libname);
902         Taout_evap = Unit_Conversion(Taout_evap,'K','C');
```

```
903         Daout_evap = Unit_Conversion(Daout_evap,'K','C');
904         WB_out = Unit_Conversion(WB_out,'K','C');
905         RH_out = airoutlet(4);
906         Taout_evap_offset = ...
907             Unit_Conversion(Taout_evap_offset,'K','C');
908
909         Tamb = Unit_Conversion(Tamb,'K','C');
910
911         P_LL = System_II.LiquidLine.Pout;
912         T_LL = Unit_Conversion(propertyRH(...
913             'T','P',P_LL,'H',System_II.LiquidLine.hout,refri ...
914         ),'K','C');
915
916         P_suc = System_II.Comp.Pin;
917         T_suc = Unit_Conversion(propertyRH(...
918             'T','P',P_suc,'H',System_II.Comp.hin,refri ...
919         ),'K','C');
920
921         P_dischg = System_II.Comp.Pout;
922         T_dischg = Unit_Conversion(propertyRH(...
923             'T','P',P_dischg,'H',System_II.Comp.hout,refri...
924         ),'K','C');
925
926         Power = System_II.Comp.W + System_II.Cond.FanPower + ...
927             System_II.Evap.FanPower;
928
```

```
929         T_air_ce = Unit_Conversion(System_II.Cond.Taout,'K','C');
930
931         T_sat_e = Unit_Conversion(propertyRH(...
932             'T','P',System_II.Comp.Pin,'Q',1,refri...
933         ),'K','C');
934         T_sat_c = Unit_Conversion(propertyRH(...
935             'T','P',System_II.Cond.Pout,'Q',1,refri...
936         ),'K','C');
937
938         Power_comp = System_II.Comp.W;
939
940         Dummy = zeros(1,3);
941
942         Q_ref = System_II.Evap.Q;
943         Q_air = Q_ref;
944         SHR = System_II.Evap.SHR;
945         COP = Q_ref/System_II.Comp.W;
946         SH = System_II.EvaporatorSH;
947         SC = System_II.CondenserSC;
948         m_ref = System_II.mdot_r;
949
950         Chrg = System_II.Charge_tuned;
951         Chrg_level = charge_level(i,1)*100;
952
953         V_i = Unit_Conversion(Vdot_evap,'/s','/h');
954         V_i_nom = Unit_Conversion(Standard_inflow,'/s','/h');
```

```
955         V_i_level = vol_air_in(i,1)*100;
956
957         V_o = Unit_Conversion(Vdot_cond,'/s','/h');
958         V_o_nom = Unit_Conversion(Standard_outflow,'/s','/h');
959         V_o_level = vol_air_out(i,1)*100;
960
961         Dummy_II= 0;
962
963         LL_res = System_II.LL_extra_ΔP;
964         ll_level(i,1) = ll_level(i,1)*100;
965
966         NC_acc = System_II.NC_amount;
967         nc_level(i,1) = nc_level(i,1)*100;
968
969         Vlv_leakage = System_II.ValveLeakFlow;
970         vl_level(i,1) = vl_level(i,1)*100;
971
972         FIR_capacity = FIR_capacity;
973         FIR_COP = FIR_COP;
974
975         Sim_Note = 0;
976         if flag(i,1) ≤ 0
977             Sim_Note = 2;
978         elseif timeout==1
979             Sim_Note = 1.5;
980         elseif sqrt(max(residual_II.^2)) > ftol || FIR_COP==65535
```

```
981             Sim_Note = 1;
982         elseif sqrt(max(residual_norm.^2)) > ftol
983             Sim_Note = 0.9;
984         elseif System_II.EXV.status_x > 0
985             Sim_Note = 0.85;
986         elseif System_II.EXV.status > 0
987             Sim_Note = 0.8;
988         elseif System_II.Charge_calculate.status > 0
989             Sim_Note = 0.7;
990         elseif System_II.Evap.status_Q > 0
991             Sim_Note = 0.6;
992         elseif System_II.Cond.status_Q > 0
993             Sim_Note = 0.5;
994         elseif System_II.Comp.W_status > 0
995             Sim_Note = 0.4;
996         elseif System_II.Comp.m_status > 0
997             Sim_Note = 0.05;
998         elseif System_II.Evap.status_SHR > 0
999             Sim_Note = 0.03;
1000        elseif System_II.HotGas.ΔP_status > 0 ...
1001            || System_II.Cond.status_ΔP > 0 ...
1002            || System_II.LiquidLine.ΔP_status > 0 ...
1003            || System_II.SuctionLine.ΔP_status > 0 ...
1004            || System_II.Evap.status_ΔP > 0
1005            Sim_Note = 0.02;
1006        elseif System_II.Comp.HL_status > 0 ...
```

```
1007            || System_II.HotGas.Q_status > 0 ...
1008            || System_II.LiquidLine.Q_status > 0 ...
1009            || System_II.SuctionLine.Q_status > 0
1010            Sim_Note = 0.01;
1011        end
1012
1013        data(i,:) = [Tain_evap Dain_evap WB RH_evap ...
1014            Taout_evap Daout_evap ...
1015            WB_out RH_out Tamb P_LL T_LL P_suc ...
1016            T_suc P_dischg T_dischg Power T_air_ce T_sat_e T_sat_c ...
1017            Power_comp Tain_evap_offset Taout_evap_offset Comp.Type ...
1018            Dummy Q_ref Q_air SHR COP SH SC m_ref Chrg Chrg_level ...
1019            V_i V_i_nom V_i_level V_o V_o_nom V_o_level Dummy_II ...
1020            LL_res ll_level(i,1) ...
1021            NC_acc nc_level(i,1) Vlv_leakage vl_level(i,1)...
1022            FIR_capacity FIR_COP Sim_Note cusp(i,1)];
1023
1024        % allocate data for writing in IP units
1025        Tain_evap = Unit_Conversion(Tain_evap,'C','F');
1026        Dain_evap = Unit_Conversion(Dain_evap,'C','F');
1027        WB = Unit_Conversion(WB,'C','F');
1028        Tain_evap_offset = Unit_Conversion(Tain_evap_offset,'C','F');
1029
1030        Taout_evap = Unit_Conversion(Taout_evap,'C','F');
1031        Daout_evap = Unit_Conversion(Daout_evap ,'C','F');
1032        WB_out = Unit_Conversion(WB_out,'C','F');
```

```
1033        Taout_evap_offset = Unit_Conversion(Taout_evap_offset,'C','F');
1034
1035        Tamb = Unit_Conversion(Tamb,'C','F');
1036
1037        P_LL = Unit_Conversion(P_LL,'kPa','psi');
1038        T_LL = Unit_Conversion(T_LL,'C','F');
1039
1040        P_suc = Unit_Conversion(P_suc,'kPa','psi');
1041        T_suc = Unit_Conversion(T_suc,'C','F');
1042
1043        P_dischg = Unit_Conversion(P_dischg,'kPa','psi');
1044        T_dischg = Unit_Conversion(T_dischg,'C','F');
1045
1046        T_air_ce = Unit_Conversion(T_air_ce,'C','F');
1047
1048        T_sat_e = Unit_Conversion(T_sat_e,'C','F');
1049        T_sat_c = Unit_Conversion(T_sat_c,'C','F');
1050
1051        Q_ref = Unit_Conversion(System_II.Evap.Q,'W','Btu/h');
1052        Q_air = Q_ref;
1053        SH = Unit_Conversion(SH,'K','R');
1054        SC = Unit_Conversion(SC,'K','R');
1055        m_ref = Unit_Conversion(...
1056            Unit_Conversion(m_ref,'/s','/min'),'kg','lbm'...
1057        );
1058
```

```
1059        Chrg = Unit_Conversion(Chrg,'kg','lbm');
1060
1061        V_i = Unit_Conversion(Unit_Conversion(...
1062            V_i,'/h','/s'...
1063        ),'m3/s','cfm');
1064        V_i_nom = Unit_Conversion(...
1065            Unit_Conversion(V_i_nom,'/h','/s'),'m3/s','cfm'...
1066        );
1067
1068        V_o = Unit_Conversion(...
1069            Unit_Conversion(V_o,'/h','/s'),'m3/s','cfm' ...
1070        );
1071        V_o_nom = Unit_Conversion(...
1072            Unit_Conversion(V_o_nom,'/h','/s'),'m3/s','cfm' ...
1073        );
1074
1075        LL_res = Unit_Conversion(LL_res,'kPa','psi');
1076
1077        NC_acc = Unit_Conversion(NC_acc,'kg','lbm');
1078
1079        Vlv_leakage = Unit_Conversion(...
1080            Unit_Conversion(Vlv_leakage,'/s','/min'),'kg','lbm' ...
1081        );
1082
1083        data_IP(i,:) = [Tain_evap Dain_evap WB RH_evap ...
1084            Taout_evap Daout_evap ...
```

```
1085            WB_out RH_out Tamb P_LL T_LL P_suc ...
1086            T_suc P_dischg T_dischg Power T_air_ce T_sat_e ...
1087            T_sat_c Power_comp ...
1088            Tain_evap_offset Taout_evap_offset Comp.Type Dummy ...
1089            Q_ref Q_air SHR COP SH SC m_ref Chrg Chrg_level V_i ...
1090            V_i_nom V_i_level V_o V_o_nom V_o_level Dummy_II ...
1091            LL_res ll_level(i,1) NC_acc nc_level(i,1) ...
1092            Vlv_leakage ...
1093            vl_level(i,1) FIR_capacity ...
1094            FIR_COP Sim_Note cusp(i,1)];
1095    display(['Ending case SIM',int2str(i),[...
1096        ' at CPU time',' ' ...
1097    ], num2str(floor(time_CPU(i,1)*100.0)/100.0),'s.']);
1098    catch err % crashing
1099        display([...
1100            'Error in case SIM',int2str(i),' in ',Comp.NBI_sys_info{1}...
1101        ]);
1102        System(i) = System_definition();
1103        data(i,:) = [data_air_in_temp(i,1:4) zeros(1,2) ...
1104            zeros(1,2) data_air_out_temp(i,1) zeros(1,3) ...
1105            zeros(1,8) ...
1106            zeros(1,13) Standard_charge*charge_level(i,1) ...
1107            charge_level(i,1)*100 ...
1108            vol_air_in(i,1)*Standard_inflow*3600 ...
1109            Standard_inflow*3600 ...
1110            vol_air_in(i,1)*100 ...
```

```
1111            Standard_outflow*3600*vol_air_out(i,1) ...
1112            Standard_outflow*3600 vol_air_out(i,1)*100 0 0 ...
1113            ll_level(i,1)*100 ...
1114            0 nc_level(i,1)*100 0 vl_level(i,1)*100 0 0 3 cusp(i,1)];
1115
1116        data_IP(i,:) = [Unit_Conversion(...
1117                data_air_in_temp(i,1),'C','F'...
1118            ) ...
1119            Unit_Conversion(data_air_in_temp(i,2),'C','F') ...
1120            Unit_Conversion(data_air_in_temp(i,3),'C','F') ...
1121            Unit_Conversion(data_air_in_temp(i,4),'C','F') ...
1122            zeros(1,2) ...
1123            zeros(1,2) Unit_Conversion(...
1124                data_air_out_temp(i,1),'C','F' ...
1125            ) ...
1126            zeros(1,11) ...
1127            zeros(1,13) Unit_Conversion(...
1128                Standard_charge*charge_level(i,1),'kg','lbm' ...
1129            ) ...
1130            charge_level(i,1)*100 Unit_Conversion(...
1131                vol_air_in(i,1)*Standard_inflow,'m3/s','cfm' ...
1132            ) ...
1133            Unit_Conversion(Standard_inflow,'m3/s','cfm') ...
1134            vol_air_in(i,1)*100 ...
1135            Unit_Conversion(...
1136                Standard_outflow*vol_air_out(i,1),'m3/s','cfm' ...
```

```
1137              ) ...
1138              Unit_Conversion(Standard_outflow,'m3/s','cfm') ...
1139              vol_air_out(i,1) 0 0 ll_level(i,1) ...
1140              0 nc_level(i,1) 0 vl_level(i,1) 0 0 3 cusp(i,1)];
1141          error_statement(i) = err;
1142          err_indicator(i,1) = 1;
1143      end
1144
1145      % write simulation case name
1146      if i/1000 < 1.e-2
1147          code = strcat('SIM-',strcat('000',num2str(i)));
1148      elseif i/1000 < 1.e-1
1149          code = strcat('SIM-',strcat('00',num2str(i)));
1150      elseif i/1000 < 1
1151          code = strcat('SIM-',strcat('0',num2str(i)));
1152      else
1153          code = strcat('SIM-',num2str(i));
1154      end
1155      code_cell(i)=cellstr(code);
1156      ppm.increment(i);
1157  end
1158
1159  try
1160    delete(ppm);
1161  catch err
1162  end
```

```
1163
1164 save(strcat(matPathName,savefilename));
1165
1166 %% Writing files
1167 % writig file
1168 % .mat file
1169 save(strcat(matPathName,savefilename));
1170
1171 % .xlsb file
1172 fig = waitbar(0,'Writing files...');
1173 % .mat file in IP units
1174 % write the header of the cell array
1175 if ¬isempty(Comp.NBI_sys_info)
1176
1177     % writing nomenclature
1178     FirstColumn = {'Choice of system';'Label';'Test Unit';...
1179         'T_RA';'DP_RA';...
1180         'WB_RA';'RH_RA';'T_SA';...
1181         'DP_SA';'WB_SA';'RH_SA';'T_amb';'P_LL';...
1182         'T_LL';'P_suc';'T_suc';...
1183         'P_dischg';'T_dischg';'Power';...
1184         'T_air_ce';'T_sat_e';'T_sat_c';...
1185         'Power_comp';'T_air_ei';'T_air_eo';'Unit / Type';...
1186         'Q_ref';'Q_air';'SHR';'COP';'SH';'SC';'m_ref';'Chrg';...
1187         'Chrg%';'V_i';'V_i_nom';'V_i_%';...
1188         'V_o';'V_o_nom';'V_o_%';'dummy';...
```

```
1189          'LL restr.';'LLrestr%';'NonCond';'NonCond%';...
1190          'VlvLeak';'VlvLeak%';...
1191          'FIRcapacity';'FIRCOP';'SimNote';'cusp'};
1192
1193      SecondColumn = {system_choice_string;'Meaning';...
1194          'System identifier';...
1195          'Return Air Dry Bulb';...
1196          'Return Air Dew Point';'Return Air Wet Bulb';...
1197          'Return Air Relative Humidity';...
1198          'Supply Air drybulb';...
1199          'Supply Air Dewpoint';'Supply Air Wet Bulb';...
1200          'Supply Air Relative Humidity';'Ambient Drybulb';...
1201          'Liquid Line pressure (absolute)';...
1202          'Liquid Line Temperature';...
1203          'Suction Pressure (absolute)';'Suction Temperature';...
1204          'Discharge pressure';'Discharge temperature';...
1205          'Total power to system';...
1206          'Condenser exit air temperature';...
1207          'Evaporator refrigerant saturation temperature';...
1208          'Condenser refrigerant saturation temperature';...
1209          'Compressor electrical power';...
1210          'Evaporator Air Inlet Temperature';...
1211          'Evaporator Air Outlet Temperature';...
1212          ['Type of system: RTU (0), RTU heat pump (1), ',...
1213          'split AC (2), split heat pump (3)'];...
1214          'Refrigerant side cooling';...
```

```
1215          'Air side cooling';'Sensible Heat Ratio';...
1216          'Coefficient of performance';...
1217          'Superheat';'Subcooling';'Refrigerant mass flow rate';...
1218          'Refrigerant charge';'Charge as %nominal';...
1219          'Indoor coil airflow rate';...
1220          'Nominally correct indoor coil airflow rate';...
1221          'Airflow as percent of nominal';'Outdoor coil airflow rate';...
1222          'Nominally correct outdoor coil airflow rate';...
1223          'Outdoor coil blockage';'';...
1224          'Liquid line restriction:  pressure loss';...
1225          'Liquid line restriction:  % pressure loss';...
1226          'Mass of non-condensables in refrigerant';...
1227          'Non-condensables as %*';...
1228          'Valve Leakage';'Valve Leakage as % of total flow';...
1229          ['Ratio of Cooling Load under Faulted Conditions ',...
1230          'to Cooling Load without Fault'];...
1231          'Ratio of COP under Faunted Conditions to COP without Fault';...
1232          ['0 = converged within tolerance, ',...
1233          '0.01 = Compressor heat loss model or pipeline',...
1234          ' heat loss model out of range, ',...
1235          '0.02 = Heat exchanger pressure drop model or ',...
1236          'pipeline pressure drop model out of range, ',...
1237          '0.03 = Evaporator sensible heat ratio model out of range, ',...
1238          '0.05 = Compressor mass flow rate model out of range, ',...
1239          '0.4 = Compressor power consumption model out of range, ',...
1240          '0.5 = Condenser heat transfer rate model out of range, ',...
```

```
1241          '0.6 = Evaporator heat transfer rate model out of range, ',...
1242          '0.7 = Charge tuning equation out of range, ',...
1243          '0.8 = Expansion valve model out of range, ',...
1244          '0.85 = Expansion valve model inlet quality too high, ',...
1245          '0.9 = Normal model divergence, ',...
1246          '1 = diverged at local minimum, ',...
1247          '2 = diverged because the maximum ',...
1248          'number of iteration has exceeded, ',...
1249          '3 = diverged because of crashing'];...
1250          'Cusp point indicator'};
1251
1252      nomen = [FirstColumn, SecondColumn];
1253      xlswrite(strcat(...
1254          matPathName,[savefilename,'.xlsb']...
1255      ),nomen,'Nomenclature');
1256
1257      mat_IP_save = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,...
1258          18,19,20,21,22,23,24,...
1259          25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,...
1260          42,43,44,45,46,47,48,...
1261          49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,...
1262          66,67,68;NaN,NaN,NaN,NaN,...
1263          NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,...
1264          'T_RA','DP_RA','WB_RA',...
1265          'RH_RA','T_SA','DP_SA','WB_SA','RH_SA','T_amb',...
1266          'P_LL','T_LL',...
```

```
1267          'P_suc',...
1268          'T_suc','P_dischg','T_dischg','Power',...
1269          'T_air_ce','T_sat_e','T_sat_c',...
1270          'Power_comp','T_air_ei','T_air_eo',...
1271          'Unit / Type',NaN,'Herrick',NaN,'Q_ref','Q_air',...
1272          'SHR','COP','SH','SC','m_ref','Chrg',...
1273          'Chrg%','V_i','V_i_nom','V_i_%',...
1274          'V_o','V_o_nom','V_o_%','Blk%',...
1275          'LL restr.','LLrestr%','NonCond',...
1276          'NonCond%','VlvLeak','VlvLeak','FIRcapacity',...
1277          'FIRCOP','SimNote','Cusp';'Test Unit',...
1278          'SysID','Expansion Type','Manufacturer','Model (indoor)',...
1279          'Model (outdoor)','Nominal Capacity',...
1280          'Refrigerant','Operating Mode ',...
1281          'Compressor Type','Compressor Model',...
1282          'Target SC','Rated EER','Rated SEER','SysID',...
1283          'Data','[ F ]','[ F ]','[ F ]','[%]','[ F ]',...
1284          '[ F ]','[ F ]','[%]','[ F ]',...
1285          '[psia]','[ F ]','[psia]','[ F ]','[psia]',...
1286          '[ F ]','[W]','[ F ]','[ F ]'...
1287          ,'[ F ]','[W]','[ F ]','[ F ]',NaN,'Herrick ID',...
1288          'Test #','Fault','[Btu/hr]',...
1289          '[Btu/hr]','[-]','[-]','[ F ]','[ F ]',...
1290          '[lbm/min]','[lbm]','[%]',...
1291          '[CFM]','[CFM]','[%]','[CFM]','[CFM]',...
1292          '[%]','[%]','[psia]','[%]',...
```

```
1293          '[lbm/lbm]','[%]','[lbm/min]','[%]','[%]','[%]','','';};
1294      mat_save = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,...
1295          18,19,20,21,22,23,24,...
1296          25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,...
1297          43,44,45,46,47,48,...
1298          49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,...
1299          67,68;NaN,NaN,NaN,NaN,...
1300          NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,'T_RA',...
1301          'DP_RA','WB_RA',...
1302          'RH_RA','T_SA','DP_SA','WB_SA','RH_SA','T_amb','P_LL',...
1303          'T_LL','P_suc',...
1304          'T_suc','P_dischg','T_dischg','Power','T_air_ce',...
1305          'T_sat_e','T_sat_c',...
1306          'Power_comp','T_air_ei','T_air_eo','Unit / Type',...
1307          NaN,'Herrick',NaN,'Q_ref','Q_air',...
1308          'SHR','COP','SH','SC','m_ref','Chrg','Chrg%',...
1309          'V_i','V_i_nom','V_i_%',...
1310          'V_o','V_o_nom','V_o_%','Blk%','LL restr.',...
1311          'LLrestr%','NonCond',...
1312          'NonCond%','VlvLeak','VlvLeak','FIRcapacity',...
1313          'FIRCOP','SimNote','Cusp';'Test Unit',...
1314          'SysID','Expansion Type','Manufacturer',...
1315          'Model (indoor)',...
1316          'Model (outdoor)','Nominal Capacity','Refrigerant',...
1317          'Operating Mode ',...
1318          'Compressor Type','Compressor Model','Target SC',...
```

```
1319          'Rated EER','Rated SEER','SysID',...
1320          'Data','[ C]','[ C]','[ C]','[%]','[ C]','[ C]',...
1321          '[ C]','[%]','[ C]',...
1322          '[kPa]','[ C]','[kPa]','[ C]','[kPa]',...
1323          '[ C]','[W]','[ C]','[ C]'...
1324          ,'[ C]','[W]','[ C]','[ C]',NaN,...
1325          'Herrick ID','Test #','Fault','[W]',...
1326          '[W]','[-]','[-]','[ C]','[ C]',...
1327          '[kg/s]','[kg]','[%]',...
1328          '[m3/s]','[m3/s]','[%]','[m3/s]',...
1329          '[m3/s]','[%]','[%]','[kPa]','[%]',...
1330          '[kg/kg]','[%]','[kg/s]','[%]','[%]','[%]','','';};
1331
1332      info_size = length(Comp.NBI_sys_info);
1333      mat_loc = 4;
1334      sys_size = size(data_IP,1);
1335      sys_info = cell(sys_size,info_size);
1336      print_IP_data = num2cell(data_IP);
1337      print_SI_data = num2cell(data);
1338      % change superheat and subcooling to zero
1339      print_code = cell(sys_size,1);
1340      for kk = 1:sys_size
1341          sys_info(kk,:) = Comp.NBI_sys_info;
1342          % fill in the fault label
1343          if strcmp(sys_info{kk,1},'Split_1')
1344              NBI_num = '04';
```

```
1345        elseif strcmp(sys_info{kk,1},'RTU_7')
1346            NBI_num = '02';
1347        elseif strcmp(sys_info{kk,1},'RTU_3')
1348            NBI_num = '01';
1349        elseif strcmp(sys_info{kk,1},'Split_2')
1350            NBI_num = '06';
1351        elseif strcmp(sys_info{kk,1},'RTU_4')
1352            NBI_num = '03';
1353        elseif strcmp(sys_info{kk,1},'RTU_2')
1354            NBI_num = '05';
1355        elseif strcmp(sys_info{kk,1},'Split_3')
1356            NBI_num = '07';
1357        elseif strcmp(sys_info{kk,1},'Split_5')
1358            NBI_num = '09';
1359        else
1360            NBI_num = '00';
1361        end
1362        print_code{kk,1} = ['1',NBI_num,sprintf('%04d',kk)];
1363        if data_IP(kk,31) < 0
1364            print_IP_data{kk,31} = 0;
1365            print_SI_data{kk,31} = 0;
1366        end
1367        if data_IP(kk,32) < 0
1368            print_IP_data{kk,32} = 0;
1369            print_SI_data{kk,32} = 0;
1370        end
```

```
1371          if data_IP(kk,35) < 100
1372              print_IP_data{kk,26} = 'UC';
1373          elseif data_IP(kk,35) > 100
1374              print_IP_data{kk,26} = 'OC';
1375          elseif data_IP(kk,38) < 100
1376              print_IP_data{kk,26} = 'EA';
1377          elseif data_IP(kk,41) < 100
1378              print_IP_data{kk,26} = 'CA';
1379          elseif data_IP(kk,44) > 0
1380              print_IP_data{kk,26} = 'LL';
1381          elseif data_IP(kk,46) > 0
1382              print_IP_data{kk,26} = 'NC';
1383          elseif data_IP(kk,48) > 0
1384              print_IP_data{kk,26} = 'VL';
1385          else
1386              print_IP_data{kk,26} = 'NoF';
1387          end
1388          print_SI_data{kk,26} = print_IP_data{kk,26};
1389      end
1390      mat_save(mat_loc:mat_loc+sys_size-1,:) = [...
1391          sys_info print_code cell(sys_size,1) print_SI_data ...
1392      ];
1393      xlswrite(...
1394          strcat(matPathName,[savefilename,'.xlsb']),...
1395          mat_save,'Data SI' ...
1396      );
```

```
1397
1398      mat_IP_save(mat_loc:mat_loc+sys_size-1,:) = [...
1399          sys_info print_code cell(sys_size,1) print_IP_data ...
1400      ];
1401      xlswrite(strcat(...
1402          matPathName,[savefilename,'.xlsb'] ...
1403      ),mat_IP_save,'Data IP');
1404  end
1405  close(fig);
1406
1407  save(strcat(matPathName,savefilename));
1408
1409  % close parallel computing
1410  try
1411      matlabpool close;
1412  catch err
1413  end
1414
1415  unloadlibrary lib;
1416
1417  % Acknowledge completion
1418  fig = msgbox(...
1419      ['Simulation Completed. Automatic opening ',...
1420      [savefilename,'.xlsb'],' after 5s'] ...
1421  );
1422  display('Simulation Completed');
```

```
1423 try
1424     pause(5);
1425     close(fig);
1426 end
1427 
1428 % open file
1429 try
1430     winopen(strcat(matPathName,'\',[savefilename,'.xlsb']));
1431 end
1432 end
```

M.2 Initial Guess Calculator

```
1  function Guess = initial_Guess(...
2      Tain_evap, Dain_evap, Vdot_evap, Tain_cond, Vdot_cond, SCSH, ...
3      P_sat_comp_in_coeff, P_sat_comp_out_coeff, SH_evap_coeff, ...
4      T_out_coeff, mdot_r_coeff, Cond, ...
5      refri, Tc, Pc)
6      Guess = ones(5,1)*0.5;
7      if Tain_evap > Tain_cond
8          Tain_guess = max(Tain_evap, Tain_cond);
9          Guess_var = [...
10             1 Tain_guess Dain_evap Vdot_evap Tain_guess Vdot_cond ...
11             SCSH.Charge ...
12         ]';
13     else
14         Guess_var = [...
```

```
15              1 Tain_evap Dain_evap Vdot_evap Tain_cond ...
16              Vdot_cond SCSH.Charge ...
17          ]';
18      end
19      P_guess = sum(P_sat_comp_in_coeff.*Guess_var);
20      if P_guess > propertyRH('P','T',Tain_evap,'Q',1,refri) | ...
21              P_guess < propertyRH('P','T',250,'Q',1,refri);
22          Guess(1,1) = (Tain_evap - 273.15)/(Tain_evap-250);
23      else
24          Guess(1,1) = (...
25              Tain_evap - propertyRH('T','P',P_guess,'Q',1,refri) ...
26          )/(Tain_evap-250);
27      end
28      P_cond_guess = sum(P_sat_comp_out_coeff.*Guess_var);
29      T_cond = propertyRH('T','P',P_cond_guess,'Q',0,refri);
30      if P_cond_guess < P_guess
31          P_cond_guess = P_guess*1.2;
32      end
33      if P_cond_guess > Pc*0.9 | ...
34              P_cond_guess < propertyRH('P','T',Tain_cond,'Q',1,refri);
35          Guess(2,1) = 0.5;
36      else
37          Guess(2,1) = (T_cond - Tain_cond)/(Tc - Tain_cond);
38      end
39      if Tain_evap > Tain_cond
40          if Guess(2,1) < 0.2
```

```
41              % low evaporating pressure for low condensing pressure
42              Guess(2,1) = Guess(2,1)*0.5;
43          else
44              Guess(2,1) = Guess(2,1)-0.1;
45          end
46          P_cond_guess = propertyRH( ...
47              'P','T',Guess(2,1)*(Tc-Tain_cond)+Tain_cond,'Q',0,refri ...
48          );
49          P_guess = min(...
50              P_cond_guess, propertyRH(...
51                  'P','T',Tain_evap-Guess(1,1)*(Tain_evap-250),'Q',1,...
52                  refri ...
53              ) ...
54          );
55          Guess(1,1) = (...
56              Tain_evap - propertyRH('T','P',P_guess,'Q',1,refri) ...
57          )/(Tain_evap-250);
58      end
59      SH_guess = sum(SH_evap_coeff.*Guess_var);
60      if SH_guess > 40 | SH_guess < -40
61          SH_guess = 0;
62      end
63      Guess(3,1) = SH_guess/6.0;
64      % adjsutment for valve leakage case
65      if SCSH.VL > 0
66          Guess(3,1) = Guess(3,1)*0.5;
```

```
67      end
68      T_guess = sum(T_out_coeff.*Guess_var);
69      if T_guess > Tc + 50 | T_guess < Tain_cond
70          Guess(4,1) = Guess(2,1)+0.1;
71      else
72          Guess(4,1) = (T_guess-T_cond)/(Tc - Tain_cond);
73      end
74      mdot_r_guess = sum(mdot_r_coeff.*Guess_var);
75      if mdot_r_guess < 1.e-8 | mdot_r_guess > 2*Cond.para.m_r
76          Guess(5,1) = 0.5;
77      else
78          Guess(5,1) = mdot_r_guess/Cond.para.m_r;
79      end
80  end
```

M.3 Solver for System Model

```
1   function [System,Real,dev2,residual_pre,flag,func_output,err] = ...
2       cycle_output_no_adj(...
3           Guess, Tain_cond, wain_cond, Vdot_cond, ...
4           Tain_evap, wain_evap, ...
5           Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
6           EXV, Evap, SuctionLine, refri, Tc, Pc, hf, hv, rhov, ...
7           DP_cond, DP_evap, libname...
8       )
9   % solve for a good temperature difference
10  if SCSH.NC_level > 1.e-8
```

```
11      lower_limit = [0.03 0.03 -Inf 0 1.e-8 1.e-8];
12      upper_limit = [...
13          2 0.95 60/6 (500-Tain_cond)/(Tc - Tain_cond) Inf 1.0 ...
14      ];
15  else
16      lower_limit = [0.03 0.03 -Inf 0 1.e-8];
17      upper_limit = [...
18          2 0.95 60/6 (500-Tain_cond)/(Tc - Tain_cond) Inf ...
19      ];
20  end
21  try
22      Options = optimset(...
23          'Display','off','TolX',SCSH.xtol,'TolFun',SCSH.ftol,...
24          'MaxFunEvals',700,'MaxIter',300 ...
25      );
26      [Real,residual_pre,flag,func_output]  = fsolve(...
27          @(x) residual_maker(...
28              x, Tain_cond, wain_cond, Vdot_cond, ...
29              Tain_evap, wain_evap, ...
30              Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, ...
31              LiquidLine, EXV, Evap, SuctionLine, refri, Tc, Pc, hf, ...
32              hv, rhov, DP_cond, DP_evap, libname...
33          ),Guess,Options);
34      dev2 = norm(residual_pre);
35      iter_old = func_output.iterations;
36      func_count_old = func_output.funcCount;
```

```
37      if sqrt(mean(residual_pre.^2)) > SCSH.ftol
38          % not converging for case with SH
39          Options = optimset(...
40              'Display','off','TolX',SCSH.xtol,'TolFun',SCSH.ftol,...
41              'Algorithm','trust-region-reflective','MaxFunEvals',750,...
42              'MaxIter',300 ...
43          );
44          [Real_II,dev2,residual_pre_II,flag_II,func_output_II] = ...
45              lsqnonlin(@(x) residual_maker(...
46                  x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
47                  wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
48                  HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
49                  refri, Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, ...
50                  libname ...
51              ),Guess,lower_limit,upper_limit,Options);
52          func_output.iterations = func_output_II.iterations + ...
53              iter_old;
54          func_output.funcCount = func_output_II.funcCount + ...
55              func_count_old;
56          if sqrt(mean(residual_pre.^2)) > sqrt(mean(residual_pre_II.^2))
57              Real = Real_II;
58              residual_pre = residual_pre_II;
59              flag = flag_II;
60          end
61      end
62      err = func_output.message;
```

```
63 catch err
64     Options = optimset(...
65         'Display','off','TolX',SCSH.xtol,'TolFun',SCSH.ftol,...
66         'Algorithm','trust-region-reflective','MaxFunEvals',750,...
67         'MaxIter',300...
68     );
69     [Real,dev2,residual_pre,flag,func_output] = lsqnonlin(...
70         @(x) residual_maker(x, Tain_cond, wain_cond, Vdot_cond, ...
71         Tain_evap, wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
72         HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, refri, ...
73         Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, libname ...
74     ),Guess,lower_limit,upper_limit,Options);
75 end
76 [residual_pre System] = cycle_calculation(...
77     Real, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
78     Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
79     EXV, Evap, SuctionLine, refri, Tc, Pc, hf, hv, rhov, DP_cond, ...
80     DP_evap, libname ...
81 );
82 if length(residual_pre) < 6
83     residual_temp = residual_pre;
84     clear residual_pre;
85     residual_pre = [residual_temp;0];
86 end
87 end
```

M.4 Residual Function for System Model

```
function [residual System] = cycle_calculation(...
    Guess, Tain_cond, ...
    wain_cond, Vdot_cond, Tain_evap, wain_evap, Vdot_evap, Tamb, ...
    Pain, SCSH, Comp, HotGas, Cond, LiquidLine, EXV, Evap, ...
    SuctionLine, refri, Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, ...
    libname...
)

% cycle_calculation is a function which generates an array of
% residual and a structure defined in the function
% System_definition(). The array of
% residual shows whether a solution is reached and the structure
% stores the data which is related to the performance of the cycle

% Input variables
% Guess        : an array of normalized guess variables for suction
%                line inlet pressure, compressor outlet pressure,
%                suction line inlet enthalpy, compressor outlet
%                temperature and refrigerant mass flow rate. If
%                the presence of non-condensable is involved,
%                an additional guess variable for the gas
%                volume inside the condenser is introduced.
% Tain_cond    : air temperature at the inlet of condenser in K
% wain_cond    : absolute humidity of air at the inlet of condenser
%                in K
% Vdot_cond    : air volumetric flow rate of condenser in m3/s
```

```
27 % Tain_evap    : air temperature at the inlet of evaporator in K
28 % wain_evap    : absolute humidity of air at the inlet of evaporator
29 %                in K
30 % Vdot_evap    : air volumetric flow rate of evaporator in m3/s
31 % Tamb         : ambient temperature in K
32 % Pain         : atmospheric pressure in kPa
33 % SCSH         : a structure storing fault information such as
34 %                charge, liquid line restriction, etc.
35 % Comp         : a structure storing the parameters of the
36 %                compressor model
37 % HotGas       : a structure storing the parameters of the
38 %                hot gas line model
39 % Cond         : a structure storing the parameters of the
40 %                condenser model
41 % LiquidLine   : a structure storing the parameters of the
42 %                liquid line model
43 % EXV          : a structure storing the parameters of the
44 %                expansion valve model
45 % Evap         : a structure storing the parameters of the
46 %                evaporator model
47 % SuctionLine  : a structure storing the parameters of the
48 %                suction line model
49 % refri        : name of refrigerant
50 % Tc           : critical temperature of refrigerant in K
51 % Pc           : critical pressure of refrigerant in kPa
52 % hf           : void
```

```
53 % hg           : void
54 % DP_evap      : void
55 % DP_cond      : void
56 % libname      : name of library for the DLL
57
58 % Output variables
59 % residual     : an array of normalized residuals to examine if
60 %                Guess is referring to a solution
61 % System       : a structure storing the system performance and
62 %                features such as superheat and subcooling.
63 %                Defined in the function System_definition()
64
65 % convert all guess values to dimensional values
66 SuctionLine.Pin = propertyRH(...
67     'P','T',Tain_evap-Guess(1)*(Tain_evap-250),'Q',0,refri...
68 );
69 SuctionLine.hin = propertyRH('H','P',SuctionLine.Pin,'Q',1,refri) +...
70     Guess(3)*6.0*propertyRH('C','P',SuctionLine.Pin,'Q',1,refri);
71 T_cond = Guess(2)*(Tc-Tain_cond)+Tain_cond;
72 Comp.Pout = propertyRH('P','T',T_cond,'Q',0,refri);
73 Tout_guess = Guess(4)*(Tc-Tain_cond)+T_cond;
74 mdot_r = Guess(5)*Cond.para.m_r;
75
76 % Solve SuctionLine
77 SuctionLine.mdot_r = mdot_r;
78 if mdot_r<0
```

```
79      residual = ones(length(Guess),1)*NaN;
80      System = System_definition();
81      return;
82  end
83  [SuctionLine.Pout SuctionLine.hout SuctionLine.Charge ...
84      SuctionLine.Q_status SuctionLine.ΔP_status] = ...
85      pipeline4(...
86          SuctionLine.mdot_r, SuctionLine.Pin, SuctionLine.hin, Tamb, ...
87          SuctionLine, refri, 1, libname...
88      );
89  if SuctionLine.Pout<0
90      residual = ones(length(Guess),1)*NaN;
91      System = System_definition();
92      return;
93  end
94
95  % other arrangements
96  Cond.Tain = Tain_cond;
97  Cond.wain = wain_cond;
98  Cond.Vdot = Vdot_cond;
99  Evap.Tain = Tain_evap;
100 Evap.wain = wain_evap;
101 Evap.Vdot = Vdot_evap;
102
103 % Solve Compressor
104 Comp.Pin = SuctionLine.Pout;
```

```
105 Comp.hin = SuctionLine.hout;
106 try
107     bp = 0;
108     [Comp.mdot_r Comp.W Comp.hout residual_Comp  Comp_m_status ...
109         Comp_W_status Comp_HL_status] = ...
110         Compressor(...
111             Comp.Pin, Comp.hin, Comp.Pout, Tamb, Tout_guess, ...
112             Comp.para, refri...
113         );
114     if SCSH.VL > 0
115         hout_temp = propertyRH(...
116             'H','T',Tout_guess,'P',Comp.Pout,refri ...
117         );
118         Comp.mdot_r = Comp.mdot_r*(1-SCSH.VL);
119         bp = fzero(@(bp) Compressor(Comp.Pin, Comp.hin*(1-bp) +...
120                 hout_temp*bp, ...
121                 Comp.Pout, Tamb, Tout_guess, Comp.para, refri...
122             )*(1-bp)-Comp.mdot_r, SCSH.VL);
123         [dum1 Comp.W Comp.hout residual_Comp Comp_m_status ...
124             Comp_W_status Comp_HL_status] = ...
125             Compressor(Comp.Pin, Comp.hin*(1-bp) +...
126                 hout_temp*bp, ...
127                 Comp.Pout, Tamb, Tout_guess, Comp.para, refri...
128             );
129     end
130
```

```
131 catch err
132     residual = ones(length(Guess),1)*NaN;
133     System = System_definition();
134     return;
135 end
136 if Comp.mdot_r<0
137     residual = ones(length(Guess),1)*NaN;
138     System = System_definition();
139     return;
140 end
141
142 % solve NC case
143 if SCSH.NC_level > 1.e-8
144     NC_mass = SCSH.NC_Total_mass*SCSH.NC_level;
145     NC_Pv = NC_mass*0.2968*Tout_guess/...
146         (pi*Cond.para.Din^2/4*Cond.para.Lt*Guess(6) + ...
147             pi*HotGas.dia^2/4*HotGas.line);
148     if SCSH.VL > 0
149         [Comp.mdot_r Comp.W Comp.hout residual_Comp] = ...
150             Compressor(Comp.Pin, Comp.hin*(1-SCSH.VL) +...
151                 propertyRH(...
152                     'H','T',Tout_guess,'P',...
153                     Comp.Pout-NC_Pv,refri...
154                 )*SCSH.VL, Comp.Pout, Tamb, Tout_guess, ...
155                 Comp.para, ...
156                 refri...
```

```
157             );
158         Comp.mdot_r = Comp.mdot_r*(1-SCSH.VL);
159     end
160     residual_Comp = propertyRH(...
161         'T','P',Comp.Pout-NC_Pv,'H',Comp.hout,refri...
162     ) - Tout_guess; % update the residual for a change at definition
163 else
164     NC_mass = 0;
165     NC_Pv = 0;
166 end
167
168 % solve Hotgas line
169 HotGas.Pin = Comp.Pout;
170 % use the average in case of failed solution
171 HotGas.mdot_r = (Comp.mdot_r+mdot_r)/2;
172 HotGas.hin = Comp.hout;
173 HotGas.Tin = propertyRH('T','P',HotGas.Pin-NC_Pv,'H',HotGas.hin,refri);
174 [HotGas.Pout HotGas.hout HotGas.Charge HotGas.Q_status ...
175     HotGas.ΔP_status] = pipeline4(...
176     HotGas.mdot_r, HotGas.Pin-NC_Pv, HotGas.hin, Tamb, HotGas, ...
177     refri, 1, libname...
178 );
179 HotGas.Pout = HotGas.Pout+NC_Pv;
180 if HotGas.Pout<0
181     residual = ones(length(Guess),1)*NaN;
182     System = System_definition();
```

```
183     return;
184 end
185 
186 % solve Condenser
187 Cond.Pin = HotGas.Pout;
188 Cond.hin = HotGas.hout;
189 Cond.mdot_r = HotGas.mdot_r;
190 Cond.Pain = Pain;
191 [Cond.Pout Cond.hout Cond.Taout Cond.Q Cond.Charge Cond.FanPower ...
192     Cond.OtherOutput Cond.mdot_a Cond.LumpedInput ...
193     Cond.rho_tp Cond.rho_sh Cond.rho_sc] = Condenser(...
194         Cond.Pin-NC_Pv, Cond.hin, Cond.mdot_r, ...
195         Cond.Tain, Cond.wain, Cond.Pain, Cond.Vdot, Cond.para, ...
196         refri, libname...
197     );
198 Cond.Pout = Cond.Pout + NC_Pv;
199 if Cond.Pout<0
200     residual = ones(length(Guess),1)*NaN;
201     System = System_definition();
202     return;
203 end
204 
205 % solve liquid line
206 LiquidLine.mdot_r = Cond.mdot_r;
207 LiquidLine.hin = Cond.hout;
208 LiquidLine.Pin = Cond.Pout;
```

```
209 try
210     LiquidLine.Tin = propertyRH(...
211         'T','P',LiquidLine.Pin,'H',LiquidLine.hin,refri...
212     );
213 catch err
214     residual = length(Guess)*NaN;
215     System = System_definition();
216     return;
217 end
218 [LiquidLine.Pout LiquidLine.hout LiquidLine.Charge ...
219     LiquidLine.Q_status LiquidLine.ΔP_status] = pipeline4(...
220     LiquidLine.mdot_r, LiquidLine.Pin, LiquidLine.hin, ...
221     Tamb, LiquidLine, refri, 0, libname...
222 );
223 if SCSH.LL_restriction > 0
224     LiquidLine.Pout = LiquidLine.Pin - SCSH.LL_restriction;
225     rho = RefrigerantDensity(LiquidLine.Pin, LiquidLine.hin, refri);
226     try
227         rho_out = RefrigerantDensity(...
228             LiquidLine.Pout, LiquidLine.hout, refri...
229         );
230     catch err
231         rho_out = rho;
232         clear err;
233     end
234     LiquidLine.Charge = .5*(rho+rho_out) *...
```

```
235         LiquidLine.line*(pi*LiquidLine.dia^2/4);
236 end
237 if LiquidLine.Pout<0
238     residual = length(Guess)*NaN;
239     System = System_definition();
240     return;
241 end
242 LiquidLine.rho_LL = .5*(...
243     RefrigerantDensity(LiquidLine.Pout, LiquidLine.hout, refri) +...
244     RefrigerantDensity(LiquidLine.Pin, LiquidLine.hin, refri)...
245 );
246
247 % solve evaporator
248 Evap.hin = LiquidLine.hout;
249 Evap.hout_est = SuctionLine.hin;
250 Evap.Pain = Pain;
251 Evap.Pout = SuctionLine.Pin;
252 Evap.mdot_r = LiquidLine.mdot_r;
253 [Evap.Pin Evap.hout Evap.Taout Evap.waout Evap.Q Evap.Charge ...
254     Evap.FanPower Evap.OtherOutput Evap.mdot_a Evap.LumpedInput ...
255     Evap.SHR Evap.rho_tp Evap.rho_sh] = Evaporator(...
256         Evap.Pout, Evap.hin, ...
257         Evap.hout_est, Evap.mdot_r, Evap.Tain, Evap.wain, ...
258         Evap.Pain, Evap.Vdot, Evap.para, refri, libname...
259     );
260
```

```
% Solve EXV
EXV.Pin = LiquidLine.Pout;
EXV.Pout = Evap.Pin;
EXV.hin = LiquidLine.hout;
EXV.hout = EXV.hin;
if strcmp(EXV.Type,'TXV')
    EXV.hout = EXV.hin;
    if strcmp(EXV.Equalizer,'Ext_Evap')
        [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
            EXV.Pin, EXV.Pout, EXV.hin, SuctionLine.Pin, ...
            propertyRH(...
                'T','P',SuctionLine.Pin,'H',SuctionLine.hin,refri...
            ), EXV.para, Pc, Tc, refri...
        );
    elseif strcmp(EXV.Equalizer,'Ext_Comp')
        [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
            EXV.Pin, EXV.Pout, EXV.hin, SuctionLine.Pout, ...
            propertyRH(...
                'T','P',SuctionLine.Pout,'H',SuctionLine.hout,refri...
            ), EXV.para, Pc, Tc, refri...
        );
    else
        [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
            EXV.Pin, EXV.Pout, EXV.hin, EXV.Pout, ...
            propertyRH(...
                'T','P',SuctionLine.Pin,'H',SuctionLine.hin,refri...
```

```
287             ), EXV.para, Pc, Tc, refri...
288         );
289     end
290 else
291     EXV.OpeningStep = 1;
292     EXV.hout = EXV.hin;
293     EXV.mdot_r = Fixed_Orifice(...
294         EXV.Pin, EXV.Pout, EXV.hin, EXV.para, Pc, Tc, refri...
295     );
296 end
297 if isfield(EXV,'mdot_adj')
298     EXV.mdot_r = EXV.mdot_r*EXV.mdot_adj;
299 end
300
301 % Superheat and Subcooling
302 SC_final = propertyRH('T','P',Cond.Pout,'Q',0,refri) - ...
303     propertyRH('T','P',Cond.Pout,'H',Cond.hout,refri);
304 if SC_final ≤ 0
305     SC_final = -(...
306             Cond.hout - propertyRH('H','P',Cond.Pout,'Q',0,refri)...
307         )/propertyRH('C','P',Cond.Pout,'Q',0,refri);
308 end
309 Comp.SH = propertyRH('T','P',Comp.Pin,'H',Comp.hin,refri) -...
310     propertyRH('T','P',Comp.Pin,'Q',1,refri);
311 Evap.SH = propertyRH('T','P',Evap.Pout,'H',Evap.hout,refri) -...
312     propertyRH('T','P',Evap.Pout,'Q',1,refri);
```

```
313 if Evap.SH ≤ 0
314     Evap.SH = (...
315             Evap.hout-propertyRH('H','P',Evap.Pout,'Q',1,refri)...
316         )/propertyRH('C','P',Evap.Pout,'Q',1,refri);
317 end
318 LL_SC = propertyRH('T','P',LiquidLine.Pout,'Q',0,refri) - ...
319     propertyRH('T','P',LiquidLine.Pout,'H',LiquidLine.hout,refri);
320 if LL_SC<0
321     LL_SC = -(...
322         LiquidLine.hout-propertyRH(...
323             'H','P',LiquidLine.Pout,'Q',0,refri ...
324         )...
325     )/propertyRH('C','P',LiquidLine.Pout,'Q',0,refri);
326 end
327
328 % storing in the system structure as output
329 System.mdot_r = Comp.mdot_r;
330
331 System.Comp.Pout = Comp.Pout;
332 System.Comp.hout = Comp.hout;
333 System.Comp.W = Comp.W;
334 System.Comp.Pin = Comp.Pin;
335 System.Comp.hin = Comp.hin;
336 System.Comp.m_status = Comp_m_status;
337 System.Comp.W_status = Comp_W_status;
338 System.Comp.HL_status = Comp_HL_status;
```

```
339
340 System.HotGas.Charge = HotGas.Charge;
341 System.HotGas.Pin = HotGas.Pin;
342 System.HotGas.Pout = HotGas.Pout;
343 System.HotGas.Q_status = HotGas.Q_status;
344 System.HotGas.ΔP_status = HotGas.ΔP_status;
345
346 System.Cond.Tain = Cond.Tain;
347 System.Cond.wain = Cond.wain;
348 System.Cond.Pin = Cond.Pin;
349 System.Cond.hin = Cond.hin;
350 System.Cond.Pout = Cond.Pout;
351 System.Cond.hout = Cond.hout;
352 System.Cond.Charge = Cond.Charge;
353 System.Cond.OtherOutput = Cond.OtherOutput;
354 System.Cond.LumpedInput = Cond.LumpedInput;
355 System.Cond.FanPower = Cond.FanPower;
356 System.Cond.Taout = Cond.Taout;
357 System.Cond.mdot_a = Cond.mdot_a;
358 System.Cond.Q = Cond.Q;
359 System.Cond.status_Q = 0;
360 System.Cond.status_ΔP = 0;
361
362 System.LiquidLine.Charge = LiquidLine.Charge;
363 System.LiquidLine.hin = LiquidLine.hin;
364 System.LiquidLine.Pin = LiquidLine.Pin;
```

```
365 System.LiquidLine.Pout = LiquidLine.Pout;
366 System.LiquidLine.hout = LiquidLine.hout;
367 System.LiquidLine.Q_status = LiquidLine.Q_status;
368 System.LiquidLine.ΔP_status = LiquidLine.ΔP_status;
369
370 System.EXV.Pin = EXV.Pin;
371 System.EXV.hin = EXV.hin;
372 System.EXV.mdot_r = EXV.mdot_r;
373 System.EXV.Pout = Evap.Pin;
374 System.EXV.hout = Evap.hin;
375 System.EXV.OpeningStep = EXV.OpeningStep;
376 System.EXV.status = 0;
377 System.EXV.status_x = 0;
378
379 System.Evap.Tain = Evap.Tain;
380 System.Evap.wain = Evap.wain;
381 System.Evap.Pin = Evap.Pin;
382 System.Evap.hin = Evap.hin;
383 System.Evap.Pout = Evap.Pout;
384 System.Evap.hout = Evap.hout;
385 System.Evap.Charge = Evap.Charge;
386 System.Evap.OtherOutput = Evap.OtherOutput;
387 System.Evap.LumpedInput = Evap.LumpedInput;
388 System.Evap.FanPower = Evap.FanPower;
389 System.Evap.Taout = Evap.Taout;
390 System.Evap.waout = Evap.waout;
```

```
391 System.Evap.mdot_a = Evap.mdot_a;
392 System.Evap.Q = Evap.Q;
393 System.Evap.SHR = Evap.SHR;
394 System.Evap.status_Q = 0;
395 System.Evap.status_SHR = 0;
396 System.Evap.status_ΔP = 0;
397
398 System.SuctionLine.Charge = SuctionLine.Charge;
399 System.SuctionLine.hin = SuctionLine.hin;
400 System.SuctionLine.Pin = SuctionLine.Pin;
401 System.SuctionLine.Pout = SuctionLine.Pout;
402 System.SuctionLine.hout = SuctionLine.hout;
403 System.SuctionLine.Q_status = SuctionLine.Q_status;
404 System.SuctionLine.ΔP_status = SuctionLine.ΔP_status;
405
406 System.Charge = System.SuctionLine.Charge+System.Evap.Charge + ...
407     System.LiquidLine.Charge + System.Cond.Charge + ...
408     System.HotGas.Charge;
409 vec_x = [...
410     (System.Cond.OtherOutput(7)-Comp.para.chargeb_UA_oil(5))...
411     (System.Cond.OtherOutput(5)-Comp.para.chargeb_UA_oil(6))...
412     (System.Evap.OtherOutput(4)-Comp.para.chargeb_UA_oil(7))...
413     ];
414 System.Charge_tuned = System.Charge + ...
415     Comp.para.chargeb_UA_oil(1) + ...
416     Comp.para.chargeb_UA_oil(2)*vec_x(1);
```

```
417 System.Charge_tuned = System.Charge_tuned + ...
418     (Comp.para.chargeb_UA_oil(3))*vec_x(2);
419 System.Charge_tuned = System.Charge_tuned + ...
420     (Comp.para.chargeb_UA_oil(4))*vec_x(3);
421 System.Charge_calculate.status = ...
422     vec_x*Comp.para.b_UA_oil_para.Cov_X*vec_x' -...
423     Comp.para.b_UA_oil_para.X_limit;
424
425 System.mdot_r = Comp.mdot_r;
426 System.CondenserSC = SC_final;
427 System.LiquidLineSC = LL_SC;
428 System.EvaporatorSH = Evap.SH;
429 System.CompressorSH = Comp.SH;
430
431 System.Heatloss = Comp.W + Evap.Q - Cond.Q;
432
433 System.Charge_calculate.Cond.wsh = System.Cond.OtherOutput(5);
434 System.Charge_calculate.Cond.wtp = System.Cond.OtherOutput(6);
435 System.Charge_calculate.Cond.wsc = System.Cond.OtherOutput(7);
436 System.Charge_calculate.Cond.rho_sh = Cond.rho_sh;
437 System.Charge_calculate.Cond.rho_tp = Cond.rho_tp;
438 System.Charge_calculate.Cond.rho_sc = Cond.rho_sc;
439 System.Charge_calculate.Evap.wsh = System.Evap.OtherOutput(4);
440 System.Charge_calculate.Evap.wtp = System.Evap.OtherOutput(5);
441 System.Charge_calculate.Evap.rho_sh = Evap.rho_sh;
442 System.Charge_calculate.Evap.rho_tp = Evap.rho_tp;
```

```
443 System.Charge_calculate.LiquidLine.rho_LL = LiquidLine.rho_LL;
444
445 System.ValveLeakage = SCSH.VL;
446 System.LL_restriction = SCSH.LLpercent;
447 System.Non_condensable = SCSH.NC_level;
448 System.ValveLeakFlow = Comp.mdot_r*(1/(1-SCSH.VL)-1);
449 System.LL_extra_ΔP = SCSH.LL_restriction*(1-1/(1+SCSH.LLpercent));
450 System.NC_amount = NC_mass; % new entry
451 System.NC_Pressure = NC_Pv;
452 System.Accumulator_stat = SCSH.Accumulator;
453 System.VL_BP = bp;
454
455 System.SOLVED = 0;
456
457 % calculate residuals
458 residual(1,1) = (Evap.hout-SuctionLine.hin) /...
459     propertyRH('C','T',Evap.Tain,'Q',1,refri)/10.0;
460 residual(2,1) = (EXV.mdot_r-Comp.mdot_r)/Cond.para.m_r;
461 if SCSH.Accumulator == 0
462     residual(3,1) = (System.Charge_tuned-SCSH.Charge)/SCSH.Charge;
463 elseif SCSH.Accumulator == 1
464     residual(3,1) = (...
465             Comp.hin-propertyRH('H','P',Comp.Pin,'Q',1,refri)...
466         ) /...
467         propertyRH('C','T',Evap.Tain,'Q',1,refri)/10.0;
468 else
```

```
469     residual(3,1) = (Comp.hin- ...
470         propertyRH('H','T',propertyRH(...
471             'T','P',Comp.Pin,'Q',1,refri...
472         )+0.1,...
473         'P',Comp.Pin,refri))/propertyRH(...
474             'C','T',Evap.Tain,'Q',1,refri...
475         )/10.0;
476 end
477 residual(4,1) = (residual_Comp)/10.0;
478 residual(5,1) = (mdot_r - Comp.mdot_r)/Cond.para.m_r;
479 if NC_Pv > 1.e-8
480     rho_v = propertyRH('D','P',Cond.Pin - NC_Pv,'Q',1.0,refri);
481     rho_l = propertyRH('D','P',Cond.Pin - NC_Pv,'Q',0.0,refri);
482     SS = (rho_l/rho_v)^0.3333;
483     CC = SS*rho_v/rho_l;
484     x1 = 1.0;
485     x2 = max(Cond.OtherOutput(10), 0);
486     alpha_average = -(...
487         CC*(log(((x1-1.0)*CC-x1)/((x2-1.0)*CC-x2))+x1-x2)-x1+x2...
488     )/(CC*CC-2*CC+1)/(x1-x2);
489     residual(6,1) = ...
490         Guess(6)-Cond.OtherOutput(5) -...
491         Cond.OtherOutput(6)*alpha_average;
492 end
493
494 end
```

M.5 Compressor Model

```
function [...
    mr W hout residual Comp_m_status Comp_W_status ...
    Comp_HL_status...
] = Compressor(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)
rho_in = propertyRH('D','P',Pevap,'H',hin,refri);
Tcond = propertyRH('T', 'P', Pcond, 'Q', 1, refri);
rho_out = propertyRH('D','T',Tout_guess,'P',Pcond,refri);
poly_n = log(Pevap/Pcond)/log(rho_in/rho_out);
mr = rho_in*(...
    para.m_para(1)+para.m_para(2)*(Pcond/Pevap)^(1/poly_n)+...
        para.m_para(3)*(Pcond-Pevap)...
    );
eta_comb = para.W_para(1)+...
    para.W_para(2)*exp(para.W_para(3)*Pevap/para.P_evap_rated)+...
    para.W_para(4)*exp(para.W_para(5)*Pcond/para.P_evap_rated);
W = mr*poly_n/(poly_n-1)*Pevap/rho_in*1000*(...
    (Pcond/Pevap)^(1-1/poly_n)-1 ...
)/eta_comb;
Tin = propertyRH('T','P',Pevap,'H',hin,refri);
hout = Compressor_Tout(Tcond, Tin, Tamb, hin, mr, W, para);
residual = propertyRH('T', 'P', Pcond, 'H', hout, refri) - Tout_guess;
vec_x = [rho_in rho_in*(Pcond/Pevap)^(1/poly_n) rho_in*(Pcond-Pevap)];
Comp_m_status = vec_x*para.COV_X*vec_x'-para.mdot_limit;
vec_x = [...
```

```
    -W/eta_comb ...
    -W/eta_comb*exp(para.W_para(3)*Pevap/para.P_evap_rated) ...
    -W/eta_comb*para.W_para(2)*exp(...
        para.W_para(3)*Pevap/para.P_evap_rated...
    )*Pevap/para.P_evap_rated ...
    -W/eta_comb*exp(para.W_para(5)*Pcond/para.P_evap_rated) ...
    -W/eta_comb*para.W_para(4)*exp(...
        para.W_para(5)*Pcond/para.P_evap_rated...
    )*Pcond/para.P_evap_rated ...
    ];
Comp_W_status = vec_x*para.COV_X_Power*vec_x'-para.Power_limit;
vec_x = [(Tin-Tamb) (Tcond-Tamb)];
Comp_HL_status = vec_x*para.COV_X_HL*vec_x'-para.HL_limit;
end
```

M.6 Residual Function for Compressor Heat Loss Model

```
function hout = Compressor_Tout(Tcond, Tin, Tamb, hin, mr, W, para)
HeatLoss = (...
    para.f_HL_factor(1).*(Tin-Tamb) +...
    para.f_HL_factor(2).*(Tcond-Tamb)...
);
hout = hin + (W-HeatLoss)/mr;
end
```

M.7 Compressor Flow Fault Model

```
function residual = Compressor_VL(...
    hout_est, Pevap, hin, Pcond, Tamb, Tout_guess, para, ...
    refri, ValveLeakage...
)
% This function adjusts the compressor outlet enthalpy by estimating
%
% residual:
%   difference between the adjusted outlet enthalpy and
%   guess outlet enthalpy
%
% based on
%
% hout_est:
%   estimated outlet enthalpy in J/kg
% Pevap:
%   suction pressure in kPa
% hin:
%   suction enthalpy in J/kg
% Pcond:
%   outlet pressure in kPa
% Tamb:
%   ambient temperature in K
% Tout_guess:
%   guess compressor outlet temperature in K
% para:
%   compressor model parameters
```

```
% refri:
%   name of refrigerant
% ValveLeakage:
%   compressor valve leakage level
[mr W hout residual_II] = Compressor(...
    Pevap, hout_est*ValveLeakage+hin*(1-ValveLeakage), Pcond, ...
    Tamb, Tout_guess, para, refri...
);
residual = hout - hout_est;
end
```

M.8 Condenser Model

```
function [...
    Pout hout Taout Qcond Charge FanPower CondOutput ma CondInput ...
    rho_tp rho_sh rho_sc ...
] = Condenser(...
    Pin, hin, mdot_r, Tain, wain, Pain, Vdot, para, refri, libname...
)

% Variable list
% Pout: Refrigenrat outlet pressure (kPa)
% hout: Refrigenrat outlet enthalpy (J/kg)
% Taout: Air outlet tmperautre (K)
% Qcond: Condenser heat transfer rate (W)
% Charge: Amount of charge in the condenser (kg)
% FanPower: Fan Power consumption of condenser (W)
```

```
15 % CondOutput: other outputs (misc.)
16
17 % Pin: Refrigenrat pressure at inlet (kPa)
18 % hin: Refrigenrat enthalpy at inlet (W)
19 % mdot_r: Refrigeraat mass flow rate (kg/s)
20 % Tain: Air temperature at inlet (K)
21 % wain: Inlet air absolute humidity (kg/kg)
22 % Pain: Atmospheric pressure (kPa)
23 % Vdot: Air volumetric flow at inlet (m^3/s)
24 % para: parameters (misc.)
25 % refri: Name of refrigerant
26 % lib: Name of dll library to be used
27
28 % UAa
29 airinlet = zeros(1,5);
30 airinlet = calllib(...
31     libname, 'HumAir_DLL', Tain, Pain, 2, wain, airinlet...
32 ); %arbitrary
33 ma = (1)/airinlet(5)*Vdot;
34 cpa = 1006 + 1860*airinlet(2);
35 CondInput(1) = para.U_a*((ma/para.m_a)^para.n)*para.A_r;
36 % no cpa as denominator as the DLL contains the cpa term
37
38 % UAsh
39 Tin = propertyRH('T','P',Pin,'H',hin,refri);
40 cp_rsh = propertyRH('C','P',Pin,'Q',1,refri);
```

```
41 mu_rsh = propertyRH('V', 'P', Pin,'Q',1, refri);
42 k_rsh = propertyRH('L', 'P', Pin,'Q',1, refri);
43 Ksh = (cp_rsh*mu_rsh/k_rsh)^0.4*(mdot_r/mu_rsh)^0.8;
44 CondInput(2) = para.U_r_sh*Ksh/para.K_sh*para.A_r;
45
46 % UAtp
47 CondInput(3) = para.U_r_tp*(mdot_r/para.m_r)*para.A_r;
48
49 % UAsc
50 cp_rsc = propertyRH('C','P',Pin,'Q',0,refri);
51 mu_rsc = propertyRH('V','P',Pin,'Q',0,refri);
52 k_rsc = propertyRH('L','P',Pin,'Q',0,refri);
53 Ksc = (cp_rsc*mu_rsc/k_rsc)^0.4*(mdot_r/mu_rsc)^0.8;
54 CondInput(4) = para.U_r_sc*Ksc/para.K_sc*para.A_r;
55
56 CondInput(5) = Tain;
57 CondInput(6) = Vdot;
58 CondInput(7) = Pain;
59 CondInput(8) = mdot_r;
60 CondInput(9) = Tin;
61 CondInput(10) = Pin;
62 CondInput(11) = para.Din;
63 CondInput(12) = para.Lt;
64 CondInput(13) = airinlet(4);
65
66 CondOutput = zeros(1,11);
```

```
67 if ¬isempty(strfind(refri,'.fld'))
68     index_end = strfind(refri,'.fld');
69     refname = refri(1:index_end-1);
70 else
71     refname = refri;
72 end
73 CondInput = real(CondInput);
74 [CondInput CondOutput] = calllib(...
75     libname, 'Matlab_Condenser_Forward_Charge_ii_NC', CondInput, ...
76     CondOutput, refname...
77 );
78
79 Qcond = CondOutput(1);
80 hout = hin - Qcond/mdot_r;
81 Taout = Tain + Qcond/ma/cpa;
82 Charge = CondOutput(11);
83
84 FanPower = para.C_fan(1)*Vdot - para.C_fan(2)*Vdot^2;
85 rho_v = propertyRH('D','P',Pin,'Q',1,refri);
86 rho_l = propertyRH('D','P',Pin,'Q',0,refri);
87 rho = propertyRH('D','P',Pin,'H',hin,refri);
88 rho_sh = (rho+rho_v)/2;
89
90 if CondOutput(7) > 0.001
91     xout = 0;
92 else
```

```
93      h_l = propertyRH('H','P',Pin,'Q',0,refri);
94      h_v = propertyRH('H','P',Pin,'Q',1,refri);
95      xout = (hout - h_l)/(h_v - h_l);
96  end
97  xin = 1;
98
99  S = (rho_l/rho_v)^(1/3)*(rho_v/rho_l);
100 gamma = -(...
101     S*(log(((xout-1)*S-xout)/((xin-1)*S-xin))+xout-xin)-xout+xin...
102 )/(S^2-2*S+1)/(xout-xin);
103 rho_tp = (gamma*rho_v+(1-gamma)*rho_l);
104
105 Pout = Pin - para.ΔP.C(1).*CondOutput(5).*(mdot_r.^2/rho_v)*(...
106     para.ΔP.rho_rated/para.ΔP.mdot_rated^2 ...
107 );
108 Pout = Pout - para.ΔP.C(2).*CondOutput(6).*(...
109     mdot_r./para.ΔP.mdot_rated...
110 ).^2.*(para.ΔP.rho_rated/rho_tp).*(...
111     (xin./rho_v.*mu_rsh+(1-xin)./rho_l.*mu_rsc)./(...
112         xin./rho_v+(1-xin)./rho_l...
113     )./para.ΔP.mu_v_rated...
114 ).^para.ΔP.C(3);
115 Pout_temp = Pout - para.ΔP.C(4).*CondOutput(7).*(...
116     mdot_r.^2./rho_l...
117 )*(para.ΔP.rho_out_rated/para.ΔP.mdot_rated^2);
118 Pout = Pout_temp - para.ΔP.C(5).*(...
```

```
119     mdot_r./para.ΔP.mdot_rated...
120 ).^2.*(...
121     1./RefrigerantDensity(Pin, hout, refri) - 1./rho...
122 )*para.ΔP.rho_rated;
123 rho_sc = rho_l;
124 end
```

M.9 Leverage Calculation of Condenser Model

```
1  function [status_Q status_ΔP] = Cond_status(...
2      System, Cond, Vdot_cond, Pain, Pc, Tc, refri, libname...
3      )
4  % compute if the condenser model is operaitng out of range
5  try
6      x_Cov = zeros(1,5);
7      x_Cov_ΔP = zeros(1,5);
8
9      Cond.para_sp = Cond.para;
10     Cond.para_sp.U_a = Cond.para_sp.U_a*(1+1.e-5);
11     [dum1 dum2 dum3 plus] = Condenser(...
12         System.Cond.Pin-System.NC_Pressure, ...
13         System.Cond.hin, System.mdot_r, System.Cond.Tain, ...
14         System.Cond.wain, ...
15         Pain, Vdot_cond, Cond.para_sp, refri, libname);
16     x_Cov(1,1) = (plus - System.Cond.Q)/1.e-5/Cond.para.U_a;
17
18     Cond.para_sp = Cond.para;
```

```
19      Cond.para_sp.n = Cond.para_sp.n*(1+1.e-5);
20      [dum1 dum2 dum3 plus] = Condenser(...
21          System.Cond.Pin-System.NC_Pressure, ...
22          System.Cond.hin, System.mdot_r, System.Cond.Tain, ...
23          System.Cond.wain, ...
24          Pain, Vdot_cond, Cond.para_sp, refri, libname);
25      x_Cov(1,2) = (plus - System.Cond.Q)/1.e-5/Cond.para.n;
26
27      Cond.para_sp = Cond.para;
28      Cond.para_sp.U_r_sh = Cond.para_sp.U_r_sh*(1+1.e-5);
29      [dum1 dum2 dum3 plus] = Condenser(...
30          System.Cond.Pin-System.NC_Pressure, ...
31          System.Cond.hin, System.mdot_r, ...
32          System.Cond.Tain, System.Cond.wain, ...
33          Pain, Vdot_cond, Cond.para_sp, refri, libname);
34      x_Cov(1,3) = (plus - System.Cond.Q)/1.e-5/Cond.para.U_r_sh;
35
36      Cond.para_sp = Cond.para;
37      Cond.para_sp.U_r_tp = Cond.para_sp.U_r_tp*(1+1.e-5);
38      [dum1 dum2 dum3 plus] = Condenser(...
39          System.Cond.Pin-System.NC_Pressure, ...
40          System.Cond.hin, System.mdot_r, ...
41          System.Cond.Tain, System.Cond.wain, ...
42          Pain, Vdot_cond, Cond.para_sp, refri, libname);
43      x_Cov(1,4) = (plus - System.Cond.Q)/1.e-5/Cond.para.U_r_tp;
44
```

```
45      Cond.para_sp = Cond.para;
46      Cond.para_sp.U_r_sc = Cond.para_sp.U_r_sc*(1+1.e-5);
47      [dum1 dum2 dum3 plus] = Condenser(...
48          System.Cond.Pin-System.NC_Pressure, ...
49          System.Cond.hin, System.mdot_r, System.Cond.Tain, ...
50          System.Cond.wain, ...
51          Pain, Vdot_cond, Cond.para_sp, refri, libname);
52      x_Cov(1,5) = (plus - System.Cond.Q)/1.e-5/Cond.para.U_r_sc;
53
54      for i = 1:length(Cond.para.ΔP.C)
55          Cond.para_sp = Cond.para;
56          if abs(Cond.para.ΔP.C(i))>1.e-8
57              Cond.para_sp.ΔP.C(i) = Cond.para_sp.ΔP.C(i)*(1+1.e-5);
58          else
59              Cond.para_sp.ΔP.C(i) = 1.e-8;
60          end
61          [Pout] = Condenser(System.Cond.Pin-System.NC_Pressure, ...
62              System.Cond.hin, System.mdot_r, System.Cond.Tain, ...
63              System.Cond.wain, ...
64              Pain, Vdot_cond, Cond.para_sp, refri, libname);
65              x_Cov_ΔP(1,i) = (Pout - System.Cond.Pout)/(...
66                  Cond.para_sp.ΔP.C(i)-Cond.para.ΔP.C(i)...
67              );
68      end
69
70      status_Q = x_Cov*Cond.para.Cov*x_Cov' - Cond.para.X_limit;
```

```
71      status_ΔP = x_Cov_ΔP*Cond.para.Cov*x_Cov_ΔP' - ...
72          Cond.para.X_limit_ΔP;
73  catch err % non-converging case
74      status_Q = 9999;
75      status_ΔP = 9999;
76  end
77  end
```

M.10 Fixed Orifice Model

```
1   function mdot_r = Fixed_Orifice(Pin, Pout, hin, para, Pc, Tc, refri)
2
3   % Variable list
4   % mdot_r: mass flow rate
5
6   % Pin: Refrigerant inlet pressure (kPa)
7   % Pout: Refrigerant outlet pressure (kPa)
8   % hin: Refrigerant inlet enthalpy (J/kg)
9   % para: parameters (misc.)
10  % Pc: critical pressure (kPa)
11  % Tc: critical temperature (K)
12  % refri: Name of refrigerant
13
14  % Payne and O'Neal model (2004)
15  % Coefficients as defined in Payne (2004)
16  para.a(1,1) = 3.8811E-01;% [-]
17  para.a(2,1) = 1.1427E+01;% [-]
```

```
18 para.a(3,1) = -1.4194E+01;% [-]
19 para.a(4,1) = 1.0703E+00;% [-]
20 para.a(5,1) = -9.1928E-02;% [-]
21 para.a(6,1) = 2.1425E+01;% [-]
22 para.a(7,1) = -5.8195E+02;% [-]
23
24 para.b(1,1) = 1.1831E+00;% [-]
25 para.b(2,1) = -1.4680E+00;% [-]
26 para.b(3,1) = -1.5285E-01;% [-]
27 para.b(4,1) = -1.4639E+01;% [-]
28 para.b(5,1) = 9.8401E+00;% [-]
29 para.b(6,1) = -1.9798E-02;% [-]
30 para.b(7,1) = -1.5348E+00;% [-]
31 para.b(8,1) = -2.0533E+00;% [-]
32 para.b(9,1) = -1.7195E+01;% [-]
33 % defining variable
34 Tsat = propertyRH('T','P',Pin,'Q',0,refri);
35 hf = propertyRH('H','P',Pin,'Q',0,refri);
36 if(hin<hf)
37     Tin = propertyRH('T','P',Pin,'H',hin,refri);
38 else
39     Tin = Tsat;
40 end
41 Psat = propertyRH('P','T',Tin,'Q',0,refri);
42 rho_g = propertyRH('D','P',Pin,'Q',1,refri);
43 rho_f = propertyRH('D','P',Pin,'Q',0,refri);
```

```
44 T_sub = Tsat - Tin;
45
46 pi_gp(1) = (Pin - Psat)/Pc;
47 if (Psat>Pin)
48     pi_gp(1) = 0;
49 end
50 pi_gp(2) = rho_g/rho_f;
51 pi_gp(3) = T_sub/Tc;
52 if pi_gp(3)<0
53     pi_gp(3) = 0;
54 end
55 pi_gp(4) = Pout/Pc;
56 pi_gp(5) = Pin/Pc;
57 para.Geometry_index(3) = 0;
58 para.Geometry_index(4) = 0;
59
60 % make sure that the pi_gp does not pass the limit
61 if -(1+para.a(6)*pi_gp(1))/para.Geometry_index(1) > 0
62     pi_gp_3_limit = sqrt(-(1+para.a(6)*pi_gp(1)) /...
63         para.Geometry_index(1));
64 else
65     pi_gp_3_limit = Inf;
66 end
67
68 C1pCoeff(1) = pi/4*sqrt(rho_f*Pc*1000)/(...
69     1+para.a(6)*pi_gp(1)+para.Geometry_index(1)*(pi_gp(3))^2 ...
```

```
)*para.a(5);
C1pCoeff(2) = pi/4*sqrt(rho_f*Pc*1000)/(...
    1+para.a(6)*pi_gp(1)+para.Geometry_index(1)*(pi_gp(3))^2 ...
)*(...
    para.a(1)+para.a(2)*pi_gp(1)+para.Geometry_index(2)*pi_gp(3) +...
    para.a(4)*pi_gp(2)+para.Geometry_index(3)*pi_gp(4) +...
    para.Geometry_index(4)*pi_gp(5) ...
);

hv = propertyRH('H','P',Pin,'Q',1,refri);
xin = (hin-hf)/(hv-hf);
mdot_r = para.adj*(C1pCoeff(1)*para.D^2*log(para.L/para.D) + ...
    C1pCoeff(2)*para.D^2);
if pi_gp(3)>pi_gp_3_limit && pi_gp_3_limit>0 && mdot_r<0
    pi_gp(3) = pi_gp_3_limit*(1-1.e-8);
    C1pCoeff(1) = pi/4*sqrt(rho_f*Pc*1000)/(...
        1+para.a(6)*pi_gp(1)+para.Geometry_index(1)*(pi_gp(3))^2 ...
    )*para.a(5);
    C1pCoeff(2) = pi/4*sqrt(rho_f*Pc*1000)/(...
        1+para.a(6)*pi_gp(1)+para.Geometry_index(1)*(pi_gp(3))^2 ...
    )*( ...
        para.a(1)+para.a(2)*pi_gp(1) +...
        para.Geometry_index(2)*pi_gp(3) +...
        para.a(4)*pi_gp(2)+para.Geometry_index(3)*pi_gp(4) +...
        para.Geometry_index(4)*pi_gp(5) ...
    );
```

```
96      mdot_r = para.adj*(C1pCoeff(1)*para.D^2*log(para.L/para.D) + ...
97          C1pCoeff(2)*para.D^2);
98  end
99  if T_sub < 3
100     pi_gp(1) = (Pin - propertyRH('P','T',Tsat-3,'Q',0,refri))/Pc;
101     pi_gp(3) = 3/Tc;
102     C1pCoeff(1) = pi/4*sqrt(rho_f*Pc*1000)/(...
103         1+para.a(6)*pi_gp(1)+para.Geometry_index(1)*(pi_gp(3))^2 ...
104     )*para.a(5);
105     C1pCoeff(2) = pi/4*sqrt(rho_f*Pc*1000)/(...
106         1+para.a(6)*pi_gp(1) +...
107         para.Geometry_index(1)*(pi_gp(3))^2 ...
108     )*(...
109         para.a(1)+para.a(2)*pi_gp(1) +...
110         para.Geometry_index(2)*pi_gp(3) +...
111         para.a(4)*pi_gp(2)+para.Geometry_index(3)*pi_gp(4) +...
112         para.Geometry_index(4)*pi_gp(5) ...
113     );
114     mdot_r_3K = para.adj*(...
115         C1pCoeff(1)*para.D^2*log(para.L/para.D) + ...
116         C1pCoeff(2)*para.D^2 ...
117     );
118     if mdot_r_3K < mdot_r
119         mdot_r = mdot_r_3K;
120     end
121 end
```

```
122 % check state at inlet
123 if(hin>hf) % for two-phase inlet
124     A = pi*para.D^2/4;
125     Ctp = Payne2004Ctp(xin, Pin, Pc, Psat, A, para, refri) /...
126         Payne2004Ctp(0, Pin, Pc, Psat, A, para, refri);
127     if Ctp > 1
128         Ctp = 1;
129     end
130     mdot_r = mdot_r*Ctp;
131 end
132 end
```

M.11 Thermostatic Expansion Valve Model

```
1  function [mdot_r OpeningStep] = Thermostatic_Expansion(...
2      Pin, Pout, hin, P_evap_out, T_evap_out, para, Pc, Tc, refri...
3  )
4  % Coefficients as defined in Payne (2004)
5  para.a(1,1) = 3.8811E-01;% [-]
6  para.a(2,1) = 1.1427E+01;% [-]
7  para.a(3,1) = -1.4194E+01;% [-]
8  para.a(4,1) = 1.0703E+00;% [-]
9  para.a(5,1) = -9.1928E-02;% [-]
10 para.a(6,1) = 2.1425E+01;% [-]
11 para.a(7,1) = -5.8195E+02;% [-]
12
13 para.b(1,1) = 1.1831E+00;% [-]
```

```
14 para.b(2,1) = -1.4680E+00;% [-]
15 para.b(3,1) = -1.5285E-01;% [-]
16 para.b(4,1) = -1.4639E+01;% [-]
17 para.b(5,1) = 9.8401E+00;% [-]
18 para.b(6,1) = -1.9798E-02;% [-]
19 para.b(7,1) = -1.5348E+00;% [-]
20 para.b(8,1) = -2.0533E+00;% [-]
21 para.b(9,1) = -1.7195E+01;% [-]
22
23 Tin = propertyRH('T','P',Pin,'H',hin,refri);
24 if T_evap_out < 294
25     P_Teout = propertyRH('P','T',T_evap_out,'Q',1,refri);
26 else
27     D_cric = propertyRH('D','T',294,'Q',1,refri);
28     P_Teout = propertyRH('P','T',T_evap_out,'D',D_cric,refri);
29 end
30 SH = P_Teout - P_evap_out;
31 if SH < 0
32     SH = 0;
33 end
34
35 A_max = para.ValvePara_A_index(1) + ...
36     para.ValvePara_A_index(2)*para.SH_upper/200 + ...
37     para.ValvePara_A_index(3)*(para.SH_upper/200)^2;
38 if SH < para.SH_upper
39     A = para.ValvePara_A_index(1) + ...
```

```
40          para.ValvePara_A_index(2)*SH/200 + ...
41          para.ValvePara_A_index(3)*(SH/200)^2;
42      if A > A_max
43          A = A_max;
44      end
45  else
46      A = A_max;
47  end
48  OpeningStep = A/A_max;
49
50  rho_f = propertyRH('D','P',Pin,'Q',0.0,refri);
51  rho_v = propertyRH('D','P',Pin,'Q',1.0,refri);
52  Psat = propertyRH('P','T',Tin,'Q',0.0,refri);
53  T_sat = propertyRH('T','P',Pin,'Q',0.0,refri);
54  T_sub = T_sat - Tin;
55  pi3 = (Pin - Psat)/Pc;
56  pi6 = rho_v/rho_f;
57  pi10 = para.L/sqrt(A*4/pi);
58  pi9 = (T_sub)/Tc;
59  if T_sub < 0
60  %     rho_in = propertyRH('D','T',T_sat,'Q',0.0,refri);
61      pi3 = 0;
62      pi9 = 0;
63  end
64  pi11 = Pout/Pc;
65  pi12 = Pin/Pc;
```

```
66 hf = propertyRH('H','T',T_sat,'Q',0.0,refri);
67 hv = propertyRH('H','T',T_sat,'Q',1,refri);
68 xin = ((hin-hf)/(hv-hf));
69 if -(1+para.ValvePara_C_index(4)*pi3)/para.ValvePara_C_index(1) > 0
70     pi9_limit = sqrt(...
71         -(1+para.ValvePara_C_index(4)*pi3) /...
72             para.ValvePara_C_index(1)...
73     );
74 else
75     pi9_limit = Inf;
76 end
77 mdot_r = (...
78     para.a(1) + para.a(2)*pi3 + ...
79     para.ValvePara_C_index(2)*pi9 + ...
80     para.ValvePara_C_index(5)*pi6 + ...
81     para.ValvePara_C_index(3)*log(pi10) ...
82 )/(...
83     1+para.ValvePara_C_index(4)*pi3 +...
84     para.ValvePara_C_index(1)*pi9^2 ...
85 )*A*sqrt(rho_f*Pc*1000);
86 if pi9>pi9_limit && pi9_limit>0 && mdot_r<0
87     pi9 = pi9_limit*(1-1.e-8);
88     % recalculate if limit is reached
89     mdot_r = (...
90         para.a(1) + para.a(2)*pi3 + ...
91         para.ValvePara_C_index(2)*pi9 + ...
```

```
92          para.ValvePara_C_index(5)*pi6 + ...
93          para.ValvePara_C_index(3)*log(pi10) ...
94          )/(...
95              1+para.ValvePara_C_index(4)*pi3 +...
96              para.ValvePara_C_index(1)*pi9^2 ...
97          )*A*sqrt(rho_f*Pc*1000);
98  end
99  if T_sub < 3
100     pi3 = (Pin - propertyRH('P','T',T_sat-3,'Q',0,refri))/Pc;
101     pi9 = 3/Tc;
102     mdot_r_3K = (...
103         para.a(1) + para.a(2)*pi3 + para.ValvePara_C_index(2)*pi9 + ...
104         para.ValvePara_C_index(5)*pi6 + ...
105         para.ValvePara_C_index(3)*log(pi10) ...
106     )/(...
107         1+para.ValvePara_C_index(4)*pi3 +...
108         para.ValvePara_C_index(1)*pi9^2 ...
109     )*A*sqrt(rho_f*Pc*1000);
110     if mdot_r_3K < mdot_r
111         mdot_r = mdot_r_3K;
112     end
113 end
114 if (xin>0) % for two-phase inlet
115     xin = ((hin-hf)/(hv-hf));
116         Ctp = Payne2004Ctp(xin, Pin, Pc, Psat, A, para, refri) /...
117             Payne2004Ctp(0, Pin, Pc, Psat, A, para, refri);
```

```
118     if Ctp > 1
119         Ctp = 1;
120     end
121     mdot_r = mdot_r*Ctp;
122 end
123 
124 end
```

M.12 Adjustment Factor Calculation for Two-Phase Flow Entering Expansion Valve

```
1  function Ctp = Payne2004Ctp(xin, Pin, Pc, Psat, A, para, refri)
2  
3  % This function estimates the adjustment factor of the expansion
4  % valve mass flow rate based on the Payne correlation (2004) with
5  % the following inputs
6  
7  % xin       : inlet quality
8  % Pin       : pressure at inlet in kPa
9  % Pc        : critical pressure in kPa
10 % Psat      : saturation pressure at inlet temperature in kPa
11 % A         : inlet area in m2
12 % para      : expansion valve parameters
13 % refri     : name of refrigerant
14 
15 rho_g = propertyRH('D','P',Pin,'Q',1,refri);
16 rho_f = propertyRH('D','P',Pin,'Q',0,refri);
17 rho_mup = ((1-xin)/rho_f+xin/rho_g)^(-1);
```

```
tp6 = rho_mup/rho_f;
tp28 = Pin/Pc;
tp32 = 1 - Pin/Pc;
tp34 = xin/(1-xin)*(rho_f/rho_g)^(.5);
tp35 = 1 - Psat/Pc;
CtpCoeff(1) = (...
    para.b(1)*tp6+para.b(2)*tp6^2+para.b(3)*(log(tp6))^2 +...
    para.b(4)*(log(tp35))^2+para.b(5)*(log(tp32))^2 ...
)/(1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3);
CtpCoeff(2) = para.b(6)/(...
    1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3 ...
);
Ctp = (CtpCoeff(1) + CtpCoeff(2)*(log(para.L/sqrt(A*4/pi)))^2);
end
```

M.13 Leverage Calculation of Expansion Valve Model

```
function [status x_Cov x_status] = EXV_status(...
    System, EXV, Pc, Tc, refri ...
)
% compute if the EXV model is operaitng out of range
try
    hf = propertyRH('H', 'P', System.EXV.Pin, 'Q', 0, refri);
    hg = propertyRH('H', 'P', System.EXV.Pin, 'Q', 1, refri);
    x_in = (System.EXV.hin - hf)/(hg - hf);
    x_status = x_in-EXV.para.x_max;
    if strcmp(EXV.Type,'TXV')
```

```
11          x_Cov = zeros(1,9);
12          for i = 1:3
13              EXV.para_sp = EXV.para;
14              if abs(EXV.para.ValvePara_A_index(i)) > 1.e-8
15                  EXV.para_sp.ValvePara_A_index(i) = ...
16                      EXV.para_sp.ValvePara_A_index(i)*(1+1.e-5);
17              else
18                  EXV.para_sp.ValvePara_A_index(i) = 1.e-8;
19              end
20              if strcmp(EXV.Equalizer,'Ext_Evap')
21                  plus = Thermostatic_Expansion(...
22                      System.EXV.Pin, System.EXV.Pout, ...
23                      System.EXV.hin, System.SuctionLine.Pin, ...
24                      propertyRH(...
25                          'T','P',System.SuctionLine.Pin,'H',...
26                          System.SuctionLine.hin,refri ...
27                      ), EXV.para_sp, Pc, Tc, refri ...
28                  );
29              elseif strcmp(EXV.Equalizer,'Ext_Comp')
30                  plus = Thermostatic_Expansion(...
31                      System.EXV.Pin, System.EXV.Pout, ...
32                      System.EXV.hin, System.SuctionLine.Pout, ...
33                      propertyRH(...
34                          'T','P',System.SuctionLine.Pout,'H',...
35                          System.SuctionLine.hout,refri ...
36                      ), EXV.para_sp, Pc, Tc, refri ...
```

```
37                  );
38              else
39                  plus = Thermostatic_Expansion(...
40                      System.EXV.Pin, System.EXV.Pout, ...
41                      System.EXV.hin, ...
42                      System.EXV.Pout, propertyRH(...
43                          'T','P',System.SuctionLine.Pin,'H',...
44                          System.SuctionLine.hin,refri ...
45                      ), EXV.para_sp, Pc, Tc, refri ...
46                  );
47              end
48              x_Cov(1,i) = (plus - System.EXV.mdot_r)/(...
49                  EXV.para_sp.ValvePara_A_index(i) - ...
50                  EXV.para.ValvePara_A_index(i) ...
51              )*EXV.para.NC_Para(i);
52          end
53
54          EXV.para_sp = EXV.para;
55          EXV.para_sp.L = EXV.para_sp.L*(1+1.e-5);
56          if strcmp(EXV.Equalizer,'Ext_Evap')
57              plus = Thermostatic_Expansion(...
58                  System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
59                  System.SuctionLine.Pin, propertyRH(...
60                      'T','P',System.SuctionLine.Pin,'H',...
61                      System.SuctionLine.hin,refri ...
62                  ), EXV.para_sp, Pc, Tc, refri ...
```

```
63              );
64          elseif strcmp(EXV.Equalizer,'Ext_Comp')
65              plus = Thermostatic_Expansion(...
66                  System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
67                  System.SuctionLine.Pout, propertyRH(...
68                      'T','P',System.SuctionLine.Pout,'H',...
69                      System.SuctionLine.hout,refri ...
70                  ), EXV.para_sp, Pc, Tc, refri ...
71              );
72          else
73              plus = Thermostatic_Expansion(...
74                  System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
75                  System.EXV.Pout, propertyRH(...
76                      'T','P',System.SuctionLine.Pin,'H',...
77                      System.SuctionLine.hin,refri ...
78                  ), EXV.para_sp, Pc, Tc, refri ...
79              );
80          end
81          x_Cov(1,4) = (plus - System.EXV.mdot_r)/1.e-5/ ...
82              EXV.para.L*EXV.para.NC_Para(4);
83
84          for i = 5:9
85              EXV.para_sp = EXV.para;
86              if abs(EXV.para.ValvePara_C_index(i-4))>1.e-8
87                  EXV.para_sp.ValvePara_C_index(i-4) = ...
88                      EXV.para_sp.ValvePara_C_index(i-4)*(1+1.e-5);
```

```
89              else
90                  EXV.para_sp.ValvePara_C_index(i-4) = 1.e-8;
91              end
92              if strcmp(EXV.Equalizer,'Ext_Evap')
93                  plus = Thermostatic_Expansion(...
94                      System.EXV.Pin, System.EXV.Pout, ...
95                      System.EXV.hin, ...
96                      System.SuctionLine.Pin, propertyRH(...
97                          'T','P',System.SuctionLine.Pin,'H',...
98                          System.SuctionLine.hin,refri ...
99                      ), EXV.para_sp, Pc, Tc, refri ...
100                 );
101             elseif strcmp(EXV.Equalizer,'Ext_Comp')
102                 plus = Thermostatic_Expansion(...
103                     System.EXV.Pin, System.EXV.Pout, ...
104                     System.EXV.hin, ...
105                     System.SuctionLine.Pout, propertyRH(...
106                         'T','P',System.SuctionLine.Pout,'H',...
107                         System.SuctionLine.hout,refri ...
108                     ), EXV.para_sp, Pc, Tc, refri ...
109                 );
110             else
111                 plus = Thermostatic_Expansion(...
112                     System.EXV.Pin, System.EXV.Pout, ...
113                     System.EXV.hin, System.EXV.Pout, ...
114                     propertyRH(...
```

```
115                         'T','P',System.SuctionLine.Pin,'H',...
116                         System.SuctionLine.hin,refri ...
117                     ), EXV.para_sp, Pc, Tc, refri ...
118                 );
119             end
120             x_Cov(1,i) = (plus - System.EXV.mdot_r)/(...
121                 EXV.para_sp.ValvePara_C_index(i-4) - ...
122                 EXV.para.ValvePara_C_index(i-4) ...
123             )*EXV.para.NC_Para(i);
124         end
125     else
126         x_Cov = zeros(1,6);
127         System.EXV.mdot_r = Fixed_Orifice(...
128             System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
129             EXV.para, Pc, Tc, refri ...
130         );
131 
132         EXV.para_sp = EXV.para;
133         if abs(EXV.para.D/EXV.para.FEO_Para(1))>1
134             EXV.para_sp.D = EXV.para_sp.D*(1+1.e-5);
135         else
136             EXV.para_sp.D = EXV.para_sp.D+1.e-5*EXV.para.FEO_Para(1);
137         end
138         plus = Fixed_Orifice(...
139             System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
140             EXV.para_sp, Pc, Tc, refri ...
```

```
141         );
142         x_Cov(1,1) = (plus - System.EXV.mdot_r)/(...
143             EXV.para_sp.D-EXV.para.D ...
144         )*EXV.para.FEO_Para(1);
145
146         EXV.para_sp = EXV.para;
147         if abs(EXV.para.L/EXV.para.FEO_Para(2))>1
148             EXV.para_sp.L = EXV.para_sp.L*(1+1.e-5);
149         else
150             EXV.para_sp.L = EXV.para_sp.L+1.e-5*EXV.para.FEO_Para(2);
151         end
152         plus = Fixed_Orifice(...
153             System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
154             EXV.para_sp, Pc, Tc, refri ...
155         );
156         x_Cov(1,2) = (plus - System.EXV.mdot_r)/(...
157             EXV.para_sp.L-EXV.para.L ...
158         )*EXV.para.FEO_Para(2);
159
160         for i = 3:6
161             EXV.para_sp = EXV.para;
162             if abs(EXV.para.Geometry_index(i-2) /...
163                 EXV.para.FEO_Para(i))> 1
164                 EXV.para_sp.Geometry_index(i-2) = ...
165                     EXV.para.Geometry_index(i-2)*(1+1.e-5);
166             else
```

```
167                 EXV.para_sp.Geometry_index(i-2) = ...
168                     EXV.para.Geometry_index(i-2)+ ...
169                     1.e-5*EXV.para.FEO_Para(i);
170             end
171             plus = Fixed_Orifice(...
172                 System.EXV.Pin, System.EXV.Pout, System.EXV.hin, ...
173                 EXV.para_sp, Pc, Tc, refri ...
174             );
175             x_Cov(1,i) = (plus - System.EXV.mdot_r)/(...
176                 EXV.para_sp.Geometry_index(i-2) -...
177                 EXV.para.Geometry_index(i-2) ...
178             )*EXV.para.FEO_Para(i);
179         end
180     end
181     status = x_Cov*EXV.para.Cov*x_Cov' - EXV.para.X_limit;
182 catch err % non-converging case
183     status = 9999;
184     if strcmp(EXV.Type,'TXV')
185         x_Cov = zeros(1,9);
186     else
187         x_Cov = zeros(1,6);
188     end
189 end
190 end
```

M.14 Evaporator Model

```
function [Pin hout Taout waout Qevap Charge ...
    FanPower EvapOutput ma EvapInput SHR rho_tp rho_sh...
] = Evaporator(...
    Pout, hin, hout_est, mdot_r, Tain, wain, Pain, Vdot, ...
    para, refri, libname ...
)

% Variable list
% Pin: Refrigenrat pressure at inlet (kPa)
% hout: Refrigerant enthalpy at outlet (kPa)
% Taout: Air temperature at outlet (K)
% waout: Air humidity at outlet (kg/kg)
% Qevap: Heat transfer rate (W)
% Charge: Amount of refrigerant in the evaporator (kg)
% FanPower: Evaporator fan power consumption (W)
% EvapOutput: All other informaiton listed in the function in DLL

% Pout: Refrigerant pressure at outlet (kPa)
% hin: Refrigenrat enthalpy at inlet (W)
% hout_est: Estimated refrigerant enthalpy, write -9999 if unknown
% mdot_r: Refrigeraat mass flow rate (kg/s)
% Tain: Air temperature at inlet (K)
% wain: Humidity at inlet (kg/kg)
% Pain: Atmospheric pressure (kPa)
% Vdot: Air volumetric flow at inlet (m^3/s)
% para: parameters (misc.)
```

```
27 % refri: Name of refrigerant
28 % lib: Name of dll library to be used
29
30 % offset fan heat
31 FanPower = para.C_fan(1) + para.C_fan(2)*Vdot + para.C_fan(3)*Vdot^2;
32 if FanPower < 0.1*para.W_rated
33     FanPower = 0.1*para.W_rated;
34 end
35 if strcmp(para.type,'Split')
36     airinlet = calllib(...
37         libname, 'HumAir_DLL', Tain, Pain, 2, wain, zeros(1,5)...
38     );
39     ma = (1)/airinlet(5)*Vdot;
40     Tain = Tain + FanPower/ma/(1006 + 1860*airinlet(2));
41     airinlet = calllib(...
42         libname, 'HumAir_DLL', Tain, Pain, 2, wain, airinlet ...
43     );
44 else
45     airinlet = zeros(1,5);
46     airinlet = calllib(...
47         libname, 'HumAir_DLL', Tain, Pain, 2, wain, airinlet ...
48     );
49 end
50
51 % UAa
52 ma = (1)/airinlet(5)*Vdot;
```

```
53 EvapInput(1) = 1;
54
55 EvapInput(2) = para.U_a*(ma/para.ma)^para.n*tanh(...
56     para.p*(para.U_a*(ma/para.ma)^para.n)^.5...
57 )/para.p*para.A_r/(para.U_a*(ma/para.ma)^para.n)^.5;
58 cpa = 1006 + 1860*airinlet(2);
59 cs = 0;
60 Tsat = propertyRH('T','P',Pout,'Q',1,refri);
61 cs = calllib(libname, 'Matlab_cair_sat', Tsat, cs)*1000;
62
63 EvapInput(3) = para.U_a*(ma/para.ma)^para.n*tanh(...
64     (cs/cpa)^0.5*para.p*(para.U_a*(ma/para.ma)^para.n)^.5...
65 )/(...
66     (cs/cpa)^0.5*para.p...
67 )/(para.U_a*(ma/para.ma)^para.n)^.5*para.A_r;
68
69 %UAtp
70 hv = propertyRH('H', 'P', Pout,'Q', 1,refri);
71 hf = propertyRH('H', 'P', Pout,'Q', 0,refri);
72 xin = (hin - hf)/(hv - hf);
73 if(xin≤0.001)
74     xin = 0.001; % avoid division by zero error
75 end
76 if((hout_est>hin)&&(hout_est<hv))
77     xout = (hout_est - hf)/(hv - hf);
78     EvapInput(5) =  h_bartp(...
```

```
79          para.Utp*para.A_r, xin, xout, mdot_r, mdot_r*(hout_est-hin), ...
80          Tsat, refri ...
81      ); %use values from previous estimations for calculation
82  else
83      EvapInput(5) =  h_bartp(...
84          para.Utp*para.A_r, xin, 1, mdot_r, mdot_r*(hv-hin), Tsat, ...
85          refri ...
86      ); %outlet refrigerant quality is an assumption
87  end
88
89  %UAsh
90  cp_v = propertyRH('C', 'P', Pout,'Q', 1, refri);
91  mu_v = propertyRH('V', 'P', Pout,'Q', 1, refri);
92  k_v = propertyRH('L', 'P', Pout,'Q', 1, refri);
93  Ksh = (cp_v*mu_v/k_v)^0.4*(mdot_r/mu_v)^0.8;
94  EvapInput(4) = para.Ush*Ksh/para.Ksh*para.A_r;
95
96  % input arrangement
97  EvapInput(6) = Tain;
98  EvapInput(7) = airinlet(4);
99  EvapInput(8) = Vdot;
100 EvapInput(9) = Pain;
101 EvapInput(10) = mdot_r;
102 EvapInput(11) = xin;
103 EvapInput(12) = Pout;
104 EvapInput(13) = Tsat;
```

```
105 EvapInput(14) = para.Din;
106 EvapInput(15) = para.Lt;
107 EvapOutput = zeros(1,14);
108 if ¬isempty(findstr(refri,'.fld'))
109     index_end = findstr(refri,'.fld');
110     refname = refri(1:index_end-1);
111 else
112     refname = refri;
113 end
114 [EvapInput EvapOutput] = calllib(...
115     libname, 'Matlab_Evaporator_Forward_Charge_ii', EvapInput, ...
116     EvapOutput, refname ...
117 );
118
119 % output arrangement and calculation of outlet pressure
120 Qevap = EvapOutput(1);
121 hout = hin + Qevap/mdot_r;
122 Taout = EvapOutput(8);
123 waout = EvapOutput(14);
124 Taout = T_homega(...
125     waout, airinlet(3)-Qevap/1000/ma, Pain, Taout, libname ...
126 );
127 airoutlet = calllib(...
128     libname, 'HumAir_DLL', Taout, Pain, 2, waout, airinlet ...
129 );
130 if airoutlet(1)>Taout
```

```
131     Taout = T_hRH(1.0, airinlet(3)-Qevap/1000/ma, Pain, Taout, libname);
132     airoutlet = calllib(...
133         libname, 'HumAir_DLL', Taout, Pain, 4, 1., airinlet ...
134     );
135     waout = airoutlet(2);
136 end
137 Charge = EvapOutput(13);
138 rho_l = propertyRH('D','P',Pout,'Q',0,refri);
139 rho_v = propertyRH('D','P',Pout,'Q',1,refri);
140 SHR = ma*cpa*(Tain - Taout)/Qevap;
141 if SHR > 1
142     SHR = 1;
143     Taout = Tain - SHR*Qevap/ma/cpa;
144 end
145 if hout > hv
146     xout = 1;
147 else
148     xout = (hout - hf)/(hv - hf);
149 end
150 S = (rho_l/rho_v)^(1/3)*(rho_v/rho_l);
151 gamma = -(...
152     S*(log(((xout-1)*S-xout)/((xin-1)*S-xin))+xout-xin)-xout+xin...
153 )/(S^2-2*S+1)/(xout-xin);
154 rho_tp = (gamma*rho_v+(1-gamma)*rho_l);
155 mu_l = propertyRH('V', 'P', Pout,'Q', 0, refri);
156 rho_out = RefrigerantDensity(Pout,hout,refri);
```

```
157 rho_sh = .5*(rho_v+rho_out);
158
159 Pin_temp = Pout + para.ΔP.C(1)*EvapOutput(5).*(...
160     mdot_r./para.ΔP.mdot_rated ...
161 ).^2./(rho_tp./para.ΔP.rho_rated).*(...
162     (xin./rho_v.*mu_v+(1-xin)./rho_l.*mu_l)./(...
163         xin./rho_v+(1-xin)./rho_l ...
164     )./para.ΔP.mu_v_rated ...
165 ).^para.ΔP.C(4);
166 Pin_temp = Pin_temp +  para.ΔP.C(2).*EvapOutput(4).*(...
167     mdot_r./para.ΔP.mdot_rated ...
168 ).^2./(rho_v./para.ΔP.rho_out_rated);
169 Pin = Pin_temp + para.ΔP.C(3).*(...
170     mdot_r./para.ΔP.mdot_rated ...
171 ).^2.*(1./rho_out-1./RefrigerantDensity(...
172     Pout,hin,refri ...
173 ))*para.ΔP.rho_rated;
174 end
```

M.15 Heat Transfer Coefficient Calculation of Evaporating Flow

```
1 function UA_bar = h_bartp(UA_corr, xin, xout, mdot_r, Q, T, refri)
2
3 % The function is used to calculate an average
4 % UA on the refrigerant side
5 % UA_corr: corrected parameter
6 % xin: inlet quality
```

```
7  % mdot_r: refrigerant flow rate (kg/s)
8  % Q: heat transfer rate (kW)
9  % T: Saturation temperature (K)
10
11 % defining intervals
12 n = 201;
13 Δx = (xout-xin)/(n-1);
14 x = xin:Δx:xout;
15
16 % defining properties
17 % enthalpy of vaporization in J/kg
18 h_fg = propertyRH('H', 'T', T,'Q', 1, refri) - ...
19     propertyRH('H', 'T', T,'Q', 0, refri);
20 % liquid density
21 rhol = propertyRH('D', 'T', T,'Q', 0, refri);
22 % vapor density
23 rhov = propertyRH('D', 'T', T,'Q', 1, refri);
24 % liquid dynamic viscosity
25 nu = propertyRH('V', 'T', T,'Q', 0, refri);
26 % liquid thermal conductivity
27 k = propertyRH('L', 'T', T,'Q', 0, refri);
28 % liquid spcific heat capacity
29 cp = propertyRH('C', 'T', T,'Q', 0, refri);
30 % liquid dynamic viscosity
31 nu_v = propertyRH('V', 'T', T,'Q', 1, refri);
32 % liquid thermal conductivity
```

```
k_v = propertyRH('L', 'T', T,'Q', 1, refri);
% liquid spcific heat capacity
cp_v = propertyRH('C', 'T', T,'Q', 1, refri);

% as defined in Shah (1982)

% calculating parameters
Bo = Q/mdot_r/h_fg; % Boiling number
if(Bo≥11*10^-4)
    F = 14.7;
else
    F = 15.43;
end

%integration
% sum_htp = 0;
bulk = UA_corr*(mdot_r)^.8*(cp/rhol/nu)^.4*k^.6;
phi_nb1 = 230*sqrt(Bo);
phi_nb2 = 1 + 46*sqrt(Bo);
phi_bs_f = F*sqrt(Bo);
Bo_limit = .3*10^-4;
rho_ratio_sq = sqrt(rhov/rhol);
x_diff_8 = (1-x).^.8;
Co = rho_ratio_sq*(x_diff_8./x.^.8);
phi_cb = (1.8./Co.^.8)';
phi_nb = zeros(n,1);
```

```
phi_bs = zeros(n,1);
phi_bs_1 = phi_bs_f.*exp(2.74./Co.^.1);
phi_bs_2 = phi_bs_f.*exp(2.74./Co.^.15);
for i = 1:n
    if(x(1,i)<1)
        if(Co(1,i)>1)
            if(Bo>Bo_limit)
                phi_nb(i,1) = phi_nb1;
            else
                phi_nb(i,1) = phi_nb2;
            end
        elseif (Co(1,i)>0.1)
            phi_bs(i,1) = phi_bs_1(1,i);
        else
            phi_bs(i,1) = phi_bs_2(1,i);
        end
    end
end
htp = bulk.*max([phi_cb phi_nb phi_bs],[],2).*x_diff_8';
if sum(x≥1)>0
    htp(x≥1) = UA_corr*(mdot_r)^.8*(cp_v/rhov/nu_v)^.4*k_v^.6;
end
htp([1,n]) = htp([1,n])/2;
sum_htp = sum(htp)*Δx;
UA_bar = sum_htp/(1-xin);
end
```

M.16 Leverage Calculation of Evaporator Model

```
function [status_Q status_SHR status_ΔP] = Evap_status(...
    System, Evap, Vdot_evap, Pain, Pc, Tc, refri, libname ...
)
% compute if the evaporator  model is operaitng out of range
try
    x_Cov = zeros(1,5);
    x_Cov_SHR = zeros(1,5);
    x_Cov_ΔP = zeros(1,length(Evap.para.ΔP.C));

    Evap.para_sp = Evap.para;
    Evap.para_sp.Utp = Evap.para_sp.Utp*(1+1.e-5);
    [dum1 dum2 dum3 dum4 plus dum5 dum6 dum7 dum8 dum9 SHR] = ...
        Evaporator(System.Evap.Pout, System.Evap.hin, ...
        System.Evap.hout, ...
        System.mdot_r, System.Evap.Tain, System.Evap.wain, Pain, ...
        Vdot_evap, Evap.para_sp, refri, libname);
    x_Cov(1,1) = (plus - System.Evap.Q)/1.e-5/Evap.para.Utp;
    x_Cov_SHR(1,1) = (SHR - System.Evap.SHR)/1.e-5/Evap.para.Utp;

    Evap.para_sp = Evap.para;
    Evap.para_sp.Ush = Evap.para_sp.Ush*(1+1.e-5);
    [dum1 dum2 dum3 dum4 plus dum5 dum6 dum7 dum8 dum9 SHR] = ...
        Evaporator(System.Evap.Pout, System.Evap.hin, ...
        System.Evap.hout, ...
```

```
25          System.mdot_r, System.Evap.Tain, System.Evap.wain, Pain, ...
26          Vdot_evap, Evap.para_sp, refri, libname);
27      x_Cov(1,2) = (plus - System.Evap.Q)/1.e-5/Evap.para.Ush;
28      x_Cov_SHR(1,2) = (SHR - System.Evap.SHR)/1.e-5/Evap.para.Ush;
29
30      Evap.para_sp = Evap.para;
31      Evap.para_sp.U_a = Evap.para_sp.U_a*(1+1.e-5);
32      [dum1 dum2 dum3 dum4 plus dum5 dum6 dum7 dum8 dum9 SHR] = ...
33          Evaporator(System.Evap.Pout, System.Evap.hin, ...
34          System.Evap.hout, ...
35          System.mdot_r, System.Evap.Tain, System.Evap.wain, Pain, ...
36          Vdot_evap, Evap.para_sp, refri, libname);
37      x_Cov(1,3) = (plus - System.Evap.Q)/1.e-5/Evap.para.U_a;
38      x_Cov_SHR(1,3) = (SHR - System.Evap.SHR)/1.e-5/Evap.para.U_a;
39
40      Evap.para_sp = Evap.para;
41      Evap.para_sp.n = Evap.para_sp.n*(1+1.e-5);
42      [dum1 dum2 dum3 dum4 plus dum5 dum6 dum7 dum8 dum9 SHR] = ...
43          Evaporator(System.Evap.Pout, System.Evap.hin, ...
44          System.Evap.hout, ...
45          System.mdot_r, System.Evap.Tain, System.Evap.wain, Pain, ...
46          Vdot_evap, Evap.para_sp, refri, libname);
47      x_Cov(1,4) = (plus - System.Evap.Q)/1.e-5/Evap.para.n;
48      x_Cov_SHR(1,4) = (SHR - System.Evap.SHR)/1.e-5/Evap.para.n;
49
50      Evap.para_sp = Evap.para;
```

```
51      Evap.para_sp.p = Evap.para_sp.p*(1+1.e-5);
52      [dum1 dum2 dum3 dum4 plus dum5 dum6 dum7 dum8 dum9 SHR] = ...
53          Evaporator(System.Evap.Pout, System.Evap.hin, ...
54          System.Evap.hout, ...
55          System.mdot_r, System.Evap.Tain, System.Evap.wain, Pain, ...
56          Vdot_evap, Evap.para_sp, refri, libname);
57      x_Cov(1,5) = (plus - System.Evap.Q)/1.e-5/Evap.para.p;
58      x_Cov_SHR(1,5) = (SHR - System.Evap.SHR)/1.e-5/Evap.para.p;
59
60      for i = 1:length(Evap.para.ΔP.C)
61          Evap.para_sp = Evap.para;
62          if abs(Evap.para.ΔP.C(i))>1.e-8
63              Evap.para_sp.ΔP.C(i) = ...
64                  Evap.para_sp.ΔP.C(i)*(1+1.e-5);
65          else
66              Evap.para_sp.ΔP.C(i) = 1.e-8;
67          end
68          [Pin] = ...
69              Evaporator(System.Evap.Pout, System.Evap.hin, ...
70              System.Evap.hout, ...
71              System.mdot_r, System.Evap.Tain, System.Evap.wain, ...
72              Pain, ...
73              Vdot_evap, Evap.para_sp, refri, libname);
74          x_Cov_ΔP(1,i) = (Pin - System.Evap.Pin)/(...
75              Evap.para_sp.ΔP.C(i)-Evap.para.ΔP.C(i) ...
76          );
```

```
77      end
78
79      status_Q = x_Cov*Evap.para.Cov*x_Cov' - Evap.para.X_limit;
80      status_SHR = x_Cov_SHR*Evap.para.Cov_SHR*x_Cov_SHR' - ...
81          Evap.para.X_limit_SHR;
82      status_ΔP = x_Cov_ΔP*Evap.para.Cov_ΔP*x_Cov_ΔP' - ...
83          Evap.para.X_limit_ΔP;
84  catch err % non-converging case
85      status_Q = 9999;
86      status_SHR = 9999;
87      status_ΔP = 9999;
88  end
89  end
```

M.17 Refrigerant Pipeline Model

```
1   function [Pout hout Charge Q_status ΔP_status] = pipeline4(...
2       mr, Pin, hin, Tamb, para, refri, phase, libname ...
3   )
4   % Heat loss model
5   Tin = propertyRH('T','P',Pin,'H',hin,refri);
6   hout = hin - para.mdot_rated/mr*para.Δh_rated*para.C_Q(1)*(...
7       abs(Tin-Tamb)./Tin...
8   ).^para.C_Q(2)*(Tin-Tamb)/para.ΔT_rated;
9   if isnan(hout)
10      hout = hin;
11      Q_status = 0;
```

```
12 else
13     vec_x = [...
14             (abs(Tin-Tamb)./Tin).^para.C_Q(2).*(...
15                 Tin-Tamb ...
16             )/para.ΔT_rated*para.mdot_rated*para.Δh_rated ...
17             para.C_Q(1).*(abs(Tin-Tamb)./Tin).^para.C_Q(2).*(...
18                 Tin-Tamb...
19             )/para.ΔT_rated*para.mdot_rated *...
20                 para.Δh_rated.*log(abs(Tin-Tamb)./Tin)
21         ];
22     Q_status = vec_x*para.COV_X_Q*vec_x'-para.Q_limit;
23 end
24
25 % solve implicitly but skip solving if no empirical coefficients are
26 % available
27 rho = RefrigerantDensity(Pin, hin, refri);
28 if sum(abs(para.C)) > 1.e-5
29     h_l = propertyRH('H','P',Pin,'Q',0,refri);
30     h_v = propertyRH('H','P',Pin,'Q',1,refri);
31     if hin < h_l || hin > h_v
32     % to avoid calculation of transport
33     % properties with two-phase refrigerant
34         try
35             V = propertyRH('V','P',Pin,'H',hin,refri);
36             rho_temp = rho;
37         catch err
```

```
38              % check if it is in liquid or gas line
39              if phase == 1
40                  V = propertyRH('V','P',Pin,'Q',1,refri);
41                  rho_temp = propertyRH('D','P',Pin,'Q',1,refri);
42              else
43                  V = propertyRH('V','P',Pin,'Q',0,refri);
44                  rho_temp = propertyRH('D','P',Pin,'Q',0,refri);
45              end
46          end
47      else
48          % check if it is in liquid or gas line
49          if phase == 1
50              V = propertyRH('V','P',Pin,'Q',1,refri);
51              rho_temp = propertyRH('D','P',Pin,'Q',1,refri);
52          else
53              V = propertyRH('V','P',Pin,'Q',0,refri);
54              rho_temp = propertyRH('D','P',Pin,'Q',0,refri);
55          end
56      end
57      X1 = (mr/para.mdot_rated);
58      X2 = (V./para.V_rated);
59      X3 = (rho_temp./para.rho_rated);
60      % use temp. values to avoid very large pressure drop
61      Pout_temp = para.C(1)*X1.^para.C(2).*X2.^para.C(3).*X3.^para.C(4);
62      Pout = Pin - Pout_temp;
63      vec_x = [...
```

```
64          X1.^para.C(2).*X2.^para.C(3).*X3.^para.C(4)...
65          (para.C(1)*X1.^para.C(2).*X2.^para.C(3).*X3.^para.C(4)).*...
66          log(X1)...
67          (para.C(1)*X1.^para.C(2).*X2.^para.C(3).*X3.^para.C(4)).*...
68          log(X2)...
69          (para.C(1)*X1.^para.C(2).*X2.^para.C(3).*X3.^para.C(4)).*...
70          log(X3)...
71          ];
72      ΔP_status = vec_x*para.COV_X_ΔP*vec_x'-para.ΔP_limit;
73  else
74      Pout = Pin;
75      ΔP_status = 0;
76  end
77  try
78      rho_out = RefrigerantDensity(Pout, hout, refri);
79  catch err
80      rho_out = rho;
81      clear err;
82  end
83  Charge = .5*(rho+rho_out)*para.line*(pi*para.dia^2/4); % added
84  end
```

M.18 Additional Functions to Assist Property Calculation

```
1  function rho = RefrigerantDensity(P, h, refri)
2  % to calculate the density by Zivi's correlation for two-phase flow
3  % and normal for thermodynamicaaly equilibrium for one-phase
```

```
4  %
5  % Input variables
6  % P          : pressure in kPa
7  % h          : enthalpy in J/kg
8  % refri      : name of refrigerant
9  %
10 % Output variables
11 % rho        : density in kg/m3
12
13 h_v = propertyRH('H','P',P,'Q',1,refri);
14 h_l = propertyRH('H','P',P,'Q',0,refri);
15 if h ≤ h_l || h ≥ h_v
16     % one-phase
17     rho = propertyRH('D','P',P,'H',h,refri);
18 else
19     % two-phase
20     rho_v = propertyRH('D','P',P,'Q',1,refri);
21     rho_l = propertyRH('D','P',P,'Q',0,refri);
22     x = (h-h_l)/(h_v-h_l);
23     mu = (rho_l/rho_v)^(1/3)*(rho_v/rho_l); % Zivi correlation
24     alpha = 1/(1+(1-x)/x*mu);
25     rho = alpha*rho_v + (1-alpha)*rho_l;
26 end
27 end
```

```
function variable = propertyRH(...
    string1, string2, value2, string3, value3, Refrigerant ...
)
%Date: 11/07/2009
%Creator: Howard Cheung
%Venue: Herrick Laboratory, Purdue University

% Function:
% to convert the property function to be usable by refrigerant mixture
% R410a

%string1 - value3: variables to be put into property function
% Refrigerant: name of refrigerant
% ratio_array: vector showing the mass fraction of refrigerant
% (leave blank for pure substance)

if(length(Refrigerant) == 5)
    if((Refrigerant == 'R410a') | (Refrigerant == 'r410a') | ...
            (Refrigerant == 'R410A'))
        if (string1 == 'T' | string1 == 'H' | string1 == 'P' ...
                | string1 == 'S' | string1 == 'D')
            variable = refpropm(...
                string1, string2, value2, string3, value3, ...
                'r410a.ppf' ...
            );
        else
```

```
27              variable = refpropm(...
28                  string1, string2, value2, string3, value3, ...
29                  'R32.fld', ...
30                  'R125.fld', [0.5 0.5] ...
31              );
32          end
33      elseif((Refrigerant == 'R407C') | (Refrigerant == 'r407c') | ...
34              (Refrigerant == 'R407C'))
35          if(string1 == 'T' | string1 == 'H' | string1 == 'P' | ...
36                  string1 == 'S' | string1 == 'D')
37              variable = refpropm(...
38                  string1, string2, value2, string3, value3, ...
39                  'r407c.ppf'...
40              );
41          else
42              variable = refpropm(...
43                  string1, string2, value2, string3, value3, ...
44                  'R32.fld', ...
45                  'R125.fld', 'R134A.fld', [0.23 0.25 0.52] ...
46              );
47          end
48      else
49          variable = refpropm(...
50              string1, string2, value2, string3, value3, Refrigerant ...
51          );
52      end
```

```
elseif (length(Refrigerant) == 3)
    if ((Refrigerant == 'Air') | (Refrigerant == 'air'))
        variable = refpropm(...
            string1, string2, value2, string3, value3, ...
            'nitrogen.fld', 'oxygen.fld', 'argon.fld', ...
            [0.78 0.21 0.01]...
        );
    else
        variable = refpropm(...
            string1, string2, value2, string3, value3, Refrigerant ...
        );
    end
else
    variable = refpropm(...
        string1, string2, value2, string3, value3, Refrigerant ...
    );
end

end
```

```
function T = T_homega(omega, h, Pa, T_guess, libname)
    % This functoin finds the air temperature based on the humidity
    % ratio and the air enthalpy

    % Input variables
```

```
6       % omega      : humidity ratio in kg of water/kg of dry air
7       % h          : air enthalpy in kJ/kg
8       % Pa         : atmospheric pressure in kPa
9       % T_guess    : guess temperature for the solution
10      % libname    : name of library
11
12      % Output variable
13      % T          : air temperautre in K
14      T = fzero(@(T_guess) T_homega_residual(...
15          omega, h, T_guess, Pa, libname ...
16      ), T_guess);
17  end
18
19  function residual = T_homega_residual(omega, h, T_guess, Pa, libname)
20      air = calllib(...
21          libname, 'HumAir_DLL', T_guess, Pa, 2, omega, zeros(1,5)...
22      );
23      residual = h - air(3);
24  end
```

```
1   function T = T_hRH(rh, h, Pa, T_guess, libname)
2       % This functoin finds the air temperature based on the
3       % relative humidity and the air enthalpy
4
5       % Input variables
```

```
6       % rh          : relative humidity
7       % h           : air enthalpy in kJ/kg
8       % Pa          : atmospheric pressure in kPa
9       % T_guess     : guess temperature for the solution
10      % libname     : name of library
11
12      % Output variable
13      % T           : air temperautre in K
14      T = fzero(@(T_guess) T_hRH_residual(...
15          rh, h, T_guess, Pa, libname ...
16      ), T_guess);
17  end
18
19  function residual = T_hRH_residual(rh, h, T_guess, Pa, libname)
20      air = calllib(...
21          libname, 'HumAir_DLL', T_guess, Pa, 4, rh, zeros(1,5) ...
22      );
23      residual = h - air(3);
24  end
```

```
1   %% Other misc. functions
2   % unit conversion
3   function WB = Wetbulb(T, TD, Pain, libname)
4       WB = fzero(@(WB) Wetbulb_residual(T, TD, WB, Pain, libname), T);
5       if isnan(WB)
```

```
        try
            WB = fzero(@(WB) Wetbulb_residual(...
                T, TD, WB, Pain, libname ...
            ), [T TD]);
        catch err
            try
                WB = fzero(@(WB) Wetbulb_residual(...
                    T, TD, WB, Pain, libname ...
                ), [T 276]);
            catch err  % if too cold
                WB_temp = TD;
                while WB_temp < T
                    res_temp = Wetbulb_residual(...
                        T, TD, WB_temp, Pain, libname ...
                    );
                    if ¬isnan(res_temp)
                        break;
                    else
                        WB_temp = WB_temp+0.3;
                    end
                end
                try
                    WB = fzero(@(WB) Wetbulb_residual(...
                        T, TD, WB, Pain, libname), [T WB_temp] ...
                    );
                catch err
```

```
                    WB = NaN;
                end
            end
        end
    end
end
```

```
function residual = Wetbulb_residual(T, TD, WB, Pain, libname)
    airinlet = zeros(1,5);
    airinlet = calllib(...
        libname, 'HumAir_DLL', T, Pain, 3, WB, airinlet ...
    );
    residual =  airinlet(1) - TD;
end
```

M.19 Miscellaneous Function for Iteration

```
% cycle solver with adjustment
function [...
    System,Real,dev2,residual_pre,flag,func_output,err...
] = cycle_output(...
    Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
    Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
    EXV, Evap, SuctionLine, refri, Tc, Pc, hf, hv, rhov, ...
    DP_cond, DP_evap, libname...
)
```

```
% solve for a good temperature difference
if SCSH.NC_level > 1.e-8
    if Guess(2,1) + 0.1 < 0.95;
        Guess(2,1) = Guess(2,1) + 0.1;
    else
        Guess(2,1) = 0.95;
    end
    if length(Guess) < 6
        Guess(length(Guess)+1,1) = 0;
    end
end
if SCSH.LL_restriction > 0
    Guess(1,1) = Guess(1,1) - 0.1;
end
if SCSH.VL > 0 & Guess(4) > 0.15
    Guess(4) = Guess(4) - 0.1;
end
[System,Real,dev2,residual_pre,flag,func_output,err] = ...
    cycle_output_no_adj(...
        Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
        wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
        HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
        refri, Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, libname...
    );
end
```

```
function lsresidual = residual_maker(...
    Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
    Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
    EXV, Evap, SuctionLine, refri, Tc, Pc, hf, hv, rhov, DP_cond, ...
    DP_evap, libname ...
)
% This function is used to direct the residual of the
% cycle_calculation_function to the solver in Matlab
[residual System] = cycle_calculation(...
    Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
    Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
    EXV, Evap, SuctionLine, refri, Tc, Pc, hf, hv, rhov, DP_cond, ...
    DP_evap, libname ...
);
lsresidual = residual;
end
```

```
function System = System_definition()
% to define the system definition for assignment
System.mdot_r = 0;

System.Comp.Pout = 0;
System.Comp.hout = 0;
System.Comp.W = 0;
System.Comp.Pin = 0;
```

```
System.Comp.hin = 0;
System.Comp.m_status = 0;
System.Comp.W_status = 0;
System.Comp.HL_status = 0;

System.HotGas.Charge = 0;
System.HotGas.Pin = 0;
System.HotGas.Pout = 0;
System.HotGas.rho_avg = 0;
System.HotGas.Q_status = 0;
System.HotGas.ΔP_status = 0;

System.Cond.Tain = 0;
System.Cond.wain = 0;
System.Cond.Pin = 0;
System.Cond.hin = 0;
System.Cond.Pout = 0;
System.Cond.hout = 0;
System.Cond.Charge = 0;
System.Cond.OtherOutput = 0;
System.Cond.LumpedInput = 0;
System.Cond.FanPower = 0;
System.Cond.Taout = 0;
System.Cond.mdot_a = 0;
System.Cond.Q = 0;
System.Cond.rho_avg = 0;
```

```
35 System.Cond.status_Q = 0;
36 System.Cond.status_ΔP = 0;
37
38 System.LiquidLine.Charge = 0;
39 System.LiquidLine.hin = 0;
40 System.LiquidLine.Pin = 0;
41 System.LiquidLine.Pout = 0;
42 System.LiquidLine.hin = 0;
43 System.LiquidLine.rho_avg = 0;
44 System.LiquidLine.Q_status = 0;
45 System.LiquidLine.ΔP_status = 0;
46
47 System.EXV.Pin = 0;
48 System.EXV.hin = 0;
49 System.EXV.mdot_r = 0;
50 System.EXV.Pout = 0;
51 System.EXV.hout = 0;
52 System.EXV.OpeningStep = 0;
53 System.EXV.status = 0;
54 System.EXV.status_x = 0;
55
56 System.Evap.Tain = 0;
57 System.Evap.wain = 0;
58 System.Evap.Pin = 0;
59 System.Evap.hin = 0;
60 System.Evap.Pout = 0;
```

```
61 System.Evap.hout = 0;
62 System.Evap.Charge = 0;
63 System.Evap.OtherOutput = 0;
64 System.Evap.LumpedInput = 0;
65 System.Evap.FanPower = 0;
66 System.Evap.Taout = 0;
67 System.Evap.waout = 0;
68 System.Evap.mdot_a = 0;
69 System.Evap.Q = 0;
70 System.Evap.SHR = 0;
71 System.Evap.rho_avg = 0;
72 System.Evap.status_Q = 0;
73 System.Evap.status_SHR = 0;
74 System.Evap.status_ΔP = 0;
75
76 System.SuctionLine.Charge = 0;
77 System.SuctionLine.hin = 0;
78 System.SuctionLine.Pin = 0;
79 System.SuctionLine.Pout = 0;
80 System.SuctionLine.hout = 0;
81 System.SuctionLine.rho_avg = 0;
82 System.SuctionLine.Q_status = 0;
83 System.SuctionLine.ΔP_status = 0;
84
85 System.Charge = 0;
86 System.Charge_tuned = 0;
```

```
87
88 System.Charge_calculate.Cond.wsh = 0;
89 System.Charge_calculate.Cond.wtp = 0;
90 System.Charge_calculate.Cond.wsc = 0;
91 System.Charge_calculate.Cond.rho_sh = 0;
92 System.Charge_calculate.Cond.rho_tp = 0;
93 System.Charge_calculate.Cond.rho_sc = 0;
94 System.Charge_calculate.Evap.wsh = 0;
95 System.Charge_calculate.Evap.wtp = 0;
96 System.Charge_calculate.Evap.rho_sh = 0;
97 System.Charge_calculate.Evap.rho_tp = 0;
98 System.Charge_calculate.LiquidLine.rho_LL = 0;
99 System.Charge_calculate.status = 0;
100
101 System.CondenserSC = 0;
102 System.LiquidLineSC = 0;
103 System.EvaporatorSH = 0;
104 System.CompressorSH = 0;
105
106 System.Heatloss = 0;
107 System.ValveLeakage = 0;
108 System.LL_restriction = 0;
109 System.Non_condensable = 0;
110 System.ValveLeakFlow = 0; % new entry
111 System.LL_extra_ΔP = 0; % new entry
112 System.NC_amount = 0; % new entry
```

```
113 System.NC_Pressure = 0; % new entry
114 System.Accumulator_stat = 0;
115 System.VL_BP = 0;
116
117 System.SOLVED = 0;
118 end
```

```
1  function bool = slightdiff(A, B, Δ)
2  % if the temperature readings at A and B difference is smaller
3  % than 1.e-3
4  % return them to be equal
5  if abs(A - B) < Δ
6      bool = 1;
7  else
8      bool = 0;
9  end
10 end
```

```
1  function values = consistent(values, varargin)
2      % form consitent values in a column
3      m = length(values);
4      gp_index = zeros(m,1);
5      start_index = 1;
6      gp_num = 1;
```

```
    if nargin ≥ 2
        Δ = varargin{1};
    else
        Δ = 1.e-3;
    end
    while prod(gp_index) == 0
        if gp_index(start_index,1) == 0
            gp_index(start_index,1) = gp_num;
            for i = start_index+1:m
                if slightdiff(values(start_index,1), values(i,1), Δ)
                    values(i,1) = values(start_index,1);
                    gp_index(i,1) = gp_num;
                end
            end
            gp_num = gp_num + 1;
        end
        start_index = start_index+1;
    end
end
```

```
function Final = Unit_Conversion(Initial,INI,FIN)
% This function deals with unit conversion of doubles in
% Intial from unit INI to unit FIN

if strcmp(INI,'C') && strcmp(FIN,'K')
```

```
6       Final = Initial + 273.15;
7   elseif strcmp(INI,'K') && strcmp(FIN,'C')
8       Final = Initial - 273.15;
9   elseif strcmp(INI,'K') && strcmp(FIN,'F')
10      Final = Unit_Conversion(Initial - 273.15,'C','F');
11  elseif strcmp(INI,'F') && strcmp(FIN,'K')
12      Final = Unit_Conversion(Initial,'F','C')+273.15;
13  elseif strcmp(INI,'F') && strcmp(FIN,'C')
14      Final = (Initial-32).*5./9;
15  elseif strcmp(INI,'C') && strcmp(FIN,'F')
16      Final = Initial.*9./5+32;
17  elseif strcmp(INI,'K') && strcmp(FIN,'R')
18      Final = Initial.*9./5;
19  elseif strcmp(INI,'R') && strcmp(FIN,'K')
20      Final = Initial./9.*5;
21  elseif strcmp(INI,'bar') && strcmp(FIN,'kPa')
22      Final = Initial.*100;
23  elseif strcmp(INI,'kPa') && strcmp(FIN,'bar')
24      Final = Initial./100;
25  elseif strcmp(INI,'psi') && strcmp(FIN,'kPa')
26      Final = Initial.*6.8947572931684;
27  elseif strcmp(INI,'kPa') && strcmp(FIN,'psi')
28      Final = Initial./6.8947572931684;
29  elseif strcmp(INI,'cfm') && strcmp(FIN,'m3/s')
30      Final = Initial.*0.0004719474432;
31  elseif strcmp(INI,'m3/s') && strcmp(FIN,'cfm')
```

```
32      Final = Initial./0.0004719474432;
33  elseif strcmp(INI,'/min') && strcmp(FIN,'/s')
34      Final = Initial./60;
35  elseif strcmp(INI,'/min') && strcmp(FIN,'/h')
36      Final = Initial.*60;
37  elseif strcmp(INI,'/s') && strcmp(FIN,'/min')
38      Final = Initial.*60;
39  elseif strcmp(INI,'/s') && strcmp(FIN,'/h')
40      Final = Initial.*3600;
41  elseif strcmp(INI,'/h') && strcmp(FIN,'/s')
42      Final = Initial./3600;
43  elseif strcmp(INI,'Btu/lbm') && strcmp(FIN,'J/kg')
44      Final = Initial.*1055.056./0.4535923700000;
45  elseif strcmp(INI,'J/kg') && strcmp(FIN,'Btu/lbm')
46      Final = Initial./1055.056.*0.4535923700000;
47  elseif strcmp(INI,'inHg') && strcmp(FIN,'kPa')
48      Final = Initial.*3.3863881578947;
49  elseif strcmp(INI,'kPa') && strcmp(FIN,'inHg')
50      Final = Initial./3.3863881578947;
51  elseif strcmp(INI,'inH2O') && strcmp(FIN,'Pa')
52      Final = Initial.*249.0823945657895;
53  elseif strcmp(INI,'Pa') && strcmp(FIN,'inH2O')
54      Final = Initial./249.0823945657895;
55  elseif strcmp(INI,'kPa') && strcmp(FIN,'Pa')
56      Final = Initial./1000;
57  elseif strcmp(INI,'Pa') && strcmp(FIN,'kPa')
```

```
58      Final = Initial.*1000;
59  elseif strcmp(INI,'in') && strcmp(FIN,'m')
60      Final = Initial.*0.0254;
61  elseif strcmp(INI,'m') && strcmp(FIN,'in')
62      Final = Initial./0.0254;
63  elseif strcmp(INI,'L') && strcmp(FIN,'m3')
64      Final = Initial.*0.001;
65  elseif strcmp(INI,'m3') && strcmp(FIN,'L')
66      Final = Initial./0.001;
67  elseif strcmp(INI,'W') && strcmp(FIN,'Btu/h')
68      Final = Initial.*3.4121416331279;
69  elseif strcmp(INI,'Btu/h') && strcmp(FIN,'W')
70      Final = Initial./3.4121416331279;
71  elseif strcmp(INI,'kg') && strcmp(FIN,'lbm')
72      Final = Initial.*2.2046226218488;
73  elseif strcmp(INI,'lbm') && strcmp(FIN,'kg')
74      Final = Initial./2.2046226218488;
75  else
76      disp('Wrong input for Unit Conversion, abandoning.....');
77  end
```

APPENDIX N. PROGRAMS IN MATLAB LANGUAGE FOR PARAMETER ESTIMATION

N.1 Data Filtering prior Compressor Modeling

```
function System = Q_gain_observation_v025_main()

% main file for data filtering before compressor modeling

% define function and folder name
version_main = '025';
version_regress = '023';
version_plot = '002';
foldername = strcat('Q_gain_observation_v',version_main);
regressname = strcat('Q_gain_observation_v',version_regress);
plotname = strcat('Q_gain_observation_plot_v',version_plot);
removal_func = str2func([...
    '@(System, libname, version_number, Q_HL_limit)',...
    regressname,'(System, libname, version_number, Q_HL_limit)' ...
]);
plot_func = str2func(['@(System, libname, version_number)',...
    plotname,'(System, libname, version_number)']);
```

```
18 mkdir(foldername);
19 savefilename = strcat(foldername,'.mat');
20
21 % Set information
22 % re-file at this stage to avoid problems in parfor loop
23 % Boshen 3-ton R410A packaged FXO system (Scroll)
24 % (Shen_030_R410A_packaged_FXO_Scroll)
25 i = 1;
26 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
27 System(i).othername = System(i).name;
28 System(i).cond_name = System(i).name;
29 System(i).LLname = System(i).name;
30 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
31 System(i).foldername = 'Boshen';
32 System(i).worksheetname = 'SI (no_invalid_CA)';
33 System(i).refname = 'R410A';
34 System(i).Standard_Airflow_evap = 0;
35 System(i).Fan_Upstream_evap = 0;
36 System(i).Standard_Airflow_cond = 1;
37 System(i).Fan_Upstream_cond = 1;
38 System(i).Heating = 0;
39 System(i).Combined = 0;
40 System(i).Accumulator = 0;
41 System(i).Diameter = 0.00853;
42 System(i).Tube_Length = 0.6066;
43 System(i).Ntube = 66;
```

```
44 System(i).Pc = 4901;
45 System(i).Estimate_Dim = 0;
46 System(i).Rating = 3*12000*0.2932972222222;
47 System(i).Uncertainty.T_couple = 0.5;
48 System(i).Uncertainty.ΔP_noz = -9999;
49 System(i).Uncertainty.T_noz = -9999;
50 System(i).Uncertainty.m_a_rel = 0.91/100;
51 System(i).Uncertainty.T_d = 0.2;
52 System(i).Uncertainty.rh = -9999;
53 System(i).Uncertainty.P_r_abs = -9999;
54 System(i).Uncertainty.P_r_rel = 0.8/100;
55 System(i).Uncertainty.m_r_abs = -9999;
56 System(i).Uncertainty.m_r_rel = 0.6/100;
57 System(i).Uncertainty.W_abs = 10;
58 System(i).Uncertainty.W_rel = -9999;
59
60 % Boshen 5-ton R407C packaged FXO system (Scroll)
61 % (Shen_050_R407C_packaged_FXO_Scroll)
62 i = i + 1;
63 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
64 System(i).othername = System(i).name;
65 System(i).cond_name = System(i).name;
66 System(i).LLname = System(i).name;
67 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
68 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
69 System(i).worksheetname = 'SI (no_invalid_CA)';
```

```
System(i).refname = 'R407C';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Accumulator = 0;
System(i).Diameter = 0.00691896;
System(i).Tube_Length = 1.505204;
System(i).Ntube = 24*3;
System(i).Pc = 3786;
System(i).Estimate_Dim = 0;
System(i).Rating = 5*12000*0.2932972222222;
System(i).Uncertainty.T_couple = 0.5;
System(i).Uncertainty.ΔP_noz = -9999;
System(i).Uncertainty.T_noz = -9999;
System(i).Uncertainty.m_a_rel = 0.91/100;
System(i).Uncertainty.T_d = 0.2;
System(i).Uncertainty.rh = -9999;
System(i).Uncertainty.P_r_abs = -9999;
System(i).Uncertainty.P_r_rel = 0.8/100;
System(i).Uncertainty.m_r_abs = -9999;
System(i).Uncertainty.m_r_rel = 0.6/100;
System(i).Uncertainty.W_abs = 10;
System(i).Uncertainty.W_rel = -9999;
```

```
96
97  % Breuker 3-ton R22 packaged FXO system (Recip)
98  % (Breuker_030_R22_packaged_FXO_Recip)
99  i = i + 1;
100 System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
101 System(i).othername = System(i).name;
102 System(i).cond_name = System(i).name;
103 System(i).LLname = System(i).name;
104 System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
105 System(i).foldername = 'Breuker';
106 System(i).worksheetname = 'SI (no_invalid_CA)';
107 System(i).refname = 'R22.fld';
108 System(i).Standard_Airflow_evap = 0;
109 System(i).Fan_Upstream_evap = 0;
110 System(i).Standard_Airflow_cond = 1;
111 System(i).Fan_Upstream_cond = 1;
112 System(i).Heating = 0;
113 System(i).Combined = 0;
114 System(i).Accumulator = 0;
115 % assumed
116 System(i).Diameter = 0.00853;
117 System(i).Tube_Length = 0.6066;
118 System(i).Ntube = 66;
119 System(i).Pc = 4990;
120 System(i).Estimate_Dim = 1;
121 System(i).Rating = 3*12000*0.2932972222222;
```

```
System(i).Uncertainty.T_couple = 0.5;
System(i).Uncertainty.ΔP_noz = 7.5;
System(i).Uncertainty.T_noz = -9999;
System(i).Uncertainty.m_a_rel = 0.91/100;
System(i).Uncertainty.T_d = 0.2;
System(i).Uncertainty.rh = -9999;
System(i).Uncertainty.P_r_abs = -9999;
System(i).Uncertainty.P_r_rel = 0.8/100;
System(i).Uncertainty.m_r_abs = -9999;
System(i).Uncertainty.m_r_rel = 0.6/100;
System(i).Uncertainty.W_abs = 10;
System(i).Uncertainty.W_rel = -9999;

% Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
% (NIST_025_R410A_split_TXV_Scroll)
i = i + 1;
System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
System(i).foldername = 'NIST';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 1;
System(i).Fan_Upstream_evap = 1;
```

```
148 System(i).Standard_Airflow_cond = 1;
149 System(i).Fan_Upstream_cond = 1;
150 System(i).Heating = 0;
151 System(i).Combined = 0;
152 System(i).Accumulator = 0;
153 System(i).Diameter = 0.007747;
154 System(i).Tube_Length = 0.508;
155 System(i).Ntube = 60;
156 System(i).Pc = 4901;
157 System(i).Estimate_Dim = 0;
158 System(i).Rating = 2.5*12000*0.2932972222222;
159 System(i).Uncertainty.T_couple = 0.3;
160 System(i).Uncertainty.ΔP_noz = 1.0;
161 System(i).Uncertainty.T_noz = 0.3;
162 System(i).Uncertainty.m_a_rel = 1.67/100;
163 System(i).Uncertainty.T_d = 0.2;
164 System(i).Uncertainty.rh = -9999;
165 System(i).Uncertainty.P_r_abs = -9999;
166 System(i).Uncertainty.P_r_rel = 0.5/100;
167 System(i).Uncertainty.m_r_abs = -9999;
168 System(i).Uncertainty.m_r_rel = 1/100;
169 System(i).Uncertainty.W_abs = -9999;
170 System(i).Uncertainty.W_rel = 0.5/100;
171
172 % Harms 5-ton R22 packaged TXV system (Scroll)
173 % (Harms_050_R22_packaged_TXV_Scroll_Heating)
```

```
174 i = i + 1;
175 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
176 System(i).othername = System(i).name;
177 System(i).cond_name = System(i).name;
178 System(i).LLname = System(i).name;
179 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
180 System(i).foldername = 'Harms_050tonR22PackagedTXV';
181 System(i).worksheetname = 'SI (no_invalid_CA)';
182 System(i).refname = 'R22.fld';
183 System(i).Standard_Airflow_evap = 0;
184 System(i).Fan_Upstream_evap = 0;
185 System(i).Standard_Airflow_cond = 1;
186 System(i).Fan_Upstream_cond = 1;
187 System(i).Heating = 0;
188 System(i).Combined = 0;
189 System(i).Accumulator = 0;
190 System(i).Diameter = 0.00889;
191 System(i).Tube_Length = 0.617;
192 System(i).Ntube = 32*4;
193 System(i).Pc = 4990;
194 System(i).Estimate_Dim = 0;
195 System(i).Rating = 5*12000*0.2932972222222;
196 System(i).Uncertainty.T_couple = 0.2;
197 System(i).Uncertainty.ΔP_noz = 7.5;
198 System(i).Uncertainty.T_noz = 0.26;
199 System(i).Uncertainty.m_a_rel = 0.91/100;
```

```
System(i).Uncertainty.T_d = 0.2;
System(i).Uncertainty.rh = -9999;
System(i).Uncertainty.P_r_abs = 20;
System(i).Uncertainty.P_r_rel = -9999;
System(i).Uncertainty.m_r_abs = 1.45/3600;
System(i).Uncertainty.m_r_rel = -9999;
System(i).Uncertainty.W_abs = 11;
System(i).Uncertainty.W_rel = -9999;

% Kim 3-ton R410A split TXV system (Scroll) (HCA3)
% (HCA3_030_R410A_split_TXV_Scroll)
i = i + 1;
System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
System(i).foldername = 'Kim-HCA3';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
```

```
System(i).Accumulator = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4901;
System(i).Estimate_Dim = 1;
System(i).Rating = 3*12000*0.2932972222222;
System(i).Uncertainty.T_couple = 1.0;
System(i).Uncertainty.ΔP_noz = -9999;
System(i).Uncertainty.T_noz = -9999;
System(i).Uncertainty.m_a_rel = 10/100;
System(i).Uncertainty.T_d = 0.2;
System(i).Uncertainty.rh = -9999;
System(i).Uncertainty.P_r_abs = -9999;
System(i).Uncertainty.P_r_rel = 1.0/100;
System(i).Uncertainty.m_r_abs = -9999;
System(i).Uncertainty.m_r_rel = 0.5/100;
System(i).Uncertainty.W_abs = -9999;
System(i).Uncertainty.W_rel = 0.2/100;

% Boshen 3-ton R410A split FXO system (Recip)
% (Shen_030_R410A_split_FXO_Recip)
% for compressor modeling, merged with TXV system data
i = i + 1;
System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
```

```
252 System(i).othername = System(i).name;
253 System(i).cond_name = System(i).name;
254 System(i).LLname = System(i).name;
255 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
256 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
257 System(i).worksheetname = 'SI (no_invalid_CA)';
258 System(i).refname = 'R410A';
259 System(i).Standard_Airflow_evap = 0;
260 System(i).Fan_Upstream_evap = 0;
261 System(i).Standard_Airflow_cond = 1;
262 System(i).Fan_Upstream_cond = 1;
263 System(i).Heating = 0;
264 System(i).Combined = 0;
265 System(i).Accumulator = 0;
266 System(i).Diameter = 0.00849;
267 System(i).Tube_Length = 0.452;
268 System(i).Ntube = 28*3;
269 System(i).Pc = 4901;
270 System(i).Estimate_Dim = 0;
271 System(i).Rating = 3*12000*0.2932972222222;
272 System(i).Uncertainty.T_couple = 0.5;
273 System(i).Uncertainty.ΔP_noz = -9999;
274 System(i).Uncertainty.T_noz = -9999;
275 System(i).Uncertainty.m_a_rel = 0.91/100;
276 System(i).Uncertainty.T_d = 0.2;
277 System(i).Uncertainty.rh = -9999;
```

```
278 System(i).Uncertainty.P_r_abs = -9999;
279 System(i).Uncertainty.P_r_rel = 0.8/100;
280 System(i).Uncertainty.m_r_abs = -9999;
281 System(i).Uncertainty.m_r_rel = 0.6/100;
282 System(i).Uncertainty.W_abs = 10;
283 System(i).Uncertainty.W_rel = -9999;
284
285 % Boshen 3-ton R410A split TXV system (Recip)
286 % (Shen_030_R410A_split_TXV_Recip)
287 % for compressor modeling, merged with TXV system data
288 i = i + 1;
289 System(i).name = 'Shen_030_R410A_split_TXV_Recip';
290 System(i).othername = System(i).name;
291 System(i).cond_name = System(i).name;
292 System(i).LLname = System(i).name;
293 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitTXV.xls';
294 System(i).foldername = 'Boshen_3tonR410ASplitTXV';
295 System(i).worksheetname = 'SI (no_invalid_CA)';
296 System(i).refname = 'R410A';
297 System(i).Standard_Airflow_evap = 0;
298 System(i).Fan_Upstream_evap = 0;
299 System(i).Standard_Airflow_cond = 1;
300 System(i).Fan_Upstream_cond = 1;
301 System(i).Heating = 0;
302 System(i).Combined = 0;
303 System(i).Accumulator = 0;
```

```
304 System(i).Diameter = 0.00849;
305 System(i).Tube_Length = 0.452;
306 System(i).Ntube = 28*3;
307 System(i).Pc = 4901;
308 System(i).Estimate_Dim = 0;
309 System(i).Rating = 3*12000*0.2932972222222;
310 System(i).Uncertainty.T_couple = 0.5;
311 System(i).Uncertainty.ΔP_noz = -9999;
312 System(i).Uncertainty.T_noz = -9999;
313 System(i).Uncertainty.m_a_rel = 0.91/100;
314 System(i).Uncertainty.T_d = 0.2;
315 System(i).Uncertainty.rh = -9999;
316 System(i).Uncertainty.P_r_abs = -9999;
317 System(i).Uncertainty.P_r_rel = 0.8/100;
318 System(i).Uncertainty.m_r_abs = -9999;
319 System(i).Uncertainty.m_r_rel = 0.6/100;
320 System(i).Uncertainty.W_abs = 10;
321 System(i).Uncertainty.W_rel = -9999;
322
323 % Boshen 3-ton R410A split TXV system (Recip)
324 % (Shen_030_R410A_split_TXV_Recip)
325 % for compressor modeling, merged with TXV system data
326 i = i + 1;
327 System(i).name = 'Shen_030_R410A_split_FXO_Recip';
328 System(i).othername = System(i).name;
329 System(i).cond_name = System(i).name;
```

```
330 System(i).LLname = System(i).name;
331 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO.xls';
332 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
333 System(i).worksheetname = 'SI (no_invalid_CA)';
334 System(i).refname = 'R410A';
335 System(i).Standard_Airflow_evap = 0;
336 System(i).Fan_Upstream_evap = 0;
337 System(i).Standard_Airflow_cond = 1;
338 System(i).Fan_Upstream_cond = 1;
339 System(i).Heating = 0;
340 System(i).Combined = 0;
341 System(i).Accumulator = 0;
342 System(i).Diameter = 0.00849;
343 System(i).Tube_Length = 0.452;
344 System(i).Ntube = 28*3;
345 System(i).Pc = 4901;
346 System(i).Estimate_Dim = 0;
347 System(i).Rating = 3*12000*0.2932972222222;
348 System(i).Uncertainty.T_couple = 0.5;
349 System(i).Uncertainty.ΔP_noz = -9999;
350 System(i).Uncertainty.T_noz = -9999;
351 System(i).Uncertainty.m_a_rel = 0.91/100;
352 System(i).Uncertainty.T_d = 0.2;
353 System(i).Uncertainty.rh = -9999;
354 System(i).Uncertainty.P_r_abs = -9999;
355 System(i).Uncertainty.P_r_rel = 0.8/100;
```

```
356 System(i).Uncertainty.m_r_abs = -9999;
357 System(i).Uncertainty.m_r_rel = 0.6/100;
358 System(i).Uncertainty.W_abs = 10;
359 System(i).Uncertainty.W_rel = -9999;
360
361 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
362 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
363 i = i + 1;
364 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
365 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
366 System(i).cond_name = System(i).name;
367 System(i).LLname = System(i).name;
368 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
369 System(i).foldername = 'Kim-HCA3-Heating';
370 System(i).worksheetname = 'SI (no_invalid_CA)';
371 System(i).refname = 'R410A';
372 System(i).Standard_Airflow_evap = 1;
373 System(i).Fan_Upstream_evap = 1;
374 System(i).Standard_Airflow_cond = 1;
375 System(i).Fan_Upstream_cond = 1;
376 System(i).Heating = 1;
377 System(i).Combined = 0;
378 System(i).Accumulator = 0;
379 % assumed
380 System(i).Diameter = 0.00849;
381 System(i).Tube_Length = 2.255;
```

```
382 System(i).Ntube = 32;
383 System(i).Pc = 4901;
384 System(i).Estimate_Dim = 1;
385 System(i).Rating = 3*12000*0.2932972222222;
386 System(i).Uncertainty.T_couple = 1.0;
387 System(i).Uncertainty.ΔP_noz = -9999;
388 System(i).Uncertainty.T_noz = -9999;
389 System(i).Uncertainty.m_a_rel = 10/100;
390 System(i).Uncertainty.T_d = 0.2;
391 System(i).Uncertainty.rh = -9999;
392 System(i).Uncertainty.P_r_abs = -9999;
393 System(i).Uncertainty.P_r_rel = 1.0/100;
394 System(i).Uncertainty.m_r_abs = -9999;
395 System(i).Uncertainty.m_r_rel = 0.5/100;
396 System(i).Uncertainty.W_abs = -9999;
397 System(i).Uncertainty.W_rel = 0.2/100;
398
399 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
400 % (YSA_030_R22_split_TXV_Scroll)
401 i = i + 1;
402 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
403 System(i).othername = System(i).name;
404 System(i).cond_name = System(i).name;
405 System(i).LLname = System(i).name;
406 System(i).filename = 'Kim_Cycle_data_R22_YSA_v02.xlsx';
407 System(i).foldername = 'Kim-YSA';
```

```
408 System(i).worksheetname = 'SI (no_invalid_CA)';
409 System(i).refname = 'R22.fld';
410 System(i).Standard_Airflow_evap = 0;
411 System(i).Fan_Upstream_evap = 1;
412 System(i).Standard_Airflow_cond = 1;
413 System(i).Fan_Upstream_cond = 1;
414 System(i).Heating = 0;
415 System(i).Combined = 0;
416 System(i).Accumulator = 1;
417 % assumed
418 System(i).Diameter = 0.00849;
419 System(i).Tube_Length = 2.255;
420 System(i).Ntube = 32;
421 System(i).Pc = 4990;
422 System(i).Estimate_Dim = 1;
423 System(i).Rating = 3*12000*0.2932972222222;
424 System(i).Uncertainty.T_couple = 1.0;
425 System(i).Uncertainty.ΔP_noz = -9999;
426 System(i).Uncertainty.T_noz = -9999;
427 System(i).Uncertainty.m_a_rel = 10/100;
428 System(i).Uncertainty.T_d = 0.2;
429 System(i).Uncertainty.rh = -9999;
430 System(i).Uncertainty.P_r_abs = -9999;
431 System(i).Uncertainty.P_r_rel = 1.0/100;
432 System(i).Uncertainty.m_r_abs = -9999;
433 System(i).Uncertainty.m_r_rel = 0.5/100;
```

```
434 System(i).Uncertainty.W_abs = -9999;
435 System(i).Uncertainty.W_rel = 0.2/100;
436
437 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
438 % (YSA_030_R22_split_TXV_Scroll_Heating)
439 i = i + 1;
440 System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
441 System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
442 System(i).cond_name = System(i).name;
443 System(i).LLname = System(i).name;
444 System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
445 System(i).foldername = 'Kim-YSA-Heating';
446 System(i).worksheetname = 'SI (no_invalid_CA)';
447 System(i).refname = 'R22.fld';
448 System(i).Standard_Airflow_evap = 1;
449 System(i).Fan_Upstream_evap = 1;
450 System(i).Standard_Airflow_cond = 1;
451 System(i).Fan_Upstream_cond = 1;
452 System(i).Heating = 1;
453 System(i).Combined = 0;
454 System(i).Accumulator = 1;
455 % assumed
456 System(i).Diameter = 0.00849;
457 System(i).Tube_Length = 2.255;
458 System(i).Ntube = 32;
459 System(i).Pc = 4990;
```

```
460 System(i).Estimate_Dim = 1;
461 System(i).Rating = 3*12000*0.2932972222222;
462 System(i).Uncertainty.T_couple = 1.0;
463 System(i).Uncertainty.ΔP_noz = -9999;
464 System(i).Uncertainty.T_noz = -9999;
465 System(i).Uncertainty.m_a_rel = 10/100;
466 System(i).Uncertainty.T_d = 0.2;
467 System(i).Uncertainty.rh = -9999;
468 System(i).Uncertainty.P_r_abs = -9999;
469 System(i).Uncertainty.P_r_rel = 1.0/100;
470 System(i).Uncertainty.m_r_abs = -9999;
471 System(i).Uncertainty.m_r_rel = 0.5/100;
472 System(i).Uncertainty.W_abs = -9999;
473 System(i).Uncertainty.W_rel = 0.2/100;
474
475 % Kim 3-ton R22 split TXV system (Scroll) (YKC)
476 % (YKC_030_R22_split_TXV_Scroll)
477 i = i + 1;
478 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
479 System(i).othername = System(i).name;
480 System(i).cond_name = System(i).name;
481 System(i).LLname = System(i).name;
482 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
483 System(i).foldername = 'Kim-YKC';
484 System(i).worksheetname = 'SI (no_invalid_CA)';
485 System(i).refname = 'R22.fld';
```

```
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Accumulator = 1;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).Estimate_Dim = 1;
System(i).Rating = 3*12000*0.2932972222222;
System(i).Uncertainty.T_couple = 1.0;
System(i).Uncertainty.ΔP_noz = -9999;
System(i).Uncertainty.T_noz = -9999;
System(i).Uncertainty.m_a_rel = 10/100;
System(i).Uncertainty.T_d = 0.2;
System(i).Uncertainty.rh = -9999;
System(i).Uncertainty.P_r_abs = -9999;
System(i).Uncertainty.P_r_rel = 1.0/100;
System(i).Uncertainty.m_r_abs = -9999;
System(i).Uncertainty.m_r_rel = 0.5/100;
System(i).Uncertainty.W_abs = -9999;
System(i).Uncertainty.W_rel = 0.2/100;
```

```
512
513 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
514 % (TM_040_R410A_packaged_TXV_Recip)
515 i = i + 1;
516 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
517 System(i).othername = System(i).name;
518 System(i).cond_name = System(i).name;
519 System(i).LLname = System(i).name;
520 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV_v03.xlsx';
521 System(i).foldername = 'Kim-TM';
522 System(i).worksheetname = 'SI (no_LL)';
523 System(i).refname = 'R410A.ppf';
524 System(i).Standard_Airflow_evap = 0;
525 System(i).Fan_Upstream_evap = 0;
526 System(i).Standard_Airflow_cond = 0;
527 System(i).Fan_Upstream_cond = 0;
528 System(i).Heating = 0;
529 System(i).Combined = 0;
530 System(i).Accumulator = 0;
531 % assumed
532 System(i).Diameter = 0.00849;
533 System(i).Tube_Length = 2.255;
534 System(i).Ntube = 32;
535 System(i).Pc = 4990;
536 System(i).Estimate_Dim = 1;
537 System(i).Rating = 3*12000*0.2932972222222;
```

```
538 System(i).Uncertainty.T_couple = 1.0;
539 System(i).Uncertainty.ΔP_noz = -9999;
540 System(i).Uncertainty.T_noz = -9999;
541 System(i).Uncertainty.m_a_rel = 10/100;
542 System(i).Uncertainty.T_d = 0.2;
543 System(i).Uncertainty.rh = -9999;
544 System(i).Uncertainty.P_r_abs = -9999;
545 System(i).Uncertainty.P_r_rel = 1.0/100;
546 System(i).Uncertainty.m_r_abs = -9999;
547 System(i).Uncertainty.m_r_rel = 0.5/100;
548 System(i).Uncertainty.W_abs = -9999;
549 System(i).Uncertainty.W_rel = 0.2/100;
550
551 % other settings for storage
552 m = length(System);
553 for i = 1:m
554     System(i).mainfoldername = foldername;
555 end
556
557 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
558 libname ='lib'; % Name of library
559
560 % start calculation
561 try
562     parfor i = 1:m
563         System(i) = 1; % a statement leads to error
```

```
564     end
565 catch err
566 end
567 for i = 1:m
568     error_statement(i) = err;
569 end
570 no_plot = zeros(m,1);
571 Q_HL_array = {};
572 Q_HL_limit = -9999;
573 % parfor i = 1:m
574 for i = 1:m
575     try
576         disp(['Starting to run ',System(i).name]);
577         [...
578             Cond_nf_Vdot_a, Evap_nf_Vdot_a,Q_HL, Q_evap, W_comp, ...
579             Q_cond, index_removed, air_index, uncertainty, ...
580             Q_HL_ratio ...
581         ] = removal_func(...
582             System(i), libname, version_main, Q_HL_limit ...
583         );
584         System(i).Cond_nf_Vdot_a = Cond_nf_Vdot_a;
585         System(i).Evap_nf_Vdot_a = Evap_nf_Vdot_a;
586         System(i).Q_HL = Q_HL;
587         System(i).Q_evap = Q_evap;
588         System(i).W_comp = W_comp;
589         System(i).Q_cond = Q_cond;
```

```
590         System(i).index_removed = index_removed;
591         System(i).air_index = air_index;
592         System(i).uncertainty = uncertainty;
593         System(i).Q_HL_ratio = Q_HL_ratio;
594         Q_HL_array(i,1) = {Q_HL_ratio};
595         disp(['Finishing simulating ',System(i).name]);
596     catch err
597         disp(['Problem found in system ',System(i).name]);
598         no_plot(i,1) = 1;
599         error_statement(i) = err;
600     end
601 end
602 save(savefilename);
603
604 % plot cannot be done in parfor. Do it in a for loop
605 for i = 1:m
606 % for i = 5:5
607     if no_plot(i,1) == 0
608         disp(['Starting to plot ',System(i).name]);
609         plot_func(System(i), libname, version_main);
610         disp(['Finishing plotting ',System(i).name]);
611     end
612 end
613
614 % printing uncertainty
615 % for i = 5:5
```

```
616 header = {'System','Evaporator refrigerant-side heat transfer [%]',...
617     'Evaporator air-side heat transfer [%]','SHR [%]',...
618     'Condenser refrigerant-side heat transfer [%]',...
619     'Condenser air mass flow rate [%]','Superheat [K]',...
620     'Subcooling [K]'};
621 for i = 1:m
622     if no_plot(i,1) == 0
623         try
624             Q_evap_r_uncertainty = mean(...
625                 System(i).uncertainty.Q_evap_r(find(...
626                     System(i).uncertainty.Q_evap_r>0 ...
627                 ))...
628             );
629         catch err
630             clear err
631             Q_evap_r_uncertainty = -9999;
632         end
633         Q_evap_a_uncertainty = mean(...
634             System(i).uncertainty.Q_evap_a(find(...
635                 System(i).uncertainty.Q_evap_a>0 ...
636             )) ...
637         );
638         SHR_uncertainty = mean(...
639             System(i).uncertainty.SHR(System(i).uncertainty.SHR>0) ...
640         );
641         Q_cond_r_uncertainty = mean(System(i).uncertainty.Q_cond_r(...
```

```
642             find(System(i).uncertainty.Q_cond_r>0)) ...
643         );
644         m_cond_a_uncertainty = mean(System(i).uncertainty.m_cond_a(...
645             find(System(i).uncertainty.m_cond_a>0)...
646         ));
647         SH_uncertainty = mean(System(i).uncertainty.SH(...
648             find(System(i).uncertainty.SH>0) ...
649         ));
650         SC_uncertainty = mean(System(i).uncertainty.SC(...
651             find(System(i).uncertainty.SC>0))...
652         );
653         para_cell = [System(i).name,num2cell([Q_evap_r_uncertainty ...
654             Q_evap_a_uncertainty ...
655             SHR_uncertainty ...
656             Q_cond_r_uncertainty ...
657             m_cond_a_uncertainty ...
658             SH_uncertainty ...
659             SC_uncertainty])];
660         header(i+1,:) = para_cell;
661     end
662 end
663 xlsxfilename = [mfilename,'_uncertainty.xlsx'];
664 xlswrite(xlsxfilename,header,'uncertainty');
665
666 % plot histogram
667 set(0,'DefaultFigureWindowStyle','docked');
```

```
668 f_system_HL = figure;
669 set(gca,'Fontsize',16);
670 Q_HL_ratio = cell2mat(Q_HL_array);
671 hist(Q_HL_ratio);
672 xlabel('Ratio of heat loss to evaporator heat transfer rate');
673 ylabel('Number of cases');
674 set(gcf,'PaperPositionMode','auto');
675 print('-dtiff',f_system_HL,strcat(...
676     mfilename,'_Q_HL_histogram_','.tif')...
677 );
678 % Q_HL_limit = mean(Q_HL_ratio) - 1.96*std(Q_HL_ratio); % 95%
679 % Q_HL_limit = -0.01;
680 Q_HL_limit = -0.08; % 97.5%
681 disp(['Eliminate Q_HL_ratio below',' ',num2str(Q_HL_limit)]);
682 % repeat to eliminate the data with appropriate limit
683 parfor i = 1:m
684     if no_plot(i,1) == 0
685         disp(['Starting to store ',System(i).name]);
686         removal_func(System(i), libname, version_main, Q_HL_limit);
687         disp(['Finishing storing ',System(i).name]);
688     end
689 end
690
691 % plot cumulative frequency diagram
692 set(0,'DefaultFigureWindowStyle','docked');
693 [freq, x] = ecdf(Q_HL_ratio);
```

```
694 freq = freq*length(Q_HL_ratio);
695 f_system_HL = figure;
696 set(gca,'Fontsize',16);
697 h = plot(x, freq);
698 set(h,'LineWidth',5);
699 xlabel('Ratio of heat loss to evaporator heat transfer rate');
700 ylabel('Cumulative number of cases');
701 set(gcf,'PaperPositionMode','auto');
702 print('-dtiff',f_system_HL,strcat(mfilename,'_Q_HL_cfreq_','.tif'));
703
704 save(savefilename);
705 movefile(savefilename,strcat(foldername,'\'));
706 movefile(strcat(mfilename,'_Q_HL_histogram_','.tif'),strcat(...
707     foldername,'\' ...
708 ));
709 movefile(strcat(mfilename,'_Q_HL_cfreq_','.tif'),strcat(...
710     foldername,'\') ...
711 );
712 copyfile(xlsxfilename,strcat(foldername,'\'));
713
714 try
715     unloadlibrary lib;
716 end
```

```
1 function [...
```

```
    Cond_nf_Vdot_a, Evap_nf_Vdot_a, Q_HL, Q_evap_r, W_comp, ...
    Q_cond_r, index_removed, air_index, uncertainty, ...
    Q_HL_ratio ...
] = Q_gain_observation_v023(...
    System, libname, version_number, Q_HL_limit ...
)

% Function to filter data according to condenser airflow and overall
% energy balance before compressor modeling

% open results
filename = System.filename;
worksheetname = System.worksheetname;
refri = System.refname;
Standard_Airflow_evap = System.Standard_Airflow_evap;
Ind_Fan_Upstream_evap = System.Fan_Upstream_evap;
Standard_Airflow_cond = System.Standard_Airflow_cond;
Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
version = strcat(System.name,'_',version_number);
savefilename = strcat('Q_gain_observation_v',version,'.mat');
foldername = strcat(strcat(...
    pwd,'\',System.mainfoldername,'\' ...
),strcat('Q_gain_observation_v',version));
if Q_HL_limit == -9999
    if isfield(System,'Combined')
        [data code] = xlsread(strcat(...
```

```
28              pwd,'\',System.foldername,'\',filename ...
29          ),worksheetname,'a2:aj65536');
30      else
31          [data code] = xlsread(strcat(...
32              pwd,'\',System.foldername,'\',filename ...
33          ),worksheetname,'a2:ai65536');
34      end
35      [ref_II fault] = xlsread(strcat(...
36          pwd,'\',System.foldername,'\',filename ...
37      ),worksheetname,'ak2:ak65536');
38      if ¬exist(foldername,'dir')
39          mkdir(foldername);
40      end
41      [pp qq] = size(data);
42
43      i = 1;
44      cell_CA = strfind(fault,'CA');
45      if isempty(cell_CA)
46          cell_CA = cell(pp,1);
47      end
48      cell_NC = strfind(fault,'NC');
49      if isempty(cell_NC)
50          cell_NC = cell(pp,1);
51      end
52      cell_VL = strfind(fault,'VL');
53      if isempty(cell_VL)
```

```
54          cell_VL = cell(pp,1);
55      end
56      cell_LL = strfind(fault,'LL');
57      if isempty(cell_LL)
58          cell_LL = cell(pp,1);
59      end
60      while i ≤ pp
61          if ¬isempty(cell2mat(cell_NC(i))) || ...
62              ¬isempty(cell2mat(cell_VL(i))) || ...
63              ¬isempty(cell2mat(cell_LL(i)))
64                  data(i,:) = [];
65                  code(i,:) = [];
66                  cell_CA(i,:) = [];
67                  fault(i,:) = [];
68                  i = i - 1;
69                  pp = pp - 1;
70          elseif ¬isempty(cell2mat(cell_CA(i)))
71              SC_checker = ...
72                  propertyRH('T','P',data(i,4),'Q',0,refri) - data(i,12);
73              SC_checker_II = ...
74                  propertyRH('T','P',data(i,5),'Q',0,refri) - data(i,13);
75              if SC_checker < 3 || SC_checker_II < 3
76                  data(i,:) = [];
77                  code(i,:) = [];
78                  cell_CA(i,:) = [];
79                  fault(i,:) = [];
```

```
80                  i = i - 1;
81                  pp = pp - 1;
82              end
83          end
84          i = i + 1;
85      end
86      [pp qq] = size(data);
87      air_index = zeros(pp,2);
88
89
90      % set relative uncertainty vectors
91      uncertainty.Q_evap_a = ones(pp,1)*-9999;
92      uncertainty.Q_evap_r = ones(pp,1)*-9999;
93      uncertainty.SHR = ones(pp,1)*-9999;
94      uncertainty.Q_cond_r = ones(pp,1)*-9999;
95      uncertainty.m_cond_a = ones(pp,1)*-9999;
96      uncertainty.SH = ones(pp,1)*-9999; % absolute
97      uncertainty.SC = ones(pp,1)*-9999; % absolute
98
99      ΔT = System.Uncertainty.T_couple;
100     ΔP_noz = System.Uncertainty.ΔP_noz;
101     ΔT_noz = System.Uncertainty.T_noz;
102     Δ_m_a_rel = System.Uncertainty.m_a_rel;
103     ΔT_d = System.Uncertainty.T_d;
104     ΔRH = System.Uncertainty.rh;
105     ΔP_r_abs = System.Uncertainty.P_r_abs;
```

```
106     ΔP_r_rel = System.Uncertainty.P_r_rel;
107     Δm_r_abs = System.Uncertainty.m_r_abs;
108     Δm_r_rel = System.Uncertainty.m_r_rel;
109     ΔW_abs = System.Uncertainty.W_abs;
110     ΔW_rel = System.Uncertainty.W_rel;
111
112     % calculate the condenser airflow rate first at non-faulted
113     % condition
114     i = 1;
115     Q_cond_r = zeros(pp,1);
116     while(i≤pp)
117         % avoid incomplete dataset
118         try
119             SC_cond(i,1) = ...
120                 propertyRH('T','P',data(i,4),'Q',0,refri) - ...
121                     data(i,12);
122             try
123                 SC_TXV(i,1) = propertyRH(...
124                     'T','P',data(i,5),'Q',0,refri ...
125                 ) - data(i,13);
126             catch err
127                 clear err;
128                 SC_TXV(i,1) = SC_cond(i,1);
129             end
130             if ΔP_r_rel > 0
131                 ΔP_r_abs = ΔP_r_rel*data(i,4);
```

```
132             end
133             SC_ΔP = (...
134                 propertyRH('T','P',data(i,4)+0.001,'Q',0,refri) -...
135                 propertyRH('T','P',data(i,4),'Q',0,refri) ...
136             )/0.001*ΔP_r_abs;
137             SC_ΔT = -1*ΔT;
138             uncertainty.SC(i,1) = sqrt(SC_ΔP^2+SC_ΔT^2);
139         catch err
140             SC_TXV(i,1) = propertyRH(...
141                 'T','P',data(i,5),'Q',0,refri ...
142             ) - data(i,13);
143             SC_cond(i,1) = SC_TXV(i,1);
144             clear err;
145         end
146         SH_comp(i,1) = data(i,9) - ...
147             propertyRH('T','P',data(i,1),'Q',1,refri);
148         SH_evap(i,1) = data(i,16) - ...
149             propertyRH('T','P',data(i,8),'Q',1,refri);
150         if ΔP_r_rel > 0
151             ΔP_r_abs = ΔP_r_rel*data(i,1);
152         end
153         SH_ΔP = -(propertyRH(...
154             'T','P',data(i,1)+0.001,'Q',1,refri ...
155         )-propertyRH('T','P',data(i,1),'Q',1,refri))/0.001*ΔP_r_abs;
156         SH_ΔT = 1*ΔT;
157         uncertainty.SH(i,1) = sqrt(SH_ΔP^2+SH_ΔT^2);
```

```
158
159         % filter for probelmatic upstream measurement
160         h_in_HG = propertyRH('H','T',data(i,10),'P',data(i,2),refri);
161         h_out_HG = propertyRH('H','T',data(i,11),'P',data(i,3),refri);
162         if System.Heating==0
163             Tamb = data(i,21);
164         else
165             Tamb = data(i,17);
166         end
167         if h_in_HG - h_out_HG ≠ 0
168             if (data(i,10)-Tamb)/(h_in_HG - h_out_HG) < 0
169                 h_out_HG = h_in_HG;
170                 data(i,11) = ...
171                     propertyRH('T','P',data(i,3),'H',h_out_HG,refri);
172             end
173         end
174
175         if (SC_cond(i,1)>3 && SC_TXV(i,1)>3 && data(i,23) > 0)
176             Q_cond_r(i,1) = data(i,23)*(...
177                 propertyRH('H','T',data(i,11),'P',data(i,3),refri) -...
178                 propertyRH('H','T',data(i,12),'P',data(i,4),refri) ...
179             );
180             if Δm_r_rel > 0
181                 Δm_r_abs = Δm_r_rel*data(i,23);
182             end
183             Q_cond_r_Δ(1,1) = Q_cond_r(i,1)/data(i,23)*Δm_r_abs;
```

```
184             if ΔP_r_rel > 0
185                 ΔP_r_abs = ΔP_r_rel*data(i,3);
186             end
187             Q_cond_r_Δ(2,1) = data(i,23)*(...
188                 propertyRH(...
189                     'H','T',data(i,11),'P',data(i,3)+0.01,refri ...
190                 )-propertyRH(...
191                     'H','T',data(i,11),'P',data(i,3),refri ...
192                 )...
193             )/0.01*ΔP_r_abs;
194             if ΔP_r_rel > 0
195                 ΔP_r_abs = ΔP_r_rel*data(i,4);
196             end
197             Q_cond_r_Δ(3,1) = -data(i,23)*(...
198                 propertyRH(...
199                     'H','T',data(i,12),'P',data(i,4),refri ...
200                 )-propertyRH(...
201                     'H','T',data(i,12),'P',data(i,4)-0.01,refri ...
202                 )...
203             )/0.01*ΔP_r_abs;
204             Q_cond_r_Δ(4,1) = data(i,23)*(...
205                 propertyRH(...
206                     'H','T',data(i,11)+0.01,'P',data(i,3),refri ...
207                 )-propertyRH(...
208                     'H','T',data(i,11),'P',data(i,3),refri...
209                 )...
```

```
210             )/0.01*ΔT;
211             Q_cond_r_Δ(5,1) = -data(i,23)*(...
212                 propertyRH(...
213                     'H','T',data(i,12),'P',data(i,4),refri ...
214                 )-propertyRH(...
215                     'H','T',data(i,12)-0.01,'P',data(i,4),refri ...
216                 )...
217             )/0.01*ΔT;
218             uncertainty.Q_cond_r(i,1) = sqrt(sum(...
219                 Q_cond_r_Δ.^2 ...
220             ))/Q_cond_r(i,1);
221
222             airinlet = calllib(...
223                 libname, 'HumAir_DLL', data(i,21), ...
224                 data(i,32), 1, 230, zeros(1,5) ...
225             );
226             rho_a = (1)/airinlet(5);
227             airinlet_ΔT = calllib(...
228                 libname, 'HumAir_DLL', data(i,21)+0.01, ...
229                 data(i,32), 1, 230, zeros(1,5) ...
230             );
231             Δ_rho_a = (1/airinlet_ΔT(5)-rho_a)/0.01*ΔT;
232             cp = 1006;
233             if Ind_Fan_Upstream_cond
234                 data(i,29) = (Q_cond_r(i,1)+data(i,26))/cp/(...
235                     data(i,22)-data(i,21) ...
```

```
236                 )/rho_a;
237                 Δ_m_a(1,1) = 1/cp/(...
238                     data(i,22)-data(i,21) ...
239                 )/rho_a*uncertainty.Q_cond_r(i,1)*Q_cond_r(i,1);
240                 Δ_m_a(2,1) = -data(i,29)/(...
241                     data(i,22)-data(i,21)...
242                 )*ΔT;
243                 Δ_m_a(3,1) = -data(i,29)/(...
244                     data(i,22)-data(i,21)...
245                 )*ΔT;
246                 Δ_m_a(4,1) = -data(i,29)/rho_a*Δ_rho_a;
247                 if ΔW_rel > 0
248                     ΔW_abs = data(i,26)*ΔW_rel;
249                 end
250                 Δ_m_a(5,1) = 1/cp/(...
251                     data(i,22)-data(i,21)...
252                 )/rho_a*ΔW_abs;
253                 uncertainty.m_cond_a(i,1) = sqrt(sum(...
254                     Δ_m_a.^2...
255                 ))/data(i,29);
256             else
257                 data(i,29) = (Q_cond_r(i,1))/cp/(...
258                     data(i,22)-data(i,21)...
259                 )/rho_a; % re-calculate airflow in m^3/s
260                 Δ_m_a(1,1) = data(i,29)*uncertainty.Q_cond_r(i,1);
261                 Δ_m_a(2,1) = -data(i,29)/(...
```

```
262                     data(i,22)-data(i,21)...
263                 )*ΔT;
264                 Δ_m_a(3,1) = -data(i,29)/(...
265                     data(i,22)-data(i,21)...
266                 )*(ΔT);
267                 Δ_m_a(4,1) = -data(i,29)/rho_a*Δ_rho_a;
268                 Δ_m_a(5,1) = 0;
269                 uncertainty.m_cond_a(i,1) = sqrt(sum(...
270                     Δ_m_a.^2 ...
271                 ))/data(i,29);
272             end
273             data(i,28) = Q_cond_r(i,1);
274         end
275
276         i = i + 1;
277     end
278     cell_CA = strfind(fault,'CA');
279     logical_CA = zeros(pp,1);
280     if ¬isempty(cell_CA)
281         for i = 1:pp
282             if ¬isempty(cell2mat(cell_CA(i)))
283                 logical_CA(i,1) = 1;
284             end
285         end
286     end
287     Cond_nf_Vdot_a = mean(data(find(...
```

```
        ¬logical_CA&(data(:,29)>0)&(SC_cond(:,1)>3&...
        SC_TXV(:,1)>3&data(:,23)>0) ...
    ),29));

    % calculate the condenser heat transfer rate with the rest of
    % the cases
    for i = 1:pp
        if Q_cond_r(i,1)≤1.e-8
            airinlet = calllib(...
                libname, 'HumAir_DLL', data(i,21), data(i,32), ...
                1, 230, zeros(1,5) ...
            );
            rho_a = (1)/airinlet(5);
            if System.Heating == 0
                data(i,29) = Cond_nf_Vdot_a;
            end
            if Ind_Fan_Upstream_cond
                Q_cond_r(i,1) = data(i,29)*rho_a*cp*(...
                    data(i,22)-data(i,21) ...
                )-data(i,26);
            else
                Q_cond_r(i,1) = data(i,29)*rho_a*cp*(...
                    data(i,22)-data(i,21) ...
                );
            end
            air_index(i,1) = 1;
```

```
314         end
315     end
316
317     % calculate evaporator heat transfer rate
318     i = 1;
319     Q_evap_r = zeros(pp,1);
320     Vdot_evap = zeros(pp,1);
321     while i ≤ pp
322         if System.Heating == 0
323             airinlet(1,1:5) = calllib(...
324                 libname, 'HumAir_DLL', data(i,17), data(i,32), ...
325                 1, data(i,19), zeros(1,5) ...
326             );
327             airoutlet(1,1:5) = calllib(...
328                 libname, 'HumAir_DLL', data(i,18), data(i,32), ...
329                 1, data(i,20), zeros(1,5)...
330             );
331             airinlet_ΔT(1,1:5) = calllib(...
332                 libname, 'HumAir_DLL', data(i,17)+0.01, ...
333                 data(i,32), 1, data(i,19), zeros(1,5) ...
334             );
335             airoutlet_ΔT(1,1:5) = calllib(...
336                 libname, 'HumAir_DLL', data(i,18)+0.01, ...
337                 data(i,32), 1, data(i,20), zeros(1,5) ...
338             );
339             if ΔT_d > 0
```

```
340                 airinlet_dD(1,1:5) = calllib(...
341                     libname, 'HumAir_DLL', data(i,17), data(i,32), ...
342                     1, data(i,19)+0.01, zeros(1,5) ...
343                 );
344                 airoutlet_dD(1,1:5) = calllib(...
345                     libname, 'HumAir_DLL', data(i,18), data(i,32), ...
346                     1, data(i,20)+0.01, zeros(1,5) ...
347                 );
348             else
349                 airinlet_dD(1,1:5) = calllib(...
350                     libname, 'HumAir_DLL', data(i,17), data(i,32), ...
351                     4, airoutlet(1,4)+0.001, zeros(1,5) ...
352                 );
353                 airoutlet_dD(1,1:5) = calllib(...
354                     libname, 'HumAir_DLL', data(i,18), data(i,32), ...
355                     4, airoutlet(1,4)+0.001, zeros(1,5) ...
356                 );
357             end
358         else
359             % set the dewpoint to be 230K as r = 1.e-8 may be
360             % out of range
361             airinlet(1,1:5) = calllib(...
362                 libname, 'HumAir_DLL', data(i,17), data(i,32), 1, ...
363                 230, zeros(1,5) ...
364             );
365             airoutlet(1,1:5) = calllib(...
```

```
366                 libname, 'HumAir_DLL', data(i,18), data(i,32), 1, ...
367                 230, zeros(1,5) ...
368             );
369             airinlet_ΔT(1,1:5) = calllib(...
370                 libname, 'HumAir_DLL', data(i,17)+0.01, ...
371                 data(i,32), 1, 230, zeros(1,5) ...
372             );
373             airoutlet_ΔT(1,1:5) = calllib(...
374                 libname, 'HumAir_DLL', data(i,18)+0.01, ...
375                 data(i,32), 1, 230, zeros(1,5) ...
376             );
377             airinlet_ΔD(1,1:5) = calllib(...
378                 libname, 'HumAir_DLL', data(i,17), data(i,32), ...
379                 1, 230, zeros(1,5) ...
380             ); % same thing
381             airoutlet_ΔD(1,1:5) = calllib(...
382                 libname, 'HumAir_DLL', data(i,18), data(i,32), ...
383                 1, 230, zeros(1,5) ...
384             ); % same thing
385             data(i,19) = airinlet(1,1);
386             data(i,20) = airoutlet(1,1);
387         end
388         air_Δh = (...
389             (airinlet_ΔT(1,3)-airinlet(1,3))/0.01*ΔT...
390         )^2 + ((airoutlet_ΔT(1,3)-airoutlet(1,3))/0.01*ΔT)^2;
391         if ΔT_d > 0
```

```
392             air_Δh = air_Δh + (...
393                 (airinlet_ΔD(1,3)-airinlet(1,3))/0.01*ΔT_d ...
394             )^2 + (...
395                 (airoutlet_ΔD(1,3)-airoutlet(1,3))/0.01*ΔT_d ...
396             )^2;
397         else
398             air_Δh = air_Δh + (...
399                 (airinlet_ΔD(1,3)-airinlet(1,3))/0.01*ΔRH ...
400             )^2 + (...
401                 (airoutlet_ΔD(1,3)-airoutlet(1,3))/0.01*ΔRH ...
402             )^2;
403         end
404         air_Δh = sqrt(air_Δh);
405         air_Δomega = (...
406             (airinlet_ΔT(1,2)-airinlet(1,2))/0.01*ΔT ...
407         )^2 + (...
408             (airoutlet_ΔT(1,2)-airoutlet(1,2))/0.01*ΔT ...
409         )^2;
410         if ΔT_d > 0
411             air_Δomega = air_Δomega + (...
412                 (airinlet_ΔD(1,2)-airinlet(1,2))/0.01*ΔT_d ...
413             )^2 + (...
414                 (airoutlet_ΔD(1,2)-airoutlet(1,2))/0.01*ΔT_d ...
415             )^2;
416         else
417             air_Δomega = air_Δomega + (...
```

```
418                 (airinlet_ΔD(1,2)-airinlet(1,2))/0.01*ΔRH ...
419             )^2 + ( ...
420                 (airoutlet_ΔD(1,2)-airoutlet(1,2))/0.01*ΔRH ...
421             )^2;
422         end
423         air_Δomega = sqrt(air_Δomega);
424         cp = 1006 + 1860*airinlet(1,2);
425         air_Δcp = 1860*air_Δomega;
426         airinlet_stored(i,1:5) = airinlet(1,1:5);
427         airoutlet_stored(i,1:5) = airoutlet(1,1:5);
428         SC_EXV_in(i,1) = propertyRH(...
429             'T','P',data(i,5),'Q',0,refri ...
430         )-data(i,13);
431         SC_cond_out(i,1) = propertyRH( ...
432             'T','P',data(i,4),'Q',0,refri ...
433         )-data(i,12);
434         if System.Heating==0
435             Tamb = data(i,21);
436         else
437             Tamb = data(i,17);
438         end
439         SH_evap_out(i,1) = data(i,16) - ...
440             propertyRH('T','P',data(i,8),'Q',1,refri);
441         if SC_EXV_in(i,1) > 3 && SC_cond_out(i,1) > 3 && ...
442             SH_evap_out(i,1) > 1 && data(i,23) > 0
443
```

```
444             Q_evap_r(i,1) = data(i,23)*(...
445                 propertyRH(...
446                     'H','T',data(i,16),'P',data(i,8),refri ...
447                 )-propertyRH('H','T',data(i,13),'P',data(i,5),refri));
448             if Δm_r_rel > 0
449                 Δm_r_abs = Δm_r_rel*data(i,23);
450             end
451             Q_evap_r_Δ(1,1) = Q_evap_r(i,1)/data(i,23)*Δm_r_abs;
452             if ΔP_r_rel > 0
453                 ΔP_r_abs = ΔP_r_rel*data(i,8);
454             end
455             Q_evap_r_Δ(2,1) = data(i,23)*(...
456                 propertyRH(...
457                     'H','T',data(i,16),'P',data(i,8)+0.01,refri ...
458                 )-propertyRH(...
459                     'H','T',data(i,16),'P',data(i,8),refri ...
460                 ))/0.01*ΔP_r_abs;
461             if ΔP_r_rel > 0
462                 ΔP_r_abs = ΔP_r_rel*data(i,5);
463             end
464             Q_evap_r_Δ(3,1) = -data(i,23)*(propertyRH(...
465                 'H','T',data(i,13),'P',data(i,5),refri ...
466             )-propertyRH(...
467                 'H','T',data(i,13),'P',data(i,5)-0.01,refri ...
468             ))/0.01*ΔP_r_abs;
469             Q_evap_r_Δ(4,1) = data(i,23)*(propertyRH(...
```

```
470                 'H','T',data(i,16)+0.01,'P',data(i,8),refri ...
471             )-propertyRH(...
472                 'H','T',data(i,16),'P',data(i,8),refri ...
473             ))/0.01*ΔT;
474             Q_evap_r_Δ(5,1) = -data(i,23)*(propertyRH(...
475                 'H','T',data(i,13),'P',data(i,5),refri ...
476             )-propertyRH(...
477                 'H','T',data(i,13)-0.01,'P',data(i,5),refri ...
478             ))/0.01*ΔT;
479             uncertainty.Q_evap_r(i,1) = sqrt(...
480                 sum(Q_evap_r_Δ.^2) ...
481             )/Q_evap_r(i,1);
482
483             if Ind_Fan_Upstream_evap
484                 ma_ori(i,1) = (Q_evap_r(i,1) - data(i,25))/(...
485                     (airinlet(1,3) - airoutlet(1,3))*1000 ...
486                 );
487                 Δ_m_a_ori(1,1) = 1/((...
488                     airinlet(1,3) - airoutlet(1,3) ...
489                 )*1000)*uncertainty.Q_evap_r(i,1)*Q_evap_r(i,1);
490                 if ΔW_rel > 0
491                     ΔW_abs = data(i,25)*ΔW_rel;
492                 end
493                 Δ_m_a_ori(2,1) = -1/((...
494                     airinlet(1,3) - airoutlet(1,3) ...
495                 )*1000)*ΔW_abs;
```

```
496                 Δ_m_a_ori(3,1) = -ma_ori(i,1)/(...
497                     airinlet(1,3) - airoutlet(1,3) ...
498                 )*air_Δh;
499                 Δ_m_a_ori_abs = sqrt(sum(Δ_m_a_ori.^2));
500                 SHR(i,1) = (ma_ori(i,1)*cp*(...
501                     data(i,17)-data(i,18) ...
502                 )+data(i,25))/Q_evap_r(i,1);
503                 Δ_SHR(1,1) = -SHR(i,1)*uncertainty.Q_evap_r(i,1);
504                 Δ_SHR(2,1) = cp*(...
505                     data(i,17)-data(i,18) ...
506                 )/Q_evap_r(i,1)*Δ_m_a_ori_abs;
507                 Δ_SHR(3,1) = ma_ori(i,1)*(...
508                     data(i,17)-data(i,18) ...
509                 )/Q_evap_r(i,1)*air_Δcp;
510                 Δ_SHR(4,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*(ΔT);
511                 Δ_SHR(5,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*(ΔT);
512                 Δ_SHR(6,1) = 1/Q_evap_r(i,1)*ΔW_abs;
513                 uncertainty.SHR(i,1) = sqrt(sum(Δ_SHR.^2))/SHR(i,1);
514             else
515                 ma_ori(i,1) = (Q_evap_r(i,1))/((...
516                     airinlet(1,3) - airoutlet(1,3) ...
517                 )*1000);
518                 Δ_m_a_ori(1,1) = 1/((...
519                     airinlet(1,3) - airoutlet(1,3) ...
520                 )*1000)*uncertainty.Q_evap_r(i,1)*Q_evap_r(i,1);
521                 Δ_m_a_ori(2,1) = 0;
```

```
522                 Δ_m_a_ori(3,1) = -ma_ori(i,1)/(...
523                     airinlet(1,3) - airoutlet(1,3) ...
524                 )*air_Δh;
525                 Δ_m_a_ori_abs = sqrt(sum(Δ_m_a_ori.^2));
526                 SHR(i,1) = (ma_ori(i,1)*cp*(...
527                     data(i,17)-data(i,18) ...
528                 ))/Q_evap_r(i,1);
529                 Δ_SHR(1,1) = -SHR(i,1)*uncertainty.Q_evap_r(i,1);
530                 Δ_SHR(2,1) = cp*(...
531                     data(i,17)-data(i,18) ...
532                 )/Q_evap_r(i,1)*Δ_m_a_ori_abs;
533                 Δ_SHR(3,1) = ma_ori(i,1)*(...
534                     data(i,17)-data(i,18) ...
535                 )/Q_evap_r(i,1)*air_Δcp;
536                 Δ_SHR(4,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*(ΔT);
537                 Δ_SHR(5,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*(ΔT);
538                 Δ_SHR(6,1) = 0;
539                 uncertainty.SHR(i,1) = sqrt(sum(Δ_SHR.^2))/SHR(i,1);
540             end
541             Vdot_evap(i,1) = ma_ori(i,1)*airinlet(1,5);
542             ma_evap_r(i,1) = ma_ori(i,1);
543             if Standard_Airflow_evap
544                 ma_evap_a(i,1) = data(i,30)*1.2;
545                 data(i,30) = ma_ori(i,1)/1.2;
546             else
547                 ma_evap_a(i,1) = data(i,30)/airoutlet(1,5);
```

```
548                 data(i,30) = ma_ori(i,1)*airoutlet(1,5);
549             end
550         end
551         i = i + 1;
552     end
553     % calculate correction ma
554     if isempty(ma_evap_r)
555         Evap_ma_r = 1;
556         Evap_ma_a = 1;
557     else
558         Evap_ma_a = mean(ma_evap_a(find(ma_evap_a>1.e-8)));
559         Evap_ma_r = mean(ma_evap_r(find(ma_evap_r>1.e-8)));
560     end
561     cell_EA = strfind(fault,'EA');
562     logical_EA = zeros(pp,1);
563     if ¬isempty(cell_EA)
564         for i = 1:pp
565             if ¬isempty(cell2mat(cell_EA(i)))
566                 logical_EA(i,1) = 1;
567             end
568         end
569     end
570     if isempty(find(¬logical_EA&(Q_evap_r>1.e-8)))
571         Evap_nf_Vdot_a = mean(Vdot_evap(:,1));
572         Evap_nf_Vdot_a_exp = mean(data(:,30));
573     else
```

```
574         Evap_nf_Vdot_a = ...
575             mean(Vdot_evap(find(¬logical_EA&(Q_evap_r>1.e-8)),1));
576         Evap_nf_Vdot_a_exp = ...
577             mean(data(find(¬logical_EA&(Q_evap_r>1.e-8)),30));
578     end
579     % solve air side data
580     for i = 1:pp
581         airinlet(1,1:5) = airinlet_stored(i,1:5);
582         airoutlet(1,1:5) = airoutlet_stored(i,1:5);
583         if SC_EXV_in(i,1) ≤ 3 || SC_cond_out(i,1) ≤ 3 || ...
584             SH_evap_out(i,1) ≤ 1 || data(i,23) ≤ 0
585             if System.Heating == 0
586                 if Standard_Airflow_evap
587                     ma_ori(i,1) = data(i,30)*1.2*Evap_ma_r/Evap_ma_a;
588                 else
589                     ma_ori(i,1) = data(i,30)/airoutlet(1,5)* ...
590                         Evap_ma_r/Evap_ma_a;
591                 end
592                 if Ind_Fan_Upstream_evap
593                     Q_evap_r(i,1) = ma_ori(i,1)*(...
594                         airinlet(1,3) - airoutlet(1,3) ...
595                     )*1000 + data(i,25);
596                     Δ_Q_evap_a(1,1) = Δ_m_a_rel*ma_ori(i,1);
597                     Δ_Q_evap_a(2,1) = ma_ori(i,1)*air_Δh*1000;
598                     if ΔW_rel > 0
599                         ΔW_abs = data(i,25)*ΔW_rel;
```

```
600                     end
601                     Δ_Q_evap_a(3,1) = ΔW_abs;
602                     uncertainty.Q_evap_a(i,1) = sqrt(...
603                         sum(Δ_Q_evap_a.^2) ...
604                     )/Q_evap_r(i,1);
605                     SHR(i,1) = (...
606                         ma_ori(i,1)*cp*(data(i,17)-data(i,18)) + ...
607                         data(i,25) ...
608                     )/Q_evap_r(i,1);
609                     Δ_SHR(1,1) = -SHR(i,1)*uncertainty.Q_evap_a(i,1);
610                     Δ_SHR(2,1) = cp*(data(i,17)-data(i,18)) /...
611                         Q_evap_r(i,1)*Δ_m_a_rel*ma_ori(i,1);
612                     Δ_SHR(3,1) = ma_ori(i,1)*(...
613                         data(i,17)-data(i,18)...
614                     )/Q_evap_r(i,1)*air_Δcp;
615                     Δ_SHR(4,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*ΔT;
616                     Δ_SHR(5,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*ΔT;
617                     Δ_SHR(6,1) = 1/Q_evap_r(i,1)*ΔW_abs;
618                     uncertainty.SHR(i,1) = sqrt(sum(...
619                         Δ_SHR.^2 ...
620                     ))/SHR(i,1);
621                 else
622                     Q_evap_r(i,1) = ma_ori(i,1)*(...
623                         airinlet(1,3)-airoutlet(1,3) ...
624                     )*1000;
625                     Δ_Q_evap_a(1,1) = Δ_m_a_rel*ma_ori(i,1);
```

```
626                     Δ_Q_evap_a(2,1) = ma_ori(i,1)*air_Δh*1000;
627                     Δ_Q_evap_a(3,1) = 0;
628                     uncertainty.Q_evap_a(i,1) = sqrt(...
629                         sum(Δ_Q_evap_a.^2) ...
630                     )/Q_evap_r(i,1);
631                     SHR(i,1) = (ma_ori(i,1)*cp*(...
632                         data(i,17)-data(i,18) ...
633                     ))/Q_evap_r(i,1);
634                     Δ_SHR(1,1) = -SHR(i,1)*uncertainty.Q_evap_a(i,1);
635                     Δ_SHR(2,1) = cp*(...
636                         data(i,17)-data(i,18) ...
637                     )/Q_evap_r(i,1)*Δ_m_a_rel*ma_ori(i,1);
638                     Δ_SHR(3,1) = ma_ori(i,1)*(...
639                         data(i,17)-data(i,18) ...
640                     )/Q_evap_r(i,1)*air_Δcp;
641                     Δ_SHR(4,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*ΔT;
642                     Δ_SHR(5,1) = ma_ori(i,1)*cp/Q_evap_r(i,1)*ΔT;
643                     Δ_SHR(6,1) = 0;
644                     uncertainty.SHR(i,1) = sqrt(...
645                         sum(Δ_SHR.^2) ...
646                     )/SHR(i,1);
647                 end
648                 air_index(i,2) = 1;
649                 Vdot_evap(i,1) = ma_ori(i,1)*airinlet(1,5);
650             end
651         end
```

```
652     end
653
654     W_comp = data(:,24);
655     Q_HL = Q_evap_r + W_comp - Q_cond_r;
656     Q_HL_ratio = Q_HL./Q_evap_r;
657     index_removed = find(Q_HL_ratio<Q_HL_limit);
658
659     % calculate mass flow rate if possible
660     for i = 1:pp
661         if (SC_TXV(i,1) ≥3 && SC_cond(i,1) ≥ 3) ...
662                 || (SC_TXV(i,1) ≥1 && SH_evap(i,1) ≥ 1) ...
663                 ||(SC_cond(i,1) ≥ 1)
664             if (SC_TXV(i,1) ≥ 3 && SC_cond(i,1) ≥ 3)
665             elseif (SC_TXV(i,1) ≥1 && SH_evap(i,1) ≥ 1)
666                 hout_evap = propertyRH(...
667                     'H','T',data(i,16),'P',data(i,8), refri ...
668                 );
669                 hout_LL = propertyRH( ...
670                     'H','T',data(i,13),'P',data(i,5), refri ...
671                 );
672                 data(i,23) = Q_evap_r(i,1)/(hout_evap - hout_LL);
673             else
674                 hin_cond = propertyRH(...
675                     'H','T',data(i,11),'P',data(i,3), refri ...
676                 );
677                 hin_LL = propertyRH(...
```

```
678                     'H','T',data(i,12),'P',data(i,4), refri ...
679                 );
680                 data(i,23) = Q_cond_r(i,1)/(hin_cond - hin_LL);
681             end
682         end
683     end
684 else
685     Q_HL_limit_now = Q_HL_limit;
686     load([strcat(foldername,'\'),savefilename]);
687     index_removed = find(Q_HL_ratio<Q_HL_limit_now);
688 end
689 
690 save(savefilename);
691 movefile(savefilename,strcat(foldername,'\'));
692 
693 end
```

N.2 Compressor Modeling

```
1 function System = Compressor_regression_v075_main()
2 
3 % main function for compressor modeling
4 
5 % define function and folder name
6 version_main = '075';
7 version_regress = '070';
8 version_plot = '007';
```

```
foldername = strcat('Compressor_regression_v',version_main);
regressname = strcat('Compressor_regression_v',version_regress);
plotname = strcat('Compressor_regression_plot_v',version_plot);
comp_func = str2func(['@(System, libname, version_number)',...
    regressname,'(System, libname, version_number)']);
plot_func = str2func(['@(System, libname, version_number, plot_num)',...
    plotname,'(System, libname, version_number, plot_num)']);
mkdir(foldername);
savefilename = strcat(foldername,'.mat');

% Set information
% re-file at this stage to avoid problems in parfor loop

% Breuker 3-ton R22 packaged FXO system (Recip)
% (Breuker_030_R22_packaged_FXO_Recip)
i = 1;
System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
System(i).LLname = System(i).name;
System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
System(i).foldername = 'Breuker';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
```

```
35 System(i).Heating = 0;
36 System(i).Combined = 0;
37
38 % Boshen 3-ton R410A packaged FXO system (Scroll)
39 % (Shen_030_R410A_packaged_FXO_Scroll)
40 i = i + 1;
41 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
42 System(i).LLname = System(i).name;
43 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
44 System(i).foldername = 'Boshen';
45 System(i).worksheetname = 'SI (no_invalid_CA)';
46 System(i).refname = 'R410A';
47 System(i).Standard_Airflow_evap = 0;
48 System(i).Fan_Upstream_evap = 0;
49 System(i).Standard_Airflow_cond = 1;
50 System(i).Fan_Upstream_cond = 1;
51 System(i).Heating = 0;
52 System(i).Combined = 0;
53
54 % Boshen 3-ton R410A split FXO system (Recip)
55 % (Shen_030_R410A_split_FXO_Recip)
56 % for compressor modeling, merged with TXV system data
57 i = i + 1;
58 System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
59 System(i).LLname = System(i).name;
60 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
```

```
61 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
62 System(i).worksheetname = 'SI (no_invalid_CA)';
63 System(i).refname = 'R410A';
64 System(i).Standard_Airflow_evap = 0;
65 System(i).Fan_Upstream_evap = 0;
66 System(i).Standard_Airflow_cond = 1;
67 System(i).Fan_Upstream_cond = 1;
68 System(i).Heating = 0;
69 System(i).Combined = 0;
70
71 % Boshen 5-ton R407C packaged FXO system (Scroll)
72 % (Shen_050_R407C_packaged_FXO_Scroll)
73 i = i + 1;
74 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
75 System(i).LLname = System(i).name;
76 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
77 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
78 System(i).worksheetname = 'SI (no_invalid_CA)';
79 System(i).refname = 'R407C';
80 System(i).Standard_Airflow_evap = 0;
81 System(i).Fan_Upstream_evap = 0;
82 System(i).Standard_Airflow_cond = 1;
83 System(i).Fan_Upstream_cond = 1;
84 System(i).Heating = 0;
85 System(i).Combined = 0;
86
```

```
87 % Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
88 % (NIST_025_R410A_split_TXV_Scroll)
89 i = i + 1;
90 System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
91 System(i).LLname = System(i).name;
92 System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
93 System(i).foldername = 'NIST';
94 System(i).worksheetname = 'SI (no_invalid_CA)';
95 System(i).refname = 'R410A';
96 System(i).Standard_Airflow_evap = 1;
97 System(i).Fan_Upstream_evap = 1;
98 System(i).Standard_Airflow_cond = 1;
99 System(i).Fan_Upstream_cond = 1;
100 System(i).Heating = 0;
101 System(i).Combined = 0;
102
103 % Harms 5-ton R22 packaged TXV system (Scroll)
104 % (Harms_050_R22_packaged_TXV_Scroll)
105 i = i + 1;
106 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
107 System(i).LLname = System(i).name;
108 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
109 System(i).foldername = 'Harms_050tonR22PackagedTXV';
110 System(i).worksheetname = 'SI (no_invalid_CA)';
111 System(i).refname = 'R22.fld';
112 System(i).Standard_Airflow_evap = 0;
```

```
113 System(i).Fan_Upstream_evap = 0;
114 System(i).Standard_Airflow_cond = 1;
115 System(i).Fan_Upstream_cond = 1;
116 System(i).Heating = 0;
117 System(i).Combined = 0;
118
119 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
120 % (HCA3_030_R410A_split_TXV_Scroll)
121 i = i + 1;
122 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
123 System(i).LLname = System(i).name;
124 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_combined.xlsx';
125 System(i).foldername = 'Kim-HCA3';
126 System(i).worksheetname = 'SI (no_invalid_CA)';
127 System(i).refname = 'R410A';
128 System(i).Standard_Airflow_evap = 0;
129 System(i).Fan_Upstream_evap = 0;
130 System(i).Standard_Airflow_cond = 1;
131 System(i).Fan_Upstream_cond = 1;
132 System(i).Heating = 0;
133 System(i).Combined = 1;
134
135 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
136 % (YSA_030_R22_split_TXV_Scroll)
137 i = i + 1;
138 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
```

```
139 System(i).LLname = System(i).name;
140 System(i).filename = 'Kim_Cycle_data_R22_YSA_combined.xlsx';
141 System(i).foldername = 'Kim-YSA';
142 System(i).worksheetname = 'SI (no_invalid_CA)';
143 System(i).refname = 'R22.fld';
144 System(i).Standard_Airflow_evap = 0;
145 System(i).Fan_Upstream_evap = 0;
146 System(i).Standard_Airflow_cond = 1;
147 System(i).Fan_Upstream_cond = 1;
148 System(i).Heating = 0;
149 System(i).Combined = 1;
150
151 % Kim 3-ton R22 split TXV system (Scroll) (YKC)
152 % (YKC_030_R22_split_TXV_Scroll)
153 i = i + 1;
154 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
155 System(i).LLname = System(i).name;
156 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
157 System(i).foldername = 'Kim-YKC';
158 System(i).worksheetname = 'SI (no_invalid_CA)';
159 System(i).refname = 'R22.fld';
160 System(i).Standard_Airflow_evap = 0;
161 System(i).Fan_Upstream_evap = 0;
162 System(i).Standard_Airflow_cond = 1;
163 System(i).Fan_Upstream_cond = 1;
164 System(i).Heating = 0;
```

```
165 System(i).Combined = 0;
166
167 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
168 % (TM_040_R410A_packaged_TXV_Recip)
169 i = i + 1;
170 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
171 System(i).LLname = System(i).name;
172 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
173 System(i).foldername = 'Kim-TM';
174 System(i).worksheetname = 'SI';
175 System(i).refname = 'R410A.ppf';
176 System(i).Standard_Airflow_evap = 0;
177 System(i).Fan_Upstream_evap = 0;
178 System(i).Standard_Airflow_cond = 1;
179 System(i).Fan_Upstream_cond = 1;
180 System(i).Heating = 0;
181 System(i).Combined = 0;
182
183 % other settings for storage
184 m = length(System);
185 for i = 1:m
186     System(i).mainfoldername = foldername;
187     System(i).b1 = zeros(3,1);
188     System(i).C = zeros(5,1);
189     System(i).f_HL = zeros(4,1);
190     System(i).poly_n_factor = zeros(3,1);
```

```
191     System(i).b1_high = zeros(3,1);
192     System(i).C_high = zeros(5,1);
193     System(i).f_HL_high = zeros(4,1);
194     System(i).poly_n_factor_high = zeros(3,1);
195     System(i).r2.r2_m = 0;
196     System(i).r2.r2_P = 0;
197     System(i).r2.r2_Δh = 0;
198     System(i).r2.r2_final = 0;
199     System(i).r2.r2_m_high = 0;
200     System(i).r2.r2_P_high = 0;
201     System(i).r2.r2_Δh_high = 0;
202     System(i).r2.r2_high_final = 0;
203     System(i).Comp.b1 = 0;
204     System(i).Comp.C = 0;
205     System(i).Comp.poly_n_factor = 0;
206     System(i).Comp.f_HL_factor = 0;
207     System(i).Comp_high.b1 = 0;
208     System(i).Comp_high.C = 0;
209     System(i).Comp_high.poly_n_factor = 0;
210     System(i).Comp_high.f_HL_factor = 0;
211 end
212
213 % setting up parallel solver
214 % try
215 %     matlabpool(2); % try setting up multiple loops
216 % catch err
```

```
217 %     matlabpool close; % if catch an error, close and reset it
218 %     matlabpool(2);
219 % end
220
221 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
222 libname ='lib'; % Name of library
223
224 % start calculation
225 % m = 2;
226 % for i = 1:m
227 try
228     parfor i = 1:m
229         System(i) = 1; % a statement leads to error
230     end
231 catch err
232 end
233 for i = 1:m
234     error_statement(i) = err;
235 end
236 no_plot = zeros(m,1);
237 % parfor i = 1:m
238 for i = 1:m
239     try
240         disp(['Starting to run ',System(i).name]);
241         [Comp, Comp_high, r2] = comp_func(System(i), libname, version_main);
242         System(i).Comp = Comp;
```

```
243         System(i).Comp_high = Comp_high;
244         System(i).r2 = r2;
245         disp(['Finishing simulating ',System(i).name]);
246     catch err
247         disp(['Problem found in system ',System(i).name]);
248         no_plot(i,1) = 1;
249         error_statement(i) = err;
250     end
251 end
252 save(savefilename);
253
254 % plot cannot be done in parfor. Do it in a for loop
255 for i = 1:m
256     if no_plot(i,1) == 0
257         disp(['Starting to plot ',System(i).name]);
258         plot_func(System(i), libname, version_main, 0);
259         disp(['Finishing plotting ',System(i).name]);
260     end
261 end
262
263 save(savefilename);
264 movefile(savefilename,strcat(foldername,'\'));
265
266 try
267     matlabpool close; % if catch an error, close and reset it
268 end
```

```
try
    unloadlibrary lib;
end
```

```
function [Comp, Comp_high, r2] = Compressor_regression_v070(...
    System, libname, version_number ...
)

% Compressor modeling function

%% File and name declatration and data acqusition
% name definition
% data reading and filtering
bin_num = 10; % number of bins for histogram
filename = System.filename;
worksheetname = System.worksheetname;
refname = System.refname;
Standard_Airflow = System.Standard_Airflow_evap;
Ind_Fan_Upstream = System.Fan_Upstream_evap;
version = strcat(System.name,'_',version_number);
savefilename = strcat('Compressor_regression_v',version,'.mat');
foldername = strcat(...
    strcat(pwd,'\',System.mainfoldername,'\'), ...
    strcat('Compressor_regression_v',version) ...
);
```

```
22 mkdir(foldername);
23
24 % read Q_gain_observation_v025 to find points to be removed
25 Q_gain_version = '025';
26 Q_gain_name = ['Q_gain_observation_v',System.name,'_',Q_gain_version];
27 Q_gain_raw = open(['Q_gain_observation_v',Q_gain_version,'\', ...
28     Q_gain_name,'\',Q_gain_name,'.mat']);
29 k = length(Q_gain_raw.index_removed);
30 if isfield(System,'Combined') & System.Combined == 1
31     [data code] = xlsread(strcat(...
32         pwd,'\',System.foldername,'\',filename ...
33     ),worksheetname,'a2:aj65536');
34     [dum fault] = xlsread(strcat(...
35         pwd,'\',System.foldername,'\',filename ...
36     ),worksheetname,'ak2:ak65536');
37 else
38     data = Q_gain_raw.data;
39     code = Q_gain_raw.code;
40     fault = Q_gain_raw.fault;
41 end
42 for i = k:-1:1 % reverse for correct indexing
43     try
44         data(Q_gain_raw.index_removed(i),:) = [];
45         fault(Q_gain_raw.index_removed(i),:) = [];
46         code(Q_gain_raw.index_removed(i),:) = [];
47         Q_gain_raw.Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
```

```
48          Q_gain_raw.Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
49      end
50  end
51  [pp qq] = size(data);
52  cell_VL = strfind(fault,'VL');
53  if isempty(cell_VL)
54      cell_VL = cell(pp,1);
55  end
56  cell_NC = strfind(fault,'NC');
57  if isempty(cell_NC)
58      cell_NC = cell(pp,1);
59  end
60  cell_LL = strfind(fault,'LL');
61  if isempty(cell_LL)
62      cell_LL = cell(pp,1);
63  end
64  i = 1;
65  while i ≤ pp
66      if ¬isempty(cell2mat(cell_VL(i))) || ...
67          ¬isempty(cell2mat(cell_NC(i))) || ...
68          ¬isempty(cell2mat(cell_LL(i)))
69          data(i,:) = [];
70          fault(i,:) = [];
71          code(i,:) = [];
72          cell_VL(i,:) = [];
73          cell_NC(i,:) = [];
```

```
74          cell_LL(i,:) = [];
75          Q_gain_raw.Q_evap_r(i,:) = [];
76          Q_gain_raw.Q_cond_r(i,:) = [];
77          i = i - 1;
78          pp = pp - 1;
79      end
80      i = i + 1;
81  end
82  [pp qq] = size(data);
83  p = pp;
84
85  % filetered for ambient temperature
86  if System.Combined == 1
87      for i = 1:pp
88          if data(i,35) == 0
89              data(i,35) = data(i,21);
90          else
91              data(i,35) = data(i,17);
92          end
93      end
94  else
95      if System.Heating == 0
96          data(:,35) = data(:,21);
97      else
98          data(:,35) = data(:,17);
99      end
```

```
100 end
101
102 % check for adjustement first
103 j = 1;
104 k = 1;
105 l = 1;
106 SH = -999;
107 for i = 1:p
108     % only use good pressure values
109     if data(i,8) < data(i,1)
110         data(i,1) = data(i,8);
111     end
112     SH_comp_check(i,1) = data(i,9) - ...
113         propertyRH('T','P',data(i,1),'Q',1,refname);
114     SH_evap_check(i,1) = data(i,16) - ...
115         propertyRH('T','P',data(i,8),'Q',1,refname);
116     LL_SC_check(i,1) = ...
117         propertyRH('T','P',data(i,5),'Q',0,refname) - data(i,13);
118     SC_check(i,1) = ...
119         propertyRH('T','P',data(i,4),'Q',0,refname) - data(i,12);
120     if SH_comp_check(i,1) ≥ 1 && ...
121         LL_SC_check(i,1) ≥ 3 && SC_check(i,1) ≥ 3
122         SH(j,1) = SH_evap_check(i,1);
123         j = j + 1;
124     end
125 end
```

```
126 % see if there is enough data to continue
127 if j < 4
128     % or not enough points to obtain the heatloss factor
129     b1 = zeros(3,1);
130     C = zeros(3,1);
131     poly_n_factor = zeros(2,1);
132     b1_high = zeros(3,1);
133     C_high = zeros(3,1);
134     poly_n_factor_high = zeros(2,1);
135     no_plot = 1;
136     r2.r2_m = 0;
137     r2.r2_P = 0;
138     r2.r2_Δh = 0;
139     r2.r2_final = 0;
140     r2.r2_m_high = 0;
141     r2.r2_P_high = 0;
142     r2.r2_Δh_high = 0;
143     r2.r2_high_final = 0;
144     save(savefilename);
145     movefile(savefilename,strcat(foldername,'\'));
146     return;
147 else
148     no_plot = 0;
149 end
150 if length(find(SH>1)) < 8
151     for j = 1:p
```

```
152         if LL_SC_check(j,1) ≥ 3 && SC_check(j,1) ≥ 3
153             airinlet(1,1:5) = calllib(...
154                 libname, 'HumAir_DLL', data(j,17), data(j,32), ...
155                 1, data(j,19), zeros(1,5) ...
156             );
157             airoutlet(1,1:5) = calllib(...
158                 libname, 'HumAir_DLL', data(j,18), data(j,32), ...
159                 1, data(j,20), zeros(1,5) ...
160             );
161             Q_evap_a(j,1) = Q_gain_raw.Q_evap_r(j,1);
162             hout_evap = propertyRH(...
163                 'H','T',data(j,13),'P',data(j,5),refname ...
164             ) + Q_evap_a(j,1)/data(j,23);
165             hin_suc(j,1) = propertyRH(...
166                 'H','T',data(j,9),'P',data(j,1),refname ...
167             );
168             if hin_suc(j,1) < hout_evap
169                 data(j,9) = propertyRH(...
170                     'T','P',data(j,1),'H',hout_evap,refname ...
171                 );
172             end
173         end
174     end
175 end
176
177 % start filtering
```

```
178 j = 1;
179 k = 1;
180 l = 1;
181
182 for i = 1:p
183     if SH_comp_check(i,1) > 1 && (...
184             (LL_SC_check(i,1) ≥3 && SC_check(i,1) ≥ 3) ...
185              || (LL_SC_check(i,1) ≥1 && SH_evap_check(i,1) ≥ 1) ...
186              ||(SC_check(i,1) ≥ 1) ...
187              )
188         SH(j,1) = data(i,16) -...
189             propertyRH('T','P',data(i,8),'Q',1,refname);
190         hin_cond = propertyRH(...
191             'H','T',data(i,11),'P',data(i,3), refname ...
192         );
193         hout_evap = propertyRH(...
194             'H','T',data(i,16),'P',data(i,8),refname ...
195         );
196         try
197             if (LL_SC_check(i,1) ≥ 3 && SC_check(i,1) ≥ 3)
198             elseif (LL_SC_check(i,1) ≥1 && SH_evap_check(i,1) ≥ 1)
199                 hout_LL = propertyRH(...
200                     'H','T',data(i,13),'P',data(i,5), refname ...
201                 );
202                 data(i,23) = ...
203                     Q_gain_raw.Q_evap_r(i,1)/(hout_evap - hout_LL);
```

```
204             else
205                 hin_LL = propertyRH(...
206                     'H','T',data(i,12),'P',data(i,4), refname ...
207                 );
208                 data(i,23) = Q_gain_raw.Q_cond_r(i,1)/(...
209                     hin_cond - hin_LL ...
210                 );
211             end
212         catch err
213             clear err;
214             data(i,23) = -9999;
215         end
216         if data(i,23) > 0
217             data_II(j,:) = data(i,:);
218             code_II(j,:) = code(i,:);
219             j = j + 1;
220         end
221     end
222     if LL_SC_check(i,1) ≥ 3 && SC_check(i,1) ≥ 3
223         data_sc(k,:) = data(i,:);
224         k = k + 1;
225     end
226     if SH_comp_check(i,1) ≥ 1
227         data_sh(l,:) = data(i,:);
228         l = l + 1;
229     end
```

```
230 end
231
232 %% Calculation for mass flow rate model
233 % check the air-side data and expansion valve inlet condition
234 % and adjust the temperature at the compressor inlet whether
235 % appropriate
236 [p q] = size(data_II);
237 [p_SC q_SC] = size(data_sc);
238 [p_sh q_sh] = size(data_sh);
239
240 % Find a mean polytropic coefficient
241 for i = 1:p_sh
242     h_in_sh = propertyRH(...
243         'H','T',data_sh(i,9),'P',data_sh(i,1),refname ...
244     );
245     rho_in(i,1) = propertyRH(...
246         'D','P',data_sh(i,1),'H',h_in_sh,refname ...
247     );
248     rho_out(i,1) = propertyRH(...
249         'D','T',data_sh(i,10),'P',data_sh(i,2),refname ...
250     );
251 end
252 n_poly_sh = ...
253     (log(data_sh(:,1)./data_sh(:,2))./ ...
254     log(rho_in(:,1)./rho_out(:,1)));
255 rho_rated = rho_in(1,1)/data(1,23);
```

```
256 P_evap_rated = data_II(1,1);
257
258 % use the data with both good subcooling and superheat to
259 % find the mass flow rate model
260 rho_in = [];
261 rho_out = []';
262 for i = 1:p
263     rho_in(i,1) = ...
264         propertyRH('D','T',data_II(i,9),'P',data_II(i,1),refname);
265     rho_out(i,1) = ...
266         propertyRH('D','T',data_II(i,10),'P',data_II(i,2),refname);
267 end
268 n_poly = log(data_II(:,1)./data_II(:,2))./log(rho_in(:,1)./ ...
269     rho_out(:,1));
270
271 X = rho_in(:,1);
272 X(:,2) = X(:,1).*((data_II(:,2)./data_II(:,1)).^(1./n_poly));
273 X(:,3) = X(:,1).*(data_II(:,2)-data_II(:,1));
274 Y = data_II(:,23);
275 weigh = weighing_function_v000(Y, bin_num);
276 for i = 1:p
277     Y(i,1) = 1/sqrt(weigh(i,1))*Y(i,1);
278     X(i,:) = 1/sqrt(weigh(i,1))*X(i,:);
279 end
280 b1 = X\Y;
281 b1 = fmincon(@(b1) sum(...
```

```
282         1./weigh.*((Y - X*b1)./Y).^2 ...
283     ),b1,[],[],[],[],[0 -Inf -Inf],[Inf 0 0],[],optimset(...
284         'display','off' ...
285     ));
286 if b1(3) > 0 % eliminate impossible b1(3)
287     X = X(:,1:2);
288     b1 = X\Y;
289     b1 = fminunc(@(b1) sum(...
290         1./weigh.*((Y - X*b1)./Y).^2 ...
291     ),b1,optimset('display','off'));
292     b1(3,1) = 0;
293     X(:,3) = zeros(p,1);
294 end
295 for i = 1:p
296     Y(i,1) = Y(i,1)*sqrt(weigh(i,1));
297     X(i,:) = X(i,:)*sqrt(weigh(i,1));
298 end
299 X_high = rho_in(:,1);
300 X_high(:,2) = X(:,1).*((data_II(:,2)./data_II(:,1)).^(1./(n_poly)));
301 X_high(:,3) = X(:,1).*(data_II(:,2)-data_II(:,1));
302 for i = 1:p
303     Y(i,1) = 1/sqrt(weigh(i,1))*Y(i,1);
304     X_high(i,:) = 1/sqrt(weigh(i,1))*X_high(i,:);
305 end
306 b1_high = X_high\Y;
307 b1_high = fminunc(@(b1_high) sum(...
```

```
308     1./weigh.*((Y - X_high*b1_high)./Y).^2 ...
309 ),b1_high,optimset('display','off'));
310 if b1_high(3) > 0 % eliminate impossible b1(3)
311     X_high = X_high(:,1:2);
312     b1_high = X_high\Y;
313     b1_high = fminunc(@(b1_high) sum(...
314         1./weigh.*((Y - X_high*b1_high)./Y).^2 ...
315     ),b1_high,optimset('display','off'));
316     b1_high(3,1) = 0;
317     X_high(:,3) = zeros(p,1);
318 end
319 for i = 1:p
320     Y(i,1) = Y(i,1)*sqrt(weigh(i,1));
321     X_high(i,:) = X_high(i,:)*sqrt(weigh(i,1));
322 end
323 mdot_est = X*b1;
324 COV_X = covariance_adj(X);
325 Comp.COV_X = COV_X;
326 Comp.mdot_limit = max(diag(X*COV_X*X'));
327 mdot_est_high = X_high*b1_high;
328 save(savefilename);
329
330
331 %% calculating for the power estimation model coefficients
332 C = [1; 0; 0; 0; 0;];
333 rho_out = [];
```

```
for i = 1:p_sh
    rho_out(i,1) = ...
        propertyRH('D','T',data_sh(i,10),'P',data_sh(i,2),refname);
end
% grouping for histogram
weigh_P = weighing_function_v000(data_sh(:,24), bin_num);
% calculate a dummy; not used in power estimation code
rho_in = (data_sh(:,1)./data_sh(:,2)).^(1./(n_poly_sh)).* ...
    rho_out(:,1);
outFD_power = gradientcheck(@(C) Power_estimation(...
    C, b1, weigh_P, data_sh, n_poly_sh, P_evap_rated ...
), C,'DifferenceType','forward');
result_power = lbfgs(@(C) Power_estimation(...
    C, b1, weigh_P, data_sh, n_poly_sh, P_evap_rated ...
), C, 'Display', 'off', 'MaxIters', 10000, ...
    'MaxFuncEvals', 10000, 'StopTol', 1.e-8, 'RelFuncTol', 1.e-8);
C = result_power.X;
[dummy1, dummy2, Power_est] = Power_estimation(...
    C, b1, weigh_P, data_sh, n_poly_sh, P_evap_rated ...
);
X = [];
X(:,1) = -Power_est./(C(1) +...
    C(2).*exp(C(3).*data_sh(:,1)./P_evap_rated) + ...
    C(4).*exp(C(5).*data_sh(:,2)./P_evap_rated));
X(:,2) = X(:,1).*exp(C(3).*data_sh(:,1)./P_evap_rated);
X(:,3) = X(:,1).*C(2).*exp(...
```

```
360     C(3).*data_sh(:,1)./P_evap_rated ...
361 ).*data_sh(:,1)./P_evap_rated;
362 X(:,4) = X(:,1).*exp(C(5).*data_sh(:,2)./P_evap_rated);
363 X(:,5) = X(:,1).*C(4).*exp(...
364     C(5).*data_sh(:,2)./P_evap_rated ...
365 ).*data_sh(:,2)./P_evap_rated;
366 COV_X = covariance_adj(X);
367 Comp.COV_X_Power = COV_X;
368 Comp.Power_limit = max(diag(X/(X'*X)*X'));
369 save(savefilename);
370
371 save(savefilename);
372
373 %% Calculation of power into refrigerant (require filtering)
374 k = 1;
375 mdot_fil_data = data_II(:,23);
376 mdot_est_fil = mdot_est;
377 mdot_est_fil_high = mdot_est_high;
378 for i = 1:pp
379     SH_comp_check(i,1) = ...
380         data(i,9) - propertyRH('T','P',data(i,1),'Q',1,refname);
381     if SH_comp_check(i,1) ≥ 1
382         h_in_data(k,1) = ...
383             propertyRH('H','T',data(i,9),'P',data(i,1),refname);
384         h_out_fil_data(k,1) = ...
385             propertyRH('H','T',data(i,10),'P',data(i,2),refname);
```

```
386         rho_in_data(k,1) = ...
387             propertyRH('D','T',data(i,9),'P',data(i,1),refname);
388         rho_out_fil_data(k,1) = ...
389             propertyRH('D','T',data(i,10),'P',data(i,2),refname);
390         mdot_est_sh(k,1) = ...
391             propertyRH('D','T',data(i,9),'P',data(i,1),refname)...
392             *(b1(1)+b1(2)*(data(i,2)/data(i,1))^(...
393                 1/n_poly_high(k,1) ...
394             )+b1(3).*(data(i,2)-data(i,1)));
395         data_sh(k,:) = data(i,:);
396         [dummy1, dummy2, Power_est_sh(k,1)] = Power_estimation(...
397             C, b1, weigh_P, data(i,:), n_poly_high(k,1), ...
398             P_evap_rated ...
399         );
400         T_evap_sh(k,1) = propertyRH('T','P',data(i,1),'Q',1,refname);
401         T_cond_sh(k,1) = propertyRH('T','P',data(i,2),'Q',1,refname);
402         k = k + 1;
403     end
404 end
405
406 %% Heat loss factor estimation
407     function [residual HeatLoss] = f_HL_factor_prediction(...
408             T_in, T_out, T_amb, h_in, hout, mdot, Power, ...
409             f_HL_factor, weigh_Δh, refname ...
410         )
411         HeatLoss = (f_HL_factor(1).*(...
```

```
412             T_in-T_amb ...
413         )+f_HL_factor(2).*(T_out-T_amb));
414         residual = 1./sqrt(weigh_Δh).*(...
415             Power - HeatLoss - mdot.*(hout - h_in) ...
416         )./(mdot.*(hout - h_in));
417     end
418
419 % initial guess
420 % use the power getting into the refrigerant stream for weighing
421 weigh_Δh = weighing_function_v000(...
422     mdot_est_sh.*(h_in_data - h_out_fil_data), bin_num ...
423 );
424 options = optimset('MaxFunEvals',2000,'Display','off');
425 f_HL_factor = ...
426     lsqnonlin(@(x) f_HL_factor_prediction(...
427         data_sh(:,9), T_cond_sh(:,1), data_sh(:,35), h_in_data, ...
428         h_out_fil_data, mdot_est_sh, Power_est_sh, x, ...
429         weigh_Δh, refname ...
430     ), [1; 1;], [0 0], [], options);
431 Δh = h_out_fil_data - h_in_data;
432     function residual = Δh_prediction(...
433         T_out_guess, T_in, T_amb, h_in, P_in, P_out, mdot_est, ...
434         Power_est, f_HL_factor, refname ...
435     )
436         h_out_1 = propertyRH('H','T',T_out_guess,'P',P_out,refname);
437         % weighing not needed
```

```
438         [residual_II HeatLoss] = f_HL_factor_prediction(...
439             T_in, T_out_guess, T_amb, h_in, h_out_1, ...
440             mdot_est, Power_est, f_HL_factor, ...
441             ones(length(h_in),1), refname ...
442         );
443         h_out_2 = (Power_est - HeatLoss)/mdot_est + h_in;
444         residual = h_out_1 - h_out_2;
445     end
446 [residual HeatLoss_pre] = f_HL_factor_prediction(...
447     data_sh(:,9), T_cond_sh(:,1), data_sh(:,35), h_in_data, ...
448     0, mdot_est_sh(:,1), Power_est_sh(:,1), f_HL_factor, ...
449     weigh_Δh, refname ...
450 );
451 Δh_pre = (Power_est_sh - HeatLoss_pre)./mdot_est_sh;
452
453 X = [];
454 X(:,1) = data_sh(:,9)-data_sh(:,35);
455 X(:,2) = T_cond_sh(:,1)-data_sh(:,35);
456 Comp.COV_X_HL = covariance_adj(X);
457 Comp.HL_limit = max(diag(X/(X'*X)*X'));
458
459 %% Statistics
460
461 r2.r2_m = 1 - sum((mdot_fil_data - mdot_est_fil).^2)./...
462     sum((mdot_fil_data - mean(mdot_fil_data)).^2);
463 r2.r2_P = 1 - sum((Power_est - data_sh(:,24)).^2)./...
```

```
464     sum((data_sh(:,24) - mean(data_sh(:,24))).^2);
465 r2.r2_Δh = 1 - sum((Δh_pre - Δh).^2)./...
466     sum((Δh - mean(Δh)).^2);
467 r2.r2_final = r2.r2_m*r2.r2_P*r2.r2_Δh;
468
469 r2.r2_m_high = 0;
470 r2.r2_P_high = 0;
471 r2.r2_Δh_high = 0; % redundant
472 r2.r2_high_final = 0; % redundant
473
474 % assignment
475 Comp.b1 = b1;
476 Comp.C = C;
477 Comp.poly_n_factor = 0;
478 Comp.f_HL_factor = f_HL_factor;
479 Comp.P_evap_rated = P_evap_rated;
480 Comp.max_HL_ratio = max(abs(1.-data_sh(:,23).*Δh./data_sh(:,24)));
481 Comp_high.b1 = 0; % redundant
482 Comp_high.C = 0; % redundant
483 Comp_high.poly_n_factor = 0; % redundant
484 Comp_high.f_HL_factor = 0; % redundant
485 Comp_high.P_evap_rated = 0; % redundant
486 Comp_high.max_HL_ratio = 0; % redundant
487
488 %% Saving files
489 save(savefilename);
```

```
490 movefile(savefilename,strcat(foldername,'\'));
491
492 end
493
494 function [residual grad_2 Power_est] = Power_estimation(...
495     C, b1, weigh, data_II, n_poly, P_evap_rated ...
496 )
497 [p q] = size(data_II);
498 grad_2 = zeros(5,1);
499 P_evap = data_II(:,1);
500 P_cond = data_II(:,2);
501 poly_n = n_poly;
502 Power_est = (b1(1)+b1(2).*(P_cond./P_evap).^(1./poly_n) +...
503     b1(3).*(P_cond-P_evap))./(poly_n-1).*poly_n.*P_evap.*1000.* ...
504     ((P_cond./P_evap).^((poly_n-1)./poly_n)-1)./(...
505         C(1)+C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
506         C(4).*exp(C(5).*P_cond./P_evap_rated) ...
507     );
508 residual = (1./weigh.*(Power_est./data_II(:,24)-1))' *...
509     ((Power_est./data_II(:,24)-1));
510 ori = 2.*1./weigh.*(Power_est./data_II(:,24)-1)./data_II(:,24).* ...
511     (...
512         b1(1)+b1(2).*(P_cond./P_evap).^(1./poly_n) +...
513         b1(3).*(P_cond-P_evap) ...
514     )./(poly_n-1).*poly_n.*P_evap.*1000.* ...
515     ((P_cond./P_evap).^((poly_n-1)./poly_n)-1);
```

```
516 grad_2(1,1) = -sum(ori./(...
517     C(1)+C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
518     C(4).*exp(C(5).*P_cond./P_evap_rated) ...
519 ).^2);
520 grad_2(2,1) = -sum(ori./(C(1)+...
521     C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
522     C(4).*exp(C(5).*P_cond./P_evap_rated)).^2.* ...
523     exp(C(3).*P_evap./P_evap_rated));
524 grad_2(3,1) = -sum(ori./(C(1)+...
525     C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
526     C(4).*exp(C(5).*P_cond./P_evap_rated)).^2.* ...
527     P_evap./P_evap_rated.*C(2).*exp(C(3).*P_evap./P_evap_rated));
528 grad_2(4,1) = -sum(ori./(C(1)+...
529     C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
530     C(4).*exp(C(5).*P_cond./P_evap_rated)).^2.* ...
531     exp(C(5).*P_cond./P_evap_rated));
532 grad_2(5,1) = -sum(ori./(C(1)+...
533     C(2).*exp(C(3).*P_evap./P_evap_rated) + ...
534     C(4).*exp(C(5).*P_cond./P_evap_rated)).^2.* ...
535     P_cond./P_evap_rated.*C(4).*exp(C(5).*P_cond./P_evap_rated));
536 end
```

N.3 Hot Gas Line Modeling

```
1 function System = HotGas_minimization_v033_main()
2
3 % Main function to model the hot gas line
```

```
4
5  % define function and folder name
6  function_name = 'HotGas_minimization';
7  version_main = '033';
8  version_regress = '028';
9  version_plot = '001';
10 foldername = strcat(function_name,'_v',version_main);
11 regressname = strcat(function_name,'_v',version_regress);
12 plotname = strcat(function_name,'_plot_v',version_plot);
13 regress_func = str2func(['@(System, libname, version_number)',...
14     regressname,'(System, libname, version_number)']);
15 plot_func = str2func(['@(System, libname, version_number)',...
16     plotname,'(System, libname, version_number)']);
17 mkdir(foldername);
18 savefilename = strcat(foldername,'.mat');
19
20 % Module Definition
21 % Define the linset lengths and inner diameters [m, m]
22 % Hot gas line
23 HotGas.line = 0;
24 HotGas.dia = 0;
25 HotGas.ΔP = 0;
26 HotGas.HeatLossUAtomr = 0;
27 HotGas.mdot_rated = 0;
28 HotGas.V_rated = 0;
29 HotGas.rho_rated = 0;
```

```
30 HotGas.C = zeros(5,1);
31 HotGas.C_Q = zeros(4,1);
32 HotGas.Δh_rated = 0;
33 HotGas.ΔT_rated = 0;s
34
35 % Set information
36 % re-file at this stage to avoid problems in parfor loop
37 % Boshen 3-ton R410A packaged FXO system (Scroll)
38 % (Shen_030_R410A_packaged_FXO_Scroll)
39 i = 1;
40 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
41 System(i).othername = System(i).name;
42 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
43 System(i).foldername = 'Boshen';
44 System(i).worksheetname = 'SI (no_invalid_CA)';
45 System(i).refname = 'R410A';
46 System(i).Standard_Airflow_evap = 0;
47 System(i).Fan_Upstream_evap = 0;
48 System(i).Standard_Airflow_cond = 1;
49 System(i).Fan_Upstream_cond = 1;
50 System(i).Heating = 0;
51 System(i).Combined = 0;
52 System(i).Diameter = 0.00853;
53 System(i).Tube_Length = 1.67075;
54 System(i).Ntube = 40;
55 System(i).Pc = 4901;
```

```
System(i).HotGas = HotGas;
System(i).HotGas.line = 0.4974;
System(i).HotGas.dia = 0.01128;

% Boshen 3-ton R410A split FXO system (Recip)
% (Shen_030_R410A_split_FXO_Recip)
% for compressor modeling, merged with TXV system data
i = i + 1;
System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
System(i).othername = System(i).name;
System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
System(i).foldername = 'Boshen_3tonR410ASplitFXO';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4901;
System(i).HotGas = HotGas;
System(i).HotGas.line = 0.1785 + 0.08993 + 1.179;
```

```
System(i).HotGas.dia = 0.01128;

% Boshen 5-ton R407C packaged FXO system (Scroll)
% (Shen_050_R407C_packaged_FXO_Scroll)
i = i + 1;
System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R407C';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00693928;
System(i).Tube_Length = 2.282;
System(i).Ntube = 56;
System(i).Pc = 3786;
System(i).HotGas = HotGas;
System(i).HotGas.line = 0;
System(i).HotGas.dia = 0.01128;

% Breuker 3-ton R22 packaged FXO system (Recip)
```

```
% (Breuker_030_R22_packaged_FXO_Recip)
i = i + 1;
System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
System(i).othername = System(i).name;
System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
System(i).foldername = 'Breuker';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).HotGas = HotGas;
System(i).HotGas.line = 0.4974;
System(i).HotGas.dia = 0.01128;

% Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
% (NIST_025_R410A_split_TXV_Scroll)
i = i + 1;
```

```
134 System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
135 System(i).othername = System(i).name;
136 System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
137 System(i).foldername = 'NIST';
138 System(i).worksheetname = 'SI (no_invalid_CA)';
139 System(i).refname = 'R410A';
140 System(i).Standard_Airflow_evap = 1;
141 System(i).Fan_Upstream_evap = 1;
142 System(i).Standard_Airflow_cond = 1;
143 System(i).Fan_Upstream_cond = 1;
144 System(i).Heating = 0;
145 System(i).Combined = 0;
146 System(i).Diameter = 0.0042;
147 System(i).Tube_Length = 1.778;
148 System(i).Ntube = 64;
149 System(i).Pc = 4901;
150 System(i).HotGas = HotGas;
151 System(i).HotGas.line = 0;
152 System(i).HotGas.dia = 0.015875;
153
154 % Harms 5-ton R22 packaged TXV system (Scroll)
155 % (Harms_050_R22_packaged_TXV_Scroll_Heating)
156 i = i + 1;
157 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
158 System(i).othername = System(i).name;
159 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
```

```
160 System(i).foldername = 'Harms_050tonR22PackagedTXV';
161 System(i).worksheetname = 'SI (no_invalid_CA)';
162 System(i).refname = 'R22.fld';
163 System(i).Standard_Airflow_evap = 0;
164 System(i).Fan_Upstream_evap = 0;
165 System(i).Standard_Airflow_cond = 1;
166 System(i).Fan_Upstream_cond = 1;
167 System(i).Heating = 0;
168 System(i).Combined = 0;
169 System(i).Diameter = 0.00889;
170 System(i).Tube_Length = 1.629;
171 System(i).Ntube = 64;
172 System(i).Pc = 4990;
173 System(i).HotGas = HotGas;
174 System(i).HotGas.line = 0;
175 System(i).HotGas.dia = 0.01128;
176
177 % Kim 3-ton R410A split TXV system (Scroll) (HCA3) (HCA3_030_R410A_split_TXV_Scroll)
178 i = i + 1;
179 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
180 System(i).othername = System(i).name;
181 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
182 System(i).foldername = 'Kim-HCA3';
183 System(i).worksheetname = 'SI (no_invalid_CA)';
184 System(i).refname = 'R410A';
185 System(i).Standard_Airflow_evap = 0;
```

```
186 System(i).Fan_Upstream_evap = 0;
187 System(i).Standard_Airflow_cond = 1;
188 System(i).Fan_Upstream_cond = 1;
189 System(i).Heating = 0;
190 System(i).Combined = 0;
191 % assumed
192 System(i).Diameter = 0.00849;
193 System(i).Tube_Length = 2.255;
194 System(i).Ntube = 32;
195 System(i).Pc = 4901;
196 System(i).HotGas = HotGas;
197 System(i).HotGas.line = 0;
198 System(i).HotGas.dia = 0.01128;
199
200 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
201 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
202 i = i + 1;
203 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
204 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
205 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
206 System(i).foldername = 'Kim-HCA3-Heating';
207 System(i).worksheetname = 'SI (no_invalid_CA)';
208 System(i).refname = 'R410A';
209 System(i).Standard_Airflow_evap = 0;
210 System(i).Fan_Upstream_evap = 0;
211 System(i).Standard_Airflow_cond = 1;
```

```
212 System(i).Fan_Upstream_cond = 1;
213 System(i).Heating = 1;
214 System(i).Combined = 0;
215 % assumed
216 System(i).Diameter = 0.00849;
217 System(i).Tube_Length = 2.255;
218 System(i).Ntube = 32;
219 System(i).Pc = 4901;
220 System(i).HotGas = HotGas;
221 System(i).HotGas.line = 0.1785 + 0.08993 + 1.179;
222 System(i).HotGas.dia = 0.01128;
223
224 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
225 % (YSA_030_R22_split_TXV_Scroll)
226 i = i + 1;
227 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
228 System(i).othername = System(i).name;
229 System(i).filename = 'Kim_Cycle_data_R22_YSA.xlsx';
230 System(i).foldername = 'Kim-YSA';
231 System(i).worksheetname = 'SI (no_invalid_CA)';
232 System(i).refname = 'R22.fld';
233 System(i).Standard_Airflow_evap = 0;
234 System(i).Fan_Upstream_evap = 0;
235 System(i).Standard_Airflow_cond = 1;
236 System(i).Fan_Upstream_cond = 1;
237 System(i).Heating = 0;
```

```
System(i).Combined = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).HotGas = HotGas;
System(i).HotGas.line = 0;
System(i).HotGas.dia = 0.01128;

% Kim 3-ton R22 split TXV system (Scroll) (YSA)
% (YSA_030_R22_split_TXV_Scroll_Heating)
i = i + 1;
System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
System(i).foldername = 'Kim-YSA-Heating';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 1;
System(i).Combined = 0;
% assumed
```

```
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).HotGas = HotGas;
System(i).HotGas.line = 0.1785 + 0.08993 + 1.179;
System(i).HotGas.dia = 0.01128;

% Kim 3-ton R22 split TXV system (Scroll) (YKC)
% (YKC_030_R22_split_TXV_Scroll)
i = i + 1;
System(i).name = 'YKC_030_R22_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
System(i).foldername = 'Kim-YKC';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
```

```
290 System(i).Ntube = 32;
291 System(i).Pc = 4990;
292 System(i).HotGas = HotGas;
293 System(i).HotGas.line = 0;
294 System(i).HotGas.dia = 0.01128;
295
296 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
297 % (TM_040_R410A_packaged_TXV_Recip)
298 i = i + 1;
299 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
300 System(i).othername = System(i).name;
301 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
302 System(i).foldername = 'Kim-TM';
303 System(i).worksheetname = 'SI';
304 System(i).refname = 'R410A';
305 System(i).Standard_Airflow_evap = 0;
306 System(i).Fan_Upstream_evap = 0;
307 System(i).Standard_Airflow_cond = 0;
308 System(i).Fan_Upstream_cond = 0;
309 System(i).Heating = 0;
310 System(i).Combined = 0;
311 System(i).Accumulator = 0;
312 % assumed
313 System(i).Diameter = 0.00889;
314 System(i).Tube_Length = 1.629;
315 System(i).Ntube = 64;
```

```
316 System(i).Pc = 4901;
317 System(i).HotGas = HotGas;
318 System(i).HotGas.line = 0;
319 System(i).HotGas.dia = 0.01128;
320
321 % other settings for storage
322 m = length(System);
323 for i = 1:m
324     System(i).mainfoldername = foldername;
325     System(i).r2_Q = 0;
326     System(i).r2_ΔP = 0;
327 end
328
329 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
330 libname ='lib'; % Name of library
331
332 % start calculation
333 % m = 2;
334 % for i = 1:m
335 no_plot = zeros(m,1);
336 % make an error statment
337 try
338     parfor i = 1:m
339         System(i) = 1; % a statement leads to error
340     end
341 catch err
```

```
342 end
343 for i = 1:m
344     error_statement(i) = err;
345 end
346 % parfor i = 1:m
347 for i = 1:m
348     try
349         disp(['Starting to run ',System(i).name]);
350         [HotGas r2_ΔP r2_Q] = regress_func(...
351             System(i), libname, version_main ...
352         );
353         System(i).HotGas = HotGas;
354         System(i).r2_Q = r2_Q;
355         System(i).r2_ΔP = r2_ΔP;
356         disp(['Finishing simulating ',System(i).name]);
357     catch err
358         disp(['Problem found in system ',System(i).name]);
359         no_plot(i,1) = 1;
360         error_statement(i) = err;
361     end
362 end
363 save(savefilename);
364
365 % plot cannot be done in parfor. Do it in a for loop
366 for i = 1:m
367     if no_plot(i,1) == 0
```

```
368         disp(['Starting to plot ',System(i).name]);
369         plot_func(System(i), libname, version_main);
370         disp(['Finishing plotting ',System(i).name]);
371     end
372 end
373 
374 save(savefilename);
375 movefile(savefilename,strcat(foldername,'\'));
376 
377 try
378     matlabpool close; % if catch an error, close and reset it
379 end
380 try
381     unloadlibrary lib;
382 end
```

```
1 function [PipeLineClass r2_ΔP r2_Q] = HotGas_minimization_v028(...
2     System,libname,version_number ...
3 )
4 
5 % Function to train hot gas line model
6 
7 %% Data reading
8 bin_num = 10; % number of bins for histogram
9 PipeLineClass = System.HotGas;
```

```
filename = System.filename;
worksheetname = System.worksheetname;
refri = System.refname;
version = strcat(System.name,'_',version_number);
savefilename = strcat('HotGas_minimization_v',version,'.mat');
foldername = strcat(strcat(pwd,'\',System.mainfoldername,'\'),...
    strcat('HotGas_minimization_v',version));
mkdir(foldername);

% read Q_gain_observation_v010 to find points to be removed
Q_gain_version = '030';
Q_gain_name = [...
    'Q_gain_observation_v',System.name,'_',Q_gain_version ...
];
Q_gain_raw = open([...
    'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,...
    '\',Q_gain_name,'.mat' ...
]);
k = length(Q_gain_raw.index_removed);
data2 = Q_gain_raw.data;
code = Q_gain_raw.code;
fault = Q_gain_raw.fault;
for i = k:-1:1 % reverse for correct indexing
    data2(Q_gain_raw.index_removed(i),:) = [];
    code(Q_gain_raw.index_removed(i),:) = [];
    fault(Q_gain_raw.index_removed(i),:) = [];
```

```
end
[n dummy] = size(data2);
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(n,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(n,1);
end
i = 1;
while i ≤ n
    if ¬isempty(cell2mat(cell_VL(i))) || ...
        ¬isempty(cell2mat(cell_NC(i)))
        data2(i,:) = [];
        fault(i,:) = [];
        code(i,:) = [];
        cell_VL(i,:) = [];
        cell_NC(i,:) = [];
        i = i - 1;
        n = n - 1;
    end
    i = i + 1;
end
[n dummy] = size(data2);

```

```
% for Compressor
version_Comp = '075';
version_func = '045';
name_Comp = 'Compressor_regression_v';
component_name = strcat(name_Comp,System.othername,'_',version_Comp);
foldername_comp = strcat(strcat(...
    pwd,'\',name_Comp,version_Comp,'\',component_name ...
));
raw = open(strcat(...
    foldername_comp,'\',strcat(component_name,'.mat') ...
));
Comp_para = raw.Comp;
Comp_func = str2func([...
    '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
    'Compressor_v',version_func, ...
    '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
]);

%% HotGas Line
data_HG = data2; % assign data to avoid re-reading later

% Obtain Superheat, Mass Flow Rate and Pressures
j = 1;
for i = 1:n
    SC_check = propertyRH(...
        'T','P',data_HG(i,5),'Q',0,refri ...
```

```
88      ) - data_HG(i,13);
89      SC_check_II = propertyRH(...
90          'T','P',data_HG(i,4),'Q',0,refri ...
91      ) - data_HG(i,12);
92      SH_comp = data_HG(i,9) - propertyRH(...
93          'T','P',data_HG(i,1),'Q',1,refri ...
94      );
95      SH_evap = data_HG(i,16) - ...
96          propertyRH('T','P',data_HG(i,8),'Q',1,refri);
97      if (((SC_check_II ≥ 3)||(...
98              SH_evap>1&&SC_check_II≥1 ...
99          )||(SC_check_II≥1)) && data_HG(i,23) > 0) || ...
100             SH_comp>1
101         if ¬(((SC_check_II ≥ 3)||(...
102             SH_evap>1&&SC_check_II≥1 ...
103         )||(SC_check_II≥1)) && data_HG(i,23) > 0)
104             if System.Heating==0
105                 Tamb_a = data_HG(i,21);
106             else
107                 Tamb_a = data_HG(i,17);
108             end
109             data_HG(i,23) = Comp_func(...
110                 data_HG(i,1), propertyRH(...
111                     'H','T',data_HG(i,9),'P',data_HG(i,1),refri ...
112                 ), data_HG(i,2), Tamb_a, data_HG(i,10), ...
113                 Comp_para, refri ...
```

```
114             );
115         end
116         mdot(j,1) = data_HG(i,23);
117         SH(j,1) = data_HG(i,10) - ...
118             propertyRH('T','P',data_HG(i,2),'Q',1,refri);
119         ΔP(j,1) = data_HG(i,2) - data_HG(i,3);
120         rho(j,1) = propertyRH(...
121             'D','T',data_HG(i,10),'P',data_HG(i,2),refri ...
122         );
123         rho_out(j,1) = propertyRH(...
124             'D','T',data_HG(i,11),'P',data_HG(i,3),refri ...
125         );
126         V(j,1) = propertyRH(...
127             'V','T',data_HG(i,10),'P',data_HG(i,2),refri ...
128         );
129         T(j,1) = data_HG(i,10);
130         if System.Heating==0
131             Tamb(j,1) = data_HG(i,21);
132         else
133             Tamb(j,1) = data_HG(i,17);
134         end
135         h(j,1) = propertyRH(...
136             'H','T',data_HG(i,10),'P',data_HG(i,2),refri ...
137         );
138         hout(j,1) = ...
139             propertyRH('H','T',data_HG(i,11),'P',data_HG(i,3),refri);
```

```
140         rho_out_est(j,1) = ...
141             propertyRH('D','P',data_HG(i,2),'H',hout(j,1),refri);
142         data_h(j,:) = data_HG(i,:);
143         data_Q(j,:) = data_HG(i,:);
144         j = j + 1;
145     end
146 end
147 mdot_rated = mdot(1,1);
148 ΔP_av = mean(ΔP);
149 rho_rated = mean(rho);
150 V_rated = mean(V);
151 SH_limit = 1;
152 SC_limit = 3;
153
154 save(savefilename);
155 movefile(savefilename,strcat(foldername,'\'));
156
157 options = optimset('MaxFunEvals',10000,'MaxIter',1000);
158 X1 = (mdot./mdot_rated);
159 X2 = (V./V_rated);
160 X3 = (rho./rho_rated);
161 X4 = (mdot./mdot_rated).^2.*(1./rho_out_est - 1./rho).*rho_rated;
162 C = zeros(4,1);
163 A = -eye(length(C));
164 A(3,3) = 0;
165 A(4,4) = 0;
```

```
166 b = zeros(length(C),1);
167 options = optimset(...
168     'MaxFunEvals',20000,'algorithm','active-set','MaxIter',5000,...
169     'Display','off' ...
170 );
171 ΔP(ΔP<0) = 0;
172 if length(ΔP) > 1 & sum(ΔP.^2)>1.e-8
173     weigh = weighing_function_v000(ΔP, bin_num);
174 elseif sum(ΔP.^2)≤1.e-8
175     weigh = ones(length(ΔP),1);
176 else
177     weigh = 1;
178 end
179 C = [0;1;0];
180 C = fmincon(@(C) PressureDrop_residual(...
181     C, X1, X2, X3, X4, ΔP, weigh ...
182 ), C, [], [], [], [], [0 1 0], [Inf 2 1], [],options);
183 if sum(isnan(C))
184     C = zeros(3,1);
185     options = optimset(...
186         'MaxFunEvals',20000,'algorithm','sqp',...
187         'MaxIter',5000,'Display','off' ...
188     );
189     C = fmincon(@(C) PressureDrop_residual(...
190         C, X1, X2, X3, X4, ΔP, weigh ...
191     ), C, [], [], [], [], [0 1 0], [Inf 2 1], [],options);
```

```
192 end
193 C(4,1) = -1;
194 ΔP_est = PressureDrop(C, X1, X2, X3, X4);
195 dev = ΔP_est - ΔP;
196 r2_ΔP = 1 - sum(dev.^2)/sum((ΔP - mean(ΔP)).^2);
197 max_dev = round(max(abs(dev)));
198 X = [];
199 X(:,1) = (X1.^C(2).*X2.^C(3).*X3.^C(4));
200 X(:,2) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X1);
201 X(:,3) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X2);
202 X(:,4) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X3);
203 COV_X = (X'*X)^(-1);
204 PipeLineClass.COV_X_ΔP = COV_X;
205 PipeLineClass.ΔP_limit = max(diag(X/(X'*X)*X'));
206
207 % more data filtering
208 i = 1;
209 while i≤ n
210     try
211         if h(i,1) - hout(i,1) == 0 % avoid division be zero error
212         elseif (T(i,1) - Tamb(i,1))/(h(i,1)-hout(i,1)) > 0
213         else
214             h(i,:) = [];
215             hout(i,:) = [];
216             mdot(i,:) = [];
217             T(i,:) = [];
```

```
218             Tamb(i,:) = [];
219             data_Q(i,:) = [];
220             i = i - 1;
221             n = n - 1;
222         end
223         i = i + 1;
224     catch err
225         % data points exceeded
226         clear err;
227         i = i + 1;
228     end
229 end
230 
231 % estimate the heat loss model
232 C_Q = [2 1]';
233 if length(mdot.*(h-hout)) > 1 & sum((mdot.*(h-hout)).^2)>1.e-8
234     weigh = weighing_function_v000(mdot.*(h-hout), bin_num);
235 elseif sum((mdot.*(h-hout)).^2)≤1.e-8
236     weigh = ones(length(mdot.*(h-hout)),1);
237 else
238     weigh = 1;
239 end
240 if ¬isempty(mdot.*(h-hout))& sum((mdot.*(h-hout)).^2)>1.e-8
241     outFD_HG = gradientcheck(@(C_Q) HeatLoss(...
242         C_Q, h, hout, mdot, T, Tamb, mdot_rated, h(1)-hout(1), ...
243         weigh ...
```

```
244     ), C_Q);
245     Q = mdot.*(h-hout);
246     DiffT = T-Tamb;
247     max_index = find(DiffT==max(DiffT));
248     Th = T(max_index);
249     Tl = Tamb(max_index);
250     Qmax = Q(max_index);
251     Ra_max = Rayleigh(Th, Tl, System.HotGas.dia/2);
252     [HTC_max, n_max] = Natural_HTC_air(Ra_max, System.HotGas.dia);
253     C_Q_1_max = Qmax/(((Th-Tl)./Th).^n_max.* ...
254         (Th-Tl)/(T(1)-Tamb(1))*mdot_rated*(h(1)-hout(1)));
255     C_Q = fmincon(@(C_Q) HeatLoss(...
256         C_Q, h, hout, mdot, T, Tamb, mdot_rated, h(1)-hout(1), ...
257         weigh ...
258     ), C_Q, [], [], [], [], [0 0.058], [C_Q_1_max 0.333], ...
259         [], optimset('GradObj', 'on', 'DerivativeCheck', 'on'));
260     Q_est = C_Q(1).*((T-Tamb)./T).^C_Q(2).*(T-Tamb)/(T(1)-Tamb(1))*...
261         mdot_rated*(h(1)-hout(1));
262     Δh_rated = h(1) - hout(1);
263     ΔT_rated = T(1)-Tamb(1);
264     dev_Q = (Q_est - Q)./Q;
265     r2_Q = 1 - sum((Q_est - Q).^2)/sum((Q - mean(Q)).^2);
266     X = [];
267     X(:,1) = ((T-Tamb)./T).^C_Q(2).*(T-Tamb)/(T(1)-Tamb(1)) * ...
268         mdot_rated*(h(1)-hout(1));
269     X(:,2) = C_Q(1).*(abs(T-Tamb)./T).^C_Q(2).* ...
```

```
270          (T-Tamb)/(T(1)-Tamb(1))*mdot_rated* ...
271          (h(1)-hout(1)).*log(abs(T-Tamb)./T);
272      COV_X = (X'*X)^(-1);
273      PipeLineClass.COV_X_Q = COV_X;
274      PipeLineClass.Q_limit = max(diag(X/(X'*X)*X'));
275  else
276      C_Q = [0 0];
277      Δh_rated = 0;
278      ΔT_rated = 0;
279      dev_Q = 0;
280      Q = 0;
281      Q_est = 0;
282      r2_Q = -9999;
283      PipeLineClass.COV_X_Q = zeros(length(C_Q),length(C_Q));
284      PipeLineClass.Q_limit = Inf;
285  end
286  max_dev_Q = round(max(abs(dev_Q)*100));
287
288  PipeLineClass.mdot_rated = mdot_rated;
289  PipeLineClass.V_rated = V_rated;
290  PipeLineClass.rho_rated = rho_rated;
291  PipeLineClass.C = C;
292  PipeLineClass.C_Q = C_Q;
293  PipeLineClass.Δh_rated = Δh_rated;
294  PipeLineClass.ΔT_rated = ΔT_rated;
295
```

```
296 save(savefilename);
297 movefile(savefilename,strcat(foldername,'\'));
298
299 end
300
301 function [residual_Q, grad_Q] = HeatLoss(...
302         C_Q, h, hout, mdot, T, Tamb, Δh_rated, mdot_rated, ...
303         weigh ...
304     )
305     residual_Q = ((h-hout).*mdot/mdot_rated/(Δh_rated) - ...
306         C_Q(1).*(abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1)));
307     n_Q = length(residual_Q);
308     gra_Q_ori(1:n_Q,1) = -2.*(residual_Q).*1./weigh;
309     residual_Q = sum(1./weigh.*residual_Q.^2);
310     grad_Q(1,1) = sum(...
311         gra_Q_ori(1:n_Q,1).* ...
312         (...
313             1.*(abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1)) ...
314         )...
315     );
316     grad_Q(2,1) = sum(gra_Q_ori(1:n_Q,1).*(...
317         C_Q(1).*(T-Tamb)./(T(1)-Tamb(1)).*log(abs(T-Tamb)./T).* ...
318         (abs(T-Tamb)./T).^C_Q(2) ...
319     ));
320 end
321
```

```
322 function residual = PressureDrop_residual(...
323     C, X1, X2, X3, X4, ΔP, weigh ...
324 )
325 n = length(X1);
326 residual = 0;
327 for i = 1:n
328     residual = residual + 1/weigh(i,1)*(ΔP(i,1) - ...
329         PressureDrop(C, X1(i,1), X2(i,1), X3(i,1), X4(i,1))).^2;
330 end
331 end
332
333 function ΔP_est = PressureDrop(C, X1, X2, X3, X4)
334 % function to estimate the pressure drop
335 ΔP_est = (C(1)*X1.^C(2).*X2.^C(3)./X3);
336 end
```

N.4 Condenser Modeling

```
1 function System = Condenser_minimization_v047_main()
2
3 % Function to traing condenser model
4
5 % define function and folder name
6 function_name = 'Condenser_minimization';
7 version_main = '047';
8 version_regress = '042';
9 version_plot = '003';
```

```
10 foldername = strcat(function_name,'_v',version_main);
11 regressname = strcat(function_name,'_v',version_regress);
12 plotname = strcat(function_name,'_plot_v',version_plot);
13 regress_func = str2func(['@(System, libname, version_number)',...
14     regressname,'(System, libname, version_number)']);
15 plot_func = str2func(['@(System, libname, version_number)',...
16     plotname,'(System, libname, version_number)']);
17 mkdir(foldername);
18 savefilename = strcat(foldername,'.mat');
19
20 % Condenser module definition
21 % Air-side rated heat transfer coefficient  [W/m^2-K]
22 Cond.para.U_a = 0;
23 % Adjustment on air-side heat transfer coefficient by air
24 % mass flow
25 Cond.para.n = 0;
26 % Rated refrigerant-side heat transfer coefficient of the
27 % superheat region [W/m^2-K]
28 Cond.para.U_r_sh = 0;
29 % Rated refrigerant-side heat transfer coefficient of the
30 % two-phase region [W/m^2-K]
31 Cond.para.U_r_tp = 0;
32 Cond.para.U_r_sc = 0;
33 % Lumped parameter for the effect of the property on the
34 % superheated heat transfer coefficient at rated condition
35 Cond.para.K_sh = 0;
```

```
% Lumped parameter for the effect of the property on the
% subcooled heat transfer coefficient at rated condition
Cond.para.K_sc = 0;
% Rated air mass flow rate for condneser model [kg/s]
Cond.para.m_a = 0;
% Rated refrigerant mass flow rate for the condenser model [kg/s]
Cond.para.m_r = 0;
% Inner diameter of tube [m]
Cond.para.Din = 0;
% Total length of heat exchanger pipes [m]
Cond.para.Lt = 0;
% Inner surface area of condenser [m^2]
Cond.para.A_r = 0;
% Average power consumption of condenser fan [W]
Cond.para.FanPower = 0;

% Average pressure drop across condenser from data [kPa]
Cond.para.ΔP_const = 0;
% Rated subcooling for condenser [K]
Cond.para.SC_rated = 0;

Cond.para.ΔP.C = zeros(1,5);
Cond.para.ΔP.mdot_rated = 0;
Cond.para.ΔP.rho_rated = 0;
Cond.para.ΔP.rho_out_rated = 0;
Cond.para.ΔP.mu_v_rated = 0;
```

```
62 Cond.para.ΔP_max = 0;
63 Cond.para.Vdot_a = 0;
64
65 % Set information
66 % re-file at this stage to avoid problems in parfor loop
67
68 % Breuker 3-ton R22 packaged FXO system (Recip)
69 % (Breuker_030_R22_packaged_FXO_Recip)
70 i = 1;
71 System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
72 System(i).othername = System(i).name;
73 System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
74 System(i).foldername = 'Breuker';
75 System(i).worksheetname = 'SI (no_invalid_CA)';
76 System(i).refname = 'R22.fld';
77 System(i).Standard_Airflow_evap = 0;
78 System(i).Fan_Upstream_evap = 0;
79 System(i).Standard_Airflow_cond = 1;
80 System(i).Fan_Upstream_cond = 1;
81 System(i).Heating = 0;
82 System(i).Combined = 0;
83 % assumed
84 System(i).Diameter = 0.00849;
85 System(i).Tube_Length = 2.255;
86 System(i).Ntube = 32;
87 System(i).Pc = 4990;
```

```
88
89 % Boshen 3-ton R410A packaged FXO system (Scroll)
90 % (Shen_030_R410A_packaged_FXO_Scroll)
91 i = i+1;
92 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
93 System(i).othername = System(i).name;
94 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
95 System(i).foldername = 'Boshen';
96 System(i).worksheetname = 'SI (no_invalid_CA)';
97 System(i).refname = 'R410A';
98 System(i).Standard_Airflow_evap = 0;
99 System(i).Fan_Upstream_evap = 0;
100 System(i).Standard_Airflow_cond = 1;
101 System(i).Fan_Upstream_cond = 1;
102 System(i).Heating = 0;
103 System(i).Combined = 0;
104 System(i).Diameter = 0.00853;
105 System(i).Tube_Length = 1.67075;
106 System(i).Ntube = 40;
107 System(i).Pc = 4901;
108
109 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
110 % (TM_040_R410A_packaged_TXV_Recip)
111 i = i + 1;
112 System(i).name =  'TM_040_R410A_packaged_TXV_Recip';
113 System(i).othername = System(i).name;
```

```
114 System(i).filename =  'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
115 System(i).foldername = 'Kim-TM';
116 System(i).worksheetname = 'SI';
117 System(i).refname = 'R410A';
118 System(i).Standard_Airflow_evap = 0;
119 System(i).Fan_Upstream_evap = 0;
120 System(i).Standard_Airflow_cond = 1;
121 System(i).Fan_Upstream_cond = 1;
122 System(i).Heating = 0;
123 System(i).Combined = 0;
124 % assumed
125 System(i).Diameter = 0.0064;
126 System(i).Tube_Length = 2.1590;
127 System(i).Ntube = 82;
128 System(i).Pc = 4901;
129
130 % Boshen 5-ton R407C packaged FXO system (Scroll)
131 % (Shen_050_R407C_packaged_FXO_Scroll)
132 i = i + 1;
133 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
134 System(i).othername = System(i).name;
135 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
136 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
137 System(i).worksheetname = 'SI (no_invalid_CA)';
138 System(i).refname = 'R407C';
139 System(i).Standard_Airflow_evap = 0;
```

```
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00693928;
System(i).Tube_Length = 2.282;
System(i).Ntube = 56;
System(i).Pc = 3786;

% Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
% (NIST_025_R410A_split_TXV_Scroll)
i = i + 1;
System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
System(i).foldername = 'NIST';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 1;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.0042;
```

```
System(i).Tube_Length = 1.778;
System(i).Ntube = 64;
System(i).Pc = 4901;

% Harms 5-ton R22 packaged TXV system (Scroll)
% (Harms_050_R22_packaged_TXV_Scroll_Heating)
i = i + 1;
System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
System(i).foldername = 'Harms_050tonR22PackagedTXV';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00889;
System(i).Tube_Length = 1.629;
System(i).Ntube = 64;
System(i).Pc = 4990;

% Kim 3-ton R410A split TXV system (Scroll) (HCA3)
% (HCA3_030_R410A_split_TXV_Scroll)
```

```
192 i = i + 1;
193 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
194 System(i).othername = System(i).name;
195 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
196 System(i).foldername = 'Kim-HCA3';
197 System(i).worksheetname = 'SI (no_invalid_CA)';
198 System(i).refname = 'R410A';
199 System(i).Standard_Airflow_evap = 0;
200 System(i).Fan_Upstream_evap = 0;
201 System(i).Standard_Airflow_cond = 1;
202 System(i).Fan_Upstream_cond = 1;
203 System(i).Heating = 0;
204 System(i).Combined = 0;
205 % assumed
206 System(i).Diameter = 0.00849;
207 System(i).Tube_Length = 2.255;
208 System(i).Ntube = 32;
209 System(i).Pc = 4901;
210
211 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
212 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
213 i = i + 1;
214 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
215 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
216 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
217 System(i).foldername = 'Kim-HCA3-Heating';
```

```
218 System(i).worksheetname = 'SI (no_invalid_CA)';
219 System(i).refname = 'R410A';
220 System(i).Standard_Airflow_evap = 0;
221 System(i).Fan_Upstream_evap = 0;
222 System(i).Standard_Airflow_cond = 1;
223 System(i).Fan_Upstream_cond = 1;
224 System(i).Heating = 1;
225 System(i).Combined = 0;
226 % assumed
227 System(i).Diameter = 0.00849;
228 System(i).Tube_Length = 2.255;
229 System(i).Ntube = 32;
230 System(i).Pc = 4901;
231
232 % Boshen 3-ton R410A split FXO system (Recip)
233 % (Shen_030_R410A_split_FXO_Recip)
234 % for compressor modeling, merged with TXV system data
235 i = i + 1;
236 System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
237 System(i).othername = System(i).name;
238 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
239 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
240 System(i).worksheetname = 'SI (no_invalid_CA)';
241 System(i).refname = 'R410A';
242 System(i).Standard_Airflow_evap = 0;
243 System(i).Fan_Upstream_evap = 0;
```

```
244 System(i).Standard_Airflow_cond = 1;
245 System(i).Fan_Upstream_cond = 1;
246 System(i).Heating = 0;
247 System(i).Combined = 0;
248 System(i).Diameter = 0.00849;
249 System(i).Tube_Length = 2.255;
250 System(i).Ntube = 32;
251 System(i).Pc = 4901;
252
253 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
254 % (YSA_030_R22_split_TXV_Scroll)
255 i = i + 1;
256 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
257 System(i).othername = System(i).name;
258 System(i).filename = 'Kim_Cycle_data_R22_YSA.xlsx';
259 System(i).foldername = 'Kim-YSA';
260 System(i).worksheetname = 'SI (no_invalid_CA)';
261 System(i).refname = 'R22.fld';
262 System(i).Standard_Airflow_evap = 0;
263 System(i).Fan_Upstream_evap = 0;
264 System(i).Standard_Airflow_cond = 1;
265 System(i).Fan_Upstream_cond = 1;
266 System(i).Heating = 0;
267 System(i).Combined = 0;
268 % assumed
269 System(i).Diameter = 0.00849;
```

```
System(i).Tube_Length = 3.255;
System(i).Ntube = 32;
System(i).Pc = 4990;

% Kim 3-ton R22 split TXV system (Scroll) (YSA)
% (YSA_030_R22_split_TXV_Scroll_Heating)
i = i + 1;
System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
System(i).foldername = 'Kim-YSA-Heating';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 1;
System(i).Combined = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 3.255;
System(i).Ntube = 32;
System(i).Pc = 4990;

% Kim 3-ton R22 split TXV system (Scroll) (YKC)
```

```
296 % (YKC_030_R22_split_TXV_Scroll)
297 i = i + 1;
298 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
299 System(i).othername = System(i).name;
300 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
301 System(i).foldername = 'Kim-YKC';
302 System(i).worksheetname = 'SI (no_invalid_CA)';
303 System(i).refname = 'R22.fld';
304 System(i).Standard_Airflow_evap = 0;
305 System(i).Fan_Upstream_evap = 0;
306 System(i).Standard_Airflow_cond = 1;
307 System(i).Fan_Upstream_cond = 1;
308 System(i).Heating = 0;
309 System(i).Combined = 0;
310 % assumed
311 System(i).Diameter = 0.00849;
312 System(i).Tube_Length = 2.255;
313 System(i).Ntube = 32;
314 System(i).Pc = 4990;
315
316 % other settings for storage
317 m = length(System);
318 % m = 1;
319 for i = 1:m
320     System(i).mainfoldername = foldername;
321     System(i).Cond = Cond;
```

```
322     System(i).r2_Q = 0;
323     System(i).r2_ΔP = 0;
324 end
325 
326 % setting up parallel solver
327 try
328     matlabpool open 3
329 catch err
330     matlabpool close; % if catch an error, close and reset it
331     matlabpool open 3
332 end
333 
334 pctRunOnAll loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
335 libname ='lib'; % Name of library
336 
337 % start calculation
338 no_plot = zeros(m,1);
339 % make an error statment
340 try
341     parfor i = 1:m
342         System(i) = 1; % a statement leads to error
343     end
344 catch err
345 end
346 for i = 1:m
347     error_statement(i) = err;
```

```
348 end
349 parfor i = 1:m
350 % for i = 3:3
351 % for i = m:m
352     try
353         disp(['Starting to run ',System(i).name]);
354         [Cond r2_Q r2_ΔP] = regress_func(...
355             System(i), libname, version_main ...
356         );
357         System(i).Cond = Cond;
358         System(i).r2_Q = r2_Q;
359         System(i).r2_ΔP = r2_ΔP;
360         disp(['Finishing simulating ',System(i).name]);
361     catch err
362         disp(['Problem found in system ',System(i).name]);
363         try
364             disp(err.message);
365             for qq = 1:length(err.stack)
366                 disp('Function Name:')
367                 disp(err.stack(qq).name);
368                 disp('Line Number:')
369                 disp(err.stack(qq).line);
370             end
371         end
372         no_plot(i,1) = 1;
373         error_statement(i) = err;
```

```
374     end
375 end
376 save(savefilename);
377 
378 % plot cannot be done in parfor. Do it in a for loop
379 for i = 1:m
380 % for i = 3:3
381     if no_plot(i,1) == 0
382         disp(['Starting to plot ',System(i).name]);
383         plot_func(System(i), libname, version_main);
384         disp(['Finishing plotting ',System(i).name]);
385     end
386 end
387 
388 save(savefilename);
389 movefile(savefilename,strcat(foldername,'\'));
390 
391 try
392     matlabpool close; % if catch an error, close and reset it
393 end
394 try
395     unloadlibrary lib;
396 end
```

```
1 function [Cond coeffQ coeff_P] = Condenser_minimization_v042(...
```

```
    System, libname, version_number ...
)

% Function to train condenser model

% Variables:
% input:
% filename: filename of the Excel file to be output

% processing:
% UA: vector for minization variables related to UAs
% m: number of rows in ref
% n: number of columns in ref
% cost: return of cost function
% stat:
% 1st entry: sigma_tt for Q
% 2nd entry: sigma_tt for Taout

% output:
% flag: reference for successful convergence

% input: input matrix
% 01st column: estimated UAa (W/K)
% 02nd column: estimated UAsh (W/K)
% 03rd column: estimated UAtp (W/K)
% 04th column: estimated UAsc (W/K)
```

```
% 05th column: original air inlet temperature (K)
% 06th column: original volumetric airflow rate (m^3/s)
% 07th column: original atmospheric pressure (kPa)
% 08th column: original refrigerant mass flow rate (kg/s)
% 09th column: original refrigerant inlet temperature (K)
% 10th column: original refrigerant saturation temperature (K)
% 11th column: original inner diamter of the tube (m)
% 12th column: original total pipe length (m)
% 13th column: original relative humidity
% 14th column: orginal refrigerant outlet temperature (K)

% ref: reference matrix
% 01st column: original heat transfer rate (W)
% 02nd column: original volumetric airflow rate (m^3/s)
% 03rd column: original refrigenrat mass flow rate (kg/s)
% 04th column: original air inlet temperature (K)
% 05th column: original air outlet temperature (K)
% 06th column: original relative humidity
% 07th column: original superheated section length ratio
% 08th column: original two-phase section length ratio
% 09th column: original subcooled section length ratio
% 10th column: original UAa (W/K)
% 11th column: original UAsh (W/K)
% 12th column: original UAtp (W/K)
% 13th column: original UAsc (W/K)
% 14th column: original atmophseric pressure (kPa)
```

```
% 15th column: original refrigerant inlet pressure (kPa)
% 16th column: original refrigerant outlet pressure (kPa)
% 17th column: original refrigerant inlet temperature (K)
% 18th column: original refrigerant outlet temperature (K)
% 19th column: original subcooling (K)

% output: output matrix
% 01st column: estimated total heat transfer (W)
% 02nd column: estimated superheated heat transfer (W)
% 03rd column: estimated two-phase heat transfer (W)
% 04th column: estimated subcooled heat transfer (W)
% 05th column: estimated superheated length ratio
% 06th column: estimated two-phase length ratio
% 07th column: estimated subcooled length ratio
% 08th column: estimated outlet refrigerant temperature (K)
% 09th column: estimated inlet quality of condensing flow
% 10th column: estimated outlet quality of condensing flow
% 11th column: air mass flow rate (kg/s)

% extra: extra output matrix
% 01st column: original overall UAsh (W/K)
% 02nd column: original overall UAtp (W/K)
% 03rd column: original overall UAsc (W/K)
% 04rd column: estimated overall UAsh (W/K)
% 05th column: estimated overall UAtp (W/K)
% 06th column: estimated overall UAsc (W/K)
```

```
80
81 %% Preparation
82 % data reading and filtering
83 bin_num = 10; % number of bins for histogram
84 filename = System.filename;
85 worksheetname = System.worksheetname;
86 refname = System.refname;
87 Standard_Airflow = System.Standard_Airflow_cond;
88 Ind_Fan_Upstream = System.Fan_Upstream_cond;
89 D = System.Diameter;
90 Ltube = System.Tube_Length;
91 Ntube = System.Ntube;
92 Pc = System.Pc;
93 version = strcat(System.name,'_',version_number);
94 savefilename = strcat('Condenser_minimization_v',version,'.mat');
95 foldername = strcat(strcat(...
96     pwd,'\',System.mainfoldername,'\' ...
97 ),strcat('Condenser_minimization_v',version));
98 if ¬exist(foldername,'dir')
99     mkdir(foldername);
100 end
101
102 % read Q_gain_observation_v010 to find points to be removed
103 Q_gain_version = '029';
104 Q_gain_name = [...
105     'Q_gain_observation_v',System.name,'_',Q_gain_version ...
```

```
106 ];
107 Q_gain_raw = open([...
108     'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,'\',...
109     Q_gain_name,'.mat' ...
110 ]);
111 k = length(Q_gain_raw.index_removed);
112 ref = Q_gain_raw.data;
113 code = Q_gain_raw.code;
114 fault = Q_gain_raw.fault;
115 Q_cond_r = Q_gain_raw.Q_cond_r;
116 for i = k:-1:1 % reverse for correct indexing
117     ref(Q_gain_raw.index_removed(i),:) = [];
118     code(Q_gain_raw.index_removed(i),:) = [];
119     fault(Q_gain_raw.index_removed(i),:) = [];
120     Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
121 end
122 [pp, qq] = size(ref);
123 cell_VL = strfind(fault,'VL');
124 if isempty(cell_VL)
125     cell_VL = cell(pp,1);
126 end
127 cell_NC = strfind(fault,'NC');
128 if isempty(cell_NC)
129     cell_NC = cell(pp,1);
130 end
131 i = 1;
```

```
132 while i ≤ pp
133     if ¬isempty(cell2mat(cell_VL(i))) || ...
134         ¬isempty(cell2mat(cell_NC(i)))
135         ref(i,:) = [];
136         fault(i,:) = [];
137         code(i,:) = [];
138         cell_VL(i,:) = [];
139         cell_NC(i,:) = [];
140         Q_cond_r(i,:) = [];
141         i = i - 1;
142         pp = pp - 1;
143     end
144     i = i + 1;
145 end
146 [pp, qq] = size(ref);
147 p = pp;
148 q = qq;
149
150 [m, n] = size(ref);
151 n = n+1;
152
153 % for Compressor
154 version_Comp = '073';
155 version_func = '045';
156 name_Comp = 'Compressor_regression_v';
157 component_name = ...
```

```
158     strcat(name_Comp,System.othername,'_',version_Comp);
159 foldername_comp = strcat(strcat(...
160     pwd,'\',name_Comp,version_Comp,'\',component_name ...
161 ));
162 raw = ...
163     open(strcat(foldername_comp,'\',strcat(component_name,'.mat')));
164 Comp_para = raw.Comp;
165 Comp_func = str2func([...
166     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
167     'Compressor_v',version_func, ...
168     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
169 ]);
170
171 % Standard area: refrigerant-side area
172 Ar = Ntube*Ltube*pi*D;
173 refri = refname;
174
175 % intial guess
176 % initilaize variables
177 output = zeros(m,13);
178 input =  zeros(m,10);
179 % matching ref and input
180 input(:,1) = ones(m,1);
181 input(:,2) = ones(m,1);
182 input(:,3) = ones(m,1);
183 input(:,4) = ones(m,1);
```

```
184 input(:,5) = ref(:,21);
185 input(:,6) = ones(m,1);
186 input(:,7) = ref(:,32);
187 input(:,8) = ref(:,23);
188 input(:,9) = ref(:,11);
189 for i = 1:m
190     input(i,10) = ref(i,3);
191     input(i,11) = D;
192     input(i,12) = Ltube*Ntube;
193 end
194 input(:,13) = 1.e-8; %arbitrary
195 input(:,14) = 0;
196 input(:,15) = ref(:,12);
197 for i = 1:m
198     airinlet = calllib(...
199         'lib', 'HumAir_DLL', input(i,5), input(i,7), ...
200         1, 230, zeros(5,1) ...
201     );
202     input(i,13) = airinlet(4);
203 end
204
205 % replace air-side heat transfer rate with ref-side
206 i = 1;
207 while(i≤m)
208     % avoid incomplete dataset
209     try
```

```
210         SC_cond = propertyRH('T','P',ref(i,4),'Q',0,refri) - ref(i,12);
211         try
212             SC_TXV = propertyRH('T','P',ref(i,5),'Q',0,refri) - ...
213                 ref(i,13);
214         catch err
215             clear err;
216             SC_TXV = SC_cond;
217         end
218     catch err
219         SC_TXV = propertyRH('T','P',ref(i,5),'Q',0,refri) - ...
220             ref(i,13);
221         SC_cond = SC_TXV;
222         clear err;
223     end
224     SH = ref(i,9) - propertyRH('T','P',ref(i,1),'Q',1,refri);
225     SH_evap = ref(i,16) - propertyRH('T','P',ref(i,8),'Q',1,refri);
226
227     % filter for probelmatic upstream measurement
228     h_in_HG = propertyRH('H','T',ref(i,10),'P',ref(i,2),refri);
229     h_out_HG = propertyRH('H','T',ref(i,11),'P',ref(i,3),refri);
230     if System.Heating==0
231         Tamb = ref(i,21);
232     else
233         Tamb = ref(i,17);
234     end
235     if h_in_HG - h_out_HG ≠ 0
```

```
236         if (ref(i,10)-Tamb)/(h_in_HG - h_out_HG) < 0
237             h_out_HG = h_in_HG;
238             ref(i,11) = propertyRH(...
239                 'T','P',ref(i,3),'H',h_out_HG,refri ...
240             );
241         end
242     end
243
244     if ((SC_cond≥3)||(SH_evap>1&&SC_TXV≥1)||(SC_cond≥1)) && ...
245         ref(i,23) > 0
246     elseif SH>1
247         ref(i,23) = Comp_func(...
248             ref(i,1), propertyRH(...
249                 'H','T',ref(i,9),'P',ref(i,1),refri ...
250             ),...
251             ref(i,2), Tamb, ref(i,10), Comp_para, refri ...
252         );
253         input(i,8) = ref(i,23);
254     end
255
256     if ((SC_cond≥3)||(SH_evap>1&&SC_TXV≥1)||(SC_cond≥1)) && ...
257         ref(i,23) > 0
258         input(i,6) = ref(i,29);
259         ref(i,28) = Q_cond_r(i,1);
260     end
261     i = i + 1;
```

```
end
Cond.para.Vdot_a = Q_gain_raw.Cond_nf_Vdot_a;
i = 1;
while i ≤ m
    try
        SC_cond = propertyRH('T','P',ref(i,4),'Q',0,refri) - ...
            ref(i,12);
        try
            SC_TXV = propertyRH(...
                'T','P',ref(i,5),'Q',0,refri
            )-ref(i,13);
        catch err
            clear err;
            SC_TXV = SC_cond;
        end
    catch err
        SC_TXV = propertyRH('T','P',ref(i,5),'Q',0,refri) - ref(i,13);
        SC_cond = SC_TXV;
        clear err;
    end
    SH_evap = ref(i,16) - propertyRH('T','P',ref(i,8),'Q',1,refri);
    SH = ref(i,9) - propertyRH('T','P',ref(i,1),'Q',1,refri);
    if ((SC_cond≥3)||(SH_evap>1&&SC_TXV≥1)||(SC_cond≥1)) && ...
        ref(i,23) > 0
    elseif SH > 1 & isempty(cell2mat(strfind(fault(i,:),'CA')))
        ref(i,29) = Cond.para.Vdot_a;
```

```
288         input(i,6) = ref(i,29);
289         ref(i,28) = Q_cond_r(i,1);
290     else
291         ref(i,:) = [];
292         input(i,:) = [];
293         output(i,:) = [];
294         fault(i,:) = [];
295         code(i,:) = [];
296         Q_cond_r(i,:) = [];
297         i = i - 1;
298         m = m - 1;
299     end
300     i = i + 1;
301 end
302 ref(:,28) = Q_cond_r(:,1);
303
304 % statistical information
305 Qbar = mean(ref(1:m,28)); %mean of heat transfer rate
306 stat(1) = 0;
307 for i = 1:m
308     stat(1) = stat(1) + (ref(i,28)-Qbar)^2;
309 end
310
311 % estimating heat transfer rate
312 for i = 1:m
313     Psat = input(i,10);
```

```
314     Tsat = propertyRH('T','P',Psat,'Q',1,refri);
315     Tin = ref(i,11);
316     cpr(i,1) = propertyRH('C','P',Psat,'Q',1,refri);
317     output(i,2) = input(i,8)*cpr(i,1)*(ref(i,11)-Tsat);
318     hfg(i) = propertyRH('H','P',Psat,'Q',1,refri) - ...
319         propertyRH('H','P',Psat,'Q',0,refri);
320     output(i,3) = input(i,8)*hfg(i);
321     cpr(i,2) = propertyRH('C','P',Psat,'Q',0,refri);
322     output(i,4) = input(i,8)*cpr(i,2)*(Tsat - ref(i,12));
323 end
324
325 % for Ksc ratio
326 %UAsh
327 P_cond = input(1,10);
328 cp_rsh = cpr(1,1);
329 miu_rsh = propertyRH('V', 'P', input(1,10),'Q',1, refri);
330 k_rsh = propertyRH('L', 'P', input(1,10),'Q',1, refri);
331 Ksh = (cp_rsh*miu_rsh/k_rsh)^0.4*(input(1,8)/miu_rsh)^0.8;
332 % UAsc
333 cp_rsc = cpr(1,2);
334 miu_rsc = propertyRH('V','P',input(1,10),'Q',0,refri);
335 k_rsc = propertyRH('L','P',input(1,10),'Q',0,refri);
336 Ksc = (cp_rsc*miu_rsc/k_rsc)^0.4*(input(1,8)/miu_rsc)^0.8;
337 % the qulaity of refrigerant which gives the highest heat
338 % transfer coefficient
339 x_max = fzero(@(x) -0.8*(1-x)^-.2 + 3.8*( ...
```

```
340     0.76*x^-.24*(1-x)^.04 - 0.04*x^.76*(1-x)^-.96 ...
341 ), [0.0001 .9999]);
342
343 hV_max = 0.023*(...
344     (input(1,8))/(pi*D^2/4)*D/miu_rsh ...
345 )^.8*(miu_rsh*cp_rsc/k_rsh)^.4*k_rsh/D;
346 hL_max = 0.023*(...
347     (input(1,8))/(pi*D^2/4)*D/miu_rsc ...
348 )^.8*(miu_rsc*cp_rsc/k_rsc)^.4*k_rsc/D;
349 % maximum heat transfer coefficient if the flow is not splited
350 htp_max = 2*hL_max/(P_cond/Pc)^.38*(...
351     (1-x_max)^.8 + 3.8*x_max^.76*(1-x_max)^.04 ...
352 );
353
354 U = ones(5,1)*0.3;
355 % index of mass flow rate from correlations RE^0.4
356 U(2) = (0.84-0.4)*rand(1)+0.4;
357
358 % Constrained optimization
359 options = optimset(...
360     'Display','off','Algorithm','sqp','MaxFunEvals',5000, ...
361     'MaxIter',500,'TolX',1.e-8,'TolFun',1.e-8 ...
362 ); %increase funevals to avoid getting out before completion
363 exitflag = 0;
364 weigh = weighing_function_v000(ref(:,28)./ref(:,23), bin_num);
365 g = coeff2( ...
```

```
366     U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
367     hL_max, refri, libname, weigh ...
368 );
369 iter_max = 10;
370 iter = 0;
371 % make sure that the initial values are not nan
372 while isnan(g) & iter < iter_max
373     U = U.*(rand(length(U),1)+1);
374     g = coeff2( ...
375         U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
376         hL_max, refri, libname, weigh ...
377     );
378     iter = iter + 1;
379 end
380 [U,fval,exitflag] = fmincon(@(U) coeff2(...
381     U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
382     hL_max, refri, libname, weigh ...
383 ),U,[],[],[],[],[0 0.4 0 0 0],[1 10.0 1 1 1],@(U) constraint(...
384     U, input, ref, output, cpr, m, Ar, hV_max, htp_max, hL_max, ...
385     refri, libname, weigh ...
386 ),options);
387 iter = 0;
388 % problem in calculation, recalculate
389 if (exitflag < 1 || fval > 0.02) && iter < iter_max
390     options = optimset(...
391         'Display','off','Algorithm','sqp','MaxFunEvals',5000,...
```

```
392         'MaxIter',500,'TolX',1.e-8,'TolFun',1.e-8 ...
393     ); %increase funevals to avoid getting out before completion
394     [U2,fval2,exitflag2] = fmincon(@(U) coeff2(...
395         U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
396         hL_max, refri, libname, weigh ...
397     ),U,[],[],[],[],[0 0.4 0 0 0],[1 10.0 1 1 1],@(U) constraint(...
398         U, input, ref, output, cpr, m, Ar, hV_max, htp_max, ...
399         hL_max, refri, libname, weigh ...
400     ),options);
401     iter = iter + 1;
402     if fval2 < fval
403         U = U2;
404         fval = fval2;
405     end
406     % problem in calculation, recalculate
407     if (exitflag < 1 || fval > 0.02) && iter < iter_max
408         options = optimset(...
409             'Display','off','Algorithm','active-set',...
410             'MaxFunEvals',5000,'MaxIter',500,...
411             'TolX',1.e-8,'TolFun',1.e-8 ...
412         ); %increase funevals to avoid getting out before completion
413         [U3,fval3,exitflag3] = fmincon(@(U) coeff2(...
414             U, input, ref, output, stat, cpr, Ar, htp_max, ...
415             hV_max, hL_max, refri, libname, weigh ...
416         ),U,[],[],[],[],[0 0.4 0 0 0],[...
417             1 10.0 1 1 1 ...
```

```
418         ],@(U) constraint(...
419             U, input, ref, output, cpr, m, Ar, hV_max, ...
420             htp_max, hL_max, refri, libname, weigh ...
421         ),options);
422         iter = iter + 1;
423         if fval3 < fval
424             U = U3;
425             fval = fval3;
426         end
427     end
428 end
429
430 [input output g] = Condenser2(...
431     U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
432     hL_max, refri, libname, weigh ...
433 );
434
435 % calculate the covriance matrix
436 X_Cov = [];
437 U_norm = [hL_max 1 hV_max htp_max hL_max];
438 for i = 1:length(U)
439     U_sp = U;
440     if abs(U(i)) > 1
441         U_sp(i) = U_sp(i)*(1+1.e-5);
442     else
443         U_sp(i) = U(i)+1.e-5;
```

```
444     end
445     [dum1 output_sp dum2] = Condenser2(...
446         U_sp, input, ref, output, stat, cpr, Ar, htp_max, ...
447         hV_max, hL_max, refri, libname, weigh ...
448     );
449     X_Cov(:,i) = (output_sp(:,1) - output(:,1))./(...
450         U_sp(i)-U(i) ...
451     )/U_norm(i);
452 end
453 Cond.para.Cov = inv(X_Cov'*X_Cov);
454 Cond.para.X_limit = max(diag(X_Cov*Cond.para.Cov*X_Cov'));
455
456 % computing the overall conductance
457 extra = 1;
458
459 stat(2) = 0;
460 for i = 1:m
461     stat(2) = stat(2) + (output(i,1)-ref(i,28))^2;
462 end
463
464 coeffQ = 1 - stat(2)/stat(1);
465 FanPower = mean(ref(:,26));
466 mdot_r_rated = input(1,8);
467 mdot_a_rated = output(1,11);
468 U(1) = U(1)*hL_max;
469 U(3) = U(3)*hV_max;
```

```
470 U(4) = U(4)*htp_max;
471 U(5) = U(5)*hL_max;
472
473 % then conduct pressure drop correlation
474
475 % Air-side rated heat transfer coefficient  [W/m^2-K]
476 Cond.para.U_a = U(1);
477 % Adjustment on air-side heat transfer coefficient by air mass flow
478 Cond.para.n = U(2);
479 % Rated refrigerant-side heat transfer coefficient
480 % of the superheat region [W/m^2-K]
481 Cond.para.U_r_sh = U(3);
482 % Rated refrigerant-side heat transfer coefficient
483 % of the two-phase region [W/m^2-K]
484 Cond.para.U_r_tp = U(4);
485 Cond.para.U_r_sc = U(5);
486 % Lumped parameter for the effect of the property on the
487 % superheated heat transfer coefficient at rated condition
488 Cond.para.K_sh = Ksh;
489 % Lumped parameter for the effect of the property on the
490 % subcooled heat transfer coefficient at rated condition
491 Cond.para.K_sc = Ksc;
492 % Rated air mass flow rate for condneser model [kg/s]
493 Cond.para.m_a = mdot_a_rated;
494 % Rated refrigerant mass flow rate for the condenser model [kg/s]
495 Cond.para.m_r = mdot_r_rated;
```

```
496 Cond.para.Din = D; % Inner diameter of tube [m]
497 Cond.para.Lt = Ltube*Ntube; % Total length of heat exchanger pipes [m]
498 Cond.para.A_r = Ar; % Inner surface area of condenser [m^2]
499 % Average power consumption of condenser fan [W]
500 Cond.para.FanPower = FanPower;
501
502 save(savefilename);
503
504 % Obtain Superheat, Mass Flow Rate and Pressures
505 % re-filter data for Breuker's system
506 SH = zeros(m,1);
507 mdot = zeros(m,1);
508 ΔP = zeros(m,1);
509 for i = 1:m
510     hin = propertyRH('H','T',ref(i,11),'P',ref(i,3),refri);
511     hout = hin - Q_cond_r(i,1)/ref(i,23);
512     SH(i,1) = ref(i,11) - propertyRH('T','P',ref(i,3),'Q',1,refri);
513     SC(i,1) = propertyRH('T','P',ref(i,4),'Q',0,refri) - ref(i,12);
514     mdot(i,1) = ref(i,23);
515     rho(i,1) = propertyRH('D','T',ref(i,11),'P',ref(i,3),refri);
516     rho_out(i,1) = RefrigerantDensity(ref(i,4), hout, refri);
517     rho_out_est(i,1) = RefrigerantDensity(ref(i,3), hout, refri);
518     ΔP(i,1) = ref(i,3) - ref(i,4);
519     hin = propertyRH('H','T',ref(i,11),'P',ref(i,3),refri);
520     hl = propertyRH('H','P',ref(i,3),'Q',0,refri);
521     hv = propertyRH('H','P',ref(i,3),'Q',1,refri);
```

```
522     rhol = propertyRH('D','P',ref(i,3),'Q',0,refri);
523     rhov = propertyRH('D','P',ref(i,3),'Q',1,refri);
524     mu = (rhol/rhov)^(1/3)*(rhov/rhol);
525     x_out = (hout - hl)/(hv-hl);
526     if x_out < 0
527         x_out = 0;
528     end
529     x_in(i,1) = 1;
530     gamma = -(mu*(log(((x_out-1)*mu-x_out)/(...
531         (x_in(i,1)-1)*mu-x_in(i,1) ...
532     ))+x_out-x_in(i,1))-x_out+x_in(i,1))/(...
533         mu*mu-2*mu+1 ...
534     )/(x_out-x_in(i,1));
535     rho_tp(i,1) = ((hin-hv)*rhov+(hv-hl)*(...
536         gamma*rhov+(1-gamma)*rhol ...
537     )+(hl-hout)*rhol)/(hin-hout);
538     rho_tp(i,1) = (gamma*rhov+(1-gamma)*rhol);
539     rho_l(i,1) = rhol;
540     rho_v(i,1) = rhov;
541     Pin = ref(i,3);
542     Tain = ref(i,21);
543     airinlet = zeros(1,5);
544     Pain = ref(i,32);
545     airinlet = calllib(...
546         libname, 'HumAir_DLL', Tain, Pain, 4, ...
547         input(i,13), airinlet ...
```

```
548     );
549     wain = airinlet(1,2);
550     Vdot = mdot(i,1)*(hin-hout)/(...
551         (1006+1860*wain)*(1+wain)/airinlet(1,5)*(...
552             ref(i,22)-ref(i,21) ...
553         ) ...
554     );
555     [Pout dumhout Taout Qcond Charge FanPower ...
556         CondOutput ma CondInput] = Condenser(...
557             Pin, hin, mdot(i,1), Tain, wain, Pain, Vdot, ...
558             Cond.para, refri, libname ...
559         );
560     w_superheat(i,1) = CondOutput(5);
561     w_twophase(i,1) = CondOutput(6);
562     w_subcool(i,1) = CondOutput(7);
563     mu_l(i,1) = propertyRH('V','P',ref(i,3),'Q',0,refri);
564     mu_v(i,1) = propertyRH('V','P',ref(i,3),'Q',1,refri);
565 end
566 mdot_rated = mdot(1,1);
567 SH_rated = mean(SH);
568 SC_rated = mean(SC);
569 ΔP_av = mean(ΔP);
570 rho_rated = mean(rho);
571 rho_out_rated = mean(rho_out_est);
572 rho_tp_rated = mean(rho_tp);
573 mu_l_rated = mean(mu_l);
```

```
574 mu_v_rated = mean(mu_v);
575 SH_limit = 1;
576 SC_limit = 3;
577
578 % initial guesses
579 C = zeros(5,1);
580 A = -eye(5);
581 A(3,3) = 0;
582 b = zeros(5,1);
583 Aeq = [];
584 beq = [];
585 options = optimset(...
586     'MaxFunEvals',20000,'algorithm','active-set',...
587     'MaxIter',5000,'Display','off' ...
588 );
589 if length(ΔP) > 1 & sum(ΔP.^2)>1.e-8
590     weigh = weighing_function_v000(ΔP, bin_num);
591 elseif sum(ΔP.^2)≤1.e-8
592     weigh = ones(length(ΔP),1);
593 else
594     weigh = 1;
595 end
596 r = PressureDrop_residual(...
597     C, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, rho_tp, ...
598     rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
599     mu_l, mu_v, mdot_r_rated, SH_rated, SC_rated, rho_rated, ...
```

```
600     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated, ...
601     SH_limit, SC_limit, ΔP, weigh, System.name, fault ...
602 );
603 iter = 0;
604 % make sure that the initial values are not nan
605 while isnan(r) & iter < iter_max
606     C = C.*(rand(length(C),1)+1);
607     r = PressureDrop_residual(...
608         C, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, ...
609         rho_tp, rho_l, rho_v, x_in, w_superheat, ...
610         w_twophase, w_subcool, mu_l, mu_v, mdot_r_rated, ...
611         SH_rated, SC_rated, rho_rated, rho_out_rated, ...
612         rho_tp_rated, mu_l_rated, mu_v_rated, SH_limit, ...
613         SC_limit, ΔP, weigh, System.name, fault ...
614     );
615     iter = iter + 1;
616 end
617 C = fmincon(@(C) PressureDrop_residual(...
618     C, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, rho_tp, ...
619     rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
620     mu_l, mu_v, mdot_rated, SH_rated, SC_rated, rho_rated, ...
621     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated, ...
622     SH_limit, SC_limit, ΔP, weigh, System.name, fault ...
623     ), C, A, b, Aeq, beq, [0 0 -100 0 0],[...
624         Inf Inf 100 Inf Inf ...
625     ],[],options);
```

```
626 r = PressureDrop_residual(...
627     C, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, rho_tp, ...
628     rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
629     mu_l, mu_v, mdot_r_rated, SH_rated, SC_rated, rho_rated, ...
630     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated, ...
631     SH_limit, SC_limit, ΔP, weigh, System.name, fault ...
632 );
633 ΔP_est = PressureDrop(...
634     C, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, rho_tp, ...
635     rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
636     mu_l, mu_v, mdot_r_rated, SH_rated, SC_rated, rho_rated, ...
637     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated ...
638 );
639
640 P_out_est = ref(:,3) - ΔP_est;
641 for i = 1:m
642     hin = propertyRH('H','T',ref(i,11),'P',ref(i,3),refri);
643     hout_est = hin - output(i,1)/ref(i,23);
644     SC_est(i,1) = propertyRH('T','P',P_out_est(i,1),'Q',0,refri)-...
645         propertyRH('T','P',P_out_est(i,1),'H',hout_est,refri);
646 end
647
648 % calculate the covriance matrix
649 X_Cov = [];
650 for i = 1:length(C)
651     C_sp = C;
```

```
652     if abs(C(i))>1
653         C_sp(i) = C_sp(i)*(1+1.e-5);
654     else
655         C_sp(i) = C_sp(i)+1.e-5;
656     end
657     ΔP_est_sp = PressureDrop(...
658         C_sp, [], ΔP_av, mdot, SH, SC, rho, rho_out_est, ...
659         rho_tp, rho_l, rho_v, x_in, w_superheat, w_twophase, ...
660         w_subcool, mu_l, mu_v, mdot_r_rated, SH_rated, SC_rated, ...
661         rho_rated, rho_out_rated, rho_tp_rated, ...
662         mu_l_rated, mu_v_rated ...
663     );
664     X_Cov(:,i) = (ΔP_est_sp - ΔP_est)./(C_sp(i)-C(i));
665 end
666 Cond.para.Cov_ΔP = inv(X_Cov'*X_Cov);
667 Cond.para.X_limit_ΔP = max(diag(X_Cov*Cond.para.Cov*X_Cov'));
668
669  % Average pressure drop across condenser from data [kPa]
670 Cond.para.ΔP_const = 0;
671 Cond.para.SC_rated = 6; % rated subcooling for condenser [K]
672
673 Cond.para.ΔP.C = C;
674 Cond.para.ΔP.mdot_rated = mdot_r_rated;
675 Cond.para.ΔP.rho_rated = rho_rated;
676 Cond.para.ΔP.rho_out_rated = rho_out_rated;
677 Cond.para.ΔP.mu_v_rated = mu_v_rated;
```

```
678 Cond.para.ΔP_max = max(ref(:,2)-ref(:,5));
679
680 coeffΔP = 1 - sum((ΔP_est - ΔP).^2)/sum(...
681     (mean(ΔP) - ΔP).^2 ...
682 );
683
684 % for Breuker's system, delete the faulted data from pressure drop
685 if strcmp(System.name,'Breuker_030_R22_packaged_FXO_Recip')
686     cell_NF = strfind(fault,'NF');
687     if isempty(cell_NF)
688         cell_NF = cell(pp,1);
689     end
690     i = 1;
691     pp = m;
692     try
693         while i ≤ pp
694             if isempty(cell2mat(cell_NF(i)))
695                 ΔP(i,:) = [];
696                 ΔP_est(i,:) = [];
697                 cell_NF(i,:) = [];
698                 i = i - 1;
699                 pp = pp - 1;
700             end
701             i = i + 1;
702         end
703     end
```

```
704 end
705
706 %% Fan power calculation
707 FanPower_mea = ref(:,26);
708
709 Cond.para.C_fan = [0 0]';
710 if ¬isempty(find(strcmp(fault,'NF'))) && ...
711     mean(ref(strcmp(fault,'NF'),26)) > 0
712     V_rated = mean(ref(find(strcmp(fault,'NF')),29));
713     V_max = V_rated;
714     W_rated = mean(ref(find(strcmp(fault,'NF')),26));
715     b = W_rated/(2*V_rated*V_max-V_rated^2);
716     a = 2*b*V_max;
717     Cond.para.C_fan = [a b]';
718 end
719
720
721 save(savefilename);
722 movefile(savefilename,strcat(foldername,'\'));
723
724 end
725
726 function g = coeff2(...
727     U, input, ref, output, stat, cpr, Ar, htpmax, hV_max, ...
728     hL_max, refri, string, weigh ...
729 )
```

```
730
731 % This function calculates the result of cost function
732
733 % Variables
734 % Inputs:
735 % UA: UAs to be minimized
736 % input: input matrix
737 % ref: reference matrix
738 % output: output matrix
739 % stat: statistical information
740 % string: name of library to be passed in
741 % Outputs:
742 % r: an array of function results at different points
743
744 [input output g] = Condenser2(...
745     U, input, ref, output, stat, cpr, Ar, htpmax, hV_max, ...
746     hL_max, refri, string, weigh ...
747 );
748 end
749
750 function [input output g] = Condenser2(...
751     U, input, ref, output, stat, cpr, Ar, htp_max, hV_max, ...
752     hL_max, refri, string, weigh ...
753 )
754 g = 0;
755
```

```
U(1) = U(1)*hL_max;
U(3) = U(3)*hV_max;
U(4) = U(4)*htp_max;
U(5) = U(5)*hL_max;

% UA initialization
if ¬isempty(findstr(refri,'.fld'))
    index_end = findstr(refri,'.fld');
    refname = refri(1:index_end-1);
else
    refname = refri;
end
[m, n] = size(ref);
output = zeros(m, 11);
mdot_r_rated = input(1,8);
airinlet = zeros(1,5);
airinlet = calllib(...
    string, 'HumAir_DLL', input(1,5), input(1,7), 4, ...
    input(1,13), airinlet ...
);
ma_rated = (1)/airinlet(5)*input(1,6);
%UAsh
P_cond = input(1,10);
cp_rsh = cpr(1,1);
miu_rsh = propertyRH('V', 'P', input(1,10),'Q',1, refri);
k_rsh = propertyRH('L', 'P', input(1,10),'Q',1, refri);
```

```
782 Ksh_ref = (cp_rsh*miu_rsh/k_rsh)^0.4*(input(1,8)/miu_rsh)^0.8;
783
784 % UAsc
785 cp_rsc = cpr(1,2);
786 T_cond = propertyRH('T','P',P_cond,'Q',1,refri);
787 miu_rsc = propertyRH('V', 'P', input(1,10),'Q', 0, refri);
788 k_rsc = propertyRH('L', 'P', input(1,10),'Q', 0, refri);
789 Ksc_ref = (cp_rsc*miu_rsc/k_rsc)^0.4*(input(1,8)/miu_rsc)^0.8;
790 for i = 1:m
791     % computing new UA
792     % UAa
793     input_II = input(i,:);
794     airinlet = zeros(1,5);
795     airinlet = calllib(...
796         string, 'HumAir_DLL', input_II(5), input_II(7), 4, ...
797         input_II(13), airinlet ...
798     );
799     ma = (1)/airinlet(5)*input_II(6);
800     input_II(1) = U(1)*((ma/ma_rated)^U(2))*Ar;
801
802     %UAsh
803     P_cond = input_II(10);
804     cp_rsh = cpr(i,1);
805     miu_rsh = propertyRH('V', 'P', input_II(10),'Q',1, refri);
806     k_rsh = propertyRH('L', 'P', input_II(10),'Q',1, refri);
807     Ksh = (cp_rsh*miu_rsh/k_rsh)^0.4*(input_II(8)/miu_rsh)^0.8;
```

```
808     if(i==1)
809        input_II(2) = U(3);
810     else
811        input_II(2) = U(3)*Ksh/Ksh_ref*Ar;
812     end
813
814     % UAsc
815     cp_rsc = cpr(i,2);
816     T_cond = propertyRH('T','P',P_cond,'Q',1,refri);
817     miu_rsc = propertyRH('V', 'P', input_II(10),'Q', 0, refri);
818     k_rsc = propertyRH('L', 'P', input_II(10),'Q', 0, refri);
819     Ksc = (cp_rsc*miu_rsc/k_rsc)^0.4*(input_II(8)/miu_rsc)^0.8;
820     if(i==1)
821        input_II(4) = U(5)*Ar;
822     else
823        input_II(4) = U(5)*Ksc/Ksc_ref*Ar;
824     end
825
826     %UAtp
827     input_II(3) = U(4)*(input_II(8)/mdot_r_rated)*Ar;
828
829     output_II = zeros(1,11);
830     input_II = real(input_II);
831     [input_II(1:14) output_II(1:11)] = calllib(...
832         string, 'Matlab_Condenser_Forward_Charge_ii_NC', ...
833         input_II(1:14), output_II, refname ...
```

```
834     );
835
836     g = g + 1/weigh(i,1)*(...
837         (output_II(1) - ref(i,28))./ref(i,28) ...
838     )^2;
839
840     output_II(11) = ma;
841     input(i,:) = input_II;
842     output(i,:) = output_II;
843 end
844 g = sqrt(g/m);
845 end
846
847 function wsh = Superheat(...
848     i, input, ref, Qsh, cpr, UA, Tcond, airinlet ...
849 )
850 % solve for the superheat section
851 ma = (1)/airinlet(5)*input(i,6);
852 wsh = -log(1-(Tcond-input(i,9))/(...
853     input(i,5)-input(i,9) ...
854 ))*input(i,8)*cpr(i,1)/ma/(...
855     1006+1860*airinlet(2) ...
856 )/(1-exp(-UA/ma/(1006+1860*airinlet(2))));
857 end
858
859 function wtp = Twophase(i, input, ref, Qtp, cpr, UA, Tcond, airinlet)
```

```
860 % solve for the two-phase section
861 ma = (1)/airinlet(5)*input(i,6);
862 epsilontp = 1 - exp(-UA/(ma*(1006+1860*airinlet(2))));
863 wtp = -Qtp/ma/(1006+1860*airinlet(2))/epsilontp/(input(i,5)-Tcond);
864 end
865
866 function wsc = Subcool(i, input, ref, Qsc, cpr, UA, Tcond, airinlet)
867 % solve for the subcool section
868 ma = (1)/airinlet(5)*input(i,6);
869 wsc = -log(1-(ref(i,12)-Tcond)/(...
870     input(i,5)-Tcond ...
871 ))*input(i,8)*cpr(i,2)/ma/(...
872     1006+1860*airinlet(2) ...
873 )/(1-exp(-UA/ma/(1006+1860*airinlet(2))));
874 end
875
876 function wsh = Superheat_limit(...
877     i, input, ref, Qsh, cpr, UA, Tcond, airinlet ...
878 )
879 % solve for the superheat section
880 ma = (1)/airinlet(5)*input(i,6);
881 wsh = -log(1-(Tcond-input(i,9))/(...
882     input(i,5)-input(i,9) ...
883 ))*input(i,8)*cpr(i,1)/ma/(1006+1860*airinlet(2))/(1);
884 end
885
```

```
886 function wtp = Twophase_limit(...
887     i, input, ref, Qtp, cpr, UA, Tcond, airinlet ...
888 )
889 % solve for the two-phase section
890 ma = (1)/airinlet(5)*input(i,6);
891 wtp = -Qtp/ma/(1006+1860*airinlet(2))/1/(input(i,5)-Tcond);
892 end
893
894 function wsc = Subcool_limit(...
895     i, input, ref, Qsc, cpr, UA, Tcond, airinlet ...
896 )
897 % solve for the subcool section
898 ma = (1)/airinlet(5)*input(i,6);
899 wsc = -log(1-(ref(i,12)-Tcond)/(...
900     input(i,5)-Tcond ...
901 ))*input(i,8)*cpr(i,2)/ma/(1006+1860*airinlet(2))/(1);
902 end
903
904 function [c,ceq] = constraint(...
905     U, input, ref, output, cpr, m, Ar, hV_max, htp_max, ...
906     hL_max, refri, string, weigh ...
907 )
908 m = min(10,length(cpr));
909 [input output g] = Condenser2(...
910     U, input, ref, output, [], cpr, Ar, htp_max, hV_max, ...
911     hL_max, refri, string, weigh ...
```

```
912 );
913 specifier = find(output(:,7)==max(output(:,7)));
914 % avoid the cases without subcooling
915 if length(specifier) > 1
916     specifier = specifier(1); % only one needed
917 end
918 wsh = output(specifier,5);
919 wtp = output(specifier,6);
920 wsc = 1 - wsh - wtp;
921 cpr = propertyRH('C','P',input(specifier,10),'Q',0,refri);
922 airinlet = calllib(...
923     string, 'HumAir_DLL', input(specifier,5), input(specifier,7), ...
924     4, input(specifier,13), zeros(1,5) ...
925 );
926 UAsh = (1/input(specifier,1)+1/input(specifier,3))^(-1);
927 UAtp = (1/input(specifier,1)+1/input(specifier,4))^(-1);
928 UAsc = (1/input(specifier,1)+1/input(specifier,5))^(-1);
929 ma = (1)/airinlet(5)*input(specifier,6);
930 epsilontp = 1 - exp(-UAtp/ma/(1006+1860*airinlet(2)));
931 % assume crossflow
932 Ntu_sc = wsc*UAsc/min(...
933     input(specifier,10)*cpr,wsc*ma*(1006+1860*airinlet(2)) ...
934 );
935 Cr = input(specifier,8)*cpr/(wsc*ma*(1006+1860*airinlet(2)));
936 if Cr > 1 % (mr*cpr > wsc*ma*cpa)
937     Cr = 1/Cr;
```

```
938     epsilonsc = 1 / Cr * (1 - exp(-Cr * (1 - exp(-Ntu_sc))));
939 else
940     epsilonsc = 1. - exp(-1. / Cr * (1. - exp(-Cr * Ntu_sc)));
941 end
942 if wsc > 0.01
943     c(1,1) = (epsilonsc - 0.9999);
944 else
945     c(1,1) = 0.0; % ignore the superheated section
946 end
947 % assume crossflow
948 cpr = propertyRH('C','P',input(specifier,10),'Q',1,refri);
949 Ntu_sh = wsh*UAsh/min(...
950     input(specifier,10)*cpr,wsh*ma*(1006+1860*airinlet(2)) ...
951 );
952 Cr = input(specifier,8)*cpr/(wsh*ma*(1006+1860*airinlet(2)));
953 if Cr > 1 % (mr*cpr > wsc*ma*cpa)
954     Cr = 1/Cr;
955     epsilonsh = 1 / Cr * (1 - exp(-Cr * (1 - exp(-Ntu_sh))));
956 else
957     epsilonsh = 1. - exp(-1. / Cr * (1. - exp(-Cr * Ntu_sh)));
958 end
959 c(2,1) = (epsilontp - 0.9999);
960 c(3,1) = epsilonsh - 0.9999;
961 % bounding the one-phase UA with two-phase
962 U(1) = U(1)*htp_max;
963 U(3) = U(3)*hV_max;
```

```
964 U(4) = U(4)*htp_max;
965 U(5) = U(5)*hL_max;
966 c(4,1) = U(3) - U(4);
967 c(5,1) = U(5) - U(4);
968 c(6,1) = U(3) - hV_max;
969 c(7,1) = U(4) - htp_max;
970 c(8,1) = U(5) - hL_max;
971 c(9,1) = 0.4-U(2);
972 c(10,1) = U(2)-0.84;
973 ceq = [];
974 end
975
976
977 function residual = PressureDrop_residual(...
978     C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, rho_l, ...
979     rho_v, x_in, w_superheat, w_twophase, w_subcool, mu_l, ...
980     mu_v, mdot_rated, SH_rated, SC_rated, rho_rated, ...
981     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated, ...
982     SH_limit, SC_limit, ΔP, weigh, name, fault ...
983 )
984 n = length(SH);
985 residual = 0;
986 for i = 1:n
987     if SH(i,1) > SH_limit && SC(i,1) > SC_limit
988         if ¬strcmp(name,'Breuker_030_R22_packaged_FXO_Recip')
989             residual = residual + 1/weigh(i,1)*(...
```

```
990                 ΔP(i,1) - PressureDrop(...
991                     C, N, ΔP_av, mdot(i,1), SH(i,1), SC(i,1), ...
992                     rho(i,1), rho_out(i,1), rho_tp(i,1), ...
993                     rho_l(i,1), rho_v(i,1), x_in(i,1), ...
994                     w_superheat(i,1), w_twophase(i,1), ...
995                     w_subcool(i,1), mu_l(i,1), mu_v(i,1), ...
996                     mdot_rated, SH_rated, SC_rated, rho_rated, ...
997                     rho_out_rated, rho_tp_rated, mu_l_rated, ...
998                     mu_v_rated ...
999                 )...
1000            )^2;
1001        elseif strcmp(fault(i,1),'NF')
1002            residual = residual + 1/weigh(i,1)*(ΔP(i,1) - ...
1003                PressureDrop(...
1004                    C, N, ΔP_av, mdot(i,1), SH(i,1), SC(i,1), ...
1005                    rho(i,1), rho_out(i,1), rho_tp(i,1), ...
1006                    rho_l(i,1), rho_v(i,1), x_in(i,1), ...
1007                    w_superheat(i,1), w_twophase(i,1), ...
1008                    w_subcool(i,1), mu_l(i,1), mu_v(i,1), ...
1009                    mdot_rated, SH_rated, SC_rated, rho_rated, ...
1010                    rho_out_rated, rho_tp_rated, ...
1011                    mu_l_rated, mu_v_rated ...
1012                )...
1013            )^2;
1014        end
1015    end
```

```
1016 end
1017 end
1018
1019 function ΔP_est = PressureDrop(...
1020     C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, ...
1021     rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
1022     mu_l, mu_v, mdot_rated, SH_rated, SC_rated, rho_rated, ...
1023     rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated...
1024 )
1025 % function to estimate the pressure drop
1026 ΔP_est = C(1).*w_superheat.* ...
1027     (mdot.^2./rho_v).*rho_rated/.mdot_rated^2;
1028 ΔP_est = ΔP_est + C(2)*w_twophase.*(mdot./mdot_rated).^2./(...
1029     rho_tp./rho_rated ...
1030 ).*((x_in./rho_v.*mu_v+(1-x_in)./rho_l.*mu_l)./(...
1031     x_in./rho_v+(1-x_in)./rho_l ...
1032 )./mu_v_rated).^C(3);
1033 ΔP_est = ΔP_est + C(4).*w_subcool.*(mdot.^2./rho_l).*(...
1034     rho_out_rated/mdot_rated^2 ...
1035 );
1036 ΔP_est = ΔP_est + C(5).*(mdot./mdot_rated).^2.*(...
1037     1./rho_out - 1./rho ...
1038 )*rho_rated;
1039 end
1040
1041 function [...
```

```
1042      Pout hout Taout Qcond Charge FanPower CondOutput ...
1043      ma CondInput ...
1044  ] = Condenser(...
1045      Pin, hin, mdot_r, Tain, wain, Pain, Vdot, para, refri, libname ...
1046  )
1047
1048  % Variable list
1049  % Pout: Refrigenrat outlet pressure (kPa)
1050  % hout: Refrigenrat outlet enthalpy (J/kg)
1051  % Taout: Air outlet tmperautre (K)
1052  % Qcond: Condenser heat transfer rate (W)
1053  % Charge: Amount of charge in the condenser (kg)
1054  % FanPower: Fan Power consumption of condenser (W)
1055  % CondOutput: other outputs (misc.)
1056
1057  % Pin: Refrigenrat pressure at inlet (kPa)
1058  % hin: Refrigenrat enthalpy at inlet (W)
1059  % mdot_r: Refrigeraat mass flow rate (kg/s)
1060  % Tain: Air temperature at inlet (K)
1061  % wain: Inlet air absolute humidity (kg/kg)
1062  % Pain: Atmospheric pressure (kPa)
1063  % Vdot: Air volumetric flow at inlet (m^3/s)
1064  % para: parameters (misc.)
1065  % refri: Name of refrigerant
1066  % lib: Name of dll library to be used
1067
```

```
% UAa
airinlet = zeros(1,5);
airinlet = calllib(...
    libname, 'HumAir_DLL', Tain, Pain, 2, wain, airinlet ...
); %arbitrary
ma = (1)/airinlet(5)*Vdot;
cpa = 1006 + 1860*airinlet(2);
CondInput(1) = para.U_a*((ma/para.m_a)^para.n)*para.A_r;

% UAsh
Tsat = propertyRH('T','P',Pin,'Q',1,refri);
Tin = propertyRH('T','P',Pin,'H',hin,refri);
cp_rsh = propertyRH('C','P',Pin,'Q',1,refri);
mu_rsh = propertyRH('V', 'T', Tsat,'Q',1, refri);
k_rsh = propertyRH('L', 'T', Tsat,'Q',1, refri);
Ksh = (cp_rsh*mu_rsh/k_rsh)^0.4*(mdot_r/mu_rsh)^0.8;
CondInput(2) = para.U_r_sh*Ksh/para.K_sh*para.A_r;

% UAtp
CondInput(3) = para.U_r_tp*(mdot_r/para.m_r)*para.A_r;

% UAsc
cp_rsc = propertyRH('C','P',Pin,'Q',0,refri);
mu_rsc = propertyRH('V','P',Pin,'Q',0,refri);
k_rsc = propertyRH('L','P',Pin,'Q',0,refri);
Ksc = (cp_rsc*mu_rsc/k_rsc)^0.4*(mdot_r/mu_rsc)^0.8;
```

```
1094 CondInput(4) = para.U_r_sc*Ksc/para.K_sc*para.A_r;
1095
1096 CondInput(5) = Tain;
1097 CondInput(6) = Vdot;
1098 CondInput(7) = Pain;
1099 CondInput(8) = mdot_r;
1100 CondInput(9) = Tin;
1101 CondInput(10) = Pin;
1102 CondInput(11) = para.Din;
1103 CondInput(12) = para.Lt;
1104 CondInput(13) = airinlet(4);
1105
1106 CondOutput = zeros(1,11);
1107 if ¬isempty(strfind(refri,'.fld'))
1108     index_end = strfind(refri,'.fld');
1109     refname = refri(1:index_end-1);
1110 else
1111     refname = refri;
1112 end
1113 CondInput = real(CondInput);
1114 [CondInput CondOutput] = calllib(...
1115     libname, 'Matlab_Condenser_Forward_Charge_ii_NC', CondInput, ...
1116     CondOutput, refname ...
1117 );
1118
1119 Qcond = CondOutput(1);
```

```
1120 hout = hin - Qcond/mdot_r;
1121 Taout = Tain + Qcond/ma/cpa;
1122 Charge = CondOutput(11);
1123
1124 FanPower = para.FanPower;
1125 Pout = Pin; %temporary, to be changed
1126 end
```

N.5 Liquid Line Modeling

```
1 function System = LiquidLine_minimization_v041_main()
2
3 % Function to pass parameters to functions to train the liquid
4 % line model
5
6 % define function and folder name
7 function_name = 'LiquidLine_minimization';
8 version_main = '041';
9 version_regress = '037';
10 version_plot = '000';
11 foldername = strcat(function_name,'_v',version_main);
12 regressname = strcat(function_name,'_v',version_regress);
13 plotname = strcat(function_name,'_plot_v',version_plot);
14 regress_func = str2func(['@(System, libname, version_number)',...
15     regressname,'(System, libname, version_number)']);
16 plot_func = str2func(['@(System, libname, version_number)',...
17     plotname,'(System, libname, version_number)']);
```

```
mkdir(foldername);
savefilename = strcat(foldername,'.mat');

% Module Definition
% Define the linset lengths and inner diameters [m, m]
LiquidLine.line = 0;
LiquidLine.dia = 0;
LiquidLine.ΔP = 0;
LiquidLine.HeatLossUAtomr = 0;
LiquidLine.mdot_rated = 0;
LiquidLine.V_rated = 0;
LiquidLine.rho_rated = 0;
LiquidLine.C = zeros(5,1);
LiquidLine.C_Q = zeros(4,1);
LiquidLine.Δh_rated = 0;
LiquidLine.ΔT_rated = 0;
LiquidLine.ΔT_rated = 0;

% Set information
% re-file at this stage to avoid problems in parfor loop
i = 1;

% Breuker 3-ton R22 packaged FXO system (Recip)
% (Breuker_030_R22_packaged_FXO_Recip)
System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
System(i).othername = System(i).name;
```

```
System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
System(i).foldername = 'Breuker';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).LiquidLine = LiquidLine;
System(i).LiquidLine.line = 0.87;
System(i).LiquidLine.dia = 0.01128;

% Boshen 3-ton R410A packaged FXO system (Scroll)
% (Shen_030_R410A_packaged_FXO_Scroll)
i = i + 1;
System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
System(i).foldername = 'Boshen';
```

```
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00853;
System(i).Tube_Length = 1.67075;
System(i).Ntube = 40;
System(i).Pc = 4901;
System(i).LiquidLine = LiquidLine;
System(i).LiquidLine.line = 0.87;
System(i).LiquidLine.dia = 0.01128;

% Boshen 3-ton R410A split FXO system (Recip)
% (Shen_030_R410A_split_FXO_Recip)
% for compressor modeling, merged with TXV system data
i = i + 1;
System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
System(i).othername = System(i).name;
System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
System(i).foldername = 'Boshen_3tonR410ASplitFXO';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
```

```
96  System(i).Standard_Airflow_evap = 0;
97  System(i).Fan_Upstream_evap = 0;
98  System(i).Standard_Airflow_cond = 1;
99  System(i).Fan_Upstream_cond = 1;
100 System(i).Heating = 0;
101 System(i).Combined = 0;
102 System(i).Diameter = 0.00849;
103 System(i).Tube_Length = 2.255;
104 System(i).Ntube = 32;
105 System(i).Pc = 4901;
106 System(i).LiquidLine = LiquidLine;
107 System(i).LiquidLine.line = 0.4318 + 4.763;
108 System(i).LiquidLine.dia = 0.008;
109
110 % Boshen 5-ton R407C packaged FXO system (Scroll)
111 % (Shen_050_R407C_packaged_FXO_Scroll)
112 i = i + 1;
113 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
114 System(i).othername = System(i).name;
115 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
116 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
117 System(i).worksheetname = 'SI (no_invalid_CA)';
118 System(i).refname = 'R407C';
119 System(i).Standard_Airflow_evap = 0;
120 System(i).Fan_Upstream_evap = 0;
121 System(i).Standard_Airflow_cond = 1;
```

```
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Diameter = 0.00693928;
System(i).Tube_Length = 2.282;
System(i).Ntube = 56;
System(i).Pc = 3786;
System(i).LiquidLine = LiquidLine;
System(i).LiquidLine.line = 0;
System(i).LiquidLine.dia = 0.008;

% Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
% (NIST_025_R410A_split_TXV_Scroll)
i = i + 1;
System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
System(i).foldername = 'NIST';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 1;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
```

```
148 System(i).Diameter = 0.0042;
149 System(i).Tube_Length = 1.778;
150 System(i).Ntube = 64;
151 System(i).Pc = 4901;
152 System(i).LiquidLine = LiquidLine;
153 System(i).LiquidLine.line = 7.9248;
154 System(i).LiquidLine.dia = 0.00635;
155
156 % Harms 5-ton R22 packaged TXV system (Scroll)
157 % (Harms_050_R22_packaged_TXV_Scroll_Heating)
158 i = i + 1;
159 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
160 System(i).othername = System(i).name;
161 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
162 System(i).foldername = 'Harms_050tonR22PackagedTXV';
163 System(i).worksheetname = 'SI (no_invalid_CA)';
164 System(i).refname = 'R22.fld';
165 System(i).Standard_Airflow_evap = 0;
166 System(i).Fan_Upstream_evap = 0;
167 System(i).Standard_Airflow_cond = 1;
168 System(i).Fan_Upstream_cond = 1;
169 System(i).Heating = 0;
170 System(i).Combined = 0;
171 System(i).Diameter = 0.00889;
172 System(i).Tube_Length = 1.629;
173 System(i).Ntube = 64;
```

```
174 System(i).Pc = 4990;
175 System(i).LiquidLine = LiquidLine;
176 System(i).LiquidLine.line = 3.1;
177 System(i).LiquidLine.dia = 0.0079;
178
179 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
180 % (HCA3_030_R410A_split_TXV_Scroll)
181 i = i + 1;
182 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
183 System(i).othername = System(i).name;
184 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
185 System(i).foldername = 'Kim-HCA3';
186 System(i).worksheetname = 'SI (no_invalid_CA)';
187 System(i).refname = 'R410A';
188 System(i).Standard_Airflow_evap = 0;
189 System(i).Fan_Upstream_evap = 0;
190 System(i).Standard_Airflow_cond = 1;
191 System(i).Fan_Upstream_cond = 1;
192 System(i).Heating = 0;
193 System(i).Combined = 0;
194 % assumed
195 System(i).Diameter = 0.00849;
196 System(i).Tube_Length = 2.255;
197 System(i).Ntube = 32;
198 System(i).Pc = 4901;
199 System(i).LiquidLine = LiquidLine;
```

```
200 System(i).LiquidLine.line = 7.9248;
201 System(i).LiquidLine.dia = 0.00635;
202
203 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
204 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
205 i = i + 1;
206 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
207 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
208 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
209 System(i).foldername = 'Kim-HCA3-Heating';
210 System(i).worksheetname = 'SI (no_invalid_CA)';
211 System(i).refname = 'R410A';
212 System(i).Standard_Airflow_evap = 0;
213 System(i).Fan_Upstream_evap = 0;
214 System(i).Standard_Airflow_cond = 1;
215 System(i).Fan_Upstream_cond = 1;
216 System(i).Heating = 1;
217 System(i).Combined = 0;
218 % assumed
219 System(i).Diameter = 0.00849;
220 System(i).Tube_Length = 2.255;
221 System(i).Ntube = 32;
222 System(i).Pc = 4901;
223 System(i).LiquidLine = LiquidLine;
224 System(i).LiquidLine.line = 0.4318 + 4.763;
225 System(i).LiquidLine.dia = 0.008;
```

```
226
227 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
228 % (YSA_030_R22_split_TXV_Scroll)
229 i = i + 1;
230 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
231 System(i).othername = System(i).name;
232 System(i).filename = 'Kim_Cycle_data_R22_YSA.xlsx';
233 System(i).foldername = 'Kim-YSA';
234 System(i).worksheetname = 'SI (no_invalid_CA)';
235 System(i).refname = 'R22.fld';
236 System(i).Standard_Airflow_evap = 0;
237 System(i).Fan_Upstream_evap = 0;
238 System(i).Standard_Airflow_cond = 1;
239 System(i).Fan_Upstream_cond = 1;
240 System(i).Heating = 0;
241 System(i).Combined = 0;
242 % assumed
243 System(i).Diameter = 0.00849;
244 System(i).Tube_Length = 2.255;
245 System(i).Ntube = 32;
246 System(i).Pc = 4990;
247 System(i).LiquidLine = LiquidLine;
248 System(i).LiquidLine.line = 0.4318 + 4.763;
249 System(i).LiquidLine.dia = 0.008;
250
251 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
```

```
252 % (YSA_030_R22_split_TXV_Scroll_Heating)
253 i = i + 1;
254 System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
255 System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
256 System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
257 System(i).foldername = 'Kim-YSA-Heating';
258 System(i).worksheetname = 'SI (no_invalid_CA)';
259 System(i).refname = 'R22.fld';
260 System(i).Standard_Airflow_evap = 0;
261 System(i).Fan_Upstream_evap = 0;
262 System(i).Standard_Airflow_cond = 1;
263 System(i).Fan_Upstream_cond = 1;
264 System(i).Heating = 1;
265 System(i).Combined = 0;
266 % assumed
267 System(i).Diameter = 0.00849;
268 System(i).Tube_Length = 2.255;
269 System(i).Ntube = 32;
270 System(i).Pc = 4990;
271 System(i).LiquidLine = LiquidLine;
272 System(i).LiquidLine.line = 0.4318 + 4.763;
273 System(i).LiquidLine.dia = 0.008;
274
275 % Kim 3-ton R22 split TXV system (Scroll) (YKC)
276 % (YKC_030_R22_split_TXV_Scroll)
277 i = i + 1;
```

```
278 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
279 System(i).othername = System(i).name;
280 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
281 System(i).foldername = 'Kim-YKC';
282 System(i).worksheetname = 'SI (no_invalid_CA)';
283 System(i).refname = 'R22.fld';
284 System(i).Standard_Airflow_evap = 0;
285 System(i).Fan_Upstream_evap = 0;
286 System(i).Standard_Airflow_cond = 1;
287 System(i).Fan_Upstream_cond = 1;
288 System(i).Heating = 0;
289 System(i).Combined = 0;
290 % assumed
291 System(i).Diameter = 0.00849;
292 System(i).Tube_Length = 2.255;
293 System(i).Ntube = 32;
294 System(i).Pc = 4990;
295 System(i).LiquidLine = LiquidLine;
296 System(i).LiquidLine.line = 0.4318 + 4.763;
297 System(i).LiquidLine.dia = 0.008;
298
299 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
300 % (TM_040_R410A_packaged_TXV_Recip)
301 i = i + 1;
302 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
303 System(i).othername = System(i).name;
```

```
304 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
305 System(i).foldername = 'Kim-TM';
306 System(i).worksheetname = 'SI (no_LL)';
307 System(i).refname = 'R410A';
308 System(i).Standard_Airflow_evap = 0;
309 System(i).Fan_Upstream_evap = 0;
310 System(i).Standard_Airflow_cond = 1;
311 System(i).Fan_Upstream_cond = 1;
312 System(i).Heating = 0;
313 System(i).Combined = 0;
314 % assumed
315 System(i).Diameter = 0.0079;
316 System(i).Tube_Length = 0;
317 System(i).Ntube = 0;
318 System(i).Pc = 4990;
319 System(i).LiquidLine = LiquidLine;
320 System(i).LiquidLine.line = 0;
321 System(i).LiquidLine.dia = 0.0079;
322
323 % other settings for storage
324 m = length(System);
325 % m = 1;
326 % m = 2;
327 for i = 1:m
328     System(i).mainfoldername = foldername;
329     System(i).r2_Q = 0;
```

```
330     System(i).r2_ΔP = 0;
331 end
332
333 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
334 libname ='lib'; % Name of library
335
336 % start calculation
337 no_plot = zeros(m,1);
338 % make an error statment
339 try
340     parfor i = 1:m
341         System(i) = 1; % a statement leads to error
342     end
343 catch err
344 end
345 for i = 1:m
346     error_statement(i) = err;
347 end
348 % parfor i = 1:m
349 for i = 1:m
350     try
351         disp(['Starting to run ',System(i).name]);
352         [LiquidLine r2_ΔP r2_Q] = regress_func(...
353             System(i), libname, version_main ...
354         );
355         System(i).LiquidLine = LiquidLine;
```

```
356         System(i).r2_Q = r2_Q;
357         System(i).r2_ΔP = r2_ΔP;
358         disp(['Finishing simulating ',System(i).name]);
359     catch err
360         disp(['Problem found in system ',System(i).name]);
361         no_plot(i,1) = 1;
362         error_statement(i) = err;
363     end
364 end
365 save(savefilename);
366
367 % plot cannot be done in parfor. Do it in a for loop
368 for i = 1:m
369     if no_plot(i,1) == 0
370         disp(['Starting to plot ',System(i).name]);
371         plot_func(System(i), libname, version_main);
372         disp(['Finishing plotting ',System(i).name]);
373     end
374 end
375
376 save(savefilename);
377 movefile(savefilename,strcat(foldername,'\'));
378
379 try
380     matlabpool close; % if catch an error, close and reset it
381 end
```

```
382 try
383     unloadlibrary lib;
384 end
```

```
1  function [PipeLineClass r2_ΔP r2_Q] = ...
2      LiquidLine_minimization_v037(...
3          System,libname,version_number ...
4      )
5
6  % Function to train liquid line model
7
8  %% Data reading
9  bin_num = 10;
10 PipeLineClass = System.LiquidLine;
11 filename = System.filename;
12 worksheetname = System.worksheetname;
13 refri = System.refname;
14 Standard_Airflow_cond = System.Standard_Airflow_cond;
15 Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
16 version = strcat(System.name,'_',version_number);
17 savefilename = ...
18     strcat('LiquidLine_minimization_v',version,'.mat');
19 foldername = strcat(strcat(...
20     pwd,'\',System.mainfoldername,'\' ...
21 ),strcat('LiquidLine_minimization_v',version));
```

```
22 mkdir(foldername);
23
24 % read Q_gain_observation_v010 to find points to be removed
25 Q_gain_version = '030';
26 Q_gain_name = [...
27     'Q_gain_observation_v',System.name,'_',Q_gain_version ...
28 ];
29 Q_gain_raw = open([...
30     'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,'\',...
31     Q_gain_name,'.mat' ...
32 ]);
33 k = length(Q_gain_raw.index_removed);
34 data2 = Q_gain_raw.data;
35 code = Q_gain_raw.code;
36 fault = Q_gain_raw.fault;
37 Q_cond_r = Q_gain_raw.Q_cond_r;
38 Q_evap_r = Q_gain_raw.Q_evap_r;
39 for i = k:-1:1 % reverse for correct indexing
40     data2(Q_gain_raw.index_removed(i),:) = [];
41     code(Q_gain_raw.index_removed(i),:) = [];
42     fault(Q_gain_raw.index_removed(i),:) = [];
43     Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
44     Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
45 end
46 [pp qq] = size(data2);
47 cell_VL = strfind(fault,'VL');
```

```
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
cell_NF = strfind(fault,'NF');
if isempty(cell_NF)
    cell_LL = cell(pp,1);
end
i = 1;

if strcmp(System.name,'Breuker_030_R22_packaged_FXO_Recip')
    while i ≤ pp
        if isempty(cell2mat(cell_NF(i)))
            data2(i,:) = [];
            fault(i,:) = [];
            code(i,:) = [];
            cell_VL(i,:) = [];
            cell_LL(i,:) = [];
            cell_NC(i,:) = [];
```

```
74              Q_cond_r(i,:) = [];
75              Q_evap_r(i,:) = [];
76              i = i - 1;
77              pp = pp - 1;
78          end
79          i = i + 1;
80      end
81  else
82      while i ≤ pp
83          if ¬isempty(cell2mat(cell_VL(i))) || ...
84              ¬isempty(cell2mat(cell_NC(i))) || ...
85              ¬isempty(cell2mat(cell_LL(i)))
86              data2(i,:) = [];
87              fault(i,:) = [];
88              code(i,:) = [];
89              cell_VL(i,:) = [];
90              cell_LL(i,:) = [];
91              cell_NC(i,:) = [];
92              Q_cond_r(i,:) = [];
93              Q_evap_r(i,:) = [];
94              i = i - 1;
95              pp = pp - 1;
96          end
97          i = i + 1;
98      end
99  end
```

```
100 [n dummy] = size(data2);
101
102 % for Compressor
103 version_Comp = '075';
104 version_func = '045';
105 name_Comp = 'Compressor_regression_v';
106 component_name = strcat(name_Comp,System.othername,'_',version_Comp);
107 foldername_comp = strcat(strcat(...
108     pwd,'\',name_Comp,version_Comp,'\',component_name ...
109 ));
110 raw = open(strcat(foldername_comp,'\',strcat(component_name,'.mat')));
111 Comp_para = raw.Comp;
112 Comp_func = str2func([...
113     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
114     'Compressor_v',version_func,...
115     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
116 ]);
117
118 %% LiquidLine Line
119 data_HG = data2; % assign data to avoid re-reading later
120
121 % Obtain Superheat, Mass Flow Rate and Pressures
122 i = 1;
123 for j = 1:n
124     % check data to see if it is correct
125     P_in_LL = data_HG(j,4);
```

```
126     if data_HG(j,5) > data_HG(j,4)
127         P_out_LL = data_HG(j,4);
128     else
129         P_out_LL = data_HG(j,5);
130     end
131     if System.Heating==0
132         Tamb_a = data_HG(j,21);
133     else
134         Tamb_a = data_HG(j,17);
135     end
136     SC_check = propertyRH('T','P',P_in_LL,'Q',0,refri) - ...
137         data_HG(j,12);
138     SC_check_II = propertyRH(...
139         'T','P',P_out_LL,'Q',0,refri ...
140     )-data_HG(j,13);
141     T_in_LL = data_HG(j,12);
142     if SC_check ≥ 1 & SC_check_II ≥ 1
143         h_in_LL = propertyRH(...
144             'H','T',data_HG(j,12),'P',P_in_LL,refri ...
145         );
146         h_out_LL = propertyRH(...
147             'H','T',data_HG(j,13),'P',P_out_LL,refri ...
148         );
149         if (data_HG(j,4) - Tamb_a)/(h_in_LL - h_out_LL) > 0
150             T_out_LL = data_HG(j,13);
151         else
```

```
152             T_out_LL = propertyRH(...
153                 'T','P',P_out_LL,'H',h_out_LL,refri ...
154             );
155             SC_check_II = propertyRH(...
156                 'T','P',P_out_LL,'Q',0,refri ...
157             )-T_out_LL;
158         end
159     else
160         T_out_LL = data_HG(j,13);
161     end
162     SH_comp = data_HG(j,9) - ...
163         propertyRH('T','P',data_HG(j,1),'Q',1,refri);
164     SH_evap = data_HG(j,16) - ...
165         propertyRH('T','P',data_HG(j,8),'Q',1,refri);
166     if ((((SC_check ≥ 3)||(...
167             SH_evap>1&&SC_check_II≥1 ...
168         )||(SC_check≥1)) && data_HG(i,23) > 0) || SH_comp>1) && ...
169             (SC_check_II>1 || SH_evap > 1)
170         if ¬(((SC_check ≥ 3)||(...
171             SH_evap>1&&SC_check_II≥1 ...
172         )||(SC_check≥1)) && data_HG(i,23) > 0)
173             if System.Heating==0
174                 Tamb_a = data_HG(j,21);
175             else
176                 Tamb_a = data_HG(j,17);
177             end
```

```
178             data_HG(j,23) = Comp_func(data_HG(j,1), propertyRH(...
179                 'H','T',data_HG(j,9),'P',data_HG(j,1),refri...
180             ), data_HG(j,2), Tamb_a, data_HG(j,10), Comp_para, refri);
181         end
182         if SC_check > 1
183             h_temp = propertyRH(...
184                 'H','T',data_HG(j,12),'P',data_HG(j,4),refri ...
185             );
186         else
187             h_temp = propertyRH(...
188                 'H','T',data_HG(j,11),'P',data_HG(j,3),refri ...
189             )-Q_cond_r(j,1)/data_HG(j,23);
190         end
191         if SC_check_II > 1
192             hout_temp = propertyRH(...
193                 'H','T',T_out_LL,'P',P_out_LL,refri ...
194             );
195         else
196             hout_temp = propertyRH(...
197                 'H','T',data_HG(j,16),'P',data_HG(j,8),refri ...
198             ) - Q_evap_r(j,1)/data_HG(j,23);
199         end
200         % calculate new SC and see if the values of expt. and
201         % sim. do match
202         if SC_check ≥ 1 & SC_check_II ≥ 1
203             h(i,1) = h_temp;
```

```
204             hout(i,1) = hout_temp;
205             SC(i,1) = SC_check;
206             SC_II(i,1) = SC_check_II;
207             mdot(i,1) = data_HG(j,23);
208             SH(i,1) = data_HG(j,10) - ...
209                 propertyRH('T','P',data_HG(j,2),'Q',1,refri);
210             Q(i,1) = mdot(i,1)*(h(i,1)-hout(i,1));
211             ΔP(i,1) = P_in_LL - P_out_LL;
212             rho(i,1) = propertyRH('D','P',P_in_LL,'H',h(i,1),refri);
213             rho_out(i,1) = ...
214                 propertyRH('D','P',P_out_LL,'H',hout(i,1),refri);
215             rho_out_est(i,1) = ...
216                 propertyRH('D','P',P_in_LL,'H',hout(i,1),refri);
217             if SC_check > 1
218                 V(i,1) = propertyRH(...
219                     'V','T',data_HG(j,12),'P',data_HG(j,4),refri...
220                 );
221             else
222                 V(i,1) = propertyRH(...
223                     'V','P',data_HG(j,4),'Q',0,refri ...
224                 );
225             end
226             T(i,1) = T_in_LL;
227             if System.Heating==0
228                 Tamb(i,1) = data_HG(j,21);
229             else
```

```
230                 Tamb(i,1) = data_HG(j,17);
231             end
232             P = data_HG(j,4);
233             code_HG(i,1) = code(j,1);
234             data_h(i,:) = data_HG(j,:);
235             data_Q(i,:) = data_HG(j,:);
236             i = i + 1;
237         end
238     end
239 end
240 mdot_rated = mdot(1,1);
241 Q_rated = mean(Q);
242 ΔP_av = mean(ΔP);
243 rho_rated = mean(rho);
244 V_rated = mean(V);
245 SH_limit = 1;
246 SC_limit = 3;
247
248 options = optimset('MaxFunEvals',10000,'MaxIter',1000);
249 X1 = (mdot./mdot_rated);
250 X2 = (V./V_rated);
251 X3 = (rho./rho_rated);
252 X4 = (mdot./mdot_rated).^2.*(1./rho_out_est - 1./rho).*rho_rated;
253 C = zeros(3,1);
254 A = -eye(length(C));
255 % A(3,3) = 1; % Higher density, smaller pressure drop
```

```
% A(5,5) = -1;
b = zeros(length(C),1);
ΔP(ΔP<0) = 0;
if length(ΔP) > 1 & sum(ΔP.^2)>1.e-8
    weigh = weighing_function_v000(ΔP, bin_num);
elseif sum(ΔP.^2)≤1.e-8
    weigh = ones(length(ΔP),1);
else
    weigh = 1;
end
if length(SC>3&SC_II>3) > 5
    options = optimset(...
        'MaxFunEvals',20000,'algorithm','active-set',...
        'MaxIter',5000,'Display','off' ...
    );
    try
        [C, fval] = fmincon(@(C) PressureDrop_residual(...
            C, X1, X2, X3, X4, ΔP, weigh ...
        ), C, [], [], [], [], [0 1 0],[Inf 2 1],[],options);
        if sum(isnan(C))
            options = optimset(...
                'MaxFunEvals',20000,'algorithm','sqp',...
                'MaxIter',5000,'Display','off' ...
            );
            C = zeros(5,1);
            [C, fval] = fmincon(@(C) PressureDrop_residual(...
```

```
282                 C, X1, X2, X3, X4, ΔP, weigh ...
283             ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
284         end
285     catch err
286         options = optimset(...
287             'MaxFunEvals',20000,'algorithm','sqp',...
288             'MaxIter',5000,'Display','off' ...
289         );
290         C = zeros(5,1);
291         [C, fval] = fmincon(@(C) PressureDrop_residual(...
292             C, X1, X2, X3, X4, ΔP, weigh ...
293         ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
294     end
295     if sqrt(fval) > 40
296         options = optimset(...
297             'MaxFunEvals',20000,'algorithm','sqp',...
298             'MaxIter',5000,'Display','off' ...
299         );
300         C = zeros(5,1);
301         [C, fval] = fmincon(@(C) PressureDrop_residual(...
302             C, X1, X2, X3, X4, ΔP, weigh ...
303         ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
304     end
305     C(4,1) = -1;
306 else
307     C(1) = mean(ΔP);
```

```
308     C(4,1) = 0;
309 end
310
311 ΔP_est = PressureDrop(C, X1, X2, X3, X4);
312 dev = ΔP_est - ΔP;
313 r2_ΔP = 1 - sum(dev.^2)/sum((ΔP - mean(ΔP)).^2);
314 max_dev = round(max(abs(dev)));
315 X = [];
316 X(:,1) = (X1.^C(2).*X2.^C(3).*X3.^C(4));
317 X(:,2) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X1);
318 X(:,3) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X2);
319 X(:,4) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X3);
320 COV_X = covariance_adj(X);
321 PipeLineClass.COV_X_ΔP = COV_X;
322 PipeLineClass.ΔP_limit = max(diag(X/(X'*X)*X'));
323
324 % estimate the heat loss model
325 Δh_rated = 100/mdot_rated;
326 C_Q = [2 1]';
327 if ¬isempty(mdot.*(h-hout))& sum((mdot.*(h-hout)).^2)>1.e-8
328     % zero the negative heat loss
329     mm = length(mdot);
330     i = 1;
331     while i ≤ mm
332         if (h(i)-hout(i))*(T(i)-Tamb(i))<0
333             h(i,:) = [];
```

```
334             hout(i,:) = [];
335             mdot(i,:) = [];
336             T(i,:) = [];
337             Tamb(i,:) = [];
338             i = i - 1;
339             mm = mm - 1;
340         end
341         i = i + 1;
342     end
343     Q = mdot.*(h-hout);
344     if length(h) > 5 & max(Q)>0
345         if length(mdot.*(h-hout)) > 1 & ...
346             sum((mdot.*(h-hout)).^2)>1.e-8
347             weigh = weighing_function_v000(mdot.*(h-hout), bin_num);
348         elseif sum((mdot.*(h-hout)).^2)≤1.e-8
349             weigh = ones(length(mdot.*(h-hout)),1);
350         else
351             weigh = 1;
352         end
353         DiffT = T-Tamb;
354         max_index = find(DiffT==max(DiffT));
355         Th = T(max_index);
356         Tl = Tamb(max_index);
357         Qmax = Q(max_index);
358         Ra_max = Rayleigh(Th, Tl, System.LiquidLine.dia);
359         [HTC_max, n_max] = Natural_HTC_air(...
```

```
360             Ra_max, System.LiquidLine.dia ...
361         );
362         C_Q_1_max = Qmax/(...
363             (abs(Th-Tl)./Th).^n_max.*(Th-Tl)/(...
364                 T(1)-Tamb(1) ...
365             )*mdot_rated*(h(1)-hout(1)) ...
366         );
367         C_Q = fmincon(@(C_Q) HeatLoss(...
368             C_Q, h, hout, mdot, T, Tamb, mdot_rated, h(1)-hout(1), ...
369             weigh ...
370         ), C_Q, [], [], [], [], [0 0.058], [...
371             C_Q_1_max 0.333 ...
372         ], [], optimset('GradObj', 'on', 'DerivativeCheck', 'on'));
373         Q_est = C_Q(1).*(...
374             abs(T-Tamb)./T ...
375         ).^C_Q(2).*(T-Tamb)/(T(1)-Tamb(1))*mdot_rated*(h(1)-hout(1));
376         ΔT_rated = T(1)-Tamb(1);
377         Δh_rated = h(1)-hout(1);
378         dev_Q = (Q_est - Q)./Q;
379         r2_Q = 1 - sum((Q_est - Q).^2)/sum((Q - mean(Q)).^2);
380         X = [];
381         X(:,1) = (abs(T-Tamb)./T).^C_Q(2).*(...
382             T-Tamb ...
383         )/(T(1)-Tamb(1))*mdot_rated*(h(1)-hout(1));
384         X(:,2) = C_Q(1).*(...
385             abs(T-Tamb)./T ...
```

```
386         ).^C_Q(2).*(T-Tamb)/(T(1)-Tamb(1))*mdot_rated*(...
387             h(1)-hout(1) ...
388         ).*log(abs(T-Tamb)./T);
389         COV_X = (X'*X)^(-1);
390         PipeLineClass.COV_X_Q = COV_X;
391         PipeLineClass.Q_limit = max(diag(X/(X'*X)*X'));
392     else
393         C_Q = [0 0];
394         Q_est = 0;
395         Q = 0;
396         ΔT_rated = 0;
397         Δh_rated = 0;
398         dev_Q = 0;
399         r2_Q = -9999;
400         PipeLineClass.COV_X_Q = zeros(length(C_Q),length(C_Q));
401         PipeLineClass.Q_limit = Inf;
402     end
403 else
404     C_Q = [Inf 0 Inf 0];
405     Q_est = 0;
406     Q = 0;
407     ΔT_rated = 0;
408     Δh_rated = 0;
409     dev_Q = 0;
410     r2_Q = -9999;
411     PipeLineClass.COV_X_Q = zeros(length(C_Q),length(C_Q));
```

```
412     PipeLineClass.Q_limit = Inf;
413 end
414 max_dev_Q = round(max(abs(dev_Q)*100));
415 
416 PipeLineClass.mdot_rated = mdot_rated;
417 PipeLineClass.V_rated = V_rated;
418 PipeLineClass.rho_rated = rho_rated;
419 PipeLineClass.C = C;
420 PipeLineClass.C_Q = C_Q;
421 PipeLineClass.Δh_rated = Δh_rated;
422 PipeLineClass.ΔT_rated = ΔT_rated;
423 
424 save(savefilename);
425 movefile(savefilename,strcat(foldername,'\'));
426 
427 end
428 
429 function [residual_Q, grad_Q] = HeatLoss(...
430     C_Q, h, hout, mdot, T, Tamb, Δh_rated, mdot_rated, weigh ...
431 )
432     residual_Q = (...
433         (h-hout).*mdot/mdot_rated/(Δh_rated) - ...
434         C_Q(1).*(abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1)) ...
435     );
436     n_Q = length(residual_Q);
437     gra_Q_ori(1:n_Q,1) = -2.*(residual_Q).*1./weigh;
```

```
438     residual_Q = sum(1./weigh.*residual_Q.^2);
439     grad_Q(1,1) = sum(gra_Q_ori(1:n_Q,1).*(1.*(...
440         abs(T-Tamb)./T ...
441     ).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1))));
442     grad_Q(2,1) = sum(gra_Q_ori(1:n_Q,1).*(...
443         C_Q(1).*(T-Tamb)./(T(1)-Tamb(1)).* ...
444         log(abs(T-Tamb)./T).*(abs(T-Tamb)./T).^C_Q(2)) ...
445     );
446 end
447
448 function residual = PressureDrop_residual(...
449     C, X1, X2, X3, X4, ΔP, weigh ...
450 )
451 n = length(X1);
452 residual = 0;
453 for i = 1:n
454     if ΔP(i,1)>0
455         residual = residual + 1/weigh(i,1)*(...
456             ΔP(i,1)-PressureDrop(...
457                 C, X1(i,1), X2(i,1), X3(i,1), X4(i,1) ...
458             ) ...
459         ).^2;
460     end
461 end
462 residual = residual/n;
463 end
```

```
464
465 function ΔP_est = PressureDrop(C, X1, X2, X3, X4)
466 % function to estimate the pressure drop
467 ΔP_est = (C(1)*X1.^C(2).*X2.^C(2)./X3);
468 end
```

N.6 FXO Modeling

```
1  function System = FEO_minimization_v070_main()
2
3  % Function to pass information to train fixed orifice model
4
5  % define function and folder name
6  function_name = 'FEO_minimization';
7  version_main = '070';
8  version_regress = '071';
9  version_plot = '002';
10 foldername = strcat(function_name,'_v',version_main);
11 regressname = strcat(function_name,'_v',version_regress);
12 plotname = strcat(function_name,'_plot_v',version_plot);
13 regress_func = str2func(['@(System, libname, version_number)',...
14     regressname,'(System, libname, version_number)']);
15 plot_func = str2func(['@(System, libname, version_number)',...
16     plotname,'(System, libname, version_number)']);
17 mkdir(foldername);
18 savefilename = strcat(foldername,'.mat');
19
```

```
20 % Fixed orifice (FEO_constant_v22.m)
21 FEO.para.adj = 1; % Tuning factor for mass flow rates [-]
22 FEO.para.D = 0;
23 FEO.para.L = 0;
24 FEO.para.depth = 0;
25 FEO.para.Geometry_index = zeros(1,3);
26 FEO.para.mode = 2;
27 FEO.para.TPADJ = 0;
28
29 % Coefficients as defined in Payne (2004)
30 FEO.para.a(1,1) = 3.8811E-01;% [-]
31 FEO.para.a(2,1) = 1.1427E+01;% [-]
32 FEO.para.a(3,1) = -1.4194E+01;% [-]
33 FEO.para.a(4,1) = 1.0703E+00;% [-]
34 FEO.para.a(5,1) = -9.1928E-02;% [-]
35 FEO.para.a(6,1) = 2.1425E+01;% [-]
36 FEO.para.a(7,1) = -5.8195E+02;% [-]
37
38 FEO.para.b(1,1) = 1.1831E+00;% [-]
39 FEO.para.b(2,1) = -1.4680E+00;% [-]
40 FEO.para.b(3,1) = -1.5285E-01;% [-]
41 FEO.para.b(4,1) = -1.4639E+01;% [-]
42 FEO.para.b(5,1) = 9.8401E+00;% [-]
43 FEO.para.b(6,1) = -1.9798E-02;% [-]
44 FEO.para.b(7,1) = -1.5348E+00;% [-]
45 FEO.para.b(8,1) = -2.0533E+00;% [-]
```

```
46 FEO.para.b(9,1) = -1.7195E+01;% [-]
47
48 % Set information
49 % re-file at this stage to avoid problems in parfor loop
50
51 % Breuker 3-ton R22 packaged FXO system (Recip)
52 % (Breuker_030_R22_packaged_FXO_Recip)
53 i = 1;
54 System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
55 System(i).othername = 'Breuker_030_R22_packaged_FXO_Recip';
56 System(i).cond_name = System(i).name;
57 System(i).LLname = System(i).name;
58 System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
59 System(i).foldername = 'Breuker';
60 System(i).worksheetname = 'SI (no_invalid_CA)';
61 System(i).refname = 'R22.fld';
62 System(i).Standard_Airflow_evap = 0;
63 System(i).Fan_Upstream_evap = 0;
64 System(i).Standard_Airflow_cond = 1;
65 System(i).Fan_Upstream_cond = 1;
66 System(i).Heating = 0;
67 System(i).Combined = 0;
68 % assumed
69 System(i).Diameter = 0.00849;
70 System(i).Tube_Length = 2.255;
71 System(i).Ntube = 32;
```

```
72 System(i).Pc = 4990;
73 System(i).Tc = 369.3;
74 System(i).FEO.para.mode = 2;
75
76 % Boshen 3-ton R410A packaged FXO system (Scroll)
77 % (Shen_030_R410A_packaged_FXO_Scroll)
78 i = i + 1;
79 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
80 System(i).othername = System(i).name;
81 System(i).cond_name = System(i).name;
82 System(i).LLname = System(i).name;
83 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
84 System(i).foldername = 'Boshen';
85 System(i).worksheetname = 'SI (no_invalid_CA)';
86 System(i).refname = 'R410A';
87 System(i).Standard_Airflow_evap = 0;
88 System(i).Fan_Upstream_evap = 0;
89 System(i).Standard_Airflow_cond = 1;
90 System(i).Fan_Upstream_cond = 1;
91 System(i).Heating = 0;
92 System(i).Combined = 0;
93 System(i).Diameter = 0.00853;
94 System(i).Tube_Length = 1.67075;
95 System(i).Ntube = 40;
96 System(i).Pc = 4901;
97 System(i).Tc = 344.49;
```

```
98  System(i).FEO.para.mode = 2;
99
100 % Boshen 3-ton R410A split FXO system (Recip)
101 % (Shen_030_R410A_split_FXO_Recip)
102 % for FEO model, ignore the TXV data
103 i = i + 1;
104 System(i).name = 'Shen_030_R410A_split_FXO_Recip';
105 System(i).othername = 'Shen_030_R410A_split_FXO_TXV_Recip';
106 System(i).cond_name = 'Shen_030_R410A_split_FXO_TXV_Recip';
107 System(i).LLname = System(i).othername;
108 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO.xls';
109 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
110 System(i).worksheetname = 'SI (no_invalid_CA)';
111 System(i).refname = 'R410A';
112 System(i).Standard_Airflow_evap = 0;
113 System(i).Fan_Upstream_evap = 0;
114 System(i).Standard_Airflow_cond = 1;
115 System(i).Fan_Upstream_cond = 1;
116 System(i).Heating = 0;
117 System(i).Combined = 0;
118 System(i).Diameter = 0.00849;
119 System(i).Tube_Length = 2.255;
120 System(i).Ntube = 32;
121 System(i).Pc = 4901;
122 System(i).Tc = 344.49;
123 System(i).FEO.para.mode = 2;
```

```
124
125 % Boshen 5-ton R407C packaged FXO system (Scroll)
126 % (Shen_050_R407C_packaged_FXO_Scroll)
127 i = i + 1;
128 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
129 System(i).othername = 'Shen_050_R407C_packaged_FXO_Scroll';
130 System(i).cond_name = System(i).name;
131 System(i).LLname = System(i).name;
132 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
133 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
134 System(i).worksheetname = 'SI (no_invalid_CA)';
135 System(i).refname = 'R407C';
136 System(i).Standard_Airflow_evap = 0;
137 System(i).Fan_Upstream_evap = 0;
138 System(i).Standard_Airflow_cond = 1;
139 System(i).Fan_Upstream_cond = 1;
140 System(i).Heating = 0;
141 System(i).Combined = 0;
142 System(i).Diameter = 0.00693928;
143 System(i).Tube_Length = 2.282;
144 System(i).Ntube = 56;
145 System(i).Pc = 3786;
146 System(i).Tc = 359.35;
147 System(i).FEO.para.mode = 2;
148
149 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
```

```
150 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
151 i = i + 1;
152 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
153 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
154 System(i).cond_name = System(i).name;
155 System(i).LLname = System(i).name;
156 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
157 System(i).foldername = 'Kim-HCA3-Heating';
158 System(i).worksheetname = 'SI (no_invalid_CA)';
159 System(i).refname = 'R410A';
160 System(i).Standard_Airflow_evap = 0;
161 System(i).Fan_Upstream_evap = 0;
162 System(i).Standard_Airflow_cond = 1;
163 System(i).Fan_Upstream_cond = 1;
164 System(i).Heating = 1;
165 System(i).Combined = 0;
166 % assumed
167 System(i).Diameter = 0.00849;
168 System(i).Tube_Length = 2.255;
169 System(i).Ntube = 32;
170 System(i).Pc = 4901;
171 System(i).Tc = 344.49;
172 System(i).FEO.para.mode = 2;
173
174 % Kim 3-ton R22 split TXV system (Scroll) (YKC)
175 % (YKC_030_R22_split_TXV_Scroll)
```

```
176 i = i + 1;
177 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
178 System(i).othername = 'YKC_030_R22_split_TXV_Scroll';
179 System(i).cond_name = System(i).name;
180 System(i).LLname = System(i).name;
181 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
182 System(i).foldername = 'Kim-YKC';
183 System(i).worksheetname = 'SI (no_invalid_CA)';
184 System(i).refname = 'R22.fld';
185 System(i).Standard_Airflow_evap = 0;
186 System(i).Fan_Upstream_evap = 0;
187 System(i).Standard_Airflow_cond = 1;
188 System(i).Fan_Upstream_cond = 1;
189 System(i).Heating = 0;
190 System(i).Combined = 0;
191 % assumed
192 System(i).Diameter = 0.00849;
193 System(i).Tube_Length = 2.255;
194 System(i).Ntube = 32;
195 System(i).Pc = 4990;
196 System(i).Tc = 369.3;
197 System(i).FEO.para.mode = 2;
198
199 % other settings for storage
200 m = length(System);
201 for i = 1:m
```

```
202     System(i).mainfoldername = foldername;
203     System(i).FEO = FEO;
204     System(i).r2_m = 0;
205 end
206 
207 % setting up parallel solver
208 try
209     matlabpool(3); % try setting up multiple loops
210 catch err
211     matlabpool close; % if catch an error, close and reset it
212     matlabpool(3);
213 end
214 
215 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
216 libname ='lib'; % Name of library
217 
218 % start calculation
219 % m = 2;
220 % for i = 1:m
221 no_plot = zeros(m,1);
222 % make an error statment
223 try
224     parfor i = 1:m
225         System(i) = 1; % a statement leads to error
226     end
227 catch err
```

```
228 end
229 for i = 1:m
230     error_statement(i) = err;
231 end
232 parfor i = 1:m
233 % for i = 1:m
234     try
235         disp(['Starting to run ',System(i).name]);
236         [FEO_class r2_m] = regress_func(...
237             System(i), libname, version_main ...
238         );
239         System(i).FEO = FEO_class;
240         System(i).r2_m = r2_m;
241         disp(['Finishing simulating ',System(i).name]);
242     catch err
243         disp(['Problem found in system ',System(i).name]);
244         no_plot(i,1) = 1;
245         error_statement(i) = err;
246     end
247 end
248 save(savefilename);
249
250 % plot cannot be done in parfor. Do it in a for loop
251 for i = 1:m
252 % for i = 1:1
253     if no_plot(i,1) == 0
```

```
254         disp(['Starting to plot ',System(i).name]);
255         plot_func(System(i), libname, version_main);
256         disp(['Finishing plotting ',System(i).name]);
257     end
258 end
259 
260 save(savefilename);
261 movefile(savefilename,strcat(foldername,'\'));
262 
263 try
264     matlabpool close; % if catch an error, close and reset it
265 end
266 try
267     unloadlibrary lib;
268 end
```

```
1 function [FEO r2_m] = FEO_minimization_v071(...
2     System, libname, version_number ...
3 )
4 
5 % Function to train fixed orifice model
6 
7 %% Preparation
8 % data reading and filtering
9 bin_num = 10;
```

```
filename = System.filename;
worksheetname = System.worksheetname;
refri = System.refname;
Standard_Airflow_cond = System.Standard_Airflow_cond;
Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
D = System.Diameter;
Ltube = System.Tube_Length;
Ntube = System.Ntube;
Pc = System.Pc;
Tc = System.Tc;
version = strcat(System.name,'_',version_number);
savefilename = strcat('FEO_minimization_v',version,'.mat');
foldername = strcat(strcat(...
    pwd,'\',System.mainfoldername,'\' ...
),strcat('FEO_minimization_v',version));
mkdir(foldername);

Q_gain_version = '030';
Q_gain_name = [...
    'Q_gain_observation_v',System.name,'_',Q_gain_version ...
];
Q_gain_raw = open([...
    'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,...
    '\',Q_gain_name,'.mat' ...
]);
k = length(Q_gain_raw.index_removed);
```

```
data2 = Q_gain_raw.data;
code = Q_gain_raw.code;
fault = Q_gain_raw.fault;
Q_cond_r = Q_gain_raw.Q_cond_r;
Q_evap_r = Q_gain_raw.Q_evap_r;
for i = k:-1:1 % reverse for correct indexing
    data2(Q_gain_raw.index_removed(i),:) = [];
    code(Q_gain_raw.index_removed(i),:) = [];
    fault(Q_gain_raw.index_removed(i),:) = [];
    Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
    Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
end
[pp qq] = size(data2);
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
i = 1;
```

```
while i ≤ pp
    if ¬isempty(cell2mat(cell_VL(i))) || ...
        ¬isempty(cell2mat(cell_NC(i))) || ...
        ¬isempty(cell2mat(cell_LL(i)))
        data2(i,:) = [];
        fault(i,:) = [];
        code(i,:) = [];
        cell_VL(i,:) = [];
        cell_LL(i,:) = [];
        cell_NC(i,:) = [];
        Q_evap_r(i,:) = [];
        Q_cond_r(i,:) = [];
        i = i - 1;
        pp = pp - 1;
    end
    i = i + 1;
end
[m n] = size(data2);

% for Compressor
version_Comp = '075';
version_func = '045';
name_Comp = 'Compressor_regression_v';
component_name = strcat(...
    name_Comp,System.othername,'_',version_Comp ...
);
```

```
88 foldername_comp = strcat(strcat(...
89     pwd,'\',name_Comp,version_Comp,'\',component_name ...
90 ));
91 raw = ..
92     open(strcat(foldername_comp,'\',strcat(component_name,'.mat')));
93 Comp_para = raw.Comp;
94 Comp_func = str2func([...
95     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
96     'Compressor_v',version_func,...
97     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
98 ]);
99
100 % for Liquid Line
101 version_LiquidLine = '040';
102 version_func = '003';
103 name_LiquidLine = 'LiquidLine_minimization_v';
104 component_name = strcat(...
105     name_LiquidLine,System.LLname,'_',version_LiquidLine ...
106 );
107 foldername_LiquidLine = strcat(strcat(...
108     pwd,'\',name_LiquidLine,version_LiquidLine,'\',component_name ...
109 ));
110 raw = open(strcat(...
111     foldername_LiquidLine,'\',strcat(component_name,'.mat') ...
112 ));
113 LiquidLine_para = raw.PipeLineClass;
```

```
114 LiquidLine_func = str2func([...
115     '@(mr, Pin, hin, Tamb, para, refri, phase, libname)',...
116     'pipeline_v',version_func,...
117     '(mr, Pin, hin, Tamb, para, refri, phase, libname)' ...
118 ]);
119
120 % set deffinition of FEO class
121 FEO.para.adj = 1; % Tuning factor for mass flow rates [-]
122 FEO.para.D = 0;
123 FEO.para.L = 0;
124 FEO.para.depth = 0;
125 FEO.para.mode = 2;
126 FEO.para.TPADJ = 0;
127
128 % Coefficients as defined in Payne (2004)
129 FEO.para.a(1,1) = 3.8811E-01;% [-]
130 FEO.para.a(2,1) = 1.1427E+01;% [-]
131 FEO.para.a(3,1) = -1.4194E+01;% [-]
132 FEO.para.a(4,1) = 1.0703E+00;% [-]
133 FEO.para.a(5,1) = -9.1928E-02;% [-]
134 FEO.para.a(6,1) = 2.1425E+01;% [-]
135 FEO.para.a(7,1) = -5.8195E+02;% [-]
136
137 FEO.para.b(1,1) = 1.1831E+00;% [-]
138 FEO.para.b(2,1) = -1.4680E+00;% [-]
139 FEO.para.b(3,1) = -1.5285E-01;% [-]
```

```
140 FEO.para.b(4,1) = -1.4639E+01;% [-]
141 FEO.para.b(5,1) = 9.8401E+00;% [-]
142 FEO.para.b(6,1) = -1.9798E-02;% [-]
143 FEO.para.b(7,1) = -1.5348E+00;% [-]
144 FEO.para.b(8,1) = -2.0533E+00;% [-]
145 FEO.para.b(9,1) = -1.7195E+01;% [-]
146
147 %% Calculation
148 % data filtering
149 i = 1;
150 while(i≤m)
151     if data2(i,5) > data2(i,4)
152         data2(i,5) = data2(i,4);
153     end
154     hout_cond = propertyRH(...
155         'H','T',data2(i,12),'P',data2(i,4),refri ...
156     );
157     hin_EXV = propertyRH(...
158         'H','T',data2(i,13),'P',data2(i,5),refri ...
159     );
160     % if heat cannot be lost in the direction of temperature,
161     % change the temperature reading
162     if data2(i,12) - data2(i,13) < 0.5
163         hin_EXV = hout_cond; % assume negligible heat loss
164         data2(i,13) = propertyRH(...
165             'T','P',data2(i,5),'H',hout_cond,refri ...
```

```
166         );
167     end
168     if (hout_cond - hin_EXV)/(data2(i,12)-data2(i,21)) < 0
169         data2(i,13) = propertyRH(...
170             'T','P',data2(i,5),'H',hout_cond,refri ...
171         );
172     end
173     Tsat = propertyRH('T','P',data2(i,5),'Q',0,refri);
174     Tsat_II = propertyRH('T','P',data2(i,4),'Q',0,refri);
175     SH_comp = data2(i,9)-propertyRH(...
176         'T','P',data2(i,1),'Q',1,refri ...
177     );
178     SH_evap = data2(i,16) - propertyRH(...
179         'T','P',data2(i,8),'Q',1,refri ...
180     );
181     removal_confirm = 0;
182     if (((Tsat_II - data2(i,12)>3 && ...
183             Tsat - data2(i,13)>3) || ...
184             (Tsat_II - data2(i,12)≥1) || ...
185             (Tsat - data2(i,13)>1 && ...
186             SH_evap > 1))&&(data2(i,23)>0)) || ...
187             SH_comp > 1
188         xin = (hin_EXV - propertyRH(...
189             'H','P',data2(i,5),'Q',0,refri ...
190         ))/(propertyRH(...
191             'H','P',data2(i,5),'Q',1,refri ...
```

```
192         )-propertyRH('H','P',data2(i,5),'Q',0,refri));
193         if strcmp(refri,'R22.fld') & xin > 0.102
194             removal_confirm = 1;
195         elseif strcmp(refri,'R410A') & xin > 0.087
196             removal_confirm = 1;
197         elseif strcmp(refri,'R407C') & xin > 0.047
198             removal_confirm = 1;
199         end
200     else
201         removal_confirm = 1;
202     end
203     if removal_confirm == 1
204         data2(i,:) = [];
205         code(i,:) = [];
206         fault(i,:) = [];
207         Q_evap_r(i,:) = [];
208         Q_cond_r(i,:) = [];
209         i = i - 1;
210         m = m - 1;
211     end
212     i = i + 1;
213 end
214
215 % calculate variables needed from filtered data
216 j = 1;
217 p = 1;
```

```
218 [m n] = size(data2);
219 while(p≤m)
220     % check subcooling first
221     i = p;
222     removal_confirm = 0;
223     Tsat = propertyRH('T','P',data2(i,5),'Q',0,refri);
224     Tsat_cond = propertyRH('T','P',data2(i,4),'Q',0,refri);
225     Tsat_evap = propertyRH('T','P',data2(i,8),'Q',1,refri);
226     SC_cond(i,1) = propertyRH(...
227         'T','P',data2(i,4),'Q',0,refri ...
228     )-data2(i,12);
229     SH_comp(i,1) = data2(i,9)-propertyRH(...
230         'T','P',data2(i,1),'Q',1,refri ...
231     );
232     SH_evap = data2(i,16)-propertyRH(...
233         'T','P',data2(i,8),'Q',1,refri ...
234     );
235     SC(i,1) = Tsat - data2(i,13);
236     % estimate the mass flow rate first before
237     % estimating the enthalpy
238     if (((SC(i,1)>3 && SC_cond(i,1)>3) || ...
239             (SC_cond(i,1)≥1) || (...
240                 SC(i,1)>1 && SH_evap > 1 ...
241             ))&&(data2(i,23)>0))
242     else
243         if System.Heating==0
```

```
244             Tamb = data2(i,21);
245         else
246             Tamb = data2(i,17);
247         end
248         data2(i,23) = Comp_func(...
249             data2(i,1), propertyRH(...
250                 'H','T',data2(i,9),'P',data2(i,1),refri ...
251             ), data2(i,2), Tamb, data2(i,10), Comp_para, refri ...
252         );
253     end
254     if Tsat - data2(i,13)≥1
255         hin_EXV = propertyRH(...
256             'H','T',data2(i,13),'P',data2(i,5),refri ...
257         );
258     elseif data2(i,12) - data2(i,13) < 0.5 & ...
259         SC_cond(i,1) > 1
260         hin_EXV = propertyRH(...
261             'H','T',data2(i,12),'P',data2(i,4),refri ...
262         );
263     elseif data2(i,16)-Tsat_evap > 1
264         hin_EXV = propertyRH(...
265             'H','T',data2(i,16),'P',data2(i,8),refri ...
266         )-Q_evap_r(i,1)/data2(i,23);
267         data2(i,13) = propertyRH(...
268             'T','P',data2(i,5),'H',hin_EXV,refri ...
269         );
```

```
270     else
271         if SC_cond(i,1) > 1
272             hin_LL = propertyRH(...
273                 'H','T',data2(i,12),'P',data2(i,4),refri ...
274             );
275         else
276             hin_LL = propertyRH(...
277                 'H','T',data2(i,11),'P',data2(i,3),refri ...
278             )-Q_cond_r(i,1)/data2(i,23);
279         end
280         if System.Heating==0
281             Tamb = data2(i,21);
282         else
283             Tamb = data2(i,17);
284         end
285         [dum9 hin_EXV dum10] = LiquidLine_func(...
286             data2(i,23), data2(i,4), hin_LL, Tamb, ...
287             LiquidLine_para, refri, 0, libname ...
288         );
289         data2(i,13) = propertyRH(...
290             'T','P',data2(i,5),'H',hin_EXV,refri ...
291         );
292     end
293     SC(i,1) = Tsat - data2(i,13);
294     mr(i,1) = data2(i,23);
295     hl = propertyRH('H','P',data2(i,5),'Q',0,refri);
```

```
296     hv = propertyRH('H','P',data2(i,5),'Q',1,refri);
297     hin(i,1) = hin_EXV;
298     xin(i,1) = (hin(i,1)-hl)/(hv-hl);
299     removal_confirm = 0;
300     if strcmp(refri,'R22.fld') & xin(i,1) > 0.102
301         removal_confirm = 1;
302     elseif strcmp(refri,'R410A') & xin(i,1) > 0.087
303         removal_confirm = 1;
304     elseif strcmp(refri,'R407C') & xin(i,1) > 0.047
305         removal_confirm = 1;
306     end
307     if removal_confirm == 1
308         data2(i,:) = [];
309         code(i,:) = [];
310         fault(i,:) = [];
311         Q_evap_r(i,:) = [];
312         Q_cond_r(i,:) = [];
313         SC_cond(i,:) = [];
314         SH_comp(i,:) = [];
315         SC(i,:) = [];
316         mr(i,:) = [];
317         hin(i,:) = [];
318         xin(i,:) = [];
319         p = p - 1;
320         m = m - 1;
321     end
```

```
322     p = p + 1;
323 end
324 [m n] = size(data2);
325 
326 % use parameters from the paper as initial guess
327 D_ori = 0.00109;
328 L_ori = 0.0127;
329 Depth_ori = 1;
330 A_ineq = [];
331 B_ineq = [];
332 weigh_mdot = weighing_function_v000(xin, bin_num);
333 weigh_mdot(:,2) = weighing_function_v000(mr, bin_num);
334 options = optimset(...
335     'display','off','algorithm','sqp','TolX',1.e-8,'TolCon',1.e-8,...
336     'MaxFunEvals',1000 ...
337 );
338 [Geometry,fval,exitflag,output] = fmincon(@(Geometry) cost_func2(...
339     Geometry, data2, m, xin, refri, Pc, Tc, weigh_mdot ...
340 ), [0.1 0.1 1 1 1 1],A_ineq,B_ineq,[],[],[...
341     0 0 -Inf -Inf -Inf ...
342 ],[Inf Inf Inf Inf Inf],[],options);
343 if fval > 2.5e-4
344     options = optimset(...
345         'display','off','algorithm','interior-point','TolX',1.e-8,...
346         'TolCon',1.e-8,'MaxFunEvals',1000 ...
347     );
```

```
348     [Geometry2,fval2,exitflag2,output2] = fmincon(...
349         @(Geometry) cost_func2(...
350             Geometry, data2, m, xin, refri, Pc, Tc, weigh_mdot ...
351         ), [Geometry(1) Geometry(2) 1 1 1 1], A_ineq, B_ineq,[],[],[...
352             0 0 -Inf -Inf -Inf ...
353         ],[Inf Inf Inf Inf Inf],@(Geometry) FEO_constraints(...
354             Geometry, data2, m, xin, refri, Pc, Tc ...
355         ),options ...
356     );
357     if fval2 < fval
358         Geometry = Geometry2;
359         fval = fval2;
360     end
361     if fval2 > 2.5e-4
362         options = optimset(...
363             'display','off','algorithm','active-set','TolX',1.e-8,...
364             'TolCon',1.e-8,'MaxFunEvals',1000 ...
365         );
366         [Geometry3,fval3,exitflag3,output3] = fmincon(...
367             @(Geometry) cost_func2(...
368                 Geometry, data2, m, xin, refri, Pc, Tc, weigh_mdot ...
369             ), [...
370                 Geometry(1) Geometry(2) 1 1 1 1 ...
371             ], A_ineq, B_ineq,[],[],[0 0 -Inf -Inf -Inf],[...
372                 Inf Inf Inf Inf Inf ...
373             ],@(Geometry) FEO_constraints(...
```

```
374                 Geometry, data2, m, xin, refri, Pc, Tc ...
375             ),options ...
376         );
377         if fval3 < fval
378             Geometry = Geometry3;
379             fval = fval3;
380         end
381     end
382 end
383 
384 mdot = FEO_funcPayne2004(Geometry, data2, m, xin, refri, Pc, Tc);
385 FEO.para.D = Geometry(1)*D_ori;
386 FEO.para.L = Geometry(2)*L_ori;
387 FEO.para.Geometry_index = Geometry(3:6).*[-5.8195E+02 -1.4194E+01 1 1];
388 
389 % calculate the covriance matrix
390 Geometry_norm = [D_ori L_ori -5.8195E+02 -1.4194E+01 1 1];
391 for i = 1:length(Geometry)
392     Geometry_sp = Geometry;
393     if abs(Geometry(i)) > 1
394         Geometry_sp(i) = Geometry(i)*(1+1.e-5);
395     else
396         Geometry_sp(i) = Geometry(i)+1.e-5;
397     end
398     X_Cov(:,i) = (FEO_funcPayne2004(...
399         Geometry_sp, data2, m, xin, refri, Pc, Tc ...
```

```
400     )-FEO_funcPayne2004(...
401         Geometry, data2, m, xin, refri, Pc, Tc ...
402     ))./(Geometry_sp(i)-Geometry(i));
403 end
404 FEO.para.Cov = covariance_adj(X_Cov);
405 for i = 1:size(X_Cov,1)
406     limit(i,1) = X_Cov(i,:)*FEO.para.Cov*X_Cov(i,:)';
407 end
408 FEO.para.X_limit = max(limit);
409 FEO.para.FEO_Para = [D_ori L_ori -5.8195E+02 -1.4194E+01 1 1];
410 if strcmp(refri, 'R410A')
411     FEO.para.x_max = 0.087;
412 elseif strcmp(refri, 'R407C')
413     FEO.para.x_max = 0.047;
414 elseif strcmp(refri, 'R134a.fld')
415     FEO.para.x_max = 0.104;
416 elseif strcmp(refri, 'R502')
417     FEO.para.x_max = 0.094;
418 elseif strcmp(refri, 'R22.fld')
419     FEO.para.x_max = 0.102;
420 else
421     FEO.para.x_max = max(0.0, max(xin));
422 end
423
424 SSE = 0; % residual sum of squares declaration
425 SST = 0; % totol sum of squares declaration
```

```
426 mdot_av = mean(data2(:,23));
427 i = 1;
428 k = 1;
429 for j = 1:m
430     if(xin(j,1)>0)
431         mdota_2p(i,1) = data2(j,23);
432         mdotb_2p(i,1) = mdot(j,1);
433         SC_2p(i,1) = SC(j,1);
434         xin_2p(i,1) = xin(j,1);
435         i = i+1;
436     else
437         mdotb_1p(k,1) = mdot(j,1);
438         mdota_1p(k,1) = data2(j,23);
439         SC_1p(k,1) = SC(j,1);
440         xin_1p(k,1) = xin(j,1);
441         k = k+1;
442     end
443     SSE = SSE + (mdot(j,1) - data2(j,23))^2;
444     SST = SST + (data2(j,23)- mdot_av)^2;
445 end
446
447 r2_m = 1 - SSE/SST;
448
449 save(savefilename);
450 movefile(savefilename,strcat(foldername,'\'));
451
```

```
452 end
453
454 function r = cost_func2(...
455     Geometry, data, m, xin, refri, Pc, Tc, weigh_mdot ...
456 )
457     r = 0;
458     mr = FEO_funcPayne2004(Geometry, data, m, xin, refri, Pc, Tc);
459     for i = 1:m
460         r = r + (1/weigh_mdot(i,1)+1/weigh_mdot(i,2))*(...
461             (mr(i,1) - data(i,23))/data(i,23) ...
462         )^2;
463     end
464     r = r/m;
465 end
466
467 function [c, ceq] = FEO_constraints(...
468     Geometry, data, m, xin, refri, Pc, Tc ...
469 )
470 xin = ones(length(xin),1)*min(xin);
471 for i = 1:m
472     T = propertyRH('T', 'P', data(i,5), 'Q', 0, refri);
473     hg = propertyRH('H', 'P', data(i,5), 'Q', 1, refri);
474     hf = propertyRH('H', 'P', data(i,5), 'Q', 0, refri);
475     h = propertyRH('H', 'T', T-20, 'P', data(i,5), refri);
476     xin(i,1) = (h-hf)/(hg-hf);
477     data(i,13) = T-20;
```

```
478 end
479 [mr, c] = FEO_funcPayne2004(Geometry, data, m, xin, refri, Pc, Tc);
480 c = -c;
481 ceq = [];
482 end
483
484 function [mr, dmdSC] = FEO_funcPayne2004(...
485     Geometry, data, m, xin, refri, Pc, Tc ...
486 )
487 % Coefficients as defined in Payne (2004)
488 para.a(1,1) = 3.8811E-01;% [-]
489 para.a(2,1) = 1.1427E+01;% [-]
490 para.a(3,1) = -1.4194E+01;% [-]
491 para.a(4,1) = 1.0703E+00;% [-]
492 para.a(5,1) = -9.1928E-02;% [-]
493 para.a(6,1) = 2.1425E+01;% [-]
494 para.a(7,1) = -5.8195E+02;% [-]
495
496 para.b(1,1) = 1.1831E+00;% [-]
497 para.b(2,1) = -1.4680E+00;% [-]
498 para.b(3,1) = -1.5285E-01;% [-]
499 para.b(4,1) = -1.4639E+01;% [-]
500 para.b(5,1) = 9.8401E+00;% [-]
501 para.b(6,1) = -1.9798E-02;% [-]
502 para.b(7,1) = -1.5348E+00;% [-]
503 para.b(8,1) = -2.0533E+00;% [-]
```

```
504 para.b(9,1) = -1.7195E+01;% [-]
505
506 Geometry(1) = Geometry(1)*0.00109;
507 Geometry(2) = Geometry(2)*0.0127;
508 Geometry(3) = Geometry(3)*para.a(7,1);
509 Geometry(4) = Geometry(4)*para.a(3,1);
510 Geometry(5) = 0;
511 Geometry(6) = 0;
512 % Geometry(5) = Geometry(5)*1;
513 % Geometry(6) = Geometry(6)*1;
514
515 for i = 1:m
516     Tin = data(i,13);
517     Pin = data(i,5);
518     Pout = data(i,6);
519     x_in = xin(i,1);
520
521     % defining variable
522     SC = propertyRH('T','P',Pin,'Q',0,refri)-Tin;
523     Tsat = propertyRH('T','P',Pin,'Q',0,refri);
524     hf = propertyRH('H','P',Pin,'Q',0,refri);
525     hv = propertyRH('H','P',Pin,'Q',1,refri);
526     if SC > 1
527         hin = propertyRH('H','T',Tin,'P',Pin,refri);
528     else
529         hin = x_in*(hv-hf)+hf;
```

```
530         Tin = propertyRH('T','P',Pin,'H',hin,refri);
531         if Tin > Tsat
532             Tin = Tsat;
533         end
534     end
535
536     Psat = propertyRH('P','T',Tin,'Q',0,refri);
537     rho_g = propertyRH('D','P',Pin,'Q',1,refri);
538     rho_f = propertyRH('D','P',Pin,'Q',0,refri);
539     T_sub = Tsat - Tin;
540
541     pi_gp(1) = (Pin - Psat)/Pc;
542     if (Psat≥Pin)
543         pi_gp(1) = 0;
544     end
545     pi_gp(2) = rho_g/rho_f;
546     pi_gp(3) = T_sub/Tc;
547     if pi_gp(3)<0
548         pi_gp(3) = 0;
549     end
550     pi_gp(4) = Pout/Pc;
551     pi_gp(5) = Pin/Pc;
552
553     Ct = pi/4*sqrt(rho_f*Pc*1000)/(...
554         1+para.a(6)*pi_gp(1)+Geometry(3)*(pi_gp(3))^2 ...
555     );
```

```
556     Vc = (...
557         para.a(1)+para.a(2)*pi_gp(1)+Geometry(4)*pi_gp(3) +...
558         para.a(4)*pi_gp(2)+Geometry(5)*pi_gp(4) +...
559         Geometry(6)*pi_gp(5) ...
560     );
561     C1pCoeff(1) = Ct*para.a(5);
562     C1pCoeff(2) = Ct*Vc;
563
564     Din_cal = Geometry(1);
565
566     D1 = Din_cal^2*log(Geometry(2)/Geometry(1));
567     D2 = Din_cal^2;
568     mr(i,1) = (C1pCoeff(1)*D1 + C1pCoeff(2)*D2);
569
570     dCtdSC = -Ct*2*Geometry(3)*2*pi_gp(3)/(...
571         1+para.a(6)*pi_gp(1)+Geometry(3)*(pi_gp(3))^2 ...
572     )/Tc;
573     dVcdSC = Geometry(4)/Tc;
574     dmdSC(i,1) = (para.a(5)*D1+Vc*D2)*dCtdSC+Ct*D2*dVcdSC;
575
576     if T_sub < 3
577 %         dmdSC(i,1) = 0.0;
578         pi_gp(1) = (Pin-propertyRH(...
579             'P','T',Tsat-3,'Q',0,refri ...
580         ))/Pc;
581         pi_gp(3) = 3/Tc;
```

```
582         C1pCoeff(1) = pi/4*sqrt(rho_f*Pc*1000)/(...
583             1+para.a(6)*pi_gp(1)+Geometry(3)*(pi_gp(3))^2 ...
584         )*para.a(5);
585         C1pCoeff(2) = pi/4*sqrt(rho_f*Pc*1000)/(...
586             1+para.a(6)*pi_gp(1)+Geometry(3)*(pi_gp(3))^2 ...
587         )*(...
588             para.a(1)+para.a(2)*pi_gp(1)+Geometry(4)*pi_gp(3) +...
589             para.a(4)*pi_gp(2)+Geometry(5)*pi_gp(4) +...
590             Geometry(6)*pi_gp(5) ...
591         );
592         mdot_r_3K = (C1pCoeff(1)*Geometry(1)^2*log(...
593             Geometry(2)/Geometry(1) ...
594         ) + C1pCoeff(2)*Geometry(1)^2);
595         if mdot_r_3K < mr(i,1)
596             mr(i,1) = mdot_r_3K;
597         end
598     end
599     % check state at inlet
600     if(hin>hf) % for two-phase inlet
601         A = pi*Geometry(1)^2/4;
602         Ctp = Payne2004Ctp(...
603             x_in, Pin, Pc, Psat, A, para, Geometry, refri ...
604         )/Payne2004Ctp(0, Pin, Pc, Psat, A, para, Geometry, refri);
605         if Ctp > 1
606             Ctp = 1;
607         end
```

```
608         mr(i,1) = mr(i,1)*Ctp;
609     end
610 end
611 mr = real(mr);
612 c = [];
613 end
614
615 function Ctp = Payne2004Ctp(...
616     xin, Pin, Pc, Psat, A, para, Geometry, refri ...
617 )
618     rho_g = propertyRH('D','P',Pin,'Q',1,refri);
619     rho_f = propertyRH('D','P',Pin,'Q',0,refri);
620     rho_mup = ((1-xin)/rho_f+xin/rho_g)^(-1);
621     tp6 = rho_mup/rho_f;
622     tp28 = Pin/Pc;
623     tp32 = 1 - Pin/Pc;
624     tp34 = xin/(1-xin)*(rho_f/rho_g)^(.5);
625     tp35 = 1 - Psat/Pc;
626     CtpCoeff(1) = (...
627         para.b(1)*tp6+para.b(2)*tp6^2+para.b(3)*(log(tp6))^2 +...
628         para.b(4)*(log(tp35))^2+para.b(5)*(log(tp32))^2 ...
629     )/(1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3);
630     CtpCoeff(2) = para.b(6)/(...
631         1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3 ...
632     );
633     Ctp = (CtpCoeff(1) + CtpCoeff(2)*(log(...
```

```
634         Geometry(2)/sqrt(A*4/pi) ...
635     ))^2);
636 end
```

N.7 TXV Modeling

```
1  function System = TXV_minimization_v081_main()
2
3  % Function to pass system information for TXV modeling
4
5  % define function and folder name
6  function_name = 'TXV_minimization';
7  version_main = '081';
8  version_regress = '078';
9  version_plot = '002';
10 foldername = strcat(function_name,'_v',version_main);
11 regressname = strcat(function_name,'_v',version_regress);
12 plotname = strcat(function_name,'_plot_v',version_plot);
13 regress_func = str2func(['@(System, libname, version_number)',...
14     regressname,'(System, libname, version_number)']);
15 plot_func = str2func(['@(System, libname, version_number)',...
16     plotname,'(System, libname, version_number)']);
17 mkdir(foldername);
18 savefilename = strcat(foldername,'.mat');
19
20 % set deffinition of TXV class
21 TXV.para.C = zeros(6,1);
```

```
22 TXV.para.T_evap_cric = 1000;
23 TXV.para.SH_upper = TXV.para.T_evap_cric;
24 SH_upper = TXV.para.SH_upper;
25 TXV.para.SH_lower = 0;
26 TXV.para.ΔT = 0;
27 TXV.para.n = 1;
28 TXV.para.mode = 2;
29 TXV.para.ValvePara_A_index = zeros(3,1);
30 TXV.para.L = 0;
31 TXV.para.ValvePara_C_index = zeros(4,1);
32 TXV.para.SH_upper = 0;
33
34 % Coefficients as defined in Payne (2004)
35 TXV.para.a(1,1) = 3.8811E-01;% [-]
36 TXV.para.a(2,1) = 1.1427E+01;% [-]
37 TXV.para.a(3,1) = -1.4194E+01;% [-]
38 TXV.para.a(4,1) = 1.0703E+00;% [-]
39 TXV.para.a(5,1) = -9.1928E-02;% [-]
40 TXV.para.a(6,1) = 2.1425E+01;% [-]
41 TXV.para.a(7,1) = -5.8195E+02;% [-]
42
43 TXV.para.b(1,1) = 1.1831E+00;% [-]
44 TXV.para.b(2,1) = -1.4680E+00;% [-]
45 TXV.para.b(3,1) = -1.5285E-01;% [-]
46 TXV.para.b(4,1) = -1.4639E+01;% [-]
47 TXV.para.b(5,1) = 9.8401E+00;% [-]
```

```
48 TXV.para.b(6,1) = -1.9798E-02;% [-]
49 TXV.para.b(7,1) = -1.5348E+00;% [-]
50 TXV.para.b(8,1) = -2.0533E+00;% [-]
51 TXV.para.b(9,1) = -1.7195E+01;% [-]
52
53 % Coefficients as defined in Payne (1999)
54 TXV.para.a2(1,1) = 3.693038038;
55 TXV.para.a2(2,1) = 0.120175996;
56 TXV.para.a2(3,1) = 0.194241638;
57 TXV.para.a2(4,1) = 0.022577667;
58
59 % Set information
60 % re-file at this stage to avoid problems in parfor loop
61 i = 1;
62 % Boshen 3-ton R410A split TXV system (Recip)
63 % (Shen_030_R410A_split_TXV_Recip)
64 System(i).name = 'Shen_030_R410A_split_TXV_Recip';
65 System(i).othername = 'Shen_030_R410A_split_FXO_TXV_Recip';
66 System(i).cond_name = System(i).othername;
67 System(i).LLname = System(i).othername;
68 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitTXV.xls';
69 System(i).foldername = 'Boshen_3tonR410ASplitTXV';
70 System(i).worksheetname = 'SI (no_invalid_CA)';
71 System(i).refname = 'R410A';
72 System(i).Standard_Airflow_evap = 0;
73 System(i).Fan_Upstream_evap = 0;
```

```
74 System(i).Standard_Airflow_cond = 1;
75 System(i).Fan_Upstream_cond = 1;
76 System(i).Heating = 0;
77 System(i).Combined = 0;
78 % System(i).Equalizer = 'Ext_Comp';
79 System(i).Equalizer = 'Ext_Evap';
80 System(i).Diameter = 0.00849;
81 System(i).Tube_Length = 0.452;
82 System(i).Ntube = 28*3;
83 System(i).Pc = 4901;
84 System(i).Tc = 344.49;
85
86 % Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
87 % (NIST_025_R410A_split_TXV_Scroll)
88 i = i + 1;
89 System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
90 System(i).othername = System(i).name;
91 System(i).cond_name = System(i).othername;
92 System(i).LLname = System(i).othername;
93 System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
94 System(i).foldername = 'NIST';
95 System(i).worksheetname = 'SI (no_invalid_CA)';
96 System(i).refname = 'R410A';
97 System(i).Standard_Airflow_evap = 1;
98 System(i).Fan_Upstream_evap = 1;
99 System(i).Standard_Airflow_cond = 1;
```

```
100 System(i).Fan_Upstream_cond = 1;
101 System(i).Heating = 0;
102 System(i).Combined = 0;
103 System(i).Equalizer = 'Ext_Evap';
104 System(i).Diameter = 0.007747;
105 System(i).Tube_Length = 0.508;
106 System(i).Ntube = 60;
107 System(i).Pc = 4901;
108 System(i).Tc = 344.49;
109
110 % Harms 5-ton R22 packaged TXV system (Scroll)
111 % (Harms_050_R22_packaged_TXV_Scroll)
112 i = i + 1;
113 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
114 System(i).othername = System(i).name;
115 System(i).cond_name = System(i).othername;
116 System(i).LLname = System(i).othername;
117 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
118 System(i).foldername = 'Harms_050tonR22PackagedTXV';
119 System(i).worksheetname = 'SI (no_invalid_CA)';
120 System(i).refname = 'R22.fld';
121 System(i).Standard_Airflow_evap = 0;
122 System(i).Fan_Upstream_evap = 0;
123 System(i).Standard_Airflow_cond = 1;
124 System(i).Fan_Upstream_cond = 1;
125 System(i).Heating = 0;
```

```
126 System(i).Combined = 0;
127 System(i).Equalizer = 'Ext_Evap';
128 System(i).Diameter = 0.00889;
129 System(i).Tube_Length = 0.617;
130 System(i).Ntube = 32*4;
131 System(i).Pc = 4990;
132 System(i).Tc = 369.3;
133
134 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
135 % (HCA3_030_R410A_split_TXV_Scroll)
136 i = i + 1;
137 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
138 System(i).othername = System(i).name;
139 System(i).cond_name = System(i).othername;
140 System(i).LLname = System(i).othername;
141 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
142 System(i).foldername = 'Kim-HCA3';
143 System(i).worksheetname = 'SI (no_invalid_CA)';
144 System(i).refname = 'R410A';
145 System(i).Standard_Airflow_evap = 0;
146 System(i).Fan_Upstream_evap = 0;
147 System(i).Standard_Airflow_cond = 1;
148 System(i).Fan_Upstream_cond = 1;
149 System(i).Heating = 0;
150 System(i).Combined = 0;
151 System(i).Equalizer = 'Ext_Evap';
```

```
152 % assumed
153 System(i).Diameter = 0.00849;
154 System(i).Tube_Length = 2.255;
155 System(i).Ntube = 32;
156 System(i).Pc = 4901;
157 System(i).Tc = 344.49;
158
159 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
160 % (YSA_030_R22_split_TXV_Scroll)
161 % Rejected because of the zero superheat at evaporator outlet
162 i = i + 1;
163 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
164 System(i).othername = System(i).name;
165 System(i).cond_name = System(i).othername;
166 System(i).LLname = System(i).othername;
167 System(i).filename = 'Kim_Cycle_data_R22_YSA_v02.xlsx';
168 System(i).foldername = 'Kim-YSA';
169 System(i).worksheetname = 'SI (no_invalid_CA)';
170 System(i).refname = 'R22.fld';
171 System(i).Standard_Airflow_evap = 0;
172 System(i).Fan_Upstream_evap = 0;
173 System(i).Standard_Airflow_cond = 1;
174 System(i).Fan_Upstream_cond = 1;
175 System(i).Heating = 0;
176 System(i).Combined = 0;
177 System(i).Equalizer = 'Ext_Evap';
```

```
178 % assumed
179 System(i).Diameter = 0.00849;
180 System(i).Tube_Length = 2.255;
181 System(i).Ntube = 32;
182 System(i).Pc = 4990;
183 System(i).Tc = 369.3;
184
185 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
186 % (TM_040_R410A_packaged_TXV_Recip)
187 i = i + 1;
188 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
189 System(i).othername = System(i).name;
190 System(i).cond_name = System(i).name;
191 System(i).LLname = System(i).name;
192 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
193 System(i).foldername = 'Kim-TM';
194 System(i).worksheetname = 'SI (no_LL)';
195 System(i).refname = 'R410A';
196 System(i).Standard_Airflow_evap = 0;
197 System(i).Fan_Upstream_evap = 0;
198 System(i).Standard_Airflow_cond = 1;
199 System(i).Fan_Upstream_cond = 1;
200 System(i).Heating = 0;
201 System(i).Combined = 0;
202 System(i).Equalizer = 'Ext_Comp';
203 % assumed
```

```
204 System(i).Diameter = 0.00849;
205 System(i).Tube_Length = 2.255;
206 System(i).Ntube = 32;
207 System(i).Pc = 4990;
208 System(i).Tc = 369.3;
209
210 % other settings for storage
211 m = length(System);
212 for i = 1:m
213     System(i).mainfoldername = foldername;
214     System(i).TXV = TXV;
215     System(i).r2_m = 0;
216 end
217
218 try
219     pctRunOnAll loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
220 catch err
221     try
222         loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
223     end
224 end
225 libname ='lib'; % Name of library
226
227 % start calculation
228 no_plot = zeros(m,1);
229 % make an error statment
```

```
try
    parfor i = 1:m
        System(i) = 1; % a statement leads to error
    end
catch err
end
for i = 1:m
    error_statement(i) = err;
end
beg_index = 1;
end_index = m;
% parfor i = beg_index:end_index
for i = 1:end_index
    try
        disp(['Starting to run ',System(i).name]);
        if strcmp(System(i).name, 'TM_040_R410A_packaged_TXV_Recip')
            [TXV_class r2_m] = TXV_minimization_v074(...
                System(i), libname, version_main ...
            );
            % for the sake of copying, no parameter estimation needed
        else
            [TXV_class r2_m] = regress_func(...
                System(i), libname, version_main ...
            );
        end
        System(i).TXV = TXV_class;
```

```
256         System(i).r2_m = r2_m;
257         disp(['Finishing simulating ',System(i).name]);
258     catch err
259         disp(['Problem found in system ',System(i).name]);
260         no_plot(i,1) = 1;
261         error_statement(i) = err;
262     end
263 end
264 save(savefilename);
265 
266 % plot cannot be done in parfor. Do it in a for loop
267 for i = beg_index:end_index
268 % for i = 1:m
269     if no_plot(i,1) == 0
270         disp(['Starting to plot ',System(i).name]);
271         plot_func(System(i), libname, version_main);
272         disp(['Finishing plotting ',System(i).name]);
273     end
274 end
275 
276 save(savefilename);
277 movefile(savefilename,strcat(foldername,'\'));
278 
279 try
280     matlabpool close; % if catch an error, close and reset it
281 end
```

```
282 try
283     unloadlibrary lib;
284 end
```

```
1  function [TXV r2_m] = TXV_minimization_v078(...
2      System, libname, version_number ...
3  )
4
5  % Function to train TXV model
6
7  %% Preparation
8  % data reading and filtering
9  bin_num = 10;
10 filename = System.filename;
11 worksheetname = System.worksheetname;
12 refri = System.refname;
13 Standard_Airflow_cond = System.Standard_Airflow_cond;
14 Standard_Airflow_evap = System.Standard_Airflow_evap;
15 Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
16 Ind_Fan_Upstream_evap = System.Fan_Upstream_evap;
17 D = System.Diameter;
18 Ltube = System.Tube_Length;
19 Ntube = System.Ntube;
20 Pc = System.Pc;
21 Tc = System.Tc;
```

```
Equalizer = System.Equalizer;

version = strcat(System.name,'_',version_number);
savefilename = strcat('TXV_minimization_v',version,'.mat');
foldername = strcat(strcat(...
    pwd,'\',System.mainfoldername,'\' ...
),strcat(...
    'TXV_minimization_v',version ...
));
mkdir(foldername);

Q_gain_version = '030';
Q_gain_name = [...
    'Q_gain_observation_v',System.name,'_',Q_gain_version ...
];
Q_gain_raw = open([...
    'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,'\',...
    Q_gain_name,'.mat' ...
]);
k = length(Q_gain_raw.index_removed);
data2 = Q_gain_raw.data;
code = Q_gain_raw.code;
fault = Q_gain_raw.fault;
Q_cond_r = Q_gain_raw.Q_cond_r;
Q_evap_r = Q_gain_raw.Q_evap_r;
for i = k:-1:1 % reverse for correct indexing
```

```
    data2(Q_gain_raw.index_removed(i),:) = [];
    code(Q_gain_raw.index_removed(i),:) = [];
    fault(Q_gain_raw.index_removed(i),:) = [];
    Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
    Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
end
[pp qq] = size(data2);
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
i = 1;
while i ≤ pp
    if ¬isempty(cell2mat(cell_VL(i))) || ...
        ¬isempty(cell2mat(cell_NC(i))) || ...
        ¬isempty(cell2mat(cell_LL(i)))
        data2(i,:) = [];
        fault(i,:) = [];
```

```
74          code(i,:) = [];
75          cell_VL(i,:) = [];
76          cell_LL(i,:) = [];
77          cell_NC(i,:) = [];
78          Q_evap_r(i,:) = [];
79          Q_cond_r(i,:) = [];
80          i = i - 1;
81          pp = pp - 1;
82      end
83      i = i + 1;
84  end
85  [m n] = size(data2);
86
87  % for Compressor
88  version_Comp = '075';
89  version_func = '045';
90  name_Comp = 'Compressor_regression_v';
91  component_name = strcat(...
92      name_Comp,System.othername,'_',version_Comp ...
93  );
94  foldername_comp = strcat(strcat(...
95      pwd,'\',name_Comp,version_Comp,'\',component_name ...
96  ));
97  raw = open(strcat(foldername_comp,'\',strcat(...
98      component_name,'.mat' ...
99  )));
```

```
100 Comp_para = raw.Comp;
101 Comp_func = str2func([...
102     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
103     'Compressor_v',version_func,...
104     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
105 ]);
106
107 % for Liquid Line
108 version_LiquidLine = '040';
109 version_func = '003';
110 name_LiquidLine = 'LiquidLine_minimization_v';
111 component_name = strcat(...
112     name_LiquidLine,System.LLname,'_',...
113     version_LiquidLine ...
114 );
115 foldername_LiquidLine = strcat(strcat(...
116     pwd,'\',name_LiquidLine,version_LiquidLine,'\',...
117     component_name ...
118 ));
119 raw = open(strcat(foldername_LiquidLine,'\',strcat(...
120     component_name,'.mat' ...
121 )));
122 LiquidLine_para = raw.PipeLineClass;
123 LiquidLine_func = str2func([...
124     '@(mr, Pin, hin, Tamb, para, refri, phase, libname)',...
125     'pipeline_v',version_func,...
```

```
126     '(mr, Pin, hin, Tamb, para, refri, phase, libname)' ...
127 ]);
128 
129 % set deffinition of TXV class
130 TXV.para.C = zeros(6,1);
131 TXV.para.T_evap_cric = propertyRH('P','T',291,'Q',1,refri) - ...
132     propertyRH('P','T',283,'Q',1,refri); % assumption
133 TXV.para.SH_upper = TXV.para.T_evap_cric;
134 TXV.para.SH_lower = 0;
135 TXV.para.ΔT = 0;
136 TXV.para.n = 1;
137 TXV.para.mode = 2;
138 
139 % Coefficients as defined in Payne (2004)
140 TXV.para.a(1,1) = 3.8811E-01;% [-]
141 TXV.para.a(2,1) = 1.1427E+01;% [-]
142 TXV.para.a(3,1) = -1.4194E+01;% [-]
143 TXV.para.a(4,1) = 1.0703E+00;% [-]
144 TXV.para.a(5,1) = -9.1928E-02;% [-]
145 TXV.para.a(6,1) = 2.1425E+01;% [-]
146 TXV.para.a(7,1) = -5.8195E+02;% [-]
147 
148 TXV.para.b(1,1) = 1.1831E+00;% [-]
149 TXV.para.b(2,1) = -1.4680E+00;% [-]
150 TXV.para.b(3,1) = -1.5285E-01;% [-]
151 TXV.para.b(4,1) = -1.4639E+01;% [-]
```

```
152 TXV.para.b(5,1) = 9.8401E+00;% [-]
153 TXV.para.b(6,1) = -1.9798E-02;% [-]
154 TXV.para.b(7,1) = -1.5348E+00;% [-]
155 TXV.para.b(8,1) = -2.0533E+00;% [-]
156 TXV.para.b(9,1) = -1.7195E+01;% [-]
157
158 % Coefficients as defined in Payne (1999)
159 TXV.para.a2(1,1) = 3.693038038;
160 TXV.para.a2(2,1) = 0.120175996;
161 TXV.para.a2(3,1) = 0.194241638;
162 TXV.para.a2(4,1) = 0.022577667;
163
164 %% Calculation
165 % data filtering
166 i = 1;
167 while(i≤m)
168     if data2(i,5) > data2(i,4)
169         data2(i,5) = data2(i,4);
170     end
171     hout_cond = propertyRH(...
172         'H','T',data2(i,12),'P',data2(i,4),refri ...
173     );
174     hin_EXV = propertyRH(...
175         'H','T',data2(i,13),'P',data2(i,5),refri ...
176     );
177     % if heat cannot be lost in the direction of temperature,
```

```
178     % change the temperature reading
179     if (hout_cond - hin_EXV)/(data2(i,12)-data2(i,21)) < 0
180         data2(i,13) = propertyRH(...
181             'T','P',data2(i,5),'H',hout_cond,refri ...
182         );
183     end
184     Tsat = propertyRH('T','P',data2(i,5),'Q',0,refri);
185     Tsat_II = propertyRH('T','P',data2(i,4),'Q',0,refri);
186     SH_comp = data2(i,9)-propertyRH(...
187         'T','P',data2(i,1),'Q',1,refri ...
188     );
189     SH_evap = data2(i,16)-propertyRH(...
190         'T','P',data2(i,8),'Q',1,refri ...
191     );
192     if strcmp(Equalizer,'Internal')
193         Tsat_III = propertyRH('T','P',data2(i,6),'Q',1,refri);
194         SH_check = data2(i,16)-Tsat_III;
195     elseif strcmp(Equalizer,'Ext_Evap')
196         Tsat_III = propertyRH('T','P',data2(i,8),'Q',1,refri);
197         SH_check = data2(i,16)-Tsat_III;
198     elseif strcmp(Equalizer,'Ext_Comp')
199         Tsat_III = propertyRH('T','P',data2(i,1),'Q',1,refri);
200         SH_check = data2(i,9)-Tsat_III;
201     elseif strcmp(Equalizer,'Ext_T_Comp')
202         Tsat_III = propertyRH('T','P',data2(i,1),'Q',1,refri);
203         SH_check = data2(i,9)-Tsat_III;
```

```
204     end
205     if System.Heating==0
206         Tamb = data2(i,21);
207     else
208         data2(i,19) = 230;
209         data2(i,20) = 230;
210         Tamb = data2(i,17);
211     end
212     if ((Tsat_II - data2(i,12)>3 ) || ...
213             (Tsat_II - data2(i,12)≥1) || ...
214             (Tsat - data2(i,13)>1 && SH_evap > 1)) && ...
215             data2(i,23) > 0
216     else
217         if System.Heating==0
218             Tamb = data2(i,21);
219         else
220             Tamb = data2(i,17);
221         end
222         data2(i,23) = Comp_func(data2(i,1), propertyRH(...
223             'H','T',data2(i,9),'P',data2(i,1),refri ...
224         ), data2(i,2), Tamb, data2(i,10), Comp_para, refri);
225     end
226     Tsat_evap = propertyRH('T','P',data2(i,8),'Q',1,refri);
227     removal_confirm = 0;
228     if Tsat - data2(i,13)≥1
229         hin_EXV = propertyRH(...
```

```
230             'H','T',data2(i,13),'P',data2(i,5),refri ...
231         );
232     elseif data2(i,16)-Tsat_evap > 1
233         hin_EXV = propertyRH(...
234             'H','T',data2(i,16),'P',data2(i,8),refri ...
235         ) - Q_evap_r(i,1)/data2(i,23);
236         data2(i,13) = propertyRH(...
237             'T','P',data2(i,5),'H',hin_EXV,refri ...
238         );
239     else
240         hin_EXV = propertyRH(...
241             'H','T',data2(i,11),'P',data2(i,3),refri ...
242         ) - Q_cond_r(i,1)/data2(i,23);
243         [dum9 hin_EXV dum10] = LiquidLine_func(...
244             data2(i,23), data2(i,4), hin_EXV, Tamb, ...
245             LiquidLine_para, refri, 0, libname ...
246         );
247         data2(i,13) = propertyRH(...
248             'T','P',data2(i,5),'H',hin_EXV,refri ...
249         );
250     end
251     hl = propertyRH('H','P',data2(i,5),'Q',0,refri);
252     hv = propertyRH('H','P',data2(i,5),'Q',1,refri);
253     hin_cal(i,1) = hin_EXV;
254     xin(i,1) = (hin_cal(i,1) - hl)/(hv - hl);
255     if ((((Tsat_II - data2(i,12)>3) || ...
```

```
256             (Tsat_II - data2(i,12)≥1) || ...
257             (Tsat - data2(i,13)>1 && SH_evap > 1))&&...
258             (data2(i,23)>0)) || ...
259             SH_comp > 1) && removal_confirm == 0
260         xin_confirm = xin(i,1);
261         if strcmp(refri,'R22.fld') & xin_confirm > 0.102
262             removal_confirm = 1;
263         elseif strcmp(refri,'R410A') & xin_confirm > 0.087
264             removal_confirm = 1;
265         elseif strcmp(refri,'R407C') & xin_confirm > 0.047
266             removal_confirm = 1;
267         else
268             SH_ori(i,1) = SH_check;
269             if strcmp(Equalizer,'Internal')
270                 SH_UP_T(i,1) = data2(i,16);
271                 SH_UP_P(i,1) = propertyRH(...
272                     'P','T',data2(i,16),'Q',1,refri ...
273                 );
274                 SH_DOWN_P(i,1) = data2(i,6);
275             elseif strcmp(Equalizer,'Ext_Evap')
276                 SH_UP_T(i,1) = data2(i,16);
277                 SH_UP_P(i,1) = propertyRH(...
278                     'P','T',data2(i,16),'Q',1,refri ...
279                 );
280                 SH_DOWN_P(i,1) = data2(i,8);
281             elseif strcmp(Equalizer,'Ext_Comp')
```

```
282                 SH_UP_T(i,1) = data2(i,9);
283                 SH_UP_P(i,1) = propertyRH(...
284                     'P','T',data2(i,9),'Q',1,refri ...
285                 );
286                 SH_DOWN_P(i,1) = data2(i,1);
287             elseif strcmp(Equalizer,'Ext_T_Comp')
288                 SH_UP_T(i,1) = data2(i,9);
289                 SH_UP_P(i,1) = data2(i,9);  % use temperature instead
290                 SH_DOWN_P(i,1) = propertyRH(...
291                     'T','P',data2(i,1),'Q',1,refri ...
292                 );
293             end
294             removal_confirm = 0;
295         end
296     else
297         removal_confirm = 1;
298     end
299     if removal_confirm == 1
300         data2(i,:) = [];
301         code(i,:) = [];
302         fault(i,:) = [];
303         hin_cal(i,:) = [];
304         xin(i,:) = [];
305         Q_evap_r(i,:) = [];
306         Q_cond_r(i,:) = [];
307         i = i - 1;
```

```
308          m = m - 1;
309      end
310      i = i + 1;
311  end
312
313  % calculate variables needed from filtered data
314  j = 1;
315  p = 1;
316  [m n] = size(data2);
317  while(p≤m)
318      % check subcooling first
319      i = p;
320      if System.Heating==0
321          Tamb = data2(i,21);
322      else
323          data2(i,19) = 230;
324          data2(i,20) = 230;
325          Tamb = data2(i,17);
326      end
327      Tsat = propertyRH('T','P',data2(i,5),'Q',0,refri);
328      SH(i,1) = data2(i,16)-propertyRH(...
329          'T','P',data2(i,8),'Q',1,refri ...
330      );
331      % if heat cannot be lost in the direction of temperature,
332      % change the temperature reading
333      SC(i,1) = Tsat - data2(i,13);
```

```
334     SH_comp(i,1) = data2(i,9)-propertyRH(...
335         'T','P',data2(i,1),'Q',1,refri ...
336     );
337     mr(i,1) = data2(i,23);
338     p = p + 1;
339 end
340 
341 % initial guess preparation
342 D_ori = mean(mr(i,1))/0.065*0.00109;
343 L_ori = mean(mr(i,1))/0.065*0.0127*0.5;
344 ΔP = SH_UP_P - SH_DOWN_P;
345 NC_Para_ori = [pi*D_ori^2/4 1 1 L_ori ...
346     -5.8195E+02 -1.4194E+01 -9.1928E-02 2.1425E+01 1.0703E+00];
347 C_para_ori = ones(1,length(NC_Para_ori));
348 C_para_ori(2) = 0;
349 C_para_ori(3) = 0;
350 weigh = weighing_function_v000(xin, bin_num);
351 if strcmp(Equalizer,'Ext_Evap')
352     weigh(:,2) = weighing_function_v000(SH, bin_num);
353 else
354     weigh(:,2) = weighing_function_v000(SH_comp, bin_num);
355 end
356 weigh(:,3) = weighing_function_v000(data2(:,23), bin_num);
357 max_SH_upper = max(SH_UP_P - SH_DOWN_P);
358 min_SH_upper = min(SH_UP_P - SH_DOWN_P);
359 SH_upper_order = sort(SH_UP_P-SH_DOWN_P,1,'descend');
```

```
360 fff = Inf;
361 old_fff = Inf;
362
363 data_norm = data2;
364 SH_UP_P_norm = SH_UP_P;
365 SH_DOWN_P_norm = SH_DOWN_P;
366 SH_UP_T_norm = SH_UP_T;
367 xin_norm = xin;
368 SH_norm = SH;
369 m_norm = length(SH_UP_P_norm);
370
371 weigh_norm = weighing_function_v000(xin_norm, bin_num);
372 weigh_norm(:,2) = weighing_function_v000(SH_norm, bin_num);
373 weigh_norm(:,3) = weighing_function_v000(data_norm(:,23), bin_num);
374
375 index_supp = find(xin == max(xin));
376 index_guess(1) = index_supp(1);
377 index_supp = find(SH_UP_P - SH_DOWN_P == min(SH_UP_P - SH_DOWN_P));
378 index_guess(4) = index_supp(1);
379
380 % iterate different SH_upper in descending order
381 old_fff = zeros(length(SH_upper_order),1);
382 old_C_para = zeros(length(SH_upper_order), length(NC_Para_ori));
383 old_ValvePara = zeros(length(SH_upper_order), length(NC_Para_ori));
384 old_NC_Para = zeros(length(SH_upper_order), length(NC_Para_ori));
385 old_SH_upper = zeros(length(SH_upper_order),1);
```

```
386 parfor iii = 1:length(SH_upper_order)
387     if m ≥ 10
388         sc_drop = 1;
389     else
390         sc_drop = 0;
391     end
392     NC_Para = NC_Para_ori;
393     index = index_guess;
394     SH_upper = SH_upper_order(iii);
395     index_supp = find(SH_UP_P-SH_DOWN_P==SH_upper);
396     index(2) = index_supp(1);
397     index_supp = find(ΔP == max(ΔP(ΔP≤median(ΔP))));
398     if isempty(index_supp)
399         index_supp = find(ΔP == max(ΔP));
400     end
401     index(3) = index_supp(1);
402     [mdot DischargeCoeff] = TXV_funcPayne2004(...
403         C_para_ori, data2, m, xin, refri, Pc, Tc, SH_upper, ...
404         NC_Para_ori, TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop...
405     );
406     C_Para = [ΔP/200 (ΔP./200).^2]\DischargeCoeff;
407     C_Para = [0 C_Para(1) C_Para(2)];
408     NC_Para(1:3) = [abs(C_Para(2)*max(ΔP/200)) C_Para(2) C_Para(3)];
409     NC_Para = real(NC_Para);
410     options = optimset(...
411         'Display','off','MaxFunEvals',length(index)*1000, ...
```

```
412         'MaxIter', 800 ...
413     );
414     C_para_temp = fsolve(@(C_para2) data2(...
415         index,23 ...
416     )-TXV_funcPayne2004(...
417         [C_para2 ones(1,length(NC_Para)-length(index))], ...
418         data2(index,:), length(index), xin(index,:), refri, Pc, ...
419         Tc, SH_upper, NC_Para, TXV, SH_UP_P(index,:), ...
420         SH_DOWN_P(index,:), SH_UP_T(index,:), sc_drop ...
421     ), ones(1,length(index)), options);
422     C_para = ones(1,length(NC_Para));
423     C_para(1:length(index)) = C_para_temp(1:length(index));
424     C_para(1) = 0;
425     ini_con = TXV_funcPayne2004_constraint(...
426         C_para, data_norm, m_norm, xin_norm, refri, Pc, Tc, ...
427         SH_upper, NC_Para, TXV, SH_UP_P_norm, SH_DOWN_P_norm, ...
428         SH_UP_T_norm, sc_drop ...
429     );
430     if isnan(sum(ini_con)) || ¬isempty(find(ini_con>0))
431         options = optimset(...
432             'display','iter','algorithm','sqp', ...
433             'TolX',1.e-8,'TolCon',1.e-8,'MaxFunEvals',1000 ...
434         );
435         C_para_temp = fmincon(@(C_para2) cost_func2(...
436             [C_para2 ones(1,length(NC_Para)-length(index))], ...
437             data_norm(index,:), length(index), xin_norm(index,:), ...
```

```
438             refri, Pc, Tc, SH_upper, NC_Para, TXV, ...
439             SH_UP_P_norm(index,:), SH_DOWN_P_norm(index,:), ...
440             SH_UP_T_norm(index,:), sc_drop, ones(length(index),3)), ...
441             ones(1,length(index)), [...
442                 zeros(length(index),1) ...
443                 -ones(length(index),1)*NC_Para(2) ...
444                 -2*NC_Para(3).*(...
445                     SH_UP_P_norm(index,:)-SH_DOWN_P_norm(index,:) ...
446                 ) ...
447                 zeros(length(index),1)...
448             ], zeros(length(index),1),[],[],[],[],...
449                 @(C_para2) TXV_funcPayne2004_constraint(...
450                 [C_para2 ones(1,length(NC_Para)-length(index))], ...
451                 data_norm(index,:), length(index), ...
452                 xin_norm(index,:), refri, Pc, Tc, SH_upper, ...
453                 NC_Para, TXV, SH_UP_P_norm(index,:), ...
454                 SH_DOWN_P_norm(index,:), ...
455                 SH_UP_T_norm(index,:), sc_drop), ...
456             options);
457         C_para = ones(1,length(NC_Para));
458         C_para(1:length(index)) = C_para_temp(1:length(index));
459     end
460     % use parameters from the paper as initial guess
461     % find the opening correlation
462     options = optimset(...
463         'display','iter','algorithm','sqp','TolX',1.e-8,...
```

```
464         'TolCon',1.e-8,'MaxFunEvals',2000 ...
465     );
466     if m ≥ length(NC_Para)+4
467         sc_drop = 1;
468         [ValvePara,fval,exitflag,output] = fmincon(...
469             @(ValvePara) cost_func2(...
470                 ValvePara, data_norm, m_norm, xin_norm, ...
471                 refri, Pc, Tc, SH_upper, NC_Para, TXV, ...
472                 SH_UP_P_norm, SH_DOWN_P_norm, SH_UP_T_norm, ...
473                 sc_drop, weigh_norm ...
474             ), C_para,[...
475                 zeros(m_norm,1) -ones(m_norm,1)*NC_Para(2) ...
476                 -2*NC_Para(3).*(SH_UP_P_norm-SH_DOWN_P_norm) ...
477                 zeros(m_norm,length(NC_Para)-3) ...
478             ],zeros(m_norm,1),[],[],[...
479                 -Inf -Inf -Inf 1.e-10 ...
480             ],[],@(ValvePara) TXV_funcPayne2004_constraint(...
481                 ValvePara, data_norm, m_norm, xin_norm, refri, ...
482                 Pc, Tc, SH_upper, NC_Para, TXV, SH_UP_P_norm, ...
483                 SH_DOWN_P_norm, SH_UP_T_norm, sc_drop ...
484             ),options);
485         if fval > 0.05^2
486             options = optimset(...
487                 'display','iter','algorithm','interior-point',...
488                 'TolX',1.e-8,'TolCon',1.e-8,'MaxFunEvals',2000 ...
489             );
```

```
490             [ValvePara_new,fval2,exitflag,output] = fmincon(...
491                 @(ValvePara) cost_func2(...
492                     ValvePara, data_norm, m_norm, xin_norm, ...
493                     refri, Pc, Tc, SH_upper, NC_Para, TXV, ...
494                     SH_UP_P_norm, SH_DOWN_P_norm, SH_UP_T_norm, ...
495                     sc_drop, weigh_norm ...
496                 ), C_para,[...
497                     zeros(m_norm,1) -ones(m_norm,1)*NC_Para(2) ...
498                     -2*NC_Para(3).*(SH_UP_P_norm-SH_DOWN_P_norm) ...
499                     zeros(m_norm,length(NC_Para)-3) ...
500                 ],zeros(m_norm,1),[],[],[...
501                     -Inf -Inf -Inf 1.e-10 ...
502                 ],[],@(ValvePara) TXV_funcPayne2004_constraint(...
503                     ValvePara, data_norm, m_norm, xin_norm, refri, ...
504                     Pc, Tc, SH_upper, NC_Para, TXV, SH_UP_P_norm, ...
505                     SH_DOWN_P_norm, SH_UP_T_norm, sc_drop ...
506                 ),options);
507             if fval2 < fval
508                 fval = fval2;
509                 ValvePara = ValvePara_new;
510             end
511         end
512     else
513         sc_drop = 0;
514         C_para_small = C_para(1:4);
515         [ValvePara,fval,exitflag,output] = fmincon(...
```

```
516                 @(ValvePara) cost_func2(...
517                     [ValvePara C_para(5:length(NC_Para))], ...
518                     data_norm, ...
519                     m_norm, xin_norm, refri, Pc, Tc, SH_upper, ...
520                     NC_Para, TXV, SH_UP_P_norm, SH_DOWN_P_norm, ...
521                     SH_UP_T_norm, sc_drop, weigh_norm ...
522                 ), C_para_small,[...
523                     zeros(m_norm,1) -ones(m_norm,1)*NC_Para(2) ...
524                     -2*NC_Para(3).*(SH_UP_P_norm-SH_DOWN_P_norm) ...
525                     zeros(m_norm,1) ...
526                 ],zeros(m_norm,1),[],[],[...
527                     -Inf -Inf -Inf 1.e-10 ...
528                 ],[],@(ValvePara) TXV_funcPayne2004_constraint([...
529                     ValvePara C_para(5:length(NC_Para)) ...
530                 ], data_norm, m_norm, xin_norm, refri, Pc, Tc, ...
531                     SH_upper, NC_Para, TXV, SH_UP_P_norm, ...
532                     SH_DOWN_P_norm, SH_UP_T_norm, sc_drop),options);
533             if fval > 0.05^2
534                 options = optimset(...
535                     'display','on','algorithm','interior-point',...
536                     'TolX',1.e-8,'TolCon',1.e-8,'MaxFunEvals',1000 ...
537                 );
538                 if fval > 1
539                     ValvePara = C_para_small;
540                 end
541                 [ValvePara_new,fval2,exitflag,output] = fmincon(...
```

```
542                 @(ValvePara) cost_func2(...
543                     [ValvePara C_para(5:length(NC_Para))], ...
544                     data_norm, ...
545                     m_norm, xin_norm, refri, Pc, Tc, SH_upper, ...
546                     NC_Para, TXV, SH_UP_P_norm, SH_DOWN_P_norm, ...
547                     SH_UP_T_norm, sc_drop, weigh_norm ...
548                 ), C_para_small,[...
549                     zeros(m_norm,1) -ones(m_norm,1)*NC_Para(2) ...
550                     -2*NC_Para(3).*(SH_UP_P_norm-SH_DOWN_P_norm) ...
551                     zeros(m_norm,1) ...
552                 ],zeros(m_norm,1),[],[],[...
553                     -Inf -Inf -Inf 1.e-10 ...
554                 ],[],@(ValvePara) TXV_funcPayne2004_constraint(...
555                     [ValvePara C_para(5:length(NC_Para))], ...
556                     data_norm, m_norm, xin_norm, refri, Pc, Tc, ...
557                     SH_upper, NC_Para, TXV, SH_UP_P_norm, ...
558                     SH_DOWN_P_norm, SH_UP_T_norm, sc_drop ...
559                 ),options);
560             if fval2 < fval
561                 fval = fval2;
562                 ValvePara = ValvePara_new;
563             end
564         end
565         ValvePara = [ValvePara C_para(5:length(NC_Para))];
566     end
567     fff = fval;
```

```
568     old_fff(iii,1) = fff;
569     old_C_para(iii,:) = C_para;
570     old_ValvePara(iii,:) = ValvePara;
571     old_NC_Para(iii,:) = NC_Para;
572     old_SH_upper(iii,1) = SH_upper;
573 %     save(savefilename);
574 %     movefile(savefilename,strcat(foldername,'\'));
575 end
576
577 iii_index = find(old_fff==min(old_fff));
578 if m ≥ 10
579     sc_drop = 1;
580 else
581     sc_drop = 0;
582 end
583
584 % assign parameters
585 C_para = old_C_para(iii_index,:);
586 ValvePara = old_ValvePara(iii_index,:);
587 NC_Para = old_NC_Para(iii_index,:);
588 SH_upper = old_SH_upper(iii_index,1);
589 TXV.para.ValvePara_A_index = ValvePara(1:3).*NC_Para(1:3);
590 TXV.para.L = ValvePara(4).*NC_Para(4);
591 TXV.para.ValvePara_C_index = ValvePara(...
592     5:length(NC_Para) ...
593 ).*NC_Para(5:length(NC_Para));
```

```
594 TXV.para.SH_upper = SH_upper;
595
596 % re-calculate to ensure that the dimensionalization is correct
597 mdot = TXV_funcPayne2004(...
598     ValvePara.*NC_Para, data2, m, xin, refri, Pc, Tc, SH_upper, ...
599     ones(1,length(NC_Para)), TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, ...
600     sc_drop ...
601 );
602
603 save(savefilename);
604 movefile(savefilename,strcat(foldername,'\'));
605
606 % re-calculate to ensure that the dimensionalization is correct
607 mdot = TXV_funcPayne2004(...
608     ValvePara.*NC_Para, data2, m, xin, refri, Pc, Tc, SH_upper, ...
609     ones(1,length(NC_Para)), TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, ...
610     sc_drop...
611 );
612
613 for i = 1:length(ValvePara)
614     ValvePara_sp = ValvePara;
615     ValvePara_sp(i) = ValvePara(i)+1.e-8;
616     X_Cov(:,i) = (TXV_funcPayne2004(...
617         ValvePara_sp.*NC_Para, data2, ...
618         m, xin, refri, Pc, Tc, SH_upper, ones(1,length(NC_Para)), ...
619         TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop ...
```

```
620     )-TXV_funcPayne2004(...
621         ValvePara.*NC_Para, data2, m, xin, refri, Pc, Tc, ...
622         SH_upper, ones(1,length(NC_Para)), TXV, SH_UP_P, ...
623         SH_DOWN_P, SH_UP_T, sc_drop ...
624     ))./(ValvePara_sp(i)-ValvePara(i));
625 end
626 TXV.para.Cov = covariance_adj(X_Cov);
627 TXV.para.X_limit = max(diag(X_Cov*TXV.para.Cov*X_Cov'));
628 TXV.para.NC_Para = NC_Para;
629 if strcmp(refri, 'R410A')
630     TXV.para.x_max = 0.087;
631 elseif strcmp(refri, 'R407C')
632     TXV.para.x_max = 0.047;
633 elseif strcmp(refri, 'R134a.fld')
634     TXV.para.x_max = 0.104;
635 elseif strcmp(refri, 'R502')
636     TXV.para.x_max = 0.094;
637 elseif strcmp(refri, 'R22.fld')
638     TXV.para.x_max = 0.102;
639 else
640     TXV.para.x_max = max(0.0, max(xin));
641 end
642
643 SSE = 0; % residual sum of squares declaration
644 SST = 0; % totol sum of squares declaration
645 mdot_av = mean(data2(:,23));
```

```
646 i = 1;
647 k = 1;
648 for j = 1:m
649     if(SC(j,1)≤0)
650         mdota_2p(i,1) = data2(j,23);
651         mdotb_2p(i,1) = mdot(j,1);
652         SH_2p(i,1) = SH(j,1);
653         xin_2p(i,1) = xin(j,1);
654         HP_2p(i,1) = data2(j,5);
655         i = i+1;
656     else
657         mdotb_1p(k,1) = mdot(j,1);
658         mdota_1p(k,1) = data2(j,23);
659         SH_1p(k,1) = SH(j,1);
660         xin_1p(k,1) = xin(j,1);
661         HP_1p(k,1) = data2(j,5);
662         k = k+1;
663     end
664     SSE = SSE + (mdot(j,1) - data2(j,23))^2;
665     SST = SST + (data2(j,23)- mdot_av)^2;
666 end
667
668 r2_m = 1 - SSE/SST;
669 save(savefilename);
670 movefile(savefilename,strcat(foldername,'\'));
671
```

```
672 end
673
674 function r = cost_func2(...
675     ValvePara, data, m, xin, refri, Pc, Tc, SH_upper, NC_Para, ...
676     TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop, weigh ...
677 )
678     r = 0;
679     mr = TXV_funcPayne2004(...
680         ValvePara, data, m, xin, refri, Pc, Tc, SH_upper, ...
681         NC_Para, TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop ...
682     );
683     for i = 1:m
684         r = r + (1/weigh(i,1)+1/weigh(i,2)+1/weigh(i,3))*(...
685             (mr(i,1) - data(i,23))/data(i,23) ...
686         )^2;
687     end
688     r = r/m;
689 end
690
691 function [c, ceq] = TXV_funcPayne2004_constraint(...
692     ValvePara, data, m, xin, refri, Pc, Tc, SH_upper, NC_Para, ...
693     TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop ...
694 )
695     ceq = [];
696     xin = ones(length(xin),1)*min(xin);
697     for i = 1:m
```

```
698         T = propertyRH('T', 'P', data(i,5), 'Q', 0, refri);
699         hg = propertyRH('H', 'P', data(i,5), 'Q', 1, refri);
700         hf = propertyRH('H', 'P', data(i,5), 'Q', 0, refri);
701         h = propertyRH('H', 'T', T-20, 'P', data(i,5), refri);
702         xin(i,1) = (h-hf)/(hg-hf);
703     end
704     [mr DischargeCoeff c] = TXV_funcPayne2004(...
705         ValvePara, data, m, xin, refri, Pc, Tc, SH_upper, ...
706         NC_Para, TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop ...
707     );
708     c = -c;
709     c(length(c)+1,1) = ValvePara(3).*NC_Para(3);
710 end
711
712 function [mr DischargeCoeff dmdSC] = TXV_funcPayne2004(...
713     ValvePara, data, m, x__in, refri, Pc, Tc, SH_upper, ...
714     NC_Para, TXV, SH_UP_P, SH_DOWN_P, SH_UP_T, sc_drop ...
715 )
716     % Coefficients as defined in Payne (2004)
717     para.a(1,1) = 3.8811E-01;% [-]
718     para.a(2,1) = 1.1427E+01;% [-]
719     para.a(3,1) = -1.4194E+01;% [-]
720     para.a(4,1) = 1.0703E+00;% [-]
721     para.a(5,1) = -9.1928E-02;% [-]
722     para.a(6,1) = 2.1425E+01;% [-]
723     para.a(7,1) = -5.8195E+02;% [-]
```

```
724
725     para.b(1,1) = 1.1831E+00;% [-]
726     para.b(2,1) = -1.4680E+00;% [-]
727     para.b(3,1) = -1.5285E-01;% [-]
728     para.b(4,1) = -1.4639E+01;% [-]
729     para.b(5,1) = 9.8401E+00;% [-]
730     para.b(6,1) = -1.9798E-02;% [-]
731     para.b(7,1) = -1.5348E+00;% [-]
732     para.b(8,1) = -2.0533E+00;% [-]
733     para.b(9,1) = -1.7195E+01;% [-]
734
735     ValvePara = ValvePara.*NC_Para;
736     Amax = ValvePara(1)+ValvePara(2)*SH_upper/200 +...
737         ValvePara(3)*(SH_upper/200)^2;
738
739     for i = 1:m
740         Pin = data(i,5);
741         Pout = data(i,6);
742         x_in = x__in(i,1);
743
744         % defining variable
745         Tsat = propertyRH('T','P',Pin,'Q',0,refri);
746         hf = propertyRH('H','P',Pin,'Q',0,refri);
747         hv = propertyRH('H','P',Pin,'Q',1,refri);
748         hin = x_in*(hv-hf)+hf;
749         if(hin≥hf)
```

```
750             Tin = Tsat;
751         else
752             Tin = propertyRH('T','P',Pin,'H',hin,refri);
753         end
754         Psat = propertyRH('P','T',Tin,'Q',0,refri);
755         rho_g = propertyRH('D','P',Pin,'Q',1,refri);
756         rho_f = propertyRH('D','P',Pin,'Q',0,refri);
757         T_sub = Tsat - Tin;
758
759         pi_gp(1) = (Pin - Psat)/Pc;
760         if (Psat>Pin)
761             pi_gp(1) = 0;
762         end
763         pi_gp(2) = rho_g/rho_f;
764         pi_gp(3) = T_sub/Tc;
765         if pi_gp(3)<0
766             pi_gp(3) = 0;
767         end
768         pi_gp(4) = Pout/Pc;
769         pi_gp(5) = Pin/Pc;
770
771         rho_use = rho_f;
772
773         % find area
774         Ct = pi/4*sqrt(rho_use*Pc*1000)/(...
775             1+ValvePara(8)*pi_gp(1)+ValvePara(5)*pi_gp(3)^2 ...
```

```
776         );
777         Vc = para.a(1)+para.a(2)*pi_gp(1)+ValvePara(6)*pi_gp(3) +...
778             ValvePara(9)*pi_gp(2);
779         C1pCoeff(1) = Ct*ValvePara(7);
780         C1pCoeff(2) = Ct*Vc;
781
782         ΔP(i,1) = SH_UP_P(i,1) - SH_DOWN_P(i,1);
783         if ΔP(i,1) > SH_upper;
784             ΔP(i,1) = SH_upper;
785         end
786
787         A = ValvePara(1)+ValvePara(2)*ΔP(i,1)/200 +...
788             ValvePara(3)*ΔP(i,1)^2/200^2;
789         if A > Amax
790             A = Amax;
791         elseif A < Amax*1.e-4
792             A = Amax*1.e-4;
793         end
794         Din_cal = sqrt(A/pi*4);
795
796         D1 = Din_cal^2*log(ValvePara(4)/Din_cal);
797         D2 = Din_cal^2;
798         mr(i,1) = (C1pCoeff(1)*D1 + C1pCoeff(2)*D2);
799         DischargeCoeff(i,1) = data(i,23)/(...
800             C1pCoeff(1)*log(ValvePara(4)/Din_cal) + C1pCoeff(2) ...
801         );
```

```
802
803         % make sure that the mass flow rate always increase
804         %  or remain the same when subcooling increases
805         % check derivative w.r.t. SC
806         dCtdSC = -Ct*2*ValvePara(5)*2*pi_gp(3)/(...
807             1+ValvePara(8)*pi_gp(1)+ValvePara(5)*pi_gp(3)^2 ...
808         )/Tc;
809         dVcdSC = ValvePara(6)/Tc;
810         dmdSC(i,1) = real((...
811             ValvePara(7)*D1+Vc*D2 ...
812         )*dCtdSC+Ct*D2*dVcdSC);
813         if T_sub < 3
814             dmdSC(i,1) = 0.0;
815             pi_gp(1) = (Pin-propertyRH(...
816                 'P','T',Tsat-3,'Q',0,refri ...
817             ))/Pc;
818             pi_gp(3) = 3/Tc;
819             C1pCoeff(1) = pi/4*sqrt(rho_use*Pc*1000)/(...
820                 1+ValvePara(8)*pi_gp(1)+ValvePara(5)*pi_gp(3)^2 ...
821             )*ValvePara(7);
822             C1pCoeff(2) = pi/4*sqrt(rho_use*Pc*1000)/(...
823                 1+ValvePara(8)*pi_gp(1)+ValvePara(5)*pi_gp(3)^2 ...
824             )*(...
825                 para.a(1)+para.a(2)*pi_gp(1)+ ...
826                 ValvePara(6)*pi_gp(3)+ValvePara(9)*pi_gp(2) ...
827             );
```

```
828             mr_3K = C1pCoeff(1)*Din_cal^2*log(...
829                 ValvePara(4)/Din_cal ...
830             ) + ...
831                 C1pCoeff(2)*Din_cal^2;
832             if mr(i,1) > mr_3K
833                 mr(i,1) = mr_3K;
834             end
835         end
836
837         % check state at inlet
838         if(hin>hf) % for two-phase inlet
839             xin = (hin-hf)/(hv-hf);
840             Ctp = Payne2004Ctp(...
841                 xin, Pin, Pc, Psat, A, para, ValvePara, ...
842                 Din_cal, refri ...
843             )/Payne2004Ctp(...
844                 0, Pin, Pc, Psat, A, para, ValvePara, ...
845                 Din_cal, refri ...
846             );
847             if Ctp > 1
848                 Ctp = 1;
849             end
850             mr(i,1) = mr(i,1)*Ctp;
851         end
852     end
853     mr = real(mr);
```

```
854
855 end
856
857 function Ctp = Payne2004Ctp(...
858         xin, Pin, Pc, Psat, A, para, ValvePara, Din_cal, refri ...
859     )
860     rho_g = propertyRH('D','P',Pin,'Q',1,refri);
861     rho_f = propertyRH('D','P',Pin,'Q',0,refri);
862     rho_mup = ((1-xin)/rho_f+xin/rho_g)^(-1);
863     tp6 = rho_mup/rho_f;
864     tp28 = Pin/Pc;
865     tp32 = 1 - Pin/Pc;
866     tp34 = xin/(1-xin)*(rho_f/rho_g)^(.5);
867     tp35 = 1 - Psat/Pc;
868     CtpCoeff(1) = (...
869         para.b(1)*tp6+para.b(2)*tp6^2+para.b(3)*(log(tp6))^2+...
870         para.b(4)*(log(tp35))^2+para.b(5)*(log(tp32))^2 ...
871     )/(1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3);
872     CtpCoeff(2) = para.b(6)/(...
873         1+para.b(7)*tp6+para.b(8)*tp34+para.b(9)*tp28^3 ...
874     );
875     Ctp = (CtpCoeff(1) + CtpCoeff(2)*(log(ValvePara(4)/Din_cal))^2);
876 end
```

N.8 Evaporator Modeling

```
1 function System = Evaporator_minimization_v068_main()
```

```
2
3  % Passing system specification to the function to train the
4  % evaporator model
5
6  % define function and folder name
7  function_name = 'Evaporator_minimization';
8  version_main = '068';
9  version_regress = '068';
10 version_plot = '005';
11 foldername = strcat(function_name,'_v',version_main);
12 regressname = strcat(function_name,'_v',version_regress);
13 plotname = strcat(function_name,'_plot_v',version_plot);
14 regress_func = str2func(['@(System, libname, version_number)',...
15     regressname,'(System, libname, version_number)']);
16 plot_func = str2func(['@(System, libname, version_number)',...
17     plotname,'(System, libname, version_number)']);
18 mkdir(foldername);
19 savefilename = strcat(foldername,'.mat');
20
21 % Evaporator module definition
22 % Inner suface area of evaproator [m^2]
23 Evap.para.A_r = 0.0;
24 % Rated heat transfer coefficient on the air-side [W/m^2-K]
25 Evap.para.U_a = 0.0;
26 % Adjustment index for air-side HTC based on airflow% [-]
27 Evap.para.n = 0.0;
```

```
28 % Adjusment for air-side HTC based on fin geometry [-]
29 Evap.para.p = 0.0;
30 Evap.para.ma = 0.0; % rated airflow [kg/s]
31 % Rated heat transfer coefficient on the ref-side for two-phase
32 % flow [W/m^2-K]
33 Evap.para.Utp = 0.0/1;
34 % Rated heat transfer coefficient on the ref-side for
35 % superheated flow [W/m^2-K]
36 Evap.para.Ush = 0.0;
37 % Rated property effect on superheated HTC
38 Evap.para.Ksh = 0.0;
39 Evap.para.Din = 0.0; % Inner diameter [m]
40 % Total length of heat exchagner pipe [m]
41 Evap.para.Lt = 0.0;
42 Evap.para.FanPower = 0.0; % Average fan power consumption [W]
43 Evap.para.ΔP = 0.0; % Rated pressure drop across evaporator [kPa]
44 % Adjustment index for refrigerant
45 % pressure drop based on flow [-]
46 Evap.para.Pn = 0.0;
47 Evap.para.mdot_r = 0.0; % Rated refrigernat mass flow rate [kg/s]
48 % Average refrigerant pressure drop [kPa]
49 Evap.para.ΔP_const = 0.0;
50
51 Evap.para.ΔP.C = zeros(4,1);
52 Evap.para.ΔP.mdot_rated = 0.0;
53 Evap.para.ΔP.rho_rated = 0.0;
```

```
54 Evap.para.ΔP.rho_out_rated = 0.0;
55 Evap.para.ΔP.mu_v_rated = 0.0;
56 Evap.para.ΔP_max = 0.0;
57 Evap.para.Evap_ref_airflow = 0.0;
58 Evap.para.Evap_mea_airflow = 0.0;
59
60 % Set information
61 % re-file at this stage to avoid problems in parfor loop
62 % Boshen 3-ton R410A packaged FXO system (Scroll)
63 % (Shen_030_R410A_packaged_FXO_Scroll)
64 i = 1;
65 System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
66 System(i).othername = System(i).name;
67 System(i).cond_name = System(i).name;
68 System(i).LLname = System(i).name;
69 System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
70 System(i).foldername = 'Boshen';
71 System(i).worksheetname = 'SI (no_invalid_CA)';
72 System(i).refname = 'R410A';
73 System(i).Standard_Airflow_evap = 0;
74 System(i).Fan_Upstream_evap = 0;
75 System(i).Standard_Airflow_cond = 1;
76 System(i).Fan_Upstream_cond = 1;
77 System(i).Heating = 0;
78 System(i).Combined = 0;
79 System(i).Accumulator = 0;
```

```
80 System(i).Diameter = 0.00853;
81 System(i).Tube_Length = 0.6066;
82 System(i).Ntube = 66;
83 System(i).Pc = 4901;
84 System(i).Dom_Valve = 'FXO';
85
86 % Boshen 5-ton R407C packaged FXO system (Scroll)
87 % (Shen_050_R407C_packaged_FXO_Scroll)
88 i = i + 1;
89 System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
90 System(i).othername = System(i).name;
91 System(i).cond_name = System(i).name;
92 System(i).LLname = System(i).name;
93 System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
94 System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
95 System(i).worksheetname = 'SI (no_invalid_CA)';
96 System(i).refname = 'R407C';
97 System(i).Standard_Airflow_evap = 0;
98 System(i).Fan_Upstream_evap = 0;
99 System(i).Standard_Airflow_cond = 1;
100 System(i).Fan_Upstream_cond = 1;
101 System(i).Heating = 0;
102 System(i).Combined = 0;
103 System(i).Accumulator = 0;
104 System(i).Diameter = 0.00691896;
105 System(i).Tube_Length = 1.505204;
```

```
106 System(i).Ntube = 24*3;
107 System(i).Pc = 3786;
108 System(i).Dom_Valve = 'FXO';
109
110 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
111 % (HCA3_030_R410A_split_TXV_Scroll)
112 i = i + 1;
113 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
114 System(i).othername = System(i).name;
115 System(i).cond_name = System(i).name;
116 System(i).LLname = System(i).name;
117 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
118 System(i).foldername = 'Kim-HCA3';
119 System(i).worksheetname = 'SI (no_invalid_CA)';
120 System(i).refname = 'R410A';
121 System(i).Standard_Airflow_evap = 0;
122 System(i).Fan_Upstream_evap = 1;
123 System(i).Standard_Airflow_cond = 1;
124 System(i).Fan_Upstream_cond = 1;
125 System(i).Heating = 0;
126 System(i).Combined = 0;
127 System(i).Accumulator = 0;
128 % assumed
129 System(i).Diameter = 0.00849;
130 System(i).Tube_Length = 2.255;
131 System(i).Ntube = 32;
```

```
System(i).Pc = 4901;
System(i).Dom_Valve = 'TXV';

% Kim 4-ton R410A packaged TXV system (Recip) (TM)
% (TM_040_R410A_packaged_TXV_Recip)
i = i + 1;
System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
System(i).othername = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
System(i).foldername = 'Kim-TM';
System(i).worksheetname = 'SI (no_LL)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Accumulator = 1;
% assumed
System(i).Diameter = 0.0089;
System(i).Tube_Length = 0.8382;
System(i).Ntube = 96;
System(i).Pc = 4901;
```

```
158 System(i).Dom_Valve = 'TXV';
159
160 % Breuker 3-ton R22 packaged FXO system (Recip)
161 % (Breuker_030_R22_packaged_FXO_Recip)
162 i = i + 1;
163 System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
164 System(i).othername = System(i).name;
165 System(i).cond_name = System(i).name;
166 System(i).LLname = System(i).name;
167 System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
168 System(i).foldername = 'Breuker';
169 System(i).worksheetname = 'SI (no_invalid_CA)';
170 System(i).refname = 'R22.fld';
171 System(i).Standard_Airflow_evap = 0;
172 System(i).Fan_Upstream_evap = 0;
173 System(i).Standard_Airflow_cond = 1;
174 System(i).Fan_Upstream_cond = 1;
175 System(i).Heating = 0;
176 System(i).Combined = 0;
177 System(i).Accumulator = 0;
178 % assumed
179 System(i).Diameter = 0.00853;
180 System(i).Tube_Length = 0.6066;
181 System(i).Ntube = 66;
182 System(i).Pc = 4990;
183 System(i).Dom_Valve = 'FXO';
```

```
184
185 % Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
186 % (NIST_025_R410A_split_TXV_Scroll)
187 i = i + 1;
188 System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
189 System(i).othername = System(i).name;
190 System(i).cond_name = System(i).name;
191 System(i).LLname = System(i).name;
192 System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
193 System(i).foldername = 'NIST';
194 System(i).worksheetname = 'SI (no_invalid_CA)';
195 System(i).refname = 'R410A';
196 System(i).Standard_Airflow_evap = 1;
197 System(i).Fan_Upstream_evap = 1;
198 System(i).Standard_Airflow_cond = 1;
199 System(i).Fan_Upstream_cond = 1;
200 System(i).Heating = 0;
201 System(i).Combined = 0;
202 System(i).Accumulator = 0;
203 System(i).Diameter = 0.007747;
204 System(i).Tube_Length = 0.508;
205 System(i).Ntube = 60;
206 System(i).Pc = 4901;
207 System(i).Dom_Valve = 'TXV';
208
209 % Harms 5-ton R22 packaged TXV system (Scroll)
```

```
% (Harms_050_R22_packaged_TXV_Scroll_Heating)
i = i + 1;
System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
System(i).othername = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
System(i).foldername = 'Harms_050tonR22PackagedTXV';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Accumulator = 0;
System(i).Diameter = 0.00889;
System(i).Tube_Length = 0.617;
System(i).Ntube = 32*4;
System(i).Pc = 4990;
System(i).Dom_Valve = 'TXV';

% Boshen 3-ton R410A split FXO system (Recip)
% (Shen_030_R410A_split_FXO_Recip)
% for compressor modeling, merged with TXV system data
```

```
236 i = i + 1;
237 System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
238 System(i).othername = System(i).name;
239 System(i).cond_name = System(i).name;
240 System(i).LLname = System(i).name;
241 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
242 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
243 System(i).worksheetname = 'SI (no_invalid_CA)';
244 System(i).refname = 'R410A';
245 System(i).Standard_Airflow_evap = 0;
246 System(i).Fan_Upstream_evap = 0;
247 System(i).Standard_Airflow_cond = 1;
248 System(i).Fan_Upstream_cond = 1;
249 System(i).Heating = 0;
250 System(i).Combined = 0;
251 System(i).Accumulator = 0;
252 System(i).Diameter = 0.00849;
253 System(i).Tube_Length = 0.452;
254 System(i).Ntube = 28*3;
255 System(i).Pc = 4901;
256 System(i).Dom_Valve = 'TXV';
257
258 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
259 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
260 i = i + 1;
261 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
```

```
System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
System(i).foldername = 'Kim-HCA3-Heating';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R410A';
System(i).Standard_Airflow_evap = 1;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 1;
System(i).Combined = 0;
System(i).Accumulator = 0;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4901;
System(i).Dom_Valve = 'TXV';

% Kim 3-ton R22 split TXV system (Scroll) (YSA)
% (YSA_030_R22_split_TXV_Scroll)
i = i + 1;
System(i).name = 'YSA_030_R22_split_TXV_Scroll';
System(i).othername = System(i).name;
```

```
288 System(i).cond_name = System(i).name;
289 System(i).LLname = System(i).name;
290 System(i).filename = 'Kim_Cycle_data_R22_YSA_v02.xlsx';
291 System(i).foldername = 'Kim-YSA';
292 System(i).worksheetname = 'SI (no_invalid_CA)';
293 System(i).refname = 'R22.fld';
294 System(i).Standard_Airflow_evap = 0;
295 System(i).Fan_Upstream_evap = 1;
296 System(i).Standard_Airflow_cond = 1;
297 System(i).Fan_Upstream_cond = 1;
298 System(i).Heating = 0;
299 System(i).Combined = 0;
300 System(i).Accumulator = 1;
301 % assumed
302 System(i).Diameter = 0.00849;
303 System(i).Tube_Length = 2.255;
304 System(i).Ntube = 32;
305 System(i).Pc = 4990;
306 System(i).Dom_Valve = 'FXO';
307
308 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
309 % (YSA_030_R22_split_TXV_Scroll_Heating)
310 i = i + 1;
311 System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
312 System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
313 System(i).cond_name = System(i).name;
```

```
System(i).LLname = System(i).name;
System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
System(i).foldername = 'Kim-YSA-Heating';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R22.fld';
System(i).Standard_Airflow_evap = 1;
System(i).Fan_Upstream_evap = 1;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 1;
System(i).Combined = 0;
System(i).Accumulator = 1;
% assumed
System(i).Diameter = 0.00849;
System(i).Tube_Length = 2.255;
System(i).Ntube = 32;
System(i).Pc = 4990;
System(i).Dom_Valve = 'TXV';

% Kim 3-ton R22 split TXV system (Scroll) (YKC)
% (YKC_030_R22_split_TXV_Scroll)
i = i + 1;
System(i).name = 'YKC_030_R22_split_TXV_Scroll';
System(i).othername = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
```

```
340 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
341 System(i).foldername = 'Kim-YKC';
342 System(i).worksheetname = 'SI (no_invalid_CA)';
343 System(i).refname = 'R22.fld';
344 System(i).Standard_Airflow_evap = 0;
345 System(i).Fan_Upstream_evap = 1;
346 System(i).Standard_Airflow_cond = 1;
347 System(i).Fan_Upstream_cond = 1;
348 System(i).Heating = 0;
349 System(i).Combined = 0;
350 System(i).Accumulator = 1;
351 % assumed
352 System(i).Diameter = 0.00849;
353 System(i).Tube_Length = 2.255;
354 System(i).Ntube = 32;
355 System(i).Pc = 4990;
356 System(i).Dom_Valve = 'FXO';
357
358 % other settings for storage
359 m = length(System);
360 % m = 1;
361 % m = 2;
362 for i = 1:m
363     System(i).mainfoldername = foldername;
364     System(i).Evap = Evap;
365     System(i).r2_Q = 0;
```

```
366     System(i).r2_SHR = 0;
367     System(i).r2_ΔP = 0;
368 end
369
370 % setting up parallel solver, (longer calculation so more cores)
371 try
372     matlabpool open 3; % try setting up multiple loops
373 catch err
374     matlabpool close; % if catch an error, close and reset it
375     matlabpool open 3; % try setting up multiple loops
376 end
377
378 %reload library
379 pctRunOnAll loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
380 libname ='lib'; % Name of library
381
382 % start calculation
383 no_plot = zeros(m,1);
384 % make an error statment
385 try
386     parfor i = 1:m
387         System(i) = 1; % a statement leads to error
388     end
389 catch err
390 end
391 for i = 1:m
```

```
392     error_statement(i) = err;
393 end
394 
395 parfor i = 1:m
396     try
397         disp(['Starting to run ',System(i).name]);
398         [Evap r2_Q r2_SHR r2_ΔP] = regress_func(...
399             System(i), libname, version_main ...
400         );
401         System(i).Evap = Evap;
402         System(i).r2_Q = r2_Q;
403         System(i).r2_SHR = r2_SHR;
404         System(i).r2_ΔP = r2_ΔP;
405         disp(['Finishing simulating ',System(i).name]);
406     catch err
407         disp(['Problem found in system ',System(i).name]);
408         try
409             disp(err.message);
410             for qq = 1:length(err.stack)
411                 disp('Function Name:')
412                 disp(err.stack(qq).name);
413                 disp('Line Number:')
414                 disp(err.stack(qq).line);
415             end
416         end
417         no_plot(i,1) = 1;
```

```
418         error_statement(i) = err;
419     end
420 end
421 save(savefilename);
422 
423 % plot cannot be done in parfor. Do it in a for loop
424 for i = 1:m
425 % for i = 4:4
426 % for i = 3:3
427     if no_plot(i,1) == 0
428         disp(['Starting to plot ',System(i).name]);
429         plot_func(System(i), libname, version_main);
430         disp(['Finishing plotting ',System(i).name]);
431     end
432 end
433 
434 save(savefilename);
435 movefile(savefilename,strcat(foldername,'\'));
436 
437 try
438     matlabpool close; % if catch an error, close and reset it
439 end
440 try
441     unloadlibrary lib;
442 end
```

```
function [Evap coeffQ coeffSHR coeffΔP] = ...
    Evaporator_minimization_v068(System, libname, version_number)

% Function to train evaporator model

%% Preparation
% data reading and filtering
bin_num = 10;
filename = System.filename;
worksheetname = System.worksheetname;
refname = System.refname;
Standard_Airflow_evap = System.Standard_Airflow_evap;
Ind_Fan_Upstream_evap = System.Fan_Upstream_evap;
Standard_Airflow_cond = System.Standard_Airflow_cond;
Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
D = System.Diameter;
Ltube = System.Tube_Length;
Ntube = System.Ntube;
Pc = System.Pc;
Dom_Valve = System.Dom_Valve;
version = strcat(System.name,'_',version_number);
savefilename = strcat('Evaporator_minimization_v',version,'.mat');
foldername = strcat(strcat(...
    pwd,'\',System.mainfoldername,'\' ...
),strcat('Evaporator_minimization_v',version));
if ¬exist(foldername,'dir')
```

```
27      mkdir(foldername);
28  end
29
30  % read Q_gain_observation_v011 to find points to be removed
31  Q_gain_version = '029';
32  Q_gain_name = [...
33      'Q_gain_observation_v',System.name,'_',Q_gain_version ...
34  ];
35  Q_gain_raw = open([...
36      'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name, ...
37      '\',Q_gain_name,'.mat' ...
38  ]);
39  k = length(Q_gain_raw.index_removed);
40  ref = Q_gain_raw.data;
41  code = Q_gain_raw.code;
42  fault = Q_gain_raw.fault;
43  Q_cond_r = Q_gain_raw.Q_cond_r;
44  Q_evap_r = Q_gain_raw.Q_evap_r;
45  for i = k:-1:1 % reverse for correct indexing
46      ref(Q_gain_raw.index_removed(i),:) = [];
47      code(Q_gain_raw.index_removed(i),:) = [];
48      fault(Q_gain_raw.index_removed(i),:) = [];
49      Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
50      Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
51  end
52  [pp qq] = size(ref);
```

```
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
i = 1;
while i ≤ pp
    if ¬isempty(cell2mat(cell_VL(i))) || ...
        ¬isempty(cell2mat(cell_NC(i))) || ...
        ¬isempty(cell2mat(cell_LL(i)))
        ref(i,:) = [];
        fault(i,:) = [];
        code(i,:) = [];
        cell_VL(i,:) = [];
        cell_LL(i,:) = [];
        cell_NC(i,:) = [];
        Q_evap_r(i,:) = [];
        Q_cond_r(i,:) = [];
        i = i - 1;
```

```
79          pp = pp - 1;
80      end
81      i = i + 1;
82  end
83  [pp qq] = size(ref);
84  p = pp;
85  q = qq;
86
87  [m, n] = size(ref);
88  n = n+1;
89  refri = refname;
90  D = 0.007747;
91  Ar = Ntube*Ltube*pi*D;
92
93  % for Compressor
94  version_Comp = '073';
95  version_func = '045';
96  name_Comp = 'Compressor_regression_v';
97  component_name = strcat(name_Comp,System.othername,'_',version_Comp);
98  foldername_comp = strcat(strcat(...
99      pwd,'\',name_Comp,version_Comp,'\',component_name ...
100 ));
101 raw = open(strcat(foldername_comp,'\',strcat(component_name,'.mat')));
102 Comp_para = raw.Comp;
103 Comp_func = str2func([...
104     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
```

```
105     'Compressor_v',version_func,...
106     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
107 ]);
108 
109 % for Liquid Line
110 version_LiquidLine = '040';
111 version_func = '003';
112 name_LiquidLine = 'LiquidLine_minimization_v';
113 component_name = strcat(...
114     name_LiquidLine,System.LLname,'_',version_LiquidLine ...
115 );
116 foldername_LiquidLine = strcat(strcat(...
117     pwd,'\',name_LiquidLine,version_LiquidLine,'\',component_name ...
118 ));
119 raw = open(strcat(...
120     foldername_LiquidLine,'\',strcat(component_name,'.mat') ...
121 ));
122 LiquidLine_para = raw.PipeLineClass;
123 LiquidLine_func = str2func([...
124     '@(mr, Pin, hin, Tamb, para, refri, phase, libname)',...
125     'pipeline_v',version_func,...
126     '(mr, Pin, hin, Tamb, para, refri, phase, libname)' ...
127 ]);
128 
129 % prefer refrigerant-side instead of air-side
130 % filtering
```

```
131 i = 1;
132 n_SH_limit = 8;
133 while i ≤ m
134     if ref(i,5) > ref(i,4)
135         ref(i,5) = ref(i,4);
136     end
137     hout_cond = propertyRH('H','T',ref(i,12),'P',ref(i,4),refri);
138     hin_EXV = propertyRH('H','T',ref(i,13),'P',ref(i,5),refri);
139     % if heat cannot be lost in the direction of temperature,
140     % change the temperature reading
141     if (hout_cond - hin_EXV)/(ref(i,12)-ref(i,21)) < 0
142         ref(i,13) = propertyRH(...
143             'T','P',ref(i,5),'H',hout_cond,refri ...
144         );
145     end
146     SC_EEV = propertyRH('T','P',ref(i,5),'Q',0,refri) - ref(i,13);
147     SC_cond = propertyRH('T','P',ref(i,4),'Q',0,refri) - ref(i,12);
148     SH_comp = ref(i,9) - propertyRH('T','P',ref(i,1),'Q',1,refri);
149     SH_evap = ref(i,16) - propertyRH('T','P',ref(i,8),'Q',1,refri);
150     if ((ref(i,23) < 0 & SH_comp<1) | ...
151             (SC_cond<1 & ¬isempty(...
152                 cell2mat(strfind(fault(i,:),'CA')) ..
153             ))) | (SC_EEV<1 & SH_evap < 1)
154         ref(i,:) = [];
155         code(i,:) = [];
156         fault(i,:) = [];
```

```
157         Q_evap_r(i,:) = [];
158         Q_cond_r(i,:) = [];
159         i = i - 1;
160         m = m - 1;
161     elseif ¬((((SC_cond ≥ 3)||(...
162             SH_evap>1&&SC_EEV≥1 ...
163         )||(SC_cond≥1)) && ref(i,23) > 0) || SH_comp>1)
164         ref(i,:) = [];
165         code(i,:) = [];
166         fault(i,:) = [];
167         Q_evap_r(i,:) = [];
168         Q_cond_r(i,:) = [];
169         i = i - 1;
170         m = m - 1;
171     else
172         SH_check(i,1) = ref(i,16) - ...
173             propertyRH('T','P',ref(i,8),'Q',1,refri);
174         SH_check_II(i,1) = SH_comp;
175         SC_check(i,1) = SC_EEV;
176         SC_check_II(i,1) = SC_cond;
177         SC(i,1) = SC_check(i,1);
178     end
179     i = i + 1;
180 end
181 [m, n] = size(ref); %
182
```

```
183 if length(find(SH_check>1)) > n_SH_limit
184     Air_side = 0;
185 else
186     i = 1;
187     Air_side = 1;
188 end
189 
190 airinlet = zeros(1,5);
191 airoutlet = airinlet;
192 
193 % initilaize variables
194 output = zeros(m,14);
195 input =  zeros(m,15);
196 % matching ref and input
197 input(1:m,2) = ones(m,1);
198 input(1:m,3) = ones(m,1);
199 input(1:m,6) = ref(1:m,17);
200 input(1:m,9) = ref(1:m,32);
201 input(1:m,10) = ref(1:m,23);
202 for i = 1:m
203     Psat(i) = ref(i,8);
204     T_sat(i) = propertyRH('T','P',Psat(i),'Q',1,refri);
205     SH(i,1) = ref(i,16)-propertyRH(...
206         'T','P',ref(i,8),'Q',1,refri ...
207     );
208     input(i,12) = Psat(i);
```

```
209     input(i,13) = T_sat(i);
210     Patm(i,1) = input(i,9);
211 end
212 input(1:m,14) = D;
213 input(1:m,15) = Ar/pi/D;
214
215 % mean airflow calculation
216 if Air_side==0
217     index_airflow = find(...
218         ref(:,30)>0&SH(:,1)>1&SC_check(:,1)>3&SC_check_II(:,1)>3 ...
219     );
220     Evap.para.Evap_mea_airflow = mean(ref(index_airflow,30));
221 else
222     index_airflow = find(ref(:,30)>0);
223     Evap.para.Evap_mea_airflow = mean(ref(index_airflow,30));
224 end
225 Vdot_a_ori = ref(:,30);
226
227 % Heat Transfer characteristic Calculation
228 i = 1;
229 while i ≤ m
230     Taout(i,1) = ref(i,18);
231     if System.Heating == 0
232         airinlet(1,1:5) = calllib(...
233             libname, 'HumAir_DLL', input(i,6), Patm(i,1), 1, ...
234             ref(i,19), zeros(1,5) ...
```

```
235         );
236         airoutlet(1,1:5) = calllib(...
237             libname, 'HumAir_DLL', Taout(i,1), Patm(i,1), 1, ...
238             ref(i,20), zeros(1,5) ...
239         );
240     else
241         % set the dewpoint to be 230K as
242         % r = 1.e-8 may be out of range
243         airinlet(1,1:5) = calllib(...
244             libname, 'HumAir_DLL', input(i,6), Patm(i,1), 1, ...
245             230, zeros(1,5) ...
246         );
247         airoutlet(1,1:5) = calllib(...
248             libname, 'HumAir_DLL', ...
249             Taout(i,1), Patm(i,1), 1, 230, zeros(1,5) ...
250         );
251         ref(i,19) = airinlet(1,1);
252         ref(i,20) = airoutlet(1,1);
253     end
254     airinlet_stored(i,1:5) = airinlet(1,1:5);
255     airoutlet_stored(i,1:5) = airoutlet(1,1:5);
256     input(i,7) = airinlet(1,4); %relative humidity
257     SC_EXV_in(i,1) = propertyRH(...
258         'T','P',ref(i,5),'Q',0,refri ...
259     )-ref(i,13);
260     SC_cond_out(i,1) = propertyRH(...
```

```
261         'T','P',ref(i,4),'Q',0,refri ...
262     )-ref(i,12);
263     if System.Heating==0
264         Tamb = ref(i,21);
265     else
266         Tamb = ref(i,17);
267     end
268     if ¬((SC_cond_out(i,1) ≥ 3) ...
269          || (SC_EXV_in(i,1) ≥1 && SH_check(i,1) ≥ 1) ...
270          ||(SC_cond_out(i,1) ≥ 1)) || ref(i,23)<0
271         ref(i,23) = Comp_func(...
272             ref(i,1), propertyRH(...
273                 'H','T',ref(i,9),'P',ref(i,1),refri ...
274             ), ref(i,2), Tamb, ref(i,10), Comp_para, refri ...
275         );
276     end
277     if SC_cond_out(i,1) > 1 & ref(i,12) - ref(i,13) < 0.5
278         h_EXV_in = ...
279             propertyRH('H','T',ref(i,12),'P',ref(i,4),refri);
280     elseif SC_EXV_in(i,1) < 1 & SH_check(i,1) > 1
281         h_EXV_in = propertyRH(...
282             'H','T',ref(i,16),'P',ref(i,8),refri ...
283         )-Q_evap_r(i,1)/ref(i,23);
284     elseif SC_EXV_in(i,1) < 1
285         if SC_cond_out(i,1) > 1
286             h_cond_out = propertyRH(...
```

```
287                 'H','T',ref(i,12),'P',ref(i,4),refri ...
288             );
289         else
290             h_cond_in = propertyRH(...
291                 'H','T',ref(i,11),'P',ref(i,3),refri ...
292             );
293             h_cond_out = h_cond_in - Q_cond_r(i,1)/ref(i,23);
294         end
295         [dum1 h_EXV_in dum2] = LiquidLine_func(...
296             ref(i,23), ref(i,4), ...
297             hcond_out, Tamb, ...
298             LiquidLine_para, refri, 0, libname ...
299         );
300     else
301         h_EXV_in = propertyRH(...
302             'H','T',ref(i,13),'P',ref(i,5),refri ...
303         );
304     end
305     hin_evap(i,1) = h_EXV_in;
306     hv(i,1) = propertyRH('H','P',ref(i,8),'Q',1,refri);
307     if SH_check(i,1) > 1
308         hout_evap(i,1) = propertyRH(...
309             'H','T',ref(i,16),'P',ref(i,8),refri ...
310         );
311     else
312         hout_evap(i,1) = h_EXV_in + Q_evap_r(i,1)/ref(i,23);
```

```
313     end
314     Q_evap(i,1) = Q_evap_r(i,1);
315     input(i,8) = ref(i,30);
316     ref(i,27) = Q_evap(i,1);
317     input(i,8) = ref(i,30);
318     cpa = 1006 + airinlet(1,2)*1860;
319     cs = calllib(libname, 'Matlab_cair_sat', input(i,13), 0)*1000;
320     cscp(i,1) = cs/cpa;
321     i = i + 1;
322 end
323 % end
324 % calculate mean airflow
325 if Air_side==0
326     if Standard_Airflow_evap
327         airflow = ref(index_airflow,30)./...
328             airinlet_stored(index_airflow,5)./1.2;
329     else
330         airflow = ref(index_airflow,30)./ ...
331             airinlet_stored(index_airflow,5).* ...
332             airoutlet_stored(index_airflow,5);
333     end
334 else
335     if Standard_Airflow_evap
336         airflow = ref(index_airflow,30)./ ...
337             airinlet_stored(index_airflow,5)./1.2;
338     else
```

```
339         airflow = ref(index_airflow,30)./ ...
340             airinlet_stored(index_airflow,5).* ...
341             airoutlet_stored(index_airflow,5);
342     end
343 end
344 Evap.para.Evap_ref_airflow = mean(airflow);
345 input(1:m,10) = ref(1:m,23);
346 input(1:m,8) = ref(1:m,30);
347
348 cp_l = propertyRH('C', 'P', Psat(1),'Q', 0, refri);
349 miu_l = propertyRH('V', 'P', Psat(1),'Q', 0, refri);
350 rho_l = propertyRH('D','P', Psat(1), 'Q', 0,refri);
351 k_l = propertyRH('L', 'P', Psat(1),'Q', 0, refri);
352 Ktp_rated = 1/((input(1,10)/miu_l/rho_l)^0.8* ...
353     (miu_l*cp_l/k_l*rho_l)^0.4*k_l);
354
355 cp_v = propertyRH('C', 'P', Psat(1),'Q', 1, refri);
356 miu_v = propertyRH('V', 'P', Psat(1),'Q', 1, refri);
357 k_v = propertyRH('L', 'P',Psat(1),'Q', 1, refri);
358 Ksh_rated = (cp_v*miu_v/k_v)^0.4*(input(1,10)/miu_v)^0.8;
359
360 % Intial guess and solving rated condition
361 hL_max = 0.023*(input(1,10)/(pi*D^2/4)*D/miu_l)^.8* ...
362     (miu_l*cp_l/k_l)^.4*k_l/D;
363 hV_max = 0.023*(input(1,10)/(pi*D^2/4)*D/miu_v)^.8* ...
364     (miu_v*cp_v/k_v)^.4*k_v/D;
```

```
365 U(1,1) = fzero(@(x) h_bartp(...
366     x/Ktp_rated*Ar, 0, 1, input(1,10), Q_evap(1,1), ...
367     input(1,13), refri ...
368 ) - hL_max*Ar, 1);
369 Utpmax = fzero(@(x) h_bartp(...
370     x/Ktp_rated*Ar, 0, 1, input(1,10), Q_evap(1,1), ...
371     input(1,13), refri ...
372 ) - hL_max*100*Ar, 1);
373 U(2,1) = hV_max/4;
374
375 U(3:5,1) = [Utpmax/2 0.4 1.e-3]';
376 U(1,1) = max(U(1), fzero(@(x) h_bartp(...
377     x/Ktp_rated*Ar, input(1,11), 1, input(1,10), ref(1,18), ...
378     input(1,13), refri ...
379 ) - U(3)*Ar, U(1)));
380
381 ma_rated = ref(1,30)*1.2;
382 U_guess = U;
383 airinlet = zeros(1,5);
384 airoutlet = zeros(1,5);
385 T_ac = input(1,6);
386 % recompute for enthalpy at the inlet of two-phase region
387 airinlet = calllib(...
388     libname, 'HumAir_DLL', T_ac(1,1), Patm(1,1), 1, ...
389     ref(1,19), airinlet ...
390 );
```

```
391 airoutlet = calllib(...
392     libname, 'HumAir_DLL', Taout(1,1), Patm(1,1), 1, ...
393     ref(1,20), airoutlet ...
394 );
395 ma = (1)/airinlet(1,5)*input(1,8); % same formula as one in ACHP
396 cpa(1,1) = 1006 + airinlet(1,2)*1860;
397
398 for i = 1:m
399     % recompute for enthalpy at the inlet of two-phase region
400     airinlet = calllib(...
401         libname, 'HumAir_DLL', input(i,6), Patm(i,1), 1, ...
402         ref(i,19), airinlet ...
403     );
404     ma(i,1) = (1)/airinlet(5)*input(i,8);
405     airoutlet = calllib(...
406         libname, 'HumAir_DLL', Taout(i,1), Patm(i,1), 1, ...
407         ref(i,20), airoutlet ...
408     );
409     SH_ori(i,1) = SH(i,1);
410     if SH_ori(i,1) < 0
411         SH_ori(i,1) = 0;
412     end
413     ref(i,37) = SH_ori(i,1);
414 end
415
416 Taoutbar = mean(ref(i,15)); % mean of air outlet temperature
```

```
417 stat(2) = 0;
418 for i = 1:m
419     stat(2) = stat(2) + (ref(i,15)-Taoutbar)^2;
420 end
421 stat(1) = sum((Q_evap-mean(Q_evap)).^2);
422
423 SHR = zeros(1,m);
424 for i = 1:m
425     airinlet(1,1:5) = calllib(...
426         libname, 'HumAir_DLL', input(i,6), Patm(i,1), 1, ...
427         ref(i,19), airinlet(1,1:5) ...
428     );
429     airoutlet(1,1:5) = calllib(...
430         libname, 'HumAir_DLL', ref(i,18), Patm(i,1), 1, ...
431         ref(i,20), airoutlet(1,1:5) ...
432     );
433     airinlet_stored(i,1:5) = airinlet(1,1:5);
434     airoutlet_stored(i,1:5) = airoutlet(1,1:5);
435     cpm = 1006 + 1860*airinlet(2);
436     if System.Heating == 0
437         % use calculated values as the original airflow
438         % is wrong, esp. in SCE data
439         if Ind_Fan_Upstream_evap
440             SHR(i) = (ref(i,30)*(1)/airinlet(1,5)*cpm*(...
441                 input(i,6)-ref(i,18) ...
442             )+ref(i,25))/(ref(i,30)*(1)/airinlet(1,5)*(...
```

```
443                 airinlet(1,3) - airoutlet(1,3) ...
444             )*1000 + ref(i,25));
445         else
446             SHR(i) = ref(i,30)*(1)/airinlet(1,5)*cpm*(...
447                 input(i,6)-ref(i,18) ...
448             )/(ref(i,30)*(1)/airinlet(1,5)*( ...
449                 airinlet(1,3) - airoutlet(1,3) ...
450             )*1000);
451         end
452     else
453         SHR(i) = Inf;
454     end
455     SHR_r(i) = cpm*(input(i,6)-ref(i,18))/Q_evap(i,1)*(1)/...
456         airinlet(1,5)*input(i,8);
457     ref(i,36) = SHR(i);
458 end
459
460 options = optimset(...
461     'Display','iter-detailed','Algorithm','sqp',...
462     'MaxFunEvals',1000,'MaxIter',200,'TolX',1.e-9,'TolFun',1.e-9 ...
463 );
464 % since UA estimated by decoupling is usually the overall UA
465 % rather than the air-side UA, it is usually underestimated. Using a
466 % refrigerant-side U which is quite small can help to shift it back to
467 % the air-side UA only
468 U(1,1) = Utpmax/5;
```

```
469 U(2,1) = hV_max/10;
470 UtpA = h_bartp(...
471     Utpmax/Ktp_rated*Ar, input(1,11), 1, input(1,10), ...
472     ref(1,18), input(1,13), refri ...
473 );
474
475 error = 1;
476 iter = 0;
477 iter_max = 3;
478 U(3:5,1) = [UtpA/Ar/10 0.4 0.01]';
479
480 U_ori = U;
481 U_norm = ones(length(U),1);
482 weigh_Q = weighing_function_v000(Q_evap, bin_num); % on Q
483 weigh_SHR = weighing_function_v000(ref(:,36), bin_num);
484 g = coeff(
485     U_norm, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
486     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, ...
487     refri, libname, weigh_Q, weigh_SHR ...
488 );
489 % make sure that the initial values are not nan
490 while isnan(g) & iter < iter_max
491     U_norm = U_norm.*(rand(length(U),1)+0.5);
492     g = coeff(...
493         U_norm, input, hin_evap, hout_evap, Dom_Valve, ref, ...
494         output, stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, ...
```

```
495         U_ori, refri, libname, weigh_Q, weigh_SHR ...
496     );
497     save(savefilename);
498     iter = iter + 1;
499 end
500 save(savefilename);
501 options = optimset(...
502     'Display','iter-detailed','Algorithm','sqp',...
503     'MaxFunEvals',2000,'MaxIter',300,'TolX',5.e-8,...
504     'TolFun',5.e-8 ...
505 );
506 [U, fval, exitflag] = fmincon(@(U) coeff(...
507     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, stat, ...
508     Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, refri, ...
509     libname, weigh_Q, weigh_SHR ...
510 ), U_norm,[],[],[],[],[1.e-10 1.e-10 1.e-10 0.4 1.e-10],[...
511     Utpmax/U_ori(1) hV_max/U_ori(2) Inf 0.84 Inf ...
512 ],@(U) constraint1(...
513     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
514     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, ...
515     refri, libname, weigh_Q, weigh_SHR ...
516 ),options);
517 % problem in calculation, recalculate
518 if (exitflag < 1 || fval > 0.5) && iter < iter_max
519     try
520         %increase funevals to avoid getting out before completion
```

```
521         options = optimset(...
522             'Display','iter-detailed',...
523             'Algorithm','interior-point',...
524             'MaxFunEvals',1000,'MaxIter',200,'TolX',5.e-6,...
525             'TolFun',5.e-6 ...
526         );
527         [U2,fval2,exitflag2] = fmincon(@(U) coeff(...
528             U, input, hin_evap, hout_evap, Dom_Valve, ref, ...
529             output, stat, Ar, Ktp_rated, Ksh_rated, ma_rated, ...
530             cscp, U_ori, refri, libname, weigh_Q, weigh_SHR ...
531         ), U_norm,[],[],[],[],[...
532             1.e-10 1.e-10 1.e-10 0.4 1.e-10 ...
533         ],[Utpmax/U_ori(1) hV_max/U_ori(2) Inf 0.84 Inf],...
534             @(U) constraint1(...
535                 U, input, hin_evap, hout_evap, Dom_Valve, ...
536                 ref, output, stat, Ar, Ktp_rated, Ksh_rated, ...
537                 ma_rated, cscp, U_ori, ...
538                 refri, libname, weigh_Q, weigh_SHR ...
539             ),options);
540         if fval2 < fval & ¬isnan(fval2)
541             U = U2;
542             fval = fval2;
543         end
544     end
545     iter = iter + 1;
546 end
```

```
547 [input output cost g h r r_II] = Evaporator(...
548     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
549     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, ...
550     refri, libname, weigh_Q, weigh_SHR ...
551 );
552
553
554 % computing coefficients of determination
555 % SHR calculation
556 HumAir = zeros(1,5);
557 airoutlet = zeros(1,5);
558 costSHR = 0;
559 SHR_m = zeros(1,m);
560 ma = [];
561 cpm = [];
562 SHR_m = [];
563 for i = 1:m
564     HumAir = calllib(...
565         libname, 'HumAir_DLL', input(i,6), Patm(i,1), 4, ...
566         input(i,7), airinlet ...
567     );
568     cpm(i,1) = 1006 + 1860*HumAir(2);
569     ma(i,1) = HumAir(5)*input(i,8);
570     SHR_m(i,1) = ...
571         ma(i,1)*cpm(i,1)*(input(i,6)-output(i,8))/output(i,1);
572     costSHR = cost + (SHR(i)-SHR_m(i))^2;
```

```
573 end
574
575 % calculate the covriance matrix
576 X_Cov = [];
577 X_Cov_SHR = [];
578 for i = 1:length(U)
579     U_sp = U;
580     if abs(U(i)) > 1
581         U_sp(i) = U_sp(i)*(1+1.e-8);
582     else
583         U_sp(i) = U(i)+1.e-8;
584     end
585     [dum1 output_sp dum2 dum3 dum4 dum5 dum6] = Evaporator(...
586         U_sp, input, hin_evap, hout_evap, Dom_Valve, ref, ...
587         output, stat, Ar, Ktp_rated, Ksh_rated, ma_rated, ...
588         cscp, U_ori, refri, libname, weigh_Q, weigh_SHR ...
589     );
590     SHR_m_sp = ma(:,1).*cpm(:,1).*(input(:,6)-output_sp(:,8))./ ...
591         output_sp(:,1);
592     X_Cov(:,i) = (output_sp(:,1) - output(:,1))./(...
593         U_sp(i)-U(i) ...
594     )/U_ori(i);
595     X_Cov_SHR(:,i) = (SHR_m_sp - SHR_m)./(U_sp(i)-U(i))/U_ori(i);
596 end
597 Evap.para.Cov = covariance_adj(X_Cov);
598 Evap.para.X_limit = max(diag(X_Cov*Evap.para.Cov*X_Cov'));
```

```
599 Evap.para.Cov_SHR = covariance_adj(X_Cov_SHR);
600 Evap.para.X_limit_SHR = max(diag(
601     X_Cov_SHR*Evap.para.Cov_SHR*X_Cov_SHR' ...
602 ));
603
604 constraints = constraint1(...
605     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, stat, ...
606     Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, refri, ...
607     libname, weigh_Q, weigh_SHR ...
608 );
609
610 U = U.*U_ori;
611
612 % statistical information
613 % (for creating stat variable only in this case)
614 % only count the ones with postivei subcooling
615 Qbar = 0; %mean of heat transfer rate
616 num = 0;
617 for i = 1:m
618     Qbar = Q_evap(i,1) + Qbar;
619     num = num + 1;
620 end
621 Qbar = Qbar/num;
622 stat(1) = 0;
623 stat_num = 0;
624 j = 1;
```

```
625 for i = 1:m
626 %     if SC(i,1) > 3
627         stat(1) = stat(1) + (Q_evap(i,1)-Qbar)^2;
628         stat_num = stat_num + (output(i,1)-Q_evap(i,1))^2;
629         Q_mea_plot(j,1) = Q_evap(i,1);
630         Q_pre_plot(j,1) = output(i,1);
631         Δh_est_final(i,1) = output(i,1)/ref(i,23);
632         Δh_ori(i,1) = hout_evap(i,1) - hin_evap(i,1);
633         SH_est(i,1) = ...
634             propertyRH(...
635                 'T','P',ref(i,8), ...
636                 'H',output(i,1)/ref(i,23)+hin_evap(i,1),refri ...
637             )-propertyRH('T','P',ref(i,8),'Q',1,refri);
638         SH_ori(i,1) = SH(i,1);
639         if SH(i,1) < 0
640             SH_ori(i,1) = 0;
641         end
642         SHR_mea_plot(j,1) = SHR(i);
643         SHR_pre_plot(j,1) = SHR_m(i);
644         j = j + 1;
645 %     end
646 end
647 coeffQ = 1 - stat_num/stat(1);
648
649 SHRbar = 0; %mean of heat transfer rate
650 num = 0;
```

```
651 for i = 1:m
652     if SC(i,1) > 3
653         SHRbar = SHR(i) + SHRbar;
654         num = num + 1;
655     end
656 end
657 SHRbar = SHRbar/num;
658
659 stat(3) = 0;
660 stat_num = 0;
661 for i = 1:m
662     if SC(i,1) > 3
663         stat(3) = stat(3) + (SHR(i)-SHRbar)^2;
664         stat_num = stat_num + (SHR_m(i) - SHR(i))^2;
665     end
666 end
667 coeffSHR = 1 - stat_num/stat(3);
668
669 % Inner suface area of evaproator [m^2]
670 Evap.para.A_r = Ar;
671 % Rated heat transfer coefficient on the air-side [W/m^2-K]
672 Evap.para.U_a = U(3);
673 % Adjustment index for air-side HTC based on airflow% [-]
674 Evap.para.n = U(4);
675 % Adjusment for air-side HTC based on fin geometry% [-]
676 Evap.para.p = U(5);
```

```
677 Evap.para.ma = ma_rated; % rated airflow [kg/s]
678 % Rated heat transfer coefficient on the ref-side
679 % for two-phase flow [W/m^2-K]
680 Evap.para.Utp = U(1)/Ktp_rated;
681 % Rated heat transfer coefficient on the ref-side
682 % for superheated flow [W/m^2-K]
683 Evap.para.Ush = U(2);
684 Evap.para.Ksh = Ksh_rated; % Rated property effect on superheated HTC
685 Evap.para.Din = D; % Inner diameter [m]
686 Evap.para.Lt = Ltube*Ntube; % Total length of heat exchagner pipe [m]
687 % Average fan power consumption [W]
688 Evap.para.FanPower = mean(ref(:,25));
689 Evap.para.ΔP = 0.0; % Rated pressure drop across evaporator [kPa]
690 % Adjustment index for refrigerant pressure drop based on flow [-]
691 Evap.para.Pn = 0.0;
692 Evap.para.mdot_r = 0.0; % Rated refrigernat mass flow rate [kg/s]
693 Evap.para.ΔP_const = 0.0; % Average refrigerant pressure drop [kPa]
694
695 save(savefilename);
696 movefile(savefilename,strcat(foldername,'\'));
697
698 %% Fan power coefficient calculation
699 FanPower_mea = ref(:,25);
700 if Evap.para.FanPower > 1.e-8
701     weigh_fan = weighing_function_v000(FanPower_mea, bin_num);
702     Vdot_air = input(:,8);
```

```
703     C_fan = (...
704         [1./weigh_fan Vdot_air./weigh_fan] ...
705     )\(FanPower_mea./weigh_fan);
706     Evap.para.C_fan = C_fan;
707     FanPower_est = [ones(m,1) Vdot_air]*Evap.para.C_fan;
708 else
709     Evap.para.C_fan = [Evap.para.FanPower 0]';
710     FanPower_est = ones(m,1)*Evap.para.FanPower;
711 end
712 
713 V_rated = mean(input(find(strcmp(fault,'NF')),8));
714 W_rated = mean(ref(find(strcmp(fault,'NF')),25));
715 Evap.para.C_fan = (...
716     [-0.3624 1.9553/V_rated -0.5914/V_rated^2]' ...
717 )*W_rated;
718 Evap.para.W_rated = W_rated;
719 Evap.para.V_rated = V_rated;
720 
721 
722 %% Pressure Drop calculation
723 clear ΔP_av mdot SH SC rho rho_out rho_tp rho_l rho_v ...
724     x_in w_superheat w_twophase w_subcool mu_l mu_v mdot_rated ...
725     SH_rated SC_rated rho_rated rho_out_rated rho_tp_rated ...
726     mu_l_rated mu_v_rated SH_limit SC_limit ΔP;
727 for j = 1:m
728     SC_LL_in(j,1) = propertyRH(...
```

```
729         'T','P',ref(j,4),'Q',0,refri ...
730     ) - ref(j,12);
731     SC_LL_out(j,1) = propertyRH(...
732         'T','P',ref(j,5),'Q',0,refri ...
733     ) - ref(j,13);
734     SH_evap_out(j,1) = ref(j,16)-propertyRH(...
735         'T','P',ref(j,8),'Q',1,refri ...
736     );
737 end
738 index_gd_ref = find(SC_LL_in>3&SC_LL_out>3&SH_evap_out>1);
739
740 j = 1;
741 i = 1;
742 while j ≤ m
743     airinlet(1,1:5) = calllib(...
744         'lib', 'HumAir_DLL', ref(j,17), ref(j,32), ...
745         1, ref(j,19), zeros(1,5) ...
746     );
747     airoutlet(1,1:5) = calllib(...
748         'lib', 'HumAir_DLL', ref(j,18), ref(j,32), ...
749         1, ref(j,20), zeros(1,5) ...
750     );
751     mdot_a(i,1) = ref(j,30)/airinlet(1,5);
752     Q(i,1) = Q_evap(i,1);
753     SC(i,1) = SC_LL_out(j,1);
754     SH(i,1) = SH_evap_out(j,1);
```

```
755     mdot(i,1) = ref(j,23);
756     ΔP(i,1) = ref(j,7) - ref(j,8);
757     hin(i,1) = hin_evap(j,1);
758     hout(i,1) = hout_evap(i,1);
759     rho_l(i,1) = propertyRH('D','P',ref(j,8),'Q',0,refri);
760     rho_v(i,1) = propertyRH('D','P',ref(j,8),'Q',1,refri);
761     rho_out(i,1) = propertyRH(...
762         'D','P',ref(j,8),'H',hout(i,1),refri ...
763     );
764     mu_l(i,1) = propertyRH('V','P',ref(j,8),'Q',0,refri);
765     mu_v(i,1) = propertyRH('V','P',ref(j,8),'Q',1,refri);
766     hl = propertyRH('H','P',ref(j,8),'Q',0,refri);
767     hv = propertyRH('H','P',ref(j,8),'Q',1,refri);
768     hl_out = propertyRH('H','P',ref(j,8),'Q',0,refri);
769     hv_out = propertyRH('H','P',ref(j,8),'Q',1,refri);
770     x_in(i,1) = (hin(i,1)-hl)/(hv-hl);
771     x_out(i,1) = (hout(i,1)-hl_out)/(hv_out-hl_out);
772     mu = (rho_l(i,1)/rho_v(i,1))^(1/3)*(rho_v(i,1)/rho_l(i,1));
773     alpha = 1/(1+(1-x_in(i,1))/x_in(i,1)*mu);
774     rho(i,1) = alpha*rho_v(i,1) + (1-alpha)*rho_l(i,1);
775     gamma = -(mu*(log(((x_out(i,1)-1)*mu-x_out(i,1))/(...
776         (x_in(i,1)-1)*mu-x_in(i,1) ...
777     ))+x_out(i,1)-x_in(i,1))-x_out(i,1)+x_in(i,1))/(...
778         mu*mu-2*mu+1 ...
779     )/(x_out(i,1)-x_in(i,1));
780     rho_tp(i,1) = (gamma*rho_v(i,1)+(1-gamma)*rho_l(i,1));
```

```
781     Pin = ref(j,7);
782     Pout_exp = ref(j,8);
783     Tain = ref(j,17);
784     Dain = ref(j,19);
785     airinlet = zeros(1,5);
786     airoutlet = zeros(1,5);
787     Pain = ref(j,32);
788     airinlet = calllib(...
789         libname, 'HumAir_DLL', Tain, Pain, 1, Dain, airinlet ...
790     );
791     airoutlet = calllib(...
792         libname, 'HumAir_DLL', ref(j,18), Pain, ...
793         1, ref(j,20), airoutlet ...
794     );
795     wain = airinlet(1,2);
796     Vdot = ref(j,30);
797     [Pout hout Taout waout Qevap Charge ...
798         FanPower EvapOutput ma EvapInput] = Evaporator_after(...
799             Pin, Pout_exp, hin(i,1), hout(i,1), mdot(i,1), ...
800             Tain, wain, Pain, Vdot, Evap.para, refri, libname ...
801         );
802     w_superheat(i,1) = EvapOutput(4);
803     w_twophase(i,1) = EvapOutput(5);
804     w_subcool(i,1) = 0;
805     i = i + 1;
806     j = j + 1;
```

```
807 end
808 mdot_rated = mdot(1,1);
809 SH_rated = mean(SH);
810 Q_rated = mean(Q);
811 ΔP_av = mean(ΔP);
812 SC_rated = mean(SC);
813 rho_rated = mean(rho);
814 rho_out_rated = mean(rho_out);
815 rho_tp_rated = mean(rho_tp);
816 mu_l_rated = mean(mu_l);
817 mu_v_rated = mean(mu_v);
818 SH_limit = 0.5;
819 SC_limit = 3;
820
821 Af = -eye(4);
822 bf = zeros(4,1);
823 Af(4,4) = 0;
824 C = [20 20 20 10];
825 options = optimset(...
826     'MaxFunEvals',20000,'algorithm','active-set','MaxIter',5000,...
827     'Display','iter-detailed' ...
828 );
829 if length(ΔP) > 1 & sum(ΔP.^2)>1.e-8
830     weigh = weighing_function_v000(ΔP, bin_num);
831 elseif sum(ΔP.^2)≤1.e-8
832     weigh = ones(length(ΔP),1);
```

```
833 else
834     weigh = 1;
835 end
836 if sum(ΔP.^2/length(ΔP)) > 1.e-1 && max(ΔP) > 0
837     C = fmincon(@(C) PressureDrop_residual(...
838         C, [], ΔP_av, mdot, SH, SC, rho, rho_out, ...
839         rho_tp, rho_l, rho_v, x_in, w_superheat, w_twophase, ...
840         w_subcool, mu_l, mu_v, mdot_rated, SH_rated, SC_rated, ...
841         rho_rated, rho_out_rated, rho_tp_rated, mu_l_rated, ...
842         mu_v_rated, SH_limit, SC_limit, ΔP, weigh ...
843     ), C, Af, bf, [], [], [0 0 0 -10],[Inf Inf Inf 10],[],options);
844     ΔP_est = PressureDrop(...
845         C, [], ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, ...
846         rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
847         mu_l, mu_v, mdot_rated, SH_rated, SC_rated, rho_rated, ...
848         rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated ...
849     );
850     dev = ΔP_est - ΔP;
851     % calculate the covriance matrix
852     X_Cov_ΔP = [];
853     for i = 1:length(C)
854         C_sp = C;
855         if abs(C(i))>1
856             C_sp(i) = C_sp(i)*(1+1.e-5);
857         else
858             C_sp(i) = C_sp(i)+1.e-5;
```

```
859         end
860         ΔP_est_sp = PressureDrop(...
861             C_sp, [], ΔP_av, mdot, SH, SC, rho, rho_out, ...
862             rho_tp, rho_l, rho_v, x_in, w_superheat, w_twophase, ...
863             w_subcool, mu_l, mu_v, mdot_rated, SH_rated, ...
864             SC_rated, rho_rated, rho_out_rated, rho_tp_rated, ...
865             mu_l_rated, mu_v_rated ...
866         );
867         X_Cov_ΔP(:,i) = (...
868             ΔP_est_sp - ΔP_est ...
869         )./(C_sp(i)-C(i));
870     end
871     Evap.para.Cov_ΔP = covariance_adj(X_Cov_ΔP);
872     Evap.para.X_limit_ΔP = max(diag(...
873         X_Cov_ΔP/(X_Cov_ΔP'*X_Cov_ΔP)*X_Cov_ΔP' ...
874     ));
875 else
876     C = [0 0 0 0];
877     ΔP_est = PressureDrop(...
878         C, [], ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, ...
879         rho_l, rho_v, x_in, w_superheat, w_twophase, w_subcool, ...
880         mu_l, mu_v, mdot_rated, SH_rated, SC_rated, rho_rated, ...
881         rho_out_rated, rho_tp_rated, mu_l_rated, mu_v_rated ...
882     );
883     dev = ΔP_est - ΔP;
884     % calculate the covriance matrix
```

```
885     Evap.para.Cov_ΔP = zeros(length(C),length(C));
886     Evap.para.X_limit_ΔP = Inf;
887 end
888 max_dev = round(max(abs(dev)));
889
890 coeffΔP = 1 - sum(dev.^2)/sum((ΔP - mean(ΔP)).^2);
891
892 Evap.para.ΔP.C = C;
893 Evap.para.ΔP.mdot_rated = mdot_rated;
894 Evap.para.ΔP.rho_rated = rho_rated;
895 Evap.para.ΔP.rho_out_rated = rho_out_rated;
896 Evap.para.ΔP.mu_v_rated = mu_v_rated;
897 Evap.para.ΔP_max = max(ref(:,7)-ref(:,8));
898
899 save(savefilename);
900 movefile(savefilename,strcat(foldername,'\'));
901
902 end
903
904 function result = coeff(...
905     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
906     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, refri, ...
907     string, weigh_Q, weigh_SHR ...
908 )
909
910 % This function calculates the result of cost function
```

```
911
912 % Variables
913 % Inputs:
914 % UA: UAs to be minimized
915 % input: input matrix
916 % ref: reference matrix
917 % output: output matrix
918 % stat: statistical information
919 % string: name of library to be passed in
920 % Outputs:
921 % r: an array of function results at different points
922
923 % computing new outputs
924 [input output f g h r r_II] = Evaporator(...
925     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
926     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, ...
927     refri, string, weigh_Q, weigh_SHR ...
928 );
929 result = norm(r)+norm(r_II);
930
931 end
932
933 function [input output f g h r r_II] = Evaporator(...
934     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, stat, ...
935     Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, refri, ...
936     string, weigh_Q, weigh_SHR ...
```

```
)
f = 0;
g = 0;
h = 0;

%     UA
cp_l = propertyRH('C', 'T', input(1,13),'Q', 0, refri);
miu_l = propertyRH('V', 'T', input(1,13),'Q', 0, refri);
rho_l = propertyRH('D','T', input(1,13), 'Q', 0,refri);
k_l = propertyRH('L', 'T', input(1,13),'Q', 0, refri);
[m, n] = size(ref);

if ¬isempty(findstr(refri,'.fld'))
    index_end = findstr(refri,'.fld');
    refname = refri(1:index_end-1);
else
    refname = refri;
end

U = U.*U_ori;

cp_v = propertyRH('C','T',26.7+273.15,'Q',1,refri);
for i = 1:m
    input_II = input(i,:);
    output_II = output(i,:);
    airinlet = zeros(1,5);
```

```
963     airinlet = calllib(...
964         string, 'HumAir_DLL', input_II(6), input_II(9), 4, ...
965         input_II(7), airinlet ...
966     );
967     airoutlet = calllib(...
968         string, 'HumAir_DLL', ref(i,18), input_II(9), 1, ...
969         ref(i,20), airinlet ...
970     );
971     hout_a = airoutlet(3)*1000;
972     ma = (1)/airinlet(5)*input_II(8);
973     % adjusted to inlet condition
974     input_II(8) = input_II(8)/airinlet(5)*airinlet(5);
975     input_II(1) = 1;
976     input_II(2) = U(3)*(ma/ma_rated)^U(4)*tanh(...
977         U(5)*(U(3)*(ma/ma_rated)^U(4))^.5 ...
978     )/U(5)/(U(3)*(ma/ma_rated)^U(4))^.5*Ar;
979
980     input_II(3) = U(3)*(ma/ma_rated)^U(4)*tanh(...
981         (cscp(i,1)*U(3)*(ma/ma_rated)^U(4))^0.5*U(5) ...
982     )/((cscp(i,1)*U(3)*(ma/ma_rated)^U(4))^0.5*U(5))*Ar;
983     % no cpa as denominator as the DLL contains the cpa term
984
985     %UAtp
986     Pevap = ref(i,8);
987     hv = propertyRH('H', 'P', Pevap,'Q', 1,refri);
988     hf = propertyRH('H', 'P', Pevap,'Q', 0,refri);
```

```
989
990     hin = hin_evap(i,1);
991 %     hout = hin + ref(i,27)/input_II(10);
992     hout = hout_evap(i,1);
993     if(hout≥hv)
994         xout = 1;
995     else
996         xout = (hout-hf)/(hv-hf);
997     end
998     input_II(11) = (hin - hf)/(hv - hf);
999     if hout_evap(i,1) > hv
1000        input_II(5) =  h_bartp(...
1001            U(1)/Ktp_rated*Ar, input_II(11), xout, input_II(10), ...
1002            input_II(10)*(hv-hin), input_II(13), refri ...
1003        );
1004    else
1005        input_II(5) =  h_bartp(...
1006            U(1)/Ktp_rated*Ar, input_II(11), xout, input_II(10), ...
1007            input_II(10)*(hout_evap(i,1)-hin), input_II(13), refri ...
1008        );
1009    end
1010
1011    %UAsh
1012    cp_v = propertyRH('C', 'P', Pevap,'Q', 1, refri);
1013    miu_v = propertyRH('V', 'P', Pevap,'Q', 1, refri);
1014    k_v = propertyRH('L', 'P', Pevap,'Q', 1, refri);
```

```
1015      Ksh = (cp_v*miu_v/k_v)^0.4*(input_II(10)/miu_v)^0.8;
1016      if(i==1)
1017         input_II(4) = U(2)*Ksh/Ksh_rated*Ar;
1018      else
1019         input_II(4) = U(2)*Ksh/Ksh_rated*Ar;
1020      end
1021
1022      input_II = real(input_II);
1023      [input_II(1,1:15) output_II(1,1:14)] = calllib(...
1024          string, 'Matlab_Evaporator_Forward_Charge_ii', ...
1025          input_II(1,1:15), output_II(1,1:14), refname ...
1026      );
1027
1028      cpm = 1006 + 1860*airinlet(2);
1029      SHR_m(i) = ma*cpm*(input_II(1,6)-output_II(1,8))/output_II(1,1);
1030      hout_a_est = airinlet(3)*1000 - output_II(1,1)/ma;
1031
1032      % no refrigerant-side data comparison if superheat is small
1033      T_sat_out = propertyRH('T','P',ref(i,8),'Q',1,refri);
1034      SH = ref(i,16) - T_sat_out;
1035      hout_est = hin + output_II(1)/input_II(10);
1036      r(i) = 1/sqrt(weigh_Q(i,1))*(output_II(1)-ref(i,27))/ref(i,27);
1037
1038      if isinf(ref(i,36))
1039          r_II(i) = 0; % no shr minimization is needed
1040      else
```

```
1041          r_II(i) = 1/sqrt(weigh_Q(i,1))*(...
1042              ref(i,36) - SHR_m(i) ...
1043          )/(ref(i,36));
1044      end
1045      input(i,:) = input_II;
1046      output(i,:) = output_II;
1047  end
1048  f = g/stat(1);
1049
1050  end
1051
1052  function UA_bar = h_bartp(UA_corr, xin, xout, mdot_r, Q, T, refri)
1053
1054  % The function is used to calculate an average UA on the
1055  % refrigerant side
1056  % UA_corr: corrected parameter
1057  % xin: inlet quality
1058  % mdot_r: refrigerant flow rate (kg/s)
1059  % Q: heat transfer rate (kW)
1060  % T: Saturation temperature (K)
1061
1062  % defining intervals
1063  n = 201;
1064  Δx = (xout-xin)/(n-1);
1065  x = xin:Δx:xout;
1066
```

```
1067 % defining properties
1068 % enthalpy of vaporization in J/kg
1069 h_fg = propertyRH('H', 'T', T,'Q', 1, refri) - ...
1070     propertyRH('H', 'T', T,'Q', 0, refri);
1071 % liquid density
1072 rhol = propertyRH('D', 'T', T,'Q', 0, refri);
1073 % vapor density
1074 rhov = propertyRH('D', 'T', T,'Q', 1, refri);
1075 % liquid dynamic viscosity
1076 nu = propertyRH('V', 'T', T,'Q', 0, refri);
1077 % liquid thermal conductivity
1078 k = propertyRH('L', 'T', T,'Q', 0, refri);
1079 % liquid spcific heat capacity
1080 cp = propertyRH('C', 'T', T,'Q', 0, refri);
1081 % liquid dynamic viscosity
1082 nu_v = propertyRH('V', 'T', T,'Q', 1, refri);
1083 % liquid thermal conductivity
1084 k_v = propertyRH('L', 'T', T,'Q', 1, refri);
1085 % liquid spcific heat capacity
1086 cp_v = propertyRH('C', 'T', T,'Q', 1, refri);
1087
1088 % as defined in Shah (1982)
1089
1090 % calculating parameters
1091 Bo = Q/mdot_r/h_fg; % Boiling number
1092 if(Bo≥11*10^-4)
```

```
1093      F = 14.7;
1094  else
1095      F = 15.43;
1096  end
1097
1098  %integration
1099  sum_htp = 0;
1100  bulk = UA_corr*(mdot_r)^.8*(cp/rhol/nu)^.4*k^.6;
1101  phi_nb1 = 230*sqrt(Bo);
1102  phi_nb2 = 1 + 46*sqrt(Bo);
1103  phi_bs_f = F*sqrt(Bo);
1104  Bo_limit = .3*10^-4;
1105  rho_ratio_sq = sqrt(rhov/rhol);
1106  x_diff_8 = (1-x).^.8;
1107  Co = rho_ratio_sq*(x_diff_8./x.^.8);
1108  phi_cb = (1.8./Co.^.8)';
1109  phi_nb = zeros(n,1);
1110  phi_bs = zeros(n,1);
1111  phi_bs_1 = phi_bs_f.*exp(2.74./Co.^.1);
1112  phi_bs_2 = phi_bs_f.*exp(2.74./Co.^.15);
1113  for i = 1:n
1114      if(x(1,i)<1)
1115          if(Co(1,i)>1)
1116              if(Bo>Bo_limit)
1117                  phi_nb(i,1) = phi_nb1;
1118              else
```

```
1119                 phi_nb(i,1) = phi_nb2;
1120             end
1121         elseif (Co(1,i)>0.1)
1122             phi_bs(i,1) = phi_bs_1(1,i);
1123         else
1124             phi_bs(i,1) = phi_bs_2(1,i);
1125         end
1126     end
1127 end
1128 htp = bulk.*max([phi_cb phi_nb phi_bs],[],2).*x_diff_8';
1129 if sum(x≥1)>0
1130     htp(x≥1) = UA_corr*(mdot_r)^.8*(cp_v/rhov/nu_v)^.4*k_v^.6;
1131 end
1132 htp([1,n]) = htp([1,n])/2;
1133 sum_htp = sum(htp)*Δx;
1134 UA_bar = sum_htp/(1-xin);
1135 end
1136
1137 function wsh = Superheat(...
1138     i, input, ref, Qsh, cpr, UA, Tevap, airinlet ...
1139 )
1140 % solve for the superheat section
1141 ma = (1)/airinlet(5)*input(i,8);
1142 wsh = -log(1-(ref(i,16)-Tevap)/(...
1143     input(i,6)-Tevap ...
1144 ))*input(i,10)*cpr(i,1)/ma/(...
```

```
1145      1006+1860*airinlet(2) ...
1146  )/(1-exp(-UA/ma/(1006+1860*airinlet(2))));
1147  end
1148
1149  function wtp = Twophase(...
1150      i, input, ref, Qtp, cpr, UA, Tevap, airinlet ...
1151  )
1152  % solve for the two-phase section
1153  ma = (1)/airinlet(5)*input(i,8);
1154  epsilontp = 1 - exp(-UA/(ma*(1006+1860*airinlet(2))));
1155  wtp = Qtp/ma/(1006+1860*airinlet(2))/epsilontp/(input(i,6)-Tevap);
1156  end
1157
1158  function wsh = Superheat_limit(...
1159      i, input, ref, Qsh, cpr, UA, Tevap, airinlet ...
1160  )
1161  % solve for the superheat section
1162  ma = (1)/airinlet(5)*input(i,8);
1163  wsh = -log(1-(ref(i,16)-Tevap)/(input(i,6)-Tevap))* ...
1164      input(i,10)*cpr(i,1)/ma/(1006+1860*airinlet(2))/(0.999);
1165  end
1166
1167  function wtp = Twophase_limit(...
1168      i, input, ref, Qtp, cpr, UA, Tevap, airinlet ...
1169  )
1170  % solve for the two-phase section
```

```
1171 ma = (1)/airinlet(5)*input(i,8);
1172 wtp = Qtp/ma/(1006+1860*airinlet(2))/0.999/(input(i,6)-Tevap);
1173 end
1174
1175 function [c,ceq] = constraint1(...
1176     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, ...
1177     stat, Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, ...
1178     refri, string, weigh_Q, weigh_SHR ...
1179 )
1180 % a constraint on the first data point: the rated one
1181 ceq = [];
1182 [input output f g h r r_II] = Evaporator(...
1183     U, input, hin_evap, hout_evap, Dom_Valve, ref, output, stat, ...
1184     Ar, Ktp_rated, Ksh_rated, ma_rated, cscp, U_ori, refri, ...
1185     string, weigh_Q, weigh_SHR ...
1186 );
1187 % avoid the cases without superheat
1188 specifier = find(output(:,4)==max(output(:,4)));
1189 if length(specifier) > 1
1190     specifier = specifier(1);
1191 end
1192 wsh = output(specifier,4);
1193 wtp = 1 - wsh;
1194 cpr = propertyRH('C','P',ref(specifier,8),'Q',1,refri);
1195 airinlet = calllib(...
1196     'lib', 'HumAir_DLL', ref(specifier,17), ...
```

```
1197      ref(specifier,32), 1, ...
1198      ref(specifier,19), zeros(1,5) ...
1199  );
1200  UAsh = (1/input(specifier,4)+1/input(specifier,2))^(-1);
1201  UAtp = (1/input(specifier,5)+1/input(specifier,2))^(-1);
1202  ma = (1)/airinlet(5)*input(specifier,8);
1203  epsilontp = 1 - exp(-UAtp/ma/(1006+1860*airinlet(2)));
1204  % assume counterflow
1205  Ntu_sh = wsh*UAsh/min(...
1206      input(specifier,10)*cpr,wsh*ma*(1006+1860*airinlet(2)) ...
1207  );
1208  Cr = input(specifier,10)*cpr/(wsh*ma*(1006+1860*airinlet(2)));
1209  if Cr > 1
1210      Cr = 1/Cr;
1211  end
1212  if Cr == 1
1213      epsilonsh = Ntu_sh/(1+Ntu_sh);
1214  else
1215      epsilonsh = (1 - exp(-Ntu_sh*(1-Cr)))/...
1216          (1 - Cr*exp(-Ntu_sh*(1-Cr)));
1217  end
1218  if wsh > 0.01
1219      c(1,1) = (epsilonsh - 0.9999);
1220  else
1221      c(1,1) = 0.0; % ignore the superheated section
1222  end
```

```
1223 c(2,1) = (epsilontp - 0.9999);
1224 U = U.*U_ori;
1225 x(1) = U(3);
1226 x(2) = U(4);
1227 x(3) = U(5);
1228 c(3,1) = -x(3);
1229 c(4,1) = -x(1);
1230 % fin efficiency at rated condition should be between 0.01 and 0.99
1231 c(5,1) = tanh(...
1232     x(3)*cscp(1)^0.5*x(1)^.5 ...
1233 )/(x(3)*cscp(1)^0.5)/x(1)^.5 - 0.99;
1234 c(6,1) = tanh(x(3)*x(1)^.5)/(x(3))/x(1)^.5 - 0.99;
1235 c(7,1) = 0.01-tanh(x(3)*x(1)^.5)/(x(3))/x(1)^.5;
1236 c(8,1) = 0.01-tanh(...
1237     x(3)*cscp(1)^0.5*x(1)^.5 ...
1238 )/(x(3)*cscp(1)^0.5)/x(1)^.5;
1239
1240 end
1241
1242 function residual = PressureDrop_residual(...
1243     C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, rho_l, ...
1244     rho_v, x_in, w_superheat, w_twophase, w_subcool, mu_l, mu_v, ...
1245     mdot_rated, SH_rated, SC_rated, rho_rated, rho_out_rated, ...
1246     rho_tp_rated, mu_l_rated, mu_v_rated, SH_limit, SC_limit, ...
1247     ΔP, weigh ...
1248 )
```

```
1249 n = length(SH);
1250 residual = 0;
1251 for i = 1:n
1252     residual = residual + 1/weigh(i,1)*(ΔP(i,1)-PressureDrop(...
1253         C, N, ΔP_av, mdot(i,1), SH(i,1), SC(i,1), rho(i,1), ...
1254         rho_out(i,1), rho_tp(i,1), rho_l(i,1), rho_v(i,1), ...
1255         x_in(i,1), w_superheat(i,1), w_twophase(i,1), ...
1256         w_subcool(i,1), mu_l(i,1), mu_v(i,1), mdot_rated, ...
1257         SH_rated, SC_rated, rho_rated, rho_out_rated, ...
1258         rho_tp_rated, mu_l_rated, mu_v_rated ...
1259     ))^2;
1260 end
1261 end
1262
1263 function [c, ceq] = DP_noncon(...
1264     C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, rho_l, ...
1265     rho_v, w_superheat, w_twophase, w_subcool, mu_l, mu_v, ...
1266     mdot_rated, SH_rated, SC_rated, rho_rated, rho_out_rated, ...
1267     rho_tp_rated, mu_l_rated, mu_v_rated, SH_limit, SC_limit, ...
1268     ΔP, r ...
1269 )
1270 ceq = [];
1271 c = PressureDrop_residual(...
1272     C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, rho_l, ...
1273     rho_v, w_superheat, w_twophase, w_subcool, mu_l, mu_v, ...
1274     mdot_rated, SH_rated, SC_rated, rho_rated, rho_out_rated, ...
```

```
1275      rho_tp_rated, mu_l_rated, mu_v_rated, SH_limit, SC_limit, ΔP ...
1276  ) - r;
1277  end
1278
1279  function ΔP_est = PressureDrop(...
1280      C, N, ΔP_av, mdot, SH, SC, rho, rho_out, rho_tp, rho_l, ...
1281      rho_v, x_in, w_superheat, w_twophase, w_subcool, mu_l, mu_v, ...
1282      mdot_rated, SH_rated, SC_rated, rho_rated, rho_out_rated, ...
1283      rho_tp_rated, mu_l_rated, mu_v_rated ...
1284  )
1285  % function to estimate the pressure drop
1286  ΔP_est = C(1)*w_twophase.*(...
1287      mdot./mdot_rated ...
1288  ).^2./(rho_tp./rho_rated).*(...
1289      (x_in./rho_v.*mu_v+(1-x_in)./rho_l.*mu_l)./(...
1290          x_in./rho_v+(1-x_in)./rho_l ...
1291      )./mu_v_rated ...
1292  ).^C(4);
1293  ΔP_est = ΔP_est + C(2).*w_superheat.*(...
1294      mdot./mdot_rated ...
1295  ).^2./(rho_v./rho_out_rated);
1296  ΔP_est = ΔP_est + C(3).*(mdot./mdot_rated).^2.* ...
1297      (1./rho_out-1./rho)*rho_rated;
1298  end
1299
1300  function [Pout hout Taout waout Qevap Charge ...
```

```
1301      FanPower EvapOutput ma EvapInput] = Evaporator_after(...
1302          Pin, Pout_exp, hin, hout_est, mdot_r, Tain, wain, Pain, ...
1303          Vdot, para, refri, libname ...
1304      )
1305
1306  % Variable list
1307  % Pout: Refrigerant pressure at outlet (kPa)
1308  % hout: Refrigerant enthalpy at outlet (kPa)
1309  % Taout: Air temperature at outlet (K)
1310  % waout: Air humidity at outlet (kg/kg)
1311  % Qevap: Heat transfer rate (W)
1312  % Charge: Amount of refrigerant in the evaporator (kg)
1313  % FanPower: Evaporator fan power consumption (W)
1314  % EvapOutput: All other informaiton listed in the function in DLL
1315
1316  % Pin: Refrigenrat pressure at inlet (kPa)
1317  % hin: Refrigenrat enthalpy at inlet (W)
1318  % hout_est: Estimated refrigerant enthalpy, write -9999 if unknown
1319  % mdot_r: Refrigeraat mass flow rate (kg/s)
1320  % Tain: Air temperature at inlet (K)
1321  % wain: Humidity at inlet (kg/kg)
1322  % Pain: Atmospheric pressure (kPa)
1323  % Vdot: Air volumetric flow at inlet (m^3/s)
1324  % para: parameters (misc.)
1325  % refri: Name of refrigerant
1326  % lib: Name of dll library to be used
```

```
1327
1328
1329 % UAa
1330 airinlet = zeros(1,5);
1331 airinlet = calllib(...
1332     libname, 'HumAir_DLL', Tain, Pain, 2, wain, airinlet ...
1333 );
1334 ma = (1)/airinlet(5)*Vdot;
1335 EvapInput(1) = 1;
1336
1337 EvapInput(2) = para.U_a*(ma/para.ma)^para.n*tanh(...
1338     para.p*(para.U_a*(ma/para.ma)^para.n)^.5 ...
1339 )/para.p*para.A_r/(para.U_a*(ma/para.ma)^para.n)^.5;
1340 cpa = 1006 + 1860*airinlet(2);
1341 cs = 0;
1342 Tsat = propertyRH('T','P',Pout_exp,'Q',1,refri);
1343 cs = calllib(libname, 'Matlab_cair_sat', Tsat, cs);
1344
1345 EvapInput(3) = para.U_a*(ma/para.ma)^para.n*tanh(...
1346     (cs/cpa)^0.5*para.p*(para.U_a*(ma/para.ma)^para.n)^.5 ...
1347 )/((cs/cpa)^0.5*para.p)/(para.U_a*(ma/para.ma)^para.n)^.5*para.A_r;
1348
1349 %UAtp
1350 hv = propertyRH('H', 'P', Pout_exp,'Q', 1,refri);
1351 hf = propertyRH('H', 'P', Pout_exp,'Q', 0,refri);
1352 xin = (hin - hf)/(hv - hf);
```

```
1353 if(xin≤0.001)
1354     xin = 0.001; % avoid division by zero error
1355 end
1356 if((hout_est>hin)&&(hout_est<hv))
1357     xout = (hout_est - hf)/(hv - hf);
1358     EvapInput(5) =  h_bartp(...
1359         para.Utp*para.A_r, xin, xout, ...
1360         mdot_r, mdot_r*(hv-hin), Tsat, refri ...
1361     ); %use values from previous estimations for calculation
1362 else
1363     EvapInput(5) =  h_bartp(...
1364         para.Utp*para.A_r, xin, 1, mdot_r, mdot_r*(hout_est-hin), ...
1365         Tsat, refri ...
1366     ); %outlet refrigerant quality is an assumption
1367 end
1368
1369 %UAsh
1370 cp_v = propertyRH('C', 'P', Pout_exp,'Q', 1, refri);
1371 miu_v = propertyRH('V', 'P', Pout_exp,'Q', 1, refri);
1372 k_v = propertyRH('L', 'P', Pout_exp,'Q', 1, refri);
1373 Ksh = (cp_v*miu_v/k_v)^0.4*(mdot_r/miu_v)^0.8;
1374 EvapInput(4) = para.Ush*Ksh/para.Ksh*para.A_r;
1375
1376 % input arrangement
1377 EvapInput(6) = Tain;
1378 EvapInput(7) = airinlet(4);
```

```
1379 EvapInput(8) = Vdot;
1380 EvapInput(9) = Pain;
1381 EvapInput(10) = mdot_r;
1382 EvapInput(11) = xin;
1383 EvapInput(12) = Pout_exp;
1384 EvapInput(13) = propertyRH('T','P',Pout_exp,'Q',1,refri);
1385 EvapInput(14) = para.Din;
1386 EvapInput(15) = para.Lt;
1387 EvapInput = real(EvapInput);
1388 EvapOutput = zeros(1,14);
1389 if ¬isempty(findstr(refri,'.fld'))
1390     index_end = findstr(refri,'.fld');
1391     refname = refri(1:index_end-1);
1392 else
1393     refname = refri;
1394 end
1395 [EvapInput EvapOutput] = calllib(...
1396     libname, 'Matlab_Evaporator_Forward_Charge_ii', EvapInput, ...
1397     EvapOutput, refname ...
1398 );
1399
1400 % output arrangement and calculation of outlet pressure
1401 Qevap = EvapOutput(1);
1402 hout = hin + Qevap/mdot_r;
1403 Taout = EvapOutput(8);
1404 waout = EvapOutput(14);
```

```
Charge = EvapOutput(13);
FanPower = para.FanPower;
Pout = Pout_exp;
end
```

N.9 Suction Line Modeling

```
function System = SuctionLine_minimization_v042_main()

% Function to pass information to train suction line model

% define function and folder name
function_name = 'SuctionLine_minimization';
version_main = '042';
version_regress = '041';
version_plot = '000';
foldername = strcat(function_name,'_v',version_main);
regressname = strcat(function_name,'_v',version_regress);
plotname = strcat(function_name,'_plot_v',version_plot);
regress_func = str2func(['@(System, libname, version_number)',...
    regressname,'(System, libname, version_number)']);
plot_func = str2func(['@(System, libname, version_number)',...
    plotname,'(System, libname, version_number)']);
mkdir(foldername);
savefilename = strcat(foldername,'.mat');

% Module Definition
```

```
% Define the linset lengths and inner diameters [m, m]
% Hot gas line
SuctionLine.line = 0;
SuctionLine.dia = 0;
SuctionLine.ΔP = 0;
SuctionLine.HeatLossUAtomr = 0;
SuctionLine.mdot_rated = 0;
SuctionLine.V_rated = 0;
SuctionLine.rho_rated = 0;
SuctionLine.C = zeros(5,1);
SuctionLine.C_Q = zeros(4,1);
SuctionLine.Δh_rated = 0;
SuctionLine.ΔT_rated = 0;

% Set information
% re-file at this stage to avoid problems in parfor loop
% Boshen 3-ton R410A packaged FXO system (Scroll)
% (Shen_030_R410A_packaged_FXO_Scroll)
i = 1;
System(i).name = 'Shen_030_R410A_packaged_FXO_Scroll';
System(i).othername = System(i).name;
System(i).evap_name = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Boshen_Cycle_data_3tonR410apackaged.xls';
System(i).foldername = 'Boshen';
```

```
47 System(i).worksheetname = 'SI (no_invalid_CA)';
48 System(i).refname = 'R410A';
49 System(i).Standard_Airflow_evap = 0;
50 System(i).Fan_Upstream_evap = 0;
51 System(i).Standard_Airflow_cond = 1;
52 System(i).Fan_Upstream_cond = 1;
53 System(i).Heating = 0;
54 System(i).Combined = 0;
55 System(i).Accumulator = 0;
56 System(i).Diameter = 0.00853;
57 System(i).Tube_Length = 1.67075;
58 System(i).Ntube = 40;
59 System(i).Pc = 4901;
60 System(i).SuctionLine = SuctionLine;
61 System(i).SuctionLine.line = 1.168;
62 System(i).SuctionLine.dia = 0.01727;
63 System(i).Charge_std = 3.24;
64 System(i).Vdot_evap_std = 0.587;
65 System(i).Vdot_cond_std = 1.10;
66
67 % Boshen 3-ton R410A split FXO system (Recip)
68 % (Shen_030_R410A_split_FXO_Recip)
69 % for compressor modeling, merged with TXV system data
70 i = i + 1;
71 System(i).name = 'Shen_030_R410A_split_FXO_TXV_Recip';
72 System(i).othername = System(i).name;
```

```
73 System(i).evap_name = System(i).name;
74 System(i).cond_name = System(i).name;
75 System(i).LLname = System(i).name;
76 System(i).filename = 'Boshen_Cycle_data_3tonR410AsplitFXO_TXV.xls';
77 System(i).foldername = 'Boshen_3tonR410ASplitFXO';
78 System(i).worksheetname = 'SI (no_invalid_CA)';
79 System(i).refname = 'R410A';
80 System(i).Standard_Airflow_evap = 0;
81 System(i).Fan_Upstream_evap = 0;
82 System(i).Standard_Airflow_cond = 1;
83 System(i).Fan_Upstream_cond = 1;
84 System(i).Heating = 0;
85 System(i).Combined = 0;
86 System(i).Accumulator = 0;
87 System(i).Diameter = 0.00849;
88 System(i).Tube_Length = 2.255;
89 System(i).Ntube = 32;
90 System(i).Pc = 4901;
91 System(i).SuctionLine = SuctionLine;
92 System(i).SuctionLine.line = 0.5715 + 4.845;
93 System(i).SuctionLine.dia = 0.01727;
94 System(i).Charge_std = 2.86;
95 System(i).Vdot_evap_std = 0.57;
96 System(i).Vdot_cond_std = 1.32;
97
98 % Boshen 5-ton R407C packaged FXO system (Scroll)
```

```
% (Shen_050_R407C_packaged_FXO_Scroll)
i = i + 1;
System(i).name = 'Shen_050_R407C_packaged_FXO_Scroll';
System(i).othername = System(i).name;
System(i).evap_name = System(i).name;
System(i).cond_name = System(i).name;
System(i).LLname = System(i).name;
System(i).filename = 'Boshen_Cycle_data_5tonR407CpackagedFXO.xls';
System(i).foldername = 'Boshen_5tonR407CPackagedFXO';
System(i).worksheetname = 'SI (no_invalid_CA)';
System(i).refname = 'R407C';
System(i).Standard_Airflow_evap = 0;
System(i).Fan_Upstream_evap = 0;
System(i).Standard_Airflow_cond = 1;
System(i).Fan_Upstream_cond = 1;
System(i).Heating = 0;
System(i).Combined = 0;
System(i).Accumulator = 0;
System(i).Diameter = 0.00693928;
System(i).Tube_Length = 2.282;
System(i).Ntube = 56;
System(i).Pc = 3786;
System(i).SuctionLine = SuctionLine;
System(i).SuctionLine.line = 0;
System(i).SuctionLine.dia = 0.01128;
System(i).Charge_std = 2.59;
```

```
125 System(i).Vdot_evap_std = 1.04;
126 System(i).Vdot_cond_std = 1.88;
127
128 % Breuker 3-ton R22 packaged FXO system (Recip)
129 % (Breuker_030_R22_packaged_FXO_Recip)
130 i = i + 1;
131 System(i).name = 'Breuker_030_R22_packaged_FXO_Recip';
132 System(i).othername = System(i).name;
133 System(i).evap_name = System(i).name;
134 System(i).cond_name = System(i).name;
135 System(i).LLname = System(i).name;
136 System(i).filename = 'Breuker_Cycle_data_3tonR22packagedFXO_v05.xls';
137 System(i).foldername = 'Breuker';
138 System(i).worksheetname = 'SI (no_invalid_CA)';
139 System(i).refname = 'R22.fld';
140 System(i).Standard_Airflow_evap = 0;
141 System(i).Fan_Upstream_evap = 0;
142 System(i).Standard_Airflow_cond = 1;
143 System(i).Fan_Upstream_cond = 1;
144 System(i).Heating = 0;
145 System(i).Combined = 0;
146 System(i).Accumulator = 0;
147 % assumed
148 System(i).Diameter = 0.00849;
149 System(i).Tube_Length = 2.255;
150 System(i).Ntube = 32;
```

```
151 System(i).Pc = 4990;
152 System(i).SuctionLine = SuctionLine;
153 System(i).SuctionLine.line = 1.168;
154 System(i).SuctionLine.dia = 0.01128;
155 System(i).Charge_std = 2.72;
156 System(i).Vdot_evap_std = 0.703;
157 System(i).Vdot_cond_std = 1.61;
158
159 % Kim (NIST) 2.5-ton R410A split TXV system (Scroll)
160 % (NIST_025_R410A_split_TXV_Scroll)
161 i = i + 1;
162 System(i).name = 'NIST_025_R410A_split_TXV_Scroll';
163 System(i).othername = System(i).name;
164 System(i).evap_name = System(i).name;
165 System(i).cond_name = System(i).name;
166 System(i).LLname = System(i).name;
167 System(i).filename = 'NIST_Cycle_data_25tonR410a_v02.xls';
168 System(i).foldername = 'NIST';
169 System(i).worksheetname = 'SI (no_invalid_CA)';
170 System(i).refname = 'R410A';
171 System(i).Standard_Airflow_evap = 1;
172 System(i).Fan_Upstream_evap = 1;
173 System(i).Standard_Airflow_cond = 1;
174 System(i).Fan_Upstream_cond = 1;
175 System(i).Heating = 0;
176 System(i).Combined = 0;
```

```
177 System(i).Accumulator = 0;
178 System(i).Diameter = 0.0042;
179 System(i).Tube_Length = 1.778;
180 System(i).Ntube = 64;
181 System(i).Pc = 4901;
182 System(i).SuctionLine = SuctionLine;
183 System(i).SuctionLine.line = 7.9248;
184 System(i).SuctionLine.dia = 0.015875;
185 System(i).Charge_std = 4.65;
186 System(i).Vdot_evap_std = 0.507;
187 System(i).Vdot_cond_std = 1.02;
188
189 % Harms 5-ton R22 packaged TXV system (Scroll)
190 % (Harms_050_R22_packaged_TXV_Scroll_Heating)
191 i = i + 1;
192 System(i).name = 'Harms_050_R22_packaged_TXV_Scroll';
193 System(i).othername = System(i).name;
194 System(i).evap_name = System(i).name;
195 System(i).cond_name = System(i).name;
196 System(i).LLname = System(i).name;
197 System(i).filename = 'Harms_Cycle_data_5tonR22packagedTXV.xls';
198 System(i).foldername = 'Harms_050tonR22PackagedTXV';
199 System(i).worksheetname = 'SI (no_invalid_CA)';
200 System(i).refname = 'R22.fld';
201 System(i).Standard_Airflow_evap = 0;
202 System(i).Fan_Upstream_evap = 0;
```

```
203 System(i).Standard_Airflow_cond = 1;
204 System(i).Fan_Upstream_cond = 1;
205 System(i).Heating = 0;
206 System(i).Combined = 0;
207 System(i).Accumulator = 0;
208 System(i).Diameter = 0.00889;
209 System(i).Tube_Length = 1.629;
210 System(i).Ntube = 64;
211 System(i).Pc = 4990;
212 System(i).SuctionLine = SuctionLine;
213 System(i).SuctionLine.line = 0;
214 System(i).SuctionLine.dia = 0.01128;
215 System(i).Charge_std = 5.22;
216 System(i).Vdot_evap_std = 1.05;
217 System(i).Vdot_cond_std = 1.97;
218
219 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
220 % (HCA3_030_R410A_split_TXV_Scroll)
221 i = i + 1;
222 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll';
223 System(i).othername = System(i).name;
224 System(i).evap_name = System(i).name;
225 System(i).cond_name = System(i).name;
226 System(i).LLname = System(i).name;
227 System(i).filename = 'Kim_Cycle_data_R410A_HCA3.xlsx';
228 System(i).foldername = 'Kim-HCA3';
```

```
229 System(i).worksheetname = 'SI (no_invalid_CA)';
230 System(i).refname = 'R410A';
231 System(i).Standard_Airflow_evap = 0;
232 System(i).Fan_Upstream_evap = 1;
233 System(i).Standard_Airflow_cond = 1;
234 System(i).Fan_Upstream_cond = 1;
235 System(i).Heating = 0;
236 System(i).Combined = 0;
237 System(i).Accumulator = 0;
238 % assumed
239 System(i).Diameter = 0.00849;
240 System(i).Tube_Length = 2.255;
241 System(i).Ntube = 32;
242 System(i).Pc = 4901;
243 System(i).SuctionLine = SuctionLine;
244 System(i).SuctionLine.line = 0.5715 + 4.845;
245 System(i).SuctionLine.dia = 0.01727;
246 System(i).Charge_std = 3.36;
247 System(i).Vdot_evap_std = 0.6;
248 System(i).Vdot_cond_std = 1.71;
249
250 % Kim 3-ton R410A split TXV system (Scroll) (HCA3)
251 % (HCA3_030_R410A_split_TXV_Scroll_Heating)
252 i = i + 1;
253 System(i).name = 'HCA3_030_R410A_split_TXV_Scroll_Heating';
254 System(i).othername = 'HCA3_030_R410A_split_TXV_Scroll';
```

```
255 System(i).evap_name = System(i).name;
256 System(i).cond_name = System(i).name;
257 System(i).LLname = System(i).name;
258 System(i).filename = 'Kim_Cycle_data_R410A_HCA3_Heating.xlsx';
259 System(i).foldername = 'Kim-HCA3-Heating';
260 System(i).worksheetname = 'SI (no_invalid_CA)';
261 System(i).refname = 'R410A';
262 System(i).Standard_Airflow_evap = 1;
263 System(i).Fan_Upstream_evap = 1;
264 System(i).Standard_Airflow_cond = 1;
265 System(i).Fan_Upstream_cond = 1;
266 System(i).Heating = 1;
267 System(i).Combined = 0;
268 System(i).Accumulator = 0;
269 % assumed
270 System(i).Diameter = 0.00849;
271 System(i).Tube_Length = 2.255;
272 System(i).Ntube = 32;
273 System(i).Pc = 4901;
274 System(i).SuctionLine = SuctionLine;
275 System(i).SuctionLine.line = 0.5715 + 4.845;
276 System(i).SuctionLine.dia = 0.01727;
277
278 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
279 % (YSA_030_R22_split_TXV_Scroll)
280 i = i + 1;
```

```
281 System(i).name = 'YSA_030_R22_split_TXV_Scroll';
282 System(i).othername = System(i).name;
283 System(i).evap_name = System(i).name;
284 System(i).cond_name = System(i).name;
285 System(i).LLname = System(i).name;
286 System(i).filename = 'Kim_Cycle_data_R22_YSA.xlsx';
287 System(i).foldername = 'Kim-YSA';
288 System(i).worksheetname = 'SI (no_invalid_CA)';
289 System(i).refname = 'R22.fld';
290 System(i).Standard_Airflow_evap = 0;
291 System(i).Fan_Upstream_evap = 1;
292 System(i).Standard_Airflow_cond = 1;
293 System(i).Fan_Upstream_cond = 1;
294 System(i).Heating = 0;
295 System(i).Combined = 0;
296 System(i).Accumulator = 1;
297 % assumed
298 System(i).Diameter = 0.00849;
299 System(i).Tube_Length = 2.255;
300 System(i).Ntube = 32;
301 System(i).Pc = 4990;
302 System(i).SuctionLine = SuctionLine;
303 System(i).SuctionLine.line = 0.5715 + 4.845;
304 System(i).SuctionLine.dia = 0.01727;
305
306 % Kim 3-ton R22 split TXV system (Scroll) (YSA)
```

```
307 % (YSA_030_R22_split_TXV_Scroll_Heating)
308 i = i + 1;
309 System(i).name = 'YSA_030_R22_split_TXV_Scroll_Heating';
310 System(i).othername = 'YSA_030_R22_split_TXV_Scroll';
311 System(i).evap_name = System(i).name;
312 System(i).cond_name = System(i).name;
313 System(i).LLname = System(i).name;
314 System(i).filename = 'Kim_Cycle_data_R22_YSA_Heating.xlsx';
315 System(i).foldername = 'Kim-YSA-Heating';
316 System(i).worksheetname = 'SI (no_invalid_CA)';
317 System(i).refname = 'R22.fld';
318 System(i).Standard_Airflow_evap = 1;
319 System(i).Fan_Upstream_evap = 1;
320 System(i).Standard_Airflow_cond = 1;
321 System(i).Fan_Upstream_cond = 1;
322 System(i).Heating = 1;
323 System(i).Combined = 0;
324 System(i).Accumulator = 1;
325 % assumed
326 System(i).Diameter = 0.00849;
327 System(i).Tube_Length = 2.255;
328 System(i).Ntube = 32;
329 System(i).Pc = 4990;
330 System(i).SuctionLine = SuctionLine;
331 System(i).SuctionLine.line = 0.5715 + 4.845;
332 System(i).SuctionLine.dia = 0.01727;
```

```
333
334 % Kim 3-ton R22 split TXV system (Scroll) (YKC)
335 % (YKC_030_R22_split_TXV_Scroll)
336 i = i + 1;
337 System(i).name = 'YKC_030_R22_split_TXV_Scroll';
338 System(i).othername = System(i).name;
339 System(i).evap_name = System(i).name;
340 System(i).cond_name = System(i).name;
341 System(i).LLname = System(i).name;
342 System(i).filename = 'Kim_Cycle_data_R22_YKC.xlsx';
343 System(i).foldername = 'Kim-YKC';
344 System(i).worksheetname = 'SI (no_invalid_CA)';
345 System(i).refname = 'R22.fld';
346 System(i).Standard_Airflow_evap = 0;
347 System(i).Fan_Upstream_evap = 1;
348 System(i).Standard_Airflow_cond = 1;
349 System(i).Fan_Upstream_cond = 1;
350 System(i).Heating = 0;
351 System(i).Combined = 0;
352 System(i).Accumulator = 1;
353 % assumed
354 System(i).Diameter = 0.00849;
355 System(i).Tube_Length = 2.255;
356 System(i).Ntube = 32;
357 System(i).Pc = 4990;
358 System(i).SuctionLine = SuctionLine;
```

```
359 System(i).SuctionLine.line = 0.5715 + 4.845;
360 System(i).SuctionLine.dia = 0.01727;
361 System(i).Charge_std = 3.32;
362 System(i).Vdot_evap_std = 0.73;
363 System(i).Vdot_cond_std = 1.43;
364
365 % Kim 4-ton R410A packaged TXV system (Recip) (TM)
366 % (TM_040_R410A_packaged_TXV_Recip)
367 i = i + 1;
368 System(i).name = 'TM_040_R410A_packaged_TXV_Recip';
369 System(i).othername = System(i).name;
370 System(i).evap_name = System(i).name;
371 System(i).cond_name = System(i).name;
372 System(i).LLname = System(i).name;
373 System(i).filename = 'Kim_Cycle_Data_4tonR410APackagedTXV.xlsx';
374 System(i).foldername = 'Kim-TM';
375 System(i).worksheetname = 'SI (no_LL)';
376 System(i).refname = 'R410A';
377 System(i).Standard_Airflow_evap = 0;
378 System(i).Fan_Upstream_evap = 0;
379 System(i).Standard_Airflow_cond = 1;
380 System(i).Fan_Upstream_cond = 1;
381 System(i).Heating = 0;
382 System(i).Combined = 0;
383 System(i).Accumulator = 0;
384 % assumed
```

```
385 System(i).Diameter = 0.00849;
386 System(i).Tube_Length = 2.255;
387 System(i).Ntube = 32;
388 System(i).Pc = 4990;
389 System(i).SuctionLine = SuctionLine;
390 System(i).SuctionLine.line = 1.168;
391 System(i).SuctionLine.dia = 0.01128;
392 System(i).Charge_std = 2.72;
393 System(i).Vdot_evap_std = 0.703;
394 System(i).Vdot_cond_std = 1.61;
395
396 % other settings for storage
397 m = length(System);
398 % m = 1;
399 % m = 2;
400 for i = 1:m
401     System(i).mainfoldername = foldername;
402     System(i).r2_Q = 0;
403     System(i).r2_ΔP = 0;
404 end
405
406 loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
407 libname ='lib'; % Name of library
408
409 % start calculation
410 no_plot = zeros(m,1);
```

```
411 % make an error statment
412 try
413     parfor i = 1:m
414         System(i) = 1; % a statement leads to error
415     end
416 catch err
417 end
418 for i = 1:m
419     error_statement(i) = err;
420 end
421 % parfor i = 1:m
422 for i = 1:m
423     try
424         disp(['Starting to run ',System(i).name]);
425         [SuctionLine r2_ΔP r2_Q] = regress_func(...
426             System(i), libname, version_main ...
427         );
428         System(i).SuctionLine = SuctionLine;
429         System(i).r2_Q = r2_Q;
430         System(i).r2_ΔP = r2_ΔP;
431         disp(['Finishing simulating ',System(i).name]);
432     catch err
433         disp(['Problem found in system ',System(i).name]);
434         no_plot(i,1) = 1;
435         error_statement(i) = err;
436     end
```

```
437 end
438 save(savefilename);
439
440 % plot cannot be done in parfor. Do it in a for loop
441 for i = 1:m
442     if no_plot(i,1) == 0
443         disp(['Starting to plot ',System(i).name]);
444         plot_func(System(i), libname, version_main);
445         disp(['Finishing plotting ',System(i).name]);
446     end
447 end
448
449 save(savefilename);
450 movefile(savefilename,strcat(foldername,'\'));
451
452 try
453     matlabpool close; % if catch an error, close and reset it
454 end
455 try
456     unloadlibrary lib;
457 end
```

```
1 function [PipeLineClass r2_ΔP r2_Q] = ...
2     SuctionLine_minimization_v041(System,libname,version_number)
3
```

```
4  % Functions to train suction line model
5
6  %% Data reading
7  bin_num = 10;
8  PipeLineClass = System.SuctionLine;
9  filename = System.filename;
10 worksheetname = System.worksheetname;
11 refri = System.refname;
12 Standard_Airflow_evap = System.Standard_Airflow_evap;
13 Ind_Fan_Upstream_evap = System.Fan_Upstream_evap;
14 Standard_Airflow_cond = System.Standard_Airflow_cond;
15 Ind_Fan_Upstream_cond = System.Fan_Upstream_cond;
16 % D = System.Diameter;
17 % Ltube = System.Tube_Length;
18 % Ntube = System.Ntube;
19 % Pc = System.Pc;
20 version = strcat(System.name,'_',version_number);
21 savefilename = strcat('SuctionLine_minimization_v',version,'.mat');
22 foldername = strcat(strcat(...
23     pwd,'\',System.mainfoldername,'\' ...
24 ),strcat('SuctionLine_minimization_v',version));
25 mkdir(foldername);
26
27 % for Compressor
28 version_Comp = '073';
29 version_func = '045';
```

```
30 name_Comp = 'Compressor_regression_v';
31 component_name = strcat(name_Comp,System.othername,'_',version_Comp);
32 foldername_comp = strcat(strcat(...
33     pwd,'\',name_Comp,version_Comp,'\',component_name ...
34 ));
35 raw = open(strcat(foldername_comp,'\',strcat(component_name,'.mat')));
36 Comp_para = raw.Comp;
37 Comp_func = str2func([...
38     '@(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)',...
39     'Compressor_v',version_func, ...
40     '(Pevap, hin, Pcond, Tamb, Tout_guess, para, refri)' ...
41 ]);
42
43 % for Liquid Line
44 version_LiquidLine = '041';
45 version_func = '003';
46 name_LiquidLine = 'LiquidLine_minimization_v';
47 component_name = strcat(...
48     name_LiquidLine,System.LLname,'_',version_LiquidLine ...
49 );
50 foldername_LiquidLine = strcat(strcat(...
51     pwd,'\',name_LiquidLine,version_LiquidLine,'\',component_name ...
52 ));
53 raw = open(strcat(...
54     foldername_LiquidLine,'\',strcat(component_name,'.mat') ...
55 ));
```

```
LiquidLine_para = raw.PipeLineClass;
LiquidLine_func = str2func([...
    '@(mr, Pin, hin, Tamb, para, refri, phase, libname)',...
    'pipeline_v',version_func,...
    '(mr, Pin, hin, Tamb, para, refri, phase, libname)' ...
]);

Q_gain_version = '029';
Q_gain_name = ['Q_gain_observation_v',System.name,'_',Q_gain_version];
Q_gain_raw = open([...
    'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,'\',...
    Q_gain_name,'.mat' ...
]);
k = length(Q_gain_raw.index_removed);
data2 = Q_gain_raw.data;
code = Q_gain_raw.code;
fault = Q_gain_raw.fault;
Q_cond_r = Q_gain_raw.Q_cond_r;
Q_evap_r = Q_gain_raw.Q_evap_r;
for i = k:-1:1 % reverse for correct indexing
    data2(Q_gain_raw.index_removed(i),:) = [];
    code(Q_gain_raw.index_removed(i),:) = [];
    fault(Q_gain_raw.index_removed(i),:) = [];
    Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
    Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
end
```

```
[pp qq] = size(data2);
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
i = 1;
while i ≤ pp
    if ¬isempty(cell2mat(cell_VL(i))) || ¬isempty(...
            cell2mat(cell_NC(i))...
        ) || ¬isempty(cell2mat(cell_LL(i)))
        data2(i,:) = [];
        fault(i,:) = [];
        code(i,:) = [];
        cell_VL(i,:) = [];
        cell_LL(i,:) = [];
        cell_NC(i,:) = [];
        Q_evap_r(i,:) = [];
        Q_cond_r(i,:) = [];
```

```
108         i = i - 1;
109         pp = pp - 1;
110     end
111     i = i + 1;
112 end
113 [n dummy] = size(data2);
114
115 %% SuctionLine Line
116 data_HG = data2; % assign data to avoid re-reading later
117
118 for j = 1:n
119     SC_LL_in(j,1) = propertyRH(...
120         'T','P',data_HG(j,4),'Q',0,refri...
121     ) - data_HG(j,12);
122     SC_LL_out(j,1) = propertyRH(...
123         'T','P',data_HG(j,5),'Q',0,refri ...
124     ) - data_HG(j,13);
125     SH_evap_out(j,1) = data_HG(j,16) - ...
126         propertyRH('T','P',data_HG(j,8),'Q',1,refri);
127     SH_comp_in(j,1) = data_HG(j,9) - ...
128         propertyRH('T','P',data_HG(j,1),'Q',1,refri);
129 end
130
131 % Obtain Superheat, Mass Flow Rate and Pressures
132 i = 1;
133 for j = 1:n
```

```
134     % data filtering for both SC and SH calculation
135     if data_HG(j,5) > data_HG(j,4)
136         data_HG(j,5) = data_HG(j,4);
137     end
138     if System.Heating==0
139         Tamb_a = data_HG(j,21);
140     else
141         Tamb_a = data_HG(j,17);
142     end
143     SC_check = propertyRH(...
144         'T','P',data_HG(j,4),'Q',0,refri ...
145     ) - data_HG(j,12);
146     SC_check_II = propertyRH('T','P',data_HG(j,5),'Q',0,refri) - ...
147         data_HG(j,13);
148     SH_check = data_HG(j,9) - propertyRH(...
149         'T','P',data_HG(j,1),'Q',1,refri ...
150     );
151     SH_check_II = data_HG(j,16) - ...
152         propertyRH('T','P',data_HG(j,8),'Q',1,refri);
153     if ¬(SC_check < 1 & ¬isempty(...
154             cell2mat(strfind(fault(j,:),'CA'))...
155         )) & SH_check> 1 & SH_check_II > 1
156         code_record(i,:) = code(j,:);
157         SC_record(i,1) = SC_check;
158         SC_record_II(i,1) = SC_check_II;
159         SH_record(i,1) = SH_check;
```

```
160         SH_record_II(i,1) = SH_check_II;
161         if System.Heating==0
162             Tamb(i,1) = data_HG(j,21);
163         else
164             Tamb(i,1) = data_HG(j,17);
165         end
166         if ¬(((SC_check ≥ 3)||(...
167                 SH_check_II>1&&SC_check_II≥1 ...
168             )||(SC_check≥1)) && data_HG(i,23) > 0)
169             data_HG(j,23) = Comp_func(...
170                 data_HG(j,1), propertyRH(...
171                     'H','T',data_HG(j,9),'P',data_HG(j,1),refri ...
172                 ), data_HG(j,2), Tamb(i,1), data_HG(j,10), ...
173             Comp_para, refri);
174         end
175         mdot(i,1) = data_HG(j,23);
176         if SC_check_II > 1
177             h_EXV_in = propertyRH(...
178                 'H','T',data_HG(j,13),'P',data_HG(j,5),refri ...
179             );
180         elseif SH_check_II > 1
181             hout_evap = propertyRH(...
182                 'H','T',data_HG(j,16),'P',data_HG(j,8),refri ...
183             );
184             h_EXV_in = hout_evap - Q_evap_r(j,1)/data_HG(j,23);
185         elseif SC_check > 1
```

```
186             [dum1 h_EXV_in dum2] = LiquidLine_func(...
187                 data_HG(j,23), data_HG(j,4), propertyRH(...
188                     'H','T',data_HG(j,12),'P',data_HG(j,4),refri ...
189                 ), Tamb(i,1), LiquidLine_para, refri, 0, libname);
190         else
191             hcond_out = propertyRH(...
192                 'H','T',data_HG(j,11),'P',data_HG(j,3),refri ...
193             ) - Q_cond_r(j,1)/data_HG(j,23);
194             [dum1 h_EXV_in dum2] = LiquidLine_func(...
195                 data_HG(j,23), data_HG(j,4), hcond_out, ...
196                 Tamb(i,1), LiquidLine_para, refri, 0, libname ...
197             );
198         end
199         if System.Heating == 1
200             data_HG(j,19) = 230;
201             data_HG(j,20) = 230;
202         end
203         airinlet(1,1:5) = calllib(...
204             libname, 'HumAir_DLL', data_HG(j,17), ...
205             data_HG(j,32), 1, data_HG(j,19), zeros(1,5) ...
206         );
207         airoutlet(1,1:5) = calllib(...
208             libname, 'HumAir_DLL', data_HG(j,18), ...
209             data_HG(j,32), 1, data_HG(j,20), zeros(1,5) ...
210         );
211         if SH_check_II < 1
```

```
212             h_in_suc = h_EXV_in + Q_evap_r(j,1)/data_HG(j,23);
213             if SH_check > 1
214                 hout(i,1) = propertyRH(...
215                     'H','T',data_HG(j,9),'P',data_HG(j,1),refri ...
216                 );
217             else
218                 hout(i,1) = fzero(@(hin_guess) Comp_inlet(...
219                     hin_guess, data_HG(j,:), Tamb(i,1), ...
220                     Comp_para, Comp_func, refri ...
221                 ), h_in_suc);
222             end
223         else
224             h_in_suc = propertyRH(...
225                 'H','T',data_HG(j,16),'P',data_HG(j,8),refri ...
226             );
227             % check validity
228             hv = propertyRH('H','P',data_HG(j,8),'Q',1,refri);
229             if SH_check > 1
230                 hout(i,1) = propertyRH(...
231                     'H','T',data_HG(j,9),'P',data_HG(j,1),refri ...
232                 );
233             else
234                 hout(i,1) = fzero(@(hin_guess) Comp_inlet(...
235                     hin_guess, data_HG(j,:), Tamb(i,1), ...
236                     Comp_para, Comp_func, refri ...
237                 ), hv);
```

```
238             end
239         end
240         Q(i,1) = mdot(i,1)*(h_in_suc - hout(i,1));
241         ΔP(i,1) = data_HG(j,8) - data_HG(j,1);
242         rho(i,1) = propertyRH(...
243             'D','P',data_HG(j,8),'H',h_in_suc,refri ...
244         );
245         rho_out(i,1) = propertyRH(...
246             'D','P',data_HG(j,1),'H',hout(i,1),refri ...
247         );
248         if SH_check_II < 1.e-8
249             V(i,1) = propertyRH(...
250                 'V','P',data_HG(j,8),'Q',1,refri ...
251             );
252         else
253             V(i,1) = propertyRH(...
254                 'V','P',data_HG(j,8),'H',h_in_suc,refri ...
255             );
256         end
257         T(i,1) = propertyRH('T','P',data_HG(j,8),'H',h_in_suc,refri);
258         h(i,1) = h_in_suc;
259         P(i,1) = data_HG(j,8);
260         code_HG(i,1) = code(j,1);
261         data_h(i,:) = data_HG(j,:);
262         data_Q(i,:) = data_HG(j,:);
263         i = i + 1;
```

```
264     end
265 end
266
267 mdot_rated = mdot(1,1);
268 Q_rated = mean(Q);
269 ΔP_av = mean(ΔP);
270 rho_rated = mean(rho);
271 V_rated = mean(V);
272 SH_limit = 1;
273 SC_limit = 3;
274
275 options = optimset('MaxFunEvals',10000,'MaxIter',1000);
276 X1 = (mdot./mdot_rated);
277 X2 = (V./V_rated);
278 X3 = (rho./rho_rated);
279 X4 = (mdot./mdot_rated).^2.*(1./rho_out - 1./rho).*rho_rated;
280 C = zeros(4,1);
281 A = -eye(length(C));
282 % Viscosity increases friction but decreases flow rate: it's overall
283 % effect is undertermined.
284 A(3,3) = 0;
285 A(4,4) = 0;
286 b = zeros(length(C),1);
287 options = optimset(...
288     'MaxFunEvals',20000,'algorithm','active-set',...
289     'MaxIter',5000,'Display','off' ...
```

```
290 );
291 ΔP(ΔP<0) = 0;
292 if length(ΔP) > 1 & sum(ΔP.^2)>1.e-8
293     weigh = weighing_function_v000(ΔP, bin_num);
294 elseif sum(ΔP.^2)≤1.e-8
295     weigh = ones(length(ΔP),1);
296 else
297     weigh = 1;
298 end
299 if sum(abs(ΔP)) < 1.e-8 % no pressure drop
300     r2_ΔP = -9999;
301     ΔP_est = zeros(length(ΔP),1);
302     dev = 0;
303     fval = 0;
304     PipeLineClass.COV_X_ΔP = zeros(length(C),length(C));
305     PipeLineClass.ΔP_limit = Inf;
306 elseif length(ΔP) ≤ 5
307     C(1,1) = mean(ΔP);
308 else
309     C = [0 1 0]';
310     try
311         [C, fval] = fmincon(@(C) PressureDrop_residual(...
312             C, X1, X2, X3, X4, ΔP, weigh...
313         ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
314         if sum(isnan(C))
315             options = optimset(...
```

```
316                 'MaxFunEvals',20000,'algorithm','sqp',...
317                 'MaxIter',5000,'Display','off' ...
318             );
319             C = zeros(5,1);
320             [C, fval] = fmincon(@(C) PressureDrop_residual(...
321                 C, X1, X2, X3, X4, ΔP, weigh ...
322             ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
323         end
324     catch err
325         options = optimset(...
326             'MaxFunEvals',20000,'algorithm','sqp',...
327             'MaxIter',5000,'Display','off' ...
328         );
329         C = zeros(5,1);
330         [C, fval] = fmincon(@(C) PressureDrop_residual(...
331             C, X1, X2, X3, X4, ΔP, weigh ...
332         ), C, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
333     end
334     ΔP_est = PressureDrop(C, X1, X2, X3, X4);
335     dev = ΔP_est - ΔP;
336     r2_ΔP = 1 - sum(dev.^2)/sum((ΔP - mean(ΔP)).^2);
337     if r2_ΔP < 0.9
338         try
339             options = optimset(...
340                 'MaxFunEvals',20000,'algorithm','sqp',...
341                 'MaxIter',5000,'Display','off' ...
```

```
342             );
343             C2 = zeros(5,1);
344             [C2, fval2] = fmincon(@(C) PressureDrop_residual(...
345                 C, X1, X2, X3, X4, ΔP, weigh ...
346             ), C2, [], [], [], [], [0 1 0], [Inf 2 1],[],options);
347             if fval2 < fval
348                 C = C2;
349                 fval = fval2;
350                 ΔP_est = PressureDrop(C, X1, X2, X3, X4);
351                 dev = ΔP_est - ΔP;
352                 r2_ΔP = 1 - sum(dev.^2)/sum(...
353                     (ΔP - mean(ΔP)).^2 ...
354                 );
355             end
356         end
357         if r2_ΔP < 0.9
358             try
359                 options = optimset(...
360                     'MaxFunEvals',20000,...
361                     'algorithm','interior-point',...
362                     'MaxIter',5000,'Display','off' ...
363                 );
364                 C3 = zeros(5,1);
365                 [C3, fval3] = fmincon(@(C) PressureDrop_residual(...
366                     C, X1, X2, X3, X4, ΔP, weigh ...
367                 ), C3, [], [], [], [], [0 1 0], [...
```

```
368                     Inf 2 1 ...
369                 ],[],options);
370                 if fval3 < fval
371                     C = C3;
372                     fval = fval3;
373                     ΔP_est = PressureDrop(C, X1, X2, X3, X4);
374                     dev = ΔP_est - ΔP;
375                     r2_ΔP = 1 - sum(dev.^2)/sum(...
376                         (ΔP - mean(ΔP)).^2 ...
377                     );
378                 end
379             end
380         end
381     end
382     C(4,1) = -1;
383     X = [];
384     X(:,1) = (X1.^C(2).*X2.^C(3).*X3.^C(4));
385     X(:,2) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X1);
386     X(:,3) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X2);
387     X(:,4) = (C(1)*X1.^C(2).*X2.^C(3).*X3.^C(4)).*log(X3);
388     COV_X = (X'*X)^(-1);
389     PipeLineClass.COV_X_ΔP = COV_X;
390     PipeLineClass.ΔP_limit = max(diag(X/(X'*X)*X'));
391     max_dev = round(max(abs(dev)));
392 end
393
```

```
% estimate the heat loss model
sign = mean(T-Tamb)/abs(mean(T-Tamb));
Δh_rated = sign*100/mdot_rated;
C_Q = [2 1]';
if ¬isempty(mdot.*(h-hout))& sum((mdot.*(h-hout)).^2)>1.e-8
    % zero the negative heat loss
    mm = length(mdot);
    i = 1;
    while i ≤ mm
        if (h(i)-hout(i))*(T(i)-Tamb(i))<0
            h(i,:) = [];
            hout(i,:) = [];
            mdot(i,:) = [];
            T(i,:) = [];
            Tamb(i,:) = [];
            i = i - 1;
            mm = mm - 1;
        end
        i = i + 1;
    end
    if length(mdot.*(h-hout)) > 1 & sum((mdot.*(h-hout)).^2)>1.e-8
        weigh = weighing_function_v000(mdot.*(h-hout), bin_num);
    elseif sum((mdot.*(h-hout)).^2)≤1.e-8
        weigh = ones(length(mdot.*(h-hout)),1);
    else
        weigh = 1;
```

```
    end
    Q = mdot.*(h-hout);
    if sum(Q.^2) > 1.e-8 & length(Q)>4
        DiffT = T-Tamb;
        max_index = find(DiffT==min(DiffT));
        Th = T(max_index);
        Tl = Tamb(max_index);
        Qmax = Q(max_index);
        Ra_max = abs(Rayleigh(Th, Tl, System.SuctionLine.dia));
        [HTC_max, n_max] = Natural_HTC_air(...
            Ra_max, System.SuctionLine.dia ...
        );
        C_Q_1_max = Qmax/((abs(Th-Tl)./Th).^n_max.* ...
            (Th-Tl)/(T(1)-Tamb(1))*mdot_rated*(h(1)-hout(1)));
        C_Q = fmincon(@(C_Q) HeatLoss(...
            C_Q, h, hout, mdot, T, Tamb, mdot_rated, Δh_rated, ...
            weigh...
        ), C_Q, [], [], [], [], [0 0.058], [...
            C_Q_1_max 0.333 ...
        ], [], optimset('GradObj', 'on', 'DerivativeCheck', 'on'));
    else
        C_Q = [0 0]'; % no heat loss or model unavailable
    end
    if ¬isempty(Q)
        Q_est = (...
            C_Q(1).*(abs(T-Tamb)./T).^C_Q(2) ...
```

```
446         ).*(T-Tamb)./(T(1,1)-Tamb(1,1))*mdot_rated*Δh_rated;
447         ΔT_rated = T(1)-Tamb(1);
448         dev_Q = (Q_est - Q)./Q;
449         r2_Q = 1 - sum((Q_est - Q).^2)/sum((Q - mean(Q)).^2);
450         max_dev_Q = round(max(abs(dev_Q)*100));
451         X = [];
452         X(:,1) = (abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)/(...
453             T(1)-Tamb(1) ...
454         )*mdot_rated*Δh_rated;
455         X(:,2) = C_Q(1).*(abs(T-Tamb)./T).^C_Q(2).* ...
456             (T-Tamb)/(T(1)-Tamb(1))*mdot_rated*Δh_rated.*log(...
457                 abs(T-Tamb)./Tamb ...
458             );
459         COV_X = covariance_adj(X);
460         PipeLineClass.COV_X_Q = COV_X;
461         PipeLineClass.Q_limit = max(diag(X*COV_X*X'));
462     else
463         C_Q = [0 0]'; % no heat loss or model unavailable
464         Δh_rated = 0;
465         ΔT_rated = 0;
466         dev_Q = 0;
467         Q = 0;
468         Q_est = 0;
469         r2_Q = -9999;
470         PipeLineClass.COV_X_Q = zeros(length(C_Q),length(C_Q));
471         PipeLineClass.Q_limit = Inf;
```

```
472     end
473 else
474     C_Q = [0 0];
475     Δh_rated = 0;
476     ΔT_rated = 0;
477     dev_Q = 0;
478     Q = 0;
479     Q_est = 0;
480     r2_Q = -9999;
481     PipeLineClass.COV_X_Q = zeros(length(C_Q),length(C_Q));
482     PipeLineClass.Q_limit = Inf;
483 end
484
485 PipeLineClass.mdot_rated = mdot_rated;
486 PipeLineClass.V_rated = V_rated;
487 PipeLineClass.rho_rated = rho_rated;
488 PipeLineClass.C = C;
489 PipeLineClass.C_Q = C_Q;
490 PipeLineClass.Δh_rated = Δh_rated;
491 PipeLineClass.ΔT_rated = ΔT_rated;
492
493 save(savefilename);
494 movefile(savefilename,strcat(foldername,'\'));
495
496 end
497
```

```
498 function residual = Comp_inlet(...
499     hin_guess, data_HG, Tamb, Comp_para, Comp_func, refri ...
500 )
501 [mr W hout residual] = Comp_func(data_HG(1,1), hin_guess,...
502                 data_HG(1,2), Tamb, data_HG(1,10), Comp_para, refri);
503 end
504
505 function [residual_Q, grad_Q] = HeatLoss(...
506     C_Q, h, hout, mdot, T, Tamb, Δh_rated, mdot_rated, weigh ...
507 )
508     residual_Q = (h-hout).*mdot/mdot_rated/(Δh_rated) - ...
509         C_Q(1).*(abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1));
510     n_Q = length(residual_Q);
511     gra_Q_ori(1:n_Q,1) = -2.*(residual_Q).*1./weigh;
512     residual_Q = sum(1./weigh.*residual_Q.^2);
513     grad_Q(1,1) = sum(gra_Q_ori(1:n_Q,1).*(...
514         1.*(abs(T-Tamb)./T).^C_Q(2).*(T-Tamb)./(T(1)-Tamb(1)) ...
515     ));
516     grad_Q(2,1) = sum(gra_Q_ori(1:n_Q,1).*(...
517         C_Q(1).*(T-Tamb)./(T(1)-Tamb(1)).*log(abs(T-Tamb)./T).* ...
518         (abs(T-Tamb)./T).^C_Q(2) ...
519     ));
520 end
521
522 function residual = PressureDrop_residual(...
523     C, X1, X2, X3, X4, ΔP, weigh ...
```

```
)
n = length(X1);
residual = 0;
for i = 1:n
    residual = residual + weigh(i,1)* ...
        ((ΔP(i,1) - PressureDrop(...
            C, X1(i,1), X2(i,1), X3(i,1), X4(i,1) ...
        ))).^2;
end
residual = residual/n;
end

function ΔP_est = PressureDrop(C, X1, X2, X3, X4)
% function to estimate the pressure drop
ΔP_est = (C(1)*X1.^C(2).*X2.^C(3)./X3);
ΔP_est = real(ΔP_est);
end
```

N.10 Pre-tuning Simulation

```

% m-file to pass system parameter for pre-tuning simulation
% and charge tuning

% clear existing data
clear;
clc;
```

```
close all;

summary_m_fil = 'para_acquistion_v083_main';
% generate the file if it does not exists
if ¬exist([summary_m_fil,'.mat'],'file')
    summary_func = str2func(['@()',summary_m_fil,'()']);
    summary_func();
end
raw = open([summary_m_fil,'.mat']);
version_number = '211';
old_version_number = '206';
load_old = 0;
testing = 0;
func_version = '175';
cycle_name = 'cycle_solver_v';
mainfolder = strcat(cycle_name,version_number);
libname = 'lib';
cycle_func = str2func([...
    '@(System_main, libname, version_number, mainfolder, ',...
    'load_old, old_version_number)',...
    strcat(cycle_name,func_version),...
    '(System_main, libname, version_number, mainfolder, ',...
    'load_old, old_version_number)' ...
]);
savefilename = strcat(cycle_name,version_number,'.mat');
mkdir(mainfolder);
```

```
34
35 % setting up parallel computing
36 if ¬load_old
37     try
38         matlabpool open 3
39     catch err
40         clear err
41         matlabpool close
42         matlabpool open 3
43     end
44 end
45
46 try
47     unloadlibrary lib
48 catch err
49     clear err
50 end
51 if ¬load_old | testing
52     try
53     pctRunOnAll loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
54     catch err
55     loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
56     end
57 else
58     loadlibrary 'Matlab_DLL.dll' 'Matlab_def.h' alias lib
59 end
```

```
60
61 System_main = raw.System;
62 try
63     parfor qq = 1:raw.n
64         System_main(qq) = 1; % a statement leads to error
65     end
66 catch err
67 end
68 r2_struct.r2_tuned_full = -9999;
69 r2_struct.r2_tuned_UA_tp = -9999;
70 r2_struct.r2_tuned_UA_oil = -9999;
71 r2_struct.r2_tuned_UA = -9999;
72 r2_struct.r2_tuned_w = -9999;
73 r2_struct.r2_tuned_oil = -9999;
74 r2_struct.r2_Q = -9999;
75 r2_struct.r2_SHR = -9999;
76 r2_struct.r2_Q_cond = -9999;
77 r2_struct.r2_W = -9999;
78 r2_struct.r2_Q_ref = -9999;
79 r2_struct.r2_SHR_ref = -9999;
80 r2_struct.r2_Q_cond_ref = -9999;
81 for i = 1:raw.n
82 % for i = 8:8
83     error_statement(i) = err;
84     r2(i) = r2_struct;
85     System_main(i).ig_coeff.P_sat_comp_out_coeff = zeros(7,1);
```

```
86      System_main(i).ig_coeff.P_sat_comp_in_coeff = zeros(7,1);
87      System_main(i).ig_coeff.SH_evap_coeff = zeros(7,1);
88      System_main(i).ig_coeff.ΔP_High_coeff = zeros(7,1);
89      System_main(i).ig_coeff.ΔP_Low_coeff = zeros(7,1);
90      System_main(i).ig_coeff.SC_EEV_coeff = zeros(7,1);
91      System_main(i).ig_coeff.T_out_coeff = zeros(7,1);
92      System_main(i).Comp.para.chargeb = zeros(7,1);
93      System_main(i).Comp.para.chargeb_full = zeros(10,1);
94      System_main(i).Comp.para.chargeb_UA_oil = zeros(8,1);
95      System_main(i).Comp.para.chargeb_UA = zeros(7,1);
96      System_main(i).Comp.para.chargeb_w = zeros(4,1);
97      System_main(i).Comp.para.chargeb_oil = zeros(5,1);
98  end
99
100 for i = 1:raw.n
101 % for i = 2:raw.n
102     try
103         [r2_m System_main(i)] = cycle_func(...
104             raw.System(i), libname, version_number, mainfolder, ...
105             load_old, old_version_number ...
106         );
107         r2(i) = r2_m;
108     catch err
109         disp(['Problem found in system ',raw.System(i).name]);
110         error_statement(i) = err;
111     end
```

```
112     close all;
113 end
114 
115 try
116     matlabpool close
117 catch err
118 end
119 unloadlibrary lib;
120 save(savefilename);
121 movefile(savefilename,[pwd,'\',mainfolder]);
```

```
1  function [r2 System_main System] = cycle_solver_v175(...
2      System_main, libname, version_number, mainfolder, load_old, ...
3      old_version_number ...
4  )
5  
6  % Function to execute pre-tuning simulation and pass results
7  % to charge tuning function
8  
9  % Reading data and importing library
10 system_name = System_main.name;
11 filename = System_main.filename;
12 worksheetname = System_main.worksheetname;
13 Standard_Airflow = System_main.Standard_Airflow_evap;
14 Ind_Fan_Upstream = System_main.Fan_Upstream_evap;
```

```
15 Ind_Fan_Upstream_cond = System_main.Fan_Upstream_cond;
16 Cond_standard_flow = System_main.Cond_standard_flow;
17 Pc = System_main.Pc;
18 Tc = System_main.Tc;
19 version = strcat(System_main.name,'_',version_number);
20 Q_gain_version = '030';
21 Q_gain_name = [...
22     'Q_gain_observation_v',System_main.name,'_',Q_gain_version ...
23 ];
24 Q_gain_raw = open([...
25     'Q_gain_observation_v',Q_gain_version,'\',Q_gain_name,'\',...
26     Q_gain_name,'.mat' ...
27 ]);
28 k = length(Q_gain_raw.index_removed);
29 data = Q_gain_raw.data;
30 code = Q_gain_raw.code;
31 fault = Q_gain_raw.fault;
32 Q_cond_r = Q_gain_raw.Q_cond_r;
33 Q_evap_r = Q_gain_raw.Q_evap_r;
34 SHR = Q_gain_raw.SHR;
35 for i = k:-1:1 % reverse for correct indexing
36     data(Q_gain_raw.index_removed(i),:) = [];
37     fault(Q_gain_raw.index_removed(i),:) = [];
38     code(Q_gain_raw.index_removed(i),:) = [];
39     Q_cond_r(Q_gain_raw.index_removed(i),:) = [];
40     Q_evap_r(Q_gain_raw.index_removed(i),:) = [];
```

```
    SHR(Q_gain_raw.index_removed(i),:) = [];
end
[m n] = size(data);
cell_VL = strfind(fault,'VL');
if isempty(cell_VL)
    cell_VL = cell(pp,1);
end
cell_NC = strfind(fault,'NC');
if isempty(cell_NC)
    cell_NC = cell(pp,1);
end
cell_LL = strfind(fault,'LL');
if isempty(cell_LL)
    cell_LL = cell(pp,1);
end
i = 1;
while i ≤ m
    if ¬isempty(cell2mat(cell_LL(i))) || ...
        ¬isempty(cell2mat(cell_VL(i))) || ...
        ¬isempty(cell2mat(cell_NC(i)))
        data(i,:) = [];
        fault(i,:) = [];
        code(i,:) = [];
        cell_LL(i,:) = [];
        cell_VL(i,:) = [];
        cell_NC(i,:) = [];
```

```
67          Q_cond_r(i,:) = [];
68          Q_evap_r(i,:) = [];
69          SHR(i,:) = [];
70          i = i - 1;
71          m = m - 1;
72      end
73      i = i + 1;
74  end
75  [m n] = size(data);
76
77  % setting directory
78  mainfoldername = strcat('cycle_solver_',version_number);
79  name = strcat(mainfoldername,'_',version);
80  savefilename = strcat(name,'.mat');
81  refri = System_main.refname;
82  path_name = strcat(pwd,'\',mainfolder,'\',name,'\');
83  mkdir(path_name);
84
85  % setting old directory
86  old_mainfoldername = strcat('cycle_solver_',old_version_number);
87  old_version = strcat(System_main.name,'_',old_version_number);
88  old_name = strcat(old_mainfoldername,'_',old_version);
89  old_savefilename = strcat(old_name,'.mat');
90  refri = System_main.refname;
91  old_mainfolder = strcat('cycle_solver_v',old_version_number);
92  old_path_name = strcat(pwd,'\',old_mainfolder,'\',old_name);
```

```
93 % mkdir(path_name);
94
95 % Component definition
96 Comp = System_main.Comp;
97 HotGas = System_main.HotGas;
98 Cond = System_main.Cond;
99 LiquidLine = System_main.LiquidLine;
100 EXV = System_main.EXV;
101 Evap = System_main.Evap;
102 SuctionLine = System_main.SuctionLine;
103 Accumulator_stat = System_main.Accumulator;
104
105 Q_evap_ref = zeros(m,1);
106 Q_cond_ref = zeros(m,1);
107 SHR_ref = zeros(m,1);
108
109 % Start calculation
110
111 % check SC
112 RES_CHECK = 1;
113 display(['Starting ',system_name]);
114 if load_old == 0
115     % filter out 20 cases (10 cases with the highest Q_evap
116     % and 10 cases with the smallest Q_evap)
117
118     % compute the original case
```

```
119     disp('Start calculating the original case');
120     [residual_opt System time_CPU ...
121         flag error_statement err_indicator ...
122         residual SH, SH_comp SC SC_EEV Q_evap ...
123         Q_cond Guess_ori_bulk EXV_status_ind] = Cycle_iterate(...
124             data, [], [], [], [], Q_evap_r, ...
125             Q_cond_r, SHR, Comp, HotGas, Cond, ...
126             LiquidLine, EXV, Evap, SuctionLine, ...
127             Accumulator_stat, Ind_Fan_Upstream, refri, Tc, ...
128             Pc, libname, system_name, [] ...
129         );
130     for i = 1:m
131         Q_evap_sim(i,1) = System(i).Evap.Q;
132         System(i).Accumulator_stat = Accumulator_stat;
133     end
134     dev_Q = (Q_evap_sim - Q_evap)./Q_evap;
135     % prepare another data matrix for sorting
136     for i = 1:m
137         residual_pack(i,1) = norm(residual(i,:));
138     end
139     mean_dev_Q = mean(dev_Q(find(residual_pack < 1.e-3)));
140     m_W_para_multiplier = [1 1 1 1]';
141     disp('No tuning needed, proceeded to charge tuning');
142 else
143     new_path_name = path_name;
144     new_savefilename = savefilename;
```

```
145     new_version = version;
146     load([old_path_name,'\',old_savefilename]);
147     path_name = new_path_name;
148     savefilename = new_savefilename;
149     version = new_version;
150     clear new_path_name new_savefilename new_version;
151 end
152 display(['Finishing simulation on ',system_name]);
153 save(savefilename);
154 
155 SC_sim = zeros(m,1);
156 SC_EEV_sim = zeros(m,1);
157 SH_sim = zeros(m,1);
158 SH_comp_sim = zeros(m,1);
159 Q_evap_sim = zeros(m,1);
160 Q_cond_sim = zeros(m,1);
161 SHR_sim = zeros(m,1);
162 for i = 1:m
163     try % to avoid error_cases
164         data_sim(i,1) = System(i).Comp.Pin;
165         data_sim(i,2) = System(i).Comp.Pout;
166         data_sim(i,3) = System(i).Cond.Pin;
167         data_sim(i,4) = System(i).Cond.Pout;
168         data_sim(i,5) = System(i).EXV.Pin;
169         data_sim(i,6) = System(i).EXV.Pout;
170         data_sim(i,7) = System(i).Evap.Pin;
```

```
171         data_sim(i,8) = System(i).Evap.Pout;
172         data_sim(i,9) = propertyRH(...
173             'T','P',data_sim(i,1),'H',System(i).Comp.hin,refri ...
174         );
175         data_sim(i,10) = propertyRH(...
176             'T','P',data_sim(i,2),'H',System(i).Comp.hout,refri ...
177         );
178         data_sim(i,11) = propertyRH(...
179             'T','P',data_sim(i,3),'H',System(i).Cond.hin,refri ...
180         );
181         data_sim(i,12) = propertyRH(...
182             'T','P',data_sim(i,4),'H',System(i).Cond.hout,refri ...
183         );
184         data_sim(i,13) = propertyRH(...
185             'T','P',data_sim(i,5),'H',System(i).EXV.hin,refri ...
186         );
187         data_sim(i,14) = propertyRH(...
188             'T','P',data_sim(i,6),'H',System(i).EXV.hout,refri ...
189         );
190         data_sim(i,15) = propertyRH(...
191             'T','P',data_sim(i,7),'H',System(i).Evap.hin,refri ...
192         );
193         data_sim(i,16) = propertyRH(...
194             'T','P',data_sim(i,8),'H',System(i).Evap.hout,refri ...
195         );
196         EXV_status_ind(i,1) = EXV_status(...
```

```
197             System(i), EXV, Pc, Tc, refri ...
198         );
199         data_sim(i,23) = System(i).mdot_r;
200         data_sim(i,24) = System(i).Comp.W;
201         SC_sim(i,1) = System(i).CondenserSC;
202         SC_EEV_sim(i,1) = System(i).LiquidLineSC;
203         SH_sim(i,1) = System(i).EvaporatorSH;
204         SH_comp_sim(i,1) = System(i).CompressorSH;
205         Q_evap_sim(i,1) = System(i).Evap.Q;
206         Q_cond_sim(i,1) = System(i).Cond.Q;
207         SHR_sim(i,1) = System(i).Evap.SHR;
208     end
209 end
210 save(savefilename);
211 W_comp = data(:,24);
212 W_comp_sim = data_sim(:,24);
213 [Charge_mea Charge_mea_w Charge_mea_UA_oil Charge_predicted_w ...
214     b_w Charge_predicted_UA_oil b_UA_oil UA_oil_range r2 ...
215     b_w_para b_UA_oil_para] = charge_tuning_v075(...
216         System, flag(1:m,:), data(1:m,:), data_sim, 1, ...
217         Q_evap(1:m,:), Q_evap_ref(1:m,:), SHR(1:m,:), ...
218         SHR_ref(1:m,:), W_comp(1:m,:), Q_cond(1:m,:), ...
219         Q_cond_ref(1:m,:), SC(1:m,:), SC_EEV(1:m,:), ...
220         SH_comp(1:m,:), Q_evap_sim, SHR_sim, W_comp_sim, ...
221         Q_cond_sim, SC_sim, SC_EEV_sim, SH_comp_sim,...
222         version,residual(1:m,:), ...
```

```
223         fault(1:m,:),refri ...
224     );
225
226 movefile(strcat('*_v',version,'.png'),path_name);
227 % close all; % prevent too many diagrams
228 System_main.Comp.para.chargeb = zeros(7,1);
229 System_main.Comp.para.chargeb_full = zeros(7,1);
230 System_main.Comp.para.chargeb_UA_oil = b_UA_oil;
231 System_main.Comp.para.UA_oil_range = UA_oil_range;
232 System_main.Comp.para.chargeb_UA = zeros(7,1);
233 System_main.Comp.para.chargeb_w = b_w;
234 System_main.Comp.para.chargeb_oil = zeros(7,1);
235 System_main.Comp.para.chargeb_UA_tp = zeros(7,1);
236 System_main.Comp.para.b_w_para = b_w_para;
237 System_main.Comp.para.b_UA_oil_para = b_UA_oil_para;
238 save(savefilename);
239
240 % calculate initial guess equation
241 gd_pt = find(residual_pack<1.e-5);
242 X = [...
243     ones(m,1) data(:,17) data(:,19) data(:,30) data(:,21) ...
244     data(:,29) data(:,31) ...
245 ];
246 System_main.ig_coeff.P_sat_comp_out_coeff = ...
247     X(gd_pt,:)\data(gd_pt,2);
248 System_main.ig_coeff.P_sat_comp_in_coeff = ...
```

```
249     X(gd_pt,:)\data(gd_pt,8);
250 System_main.ig_coeff.SH_evap_coeff = X(...
251     residual_pack<1.e-5&SH>1,: ...
252 )\SH(residual_pack<1.e-5&SH>1,1);
253 System_main.ig_coeff.ΔP_High_coeff = X(gd_pt,:)\(...
254     data(gd_pt,2)-data(gd_pt,5) ...
255 );
256 System_main.ig_coeff.ΔP_Low_coeff = X(gd_pt,:)\(...
257     data(gd_pt,6)-data(gd_pt,8) ...
258 );
259 System_main.ig_coeff.SC_EEV_coeff = X(...
260     residual_pack<1.e-5&SC_EEV>1,: ...
261 )\SC_EEV(residual_pack<1.e-5&SC_EEV>1,1);
262 System_main.ig_coeff.T_out_coeff = X(gd_pt,:)\data(gd_pt,10);
263 System_main.ig_coeff.mdot_r = X(gd_pt,:)\data(gd_pt,23);
264
265 P_out_regress = X*System_main.ig_coeff.P_sat_comp_out_coeff;
266 P_in_regress = X*System_main.ig_coeff.P_sat_comp_in_coeff;
267 SH_regress = X*System_main.ig_coeff.SH_evap_coeff;
268 T_out_regress = X*System_main.ig_coeff.T_out_coeff;
269 mdot_r_regress = X*System_main.ig_coeff.mdot_r;
270
271 save(savefilename);
272 % close all;
273 display(['Finishing charge tuning on ',system_name]);
274 movefile(savefilename,path_name);
```

```
275 end
276
277 function [residual_opt System time_CPU ...
278         flag error_statement err_indicator ...
279         residual SH, SH_comp SC SC_EEV Q_evap ...
280         Q_cond Real_final EXV_status_ind] = Cycle_iterate(data, ...
281         para_multiplier, W_para, norm_m_para, norm_W_para, ...
282         Q_evap_r, Q_cond_r, SHR, Comp, HotGas, Cond, ...
283         LiquidLine, EXV, Evap, SuctionLine, Accumulator_stat,...
284         Ind_Fan_Upstream, refri, Tc, Pc, libname, system_name, ...
285         Guess_ori)
286 m = size(data,1);
287 % declaration of variables to be output after parfor
288 DP_cond_change = zeros(m,2);
289 DP_evap_change = zeros(m,2);
290 iter_f.iter = zeros(3,1);
291 dev = zeros(m,1);
292 flag = zeros(m,1);
293 dev2 = zeros(m,1);
294 EXV_status_ind = zeros(m,1);
295 for i = 1:m
296     System(i) = System_definition();
297 end
298 residual = ones(m,5)*9999;
299 residual_pack = ones(m,1)*9999;
300 Q_evap = zeros(m,1);
```

```
301 SC = zeros(m,1);
302 SH = zeros(m,1);
303 SC_EEV = zeros(m,1);
304 SH_comp = zeros(m,1);
305 Q_cond = zeros(m,1);
306 try
307     parfor qq = 1:raw.n
308         System(qq) = 1; % a statement leads to error
309     end
310 catch err
311 end
312 for i = 1:m
313     error_statement(i) = err;
314 end
315 clear err;
316 err_indicator = zeros(m,1);
317 time_CPU = zeros(m,1);
318 % if isempty(Guess_ori)
319 Real_final = zeros(m,5);
320 % end
321 parfor i = 1:m
322 % for i = 1:m
323     try
324         display([...
325             strcat(...
326                 'Starting case C',int2str(i),'/',int2str(m) ...
```

```
327             ), ' in system ',system_name ...
328         ]);
329         data_II = data(i,:);
330 
331         % Re-calculating the loads for input variables
332         airinlet = zeros(1,5);
333         Pain = data_II(32);
334         Tain_cond = data_II(21);
335         airinlet_cond = calllib(...
336             libname, 'HumAir_DLL', Tain_cond, Pain, ...
337             1, 230, airinlet ...
338         );
339         wain_cond = airinlet_cond(2);
340         Tain_evap = data_II(17);
341         Taout_evap = data_II(18);
342         Dain_evap = data_II(19);
343         Daout_evap = data_II(20);
344         Tamb = Tain_cond;
345         airinlet = calllib(...
346             libname, 'HumAir_DLL', Tain_evap, Pain, ...
347             1, Dain_evap, airinlet ...
348         );
349         airoutlet = calllib(...
350             libname, 'HumAir_DLL', Taout_evap, Pain, ...
351             1, Daout_evap, airinlet ...
352         );
```

```
353         SH(i,1) = data_II(16) - ...
354             propertyRH('T','P',data_II(8),'Q',1,refri);
355         SH_comp(i,1) = data_II(9) - ...
356             propertyRH('T','P',data_II(1),'Q',1,refri);
357         SC(i,1) = propertyRH('T','P',data_II(4),'Q',0,refri) - ...
358             data_II(12);
359         SC_EEV(i,1) = propertyRH(...
360             'T','P',data_II(5),'Q',0,refri ...
361         )-data_II(13);
362         cpm = 1006 + 1860*airinlet(2);
363         Q_evap(i,1) = Q_evap_r(i,1);
364         Vdot_evap = data_II(30);
365         % calculate the measured SHRt
366         if Ind_Fan_Upstream
367             SHR(i,1) = (...
368                 Vdot_evap/airinlet(5)*cpm*(...
369                     Tain_evap-Taout_evap ...
370                 )+data_II(25) ...
371             )/(...
372                 Vdot_evap/airinlet(5)*(...
373                     airinlet(3)-airoutlet(3) ...
374                 )*1000+data_II(25) ...
375             );
376         else
377             SHR(i,1) = (cpm*(Tain_evap-Taout_evap))/(...
378                 (airinlet(3)-airoutlet(3))*1000 ...
```

```
379             );
380         end
381         data_II(30) = Vdot_evap;
382         % calculate the measured condenser heat transfer rate
383         Q_cond(i,1) = Q_cond_r(i,1);
384         Vdot_cond = data_II(29);
385         wain_evap = airinlet(2);
386         hf = propertyRH('H','P',data_II(1),'Q',0,refri);
387         hv = propertyRH('H','P',data_II(1),'Q',1,refri);
388         rhov = propertyRH('D','P',data_II(1),'Q',1,refri);
389         SCSH = struct(...
390             'SC',SC(i,1),'CompPin',data_II(1),...
391             'CompPout',data_II(2),'EEVPin',data_II(5),...
392             'EvapSH',SH(i,1),'Charge',data_II(31),...
393             'SC_EEV',SC_EEV(i,1),'CompTout',data_II(10),...
394             'DiscSH',data_II(10)-propertyRH(...
395                 'T','P',data_II(2),'Q',1,refri ...
396             ),'CompSH',SH_comp(i,1),...
397             'CompDout',propertyRH(...
398                 'D','T',data_II(10),'P',data_II(2),refri ...
399             ),'Comphout',propertyRH( ...
400                 'H','T',data_II(10),'P',data_II(2),refri ...
401             ),'Comphin_sat',propertyRH(...
402                 'H','P',data_II(1),'Q',1,refri ...
403             ),'CondDin',propertyRH(...
404                 'D','T',data_II(11),'P',data_II(3),refri ...
```

```
405             ),...
406             'CondQ',Q_cond(i,1),'CHANGE_RES',0,...
407             'CompTdiff',data_II(10)-data_II(9),...
408             'Pratio',data_II(2)/data_II(1),'EvapQ',Q_evap(i,1),...
409             'Accumulator',0,'System_name',system_name ...
410         );
411
412         % guesses for data known cases
413         Guess = zeros(5,1);
414
415         if isempty(Guess_ori)
416             Guess(1,1) = (...
417                 Tain_evap - propertyRH(...
418                     'T','P',data_II(8),'Q',0,refri ...
419                 ) ...
420             )/(data_II(17)-250);
421             Guess(2,1) = (...
422                 propertyRH(...
423                     'T','P',data_II(2),'Q',0,refri ...
424                 )-Tain_cond ...
425             )/(Tc - Tain_cond);
426             Guess(3,1) = (...
427                 data_II(16) - propertyRH(...
428                     'T','P',data_II(8),'Q',1,refri ...
429                 ) ...
430             )/6.0;
```

```
431             Guess(4,1) = (data_II(10) - Tain_cond)/(...
432                 Tc - Tain_cond ...
433             );
434             Guess(5,1) = data_II(23)/Cond.para.m_r;
435             if Guess(5,1) < 0
436                 if SH_comp(i,1) > 1
437                     data_II(23) = Compressor(...
438                         data_II(1), propertyRH(...
439                             'H','T',data_II(9),'P',...
440                             data_II(1),refri ...
441                         ), data_II(2), Tamb, data_II(16), ...
442                         Comp.para, refri ...
443                     );
444                 else
445                     data_II(23) = Compressor(...
446                         data_II(1), propertyRH(...
447                             'H','P',data_II(1),'Q',1,refri ...
448                         ), data_II(2), Tamb, data_II(16), ...
449                         Comp.para, refri ...
450                     );
451                 end
452                 Guess(5,1) = data_II(23)/Cond.para.m_r;
453             end
454         else
455             Guess = Guess_ori(i,:)';
456         end
```

```
457         Guess_ori_II = Guess;
458
459         % setting up the solver
460         k = 0;
461         iter_detail = 0;
462         residual_II = ones(1,5)*Inf;
463         Guess(4,1) = Guess(4,1)*0.5;
464         % solve for a good temperature difference
465         DP_cond = [];
466         DP_evap = [];
467         oldcputime = cputime;
468         System_II = System_definition();
469         while (sqrt(mean(residual_II.^2))>5e-4) && k < 4
470             if k > 0
471                 Guess(3,1) = Guess(3,1) + 0.2;
472                 if k > 1
473                     Guess = Guess.*(rand(length(Guess),1)-1);
474                     if Guess(1,1) < 0.1
475                         Guess(1,1) = 0.1;
476                     end
477                     if Guess(2,1) < 0.1
478                         Guess(2,1) = 0.1;
479                     end
480                     if Guess(2,1) > 0.95
481                         Guess(2,1) = 0.95;
482                     end
```

```
483                     if Guess(4,1) < 0;
484                         Guess(4,1) = 0;
485                     end
486                 end
487             end
488             try
489                 [...
490                     System_II,Real,dev2(i,1),residual_II,...
491                     flag(i,1),func_output,err ...
492                 ]  = cycle_output(...
493                     Guess, Tain_cond, wain_cond, Vdot_cond, ...
494                     Tain_evap, wain_evap, Vdot_evap, Tamb, ...
495                     Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
496                     EXV, Evap, SuctionLine, Accumulator_stat, ...
497                     refri, Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, ...
498                     libname ...
499                 );
500                 iter_detail = iter_detail + func_output.iterations;
501             catch err
502                 residual_II = ones(length(Guess),1)*9999;
503                 Real = ones(length(Guess),1)*9999;
504             end
505             k = k + 1;
506         end
507         % recalculate if the accumulator assumption is violated
508         if Accumulator_stat == 1 & (...
```

```
509                 System_II.CompressorSH ≤ 0 | ...
510                 sqrt(mean(residual_II.^2))>5e-4 | ...
511                 isnan(residual_II) ...
512             )
513             SCSH.Accumulator = 1;
514             [...
515                 System_II,Real,dev2(i,1),residual_II,flag(i,1),...
516                 func_output,err ...
517             ]  = cycle_output(...
518                 Guess_ori_II, Tain_cond, wain_cond, Vdot_cond, ...
519                 Tain_evap, wain_evap, Vdot_evap, Tamb, Pain, ...
520                 SCSH, Comp, HotGas, Cond, LiquidLine, EXV, ...
521                 Evap, SuctionLine, Accumulator_stat, refri, ...
522                 Tc, Pc, hf, hv, rhov, DP_cond, DP_evap, libname ...
523             );
524         end
525         time_CPU(i,1) = cputime-oldcputime;
526         System(i) = System_II;
527         Real_final(i,:) = Real';
528
529         % iteration record
530         iter_f(i).iter(1) = iter_detail/k;
531         iter_f(i).iter(2) = k;
532         iter_f(i).iter(3) = iter_detail;
533
534         % dimensionalize the residual to a temperature for checking
```

```
535         dev(i) = (System(i).SuctionLine.hin-System(i).Evap.hout) /...
536             propertyRH('C','P',System(i).Comp.Pin,'Q',1,refri);
537         data(i,:) = data_II; % feedback
538         residual(i,:) = residual_II;
539         residual_pack(i,1) = norm(residual(i,:));
540         display([strcat(...
541             'Finishing case C',int2str(i),'/',int2str(m) ...
542         ),' in system ',system_name]);
543         display(['with residual ',num2str(residual_pack(i,1))]);
544     catch err
545         time_CPU(i,1) = Inf;
546         flag(i,1) = 9999;
547         error_statement(i) = err;
548         err_indicator(i,1) = 1;
549     end
550 end
551 for i = 1:m
552     Q_evap_sim(i,1) = System(i).Evap.Q;
553     Q_cond_sim(i,1) = System(i).Cond.Q;
554     W_comp_sim(i,1) = System(i).Comp.W;
555     mdot_r_sim(i,1) = System(i).mdot_r;
556     SHR_sim(i,1) = System(i).Evap.SHR;
557     Comp_Pout(i,1) = System(i).Comp.Pout;
558     Comp_Pin(i,1) = System(i).Comp.Pin;
559 end
560     residual_opt(1,1) = sum((log(residual_pack+1)+1).* ...
```

```
561         ((Q_evap_sim-Q_evap)./Q_evap).^2);
562     residual_opt(2,1) = sum((log(residual_pack+1)+1).* ...
563         ((W_comp_sim-data(:,24))./data(:,24)).^2);
564     residual_opt(3,1) = sum((log(residual_pack+1)+1).* ...
565         ((Q_cond_sim-Q_cond)./Q_cond).^2);
566     residual_opt(4,1) = sum((log(residual_pack(find(SC>3))+1)+1).*...
567         ((...
568             mdot_r_sim(find(SC>3))-data(find(SC>3),23) ...
569         )./mdot_r_sim(find(SC>3))).^2);
570     residual_opt(5,1) = sum(...
571         (log(residual_pack+1)+1).*((SHR_sim-SHR)./SHR).^2 ...
572     );
573 end
574
575 % The functions below are different from the final simulation functions
576 % because it is a pre-tuning simulation
577
578 function [System,Real,dev2,residual_pre,flag,func_output,err] = ...
579     cycle_output(...
580         Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
581         wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas,
582         Cond, LiquidLine, EXV, Evap, SuctionLine, ...
583         Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, DP_cond, ...
584         DP_evap, libname ...
585     )
586 TolX = 1.e-6;
```

```
587 TolFun = 1.e-8;
588 % solve for a good temperature difference
589 try
590     Options = optimset(...
591         'Display','off','TolX',TolX,'TolFun',TolFun,...
592         'MaxFunEvals',1400,'MaxIter',600 ...
593     );
594     [Real,residual_pre,flag,func_output]  = fsolve(...
595         @(x) residual_maker(...
596             x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
597             wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
598             HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
599             Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, ...
600             DP_cond, DP_evap, libname ...
601         ),Guess,Options ...
602     );
603     dev2 = norm(residual_pre);
604     if sqrt(mean(residual_pre.^2)) > 1.e-4
605         Options = optimset(...
606             'Display','off','TolX',TolX,'TolFun',TolFun,...
607             'Algorithm','trust-region-reflective',...
608             'MaxFunEvals',1400,'MaxIter',600 ...
609         );
610         [Real,dev2,residual_pre,flag,func_output] = lsqnonlin(...
611             @(x) residual_maker(...
612                 x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
```

```
613                 wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
614                 HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
615                 Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, ...
616                 DP_cond, DP_evap, libname ...
617             ),Guess,[0.03 0.03 -Inf 0 1.e-8],[2 0.95 40/6 Inf Inf],...
618             Options ...
619         );
620     end
621     if sqrt(mean(residual_pre.^2)) > 1.e-4
622         SCSH.CHANGE_RES = 1;
623         [Real,dev2,residual_pre,flag,func_output] = lsqnonlin(...
624             @(x) residual_maker(...
625                 x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
626                 wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
627                 HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
628                 Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, ...
629                 DP_cond, DP_evap, libname ...
630             ),Guess,[0.03 0.03 -Inf 0 1.e-8],[...
631                 2 0.95 40/6 Inf Inf...
632             ],Options ...
633         );
634     end
635     err = func_output.message;
636 catch err
637     Options = optimset(...
638         'Display','off','TolX',TolX,'TolFun',TolFun,...
```

```
639         'Algorithm','trust-region-reflective',...
640         'MaxFunEvals',1400,'MaxIter',600 ...
641     );
642     [Real,dev2,residual_pre,flag,func_output]  = lsqnonlin(...
643         @(x) residual_maker(...
644             x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
645             wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
646             HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
647             Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, ...
648             DP_cond, DP_evap, libname ...
649         ),Guess,[0.03 0.03 -Inf 0 1.e-8],[2 0.95 40/6 Inf Inf],...
650         Options ...
651     );
652     if sqrt(mean(residual_pre.^2)) > 1.e-4
653         SCSH.CHANGE_RES = 1;
654         [Real,dev2,residual_pre,flag,func_output]  = lsqnonlin(...
655             @(x) residual_maker(...
656                 x, Tain_cond, wain_cond, Vdot_cond, Tain_evap, ...
657                 wain_evap, Vdot_evap, Tamb, Pain, SCSH, Comp, ...
658                 HotGas, Cond, LiquidLine, EXV, Evap, SuctionLine, ...
659                 Accumulator_stat, refri, Tc, Pc, hf, hv, rhov, ...
660                 DP_cond, DP_evap, libname ...
661             ),Guess,[0.03 0.03 -Inf 0 1.e-8],...
662             [2 0.95 40/6 Inf Inf],Options ...
663         );
664     end
```

```
665 end
666 [residual_pre System] = cycle_calculation(...
667     Real, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
668     Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
669     EXV, Evap, SuctionLine, Accumulator_stat, refri, Tc, Pc, hf, ...
670     hv, rhov, DP_cond, DP_evap, libname ...
671 );
672 end
673
674 function lsresidual = residual_maker(...
675     Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
676     Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
677     EXV, Evap, SuctionLine, Accumulator_stat, refri, Tc, Pc, hf, ...
678     hv, rhov, DP_cond, DP_evap, libname ...
679 )
680 [residual System] = cycle_calculation(...
681     Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
682     Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
683     EXV, Evap, SuctionLine, Accumulator_stat, refri, Tc, Pc, hf, ...
684     hv, rhov, DP_cond, DP_evap, libname ...
685 );
686 lsresidual = residual;
687 end
688
689 function [residual System] = cycle_calculation(...
690     Guess, Tain_cond, wain_cond, Vdot_cond, Tain_evap, wain_evap, ...
```

```
691     Vdot_evap, Tamb, Pain, SCSH, Comp, HotGas, Cond, LiquidLine, ...
692     EXV, Evap, SuctionLine, Accumulator_stat, refri, Tc, Pc, hf, ...
693     hv, rhov, DP_cond, DP_evap, libname ...
694 )
695
696 % resolve all guess values
697 SuctionLine.Pin = propertyRH(...
698     'P','T',Tain_evap - Guess(1)*(Tain_evap-250),'Q',0,refri ...
699 );
700 SuctionLine.hin = propertyRH(...
701     'H','P',SuctionLine.Pin,'Q',1,refri ...
702 ) + Guess(3)*6.0*propertyRH('C','P',SuctionLine.Pin,'Q',1,refri);
703 Comp.Pout = propertyRH(...
704     'P','T',Guess(2)*(Tc-Tain_cond)+Tain_cond,'Q',0,refri ...
705 );
706 Tout_guess = Guess(4)*(Tc-Tain_cond)+Tain_cond;
707 mdot_r = Guess(5)*Cond.para.m_r;
708
709 % Solve SuctionLine
710 SuctionLine.mdot_r = mdot_r;
711 if mdot_r<0
712     residual = ones(length(Guess),1)*NaN;
713     System = System_definition();
714     return;
715 end
716 [SuctionLine.Pout SuctionLine.hout SuctionLine.Charge ...
```

```
717     SuctionLine.rho_avg] = pipeline4(...
718         SuctionLine.mdot_r, SuctionLine.Pin, SuctionLine.hin, ...
719         Tamb, SuctionLine, refri, 1, libname ...
720     );
721 if SuctionLine.Pout<0
722     residual = ones(length(Guess),1)*NaN;
723     System = System_definition();
724     return;
725 end
726 
727 % other arrangements
728 Cond.Tain = Tain_cond;
729 Cond.wain = wain_cond;
730 Cond.Vdot = Vdot_cond;
731 Evap.Tain = Tain_evap;
732 Evap.wain = wain_evap;
733 Evap.Vdot = Vdot_evap;
734 
735 % Solve Compressor
736 Comp.Pin = SuctionLine.Pout;
737 Comp.hin = SuctionLine.hout;
738 try
739     [Comp.mdot_r Comp.W Comp.hout residual_Comp] = Compressor(...
740         Comp.Pin, Comp.hin, Comp.Pout, Tamb, Tout_guess, ...
741         Comp.para, refri ...
742     );
```

```
743 catch err
744     residual = ones(length(Guess),1)*NaN;
745     System = System_definition();
746     return;
747 end
748 if Comp.mdot_r<0
749     residual = ones(length(Guess),1)*NaN;
750     System = System_definition();
751     return;
752 end
753
754 % solve Hotgas line
755 HotGas.Pin = Comp.Pout;
756 HotGas.mdot_r = (Comp.mdot_r+mdot_r)/2;
757 HotGas.hin = Comp.hout;
758 HotGas.Tin = propertyRH(...
759     'T','P',HotGas.Pin,'H',HotGas.hin,refri ...
760 );
761 [HotGas.Pout HotGas.hout HotGas.Charge HotGas.rho_avg] = ...
762     pipeline4(...
763         HotGas.mdot_r, HotGas.Pin, HotGas.hin, Tamb, ...
764         HotGas, refri, 1, libname ...
765     );
766 if HotGas.Pout<0
767     residual = ones(length(Guess),1)*NaN;
768     System = System_definition();
```

```
769     return;
770 end
771
772 % solve Condenser
773 Cond.Pin = HotGas.Pout;
774 Cond.hin = HotGas.hout;
775 Cond.mdot_r = HotGas.mdot_r;
776 Cond.Pain = Pain;
777 [Cond.Pout Cond.hout Cond.Taout Cond.Q Cond.Charge Cond.FanPower ...
778     Cond.OtherOutput Cond.mdot_a Cond.LumpedInput Cond.rho_tp ...
779     Cond.rho_sh Cond.rho_sc] = Condenser(...
780         Cond.Pin, Cond.hin, Cond.mdot_r, Cond.Tain, Cond.wain, ...
781         Cond.Pain, Cond.Vdot, Cond.para, refri, libname ...
782 );
783 if Cond.Pout<0
784     residual = ones(length(Guess),1)*NaN;
785     System = System_definition();
786     return;
787 end
788 Cond.rho_avg = Cond.OtherOutput(5)*Cond.rho_sh + ...
789     Cond.OtherOutput(6)*Cond.rho_tp + ...
790     Cond.OtherOutput(7)*Cond.rho_sc;
791
792 % solve liquid line
793 LiquidLine.mdot_r = Cond.mdot_r;
794 LiquidLine.hin = Cond.hout;
```

```
795 LiquidLine.Pin = Cond.Pout;
796 try
797     LiquidLine.Tin = propertyRH(...
798         'T','P',LiquidLine.Pin,'H',LiquidLine.hin,refri ...
799     );
800 catch err
801     residual = length(Guess)*NaN;
802     System = System_definition();
803     return;
804 end
805 [LiquidLine.Pout LiquidLine.hout ...
806     LiquidLine.Charge LiquidLine.rho_avg] = pipeline4(...
807         LiquidLine.mdot_r, LiquidLine.Pin, LiquidLine.hin, Tamb, ...
808         LiquidLine, refri, 0, libname ...
809     );
810 if LiquidLine.Pout<0
811     residual = length(Guess)*NaN;
812     System = System_definition();
813     return;
814 end
815
816 % solve evaporator
817 Evap.hin = LiquidLine.hout;
818 Evap.hout_est = SuctionLine.hin;
819 Evap.Pain = Pain;
820 Evap.Pout = SuctionLine.Pin;
```

```
821 Evap.mdot_r = LiquidLine.mdot_r;
822 [Evap.Pin Evap.hout Evap.Taout Evap.waout Evap.Q Evap.Charge ...
823     Evap.FanPower Evap.OtherOutput Evap.mdot_a Evap.LumpedInput ...
824     Evap.SHR Evap.rho_tp Evap.rho_sh] = Evaporator(...
825         Evap.Pout, Evap.hin, Evap.hout_est, Evap.mdot_r, ...
826         Evap.Tain, Evap.wain, Evap.Pain, Evap.Vdot, Evap.para, ...
827         refri, libname ...
828     );
829 Evap.rho_avg = Evap.rho_tp*(1-Evap.OtherOutput(4)) + ...
830     Evap.rho_sh*Evap.OtherOutput(4);
831
832 % Solve EXV
833 EXV.hin = LiquidLine.hout;
834 EXV.Pin = LiquidLine.Pout;
835 EXV.hout = EXV.hin;
836 EXV.Pout = Evap.Pin;
837 if strcmp(SCSH.System_name, 'TM_040_R410A_packaged_TXV_Recip')
838     EXV.mdot_r = Comp.mdot_r;
839     EXV.OpeningStep = 1;
840     EXV.hout = EXV.hin;
841 elseif strcmp(EXV.Type,'TXV')
842     EXV.hout = EXV.hin;
843     if strcmp(EXV.Equalizer,'Ext_Evap')
844         [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
845             EXV.Pin, EXV.Pout, EXV.hin, SuctionLine.Pin, ...
846             propertyRH(...
```

```
847                 'T','P',SuctionLine.Pin,...
848                 'H',SuctionLine.hin,refri ...
849             ), EXV.para, Pc, Tc, refri ...
850         );
851     elseif strcmp(EXV.Equalizer,'Ext_Comp')
852         [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
853             EXV.Pin, EXV.Pout, EXV.hin, SuctionLine.Pout, ...
854             propertyRH(...
855                 'T','P',SuctionLine.Pout,...
856                 'H',SuctionLine.hout,refri ...
857             ), EXV.para, Pc, Tc, refri ...
858         );
859     else
860         [EXV.mdot_r EXV.OpeningStep] = Thermostatic_Expansion(...
861             EXV.Pin, EXV.Pout, EXV.hin, ...
862             EXV.Pout, propertyRH(...
863                 'T','P',SuctionLine.Pin,...
864                 'H',SuctionLine.hin,refri ...
865             ), EXV.para, Pc, Tc, refri ...
866         );
867     end
868     if EXV.mdot_r<0
869         residual = ones(length(Guess),1)*NaN;
870         System = System_definition();
871         return;
872     end
```

```
873 else
874     EXV.OpeningStep = 1;
875     EXV.hout = EXV.hin;
876     EXV.mdot_r = Fixed_Orifice(...
877         EXV.Pin, EXV.Pout, EXV.hin, EXV.para, Pc, Tc, refri ...
878     );
879     if EXV.mdot_r<0
880         residual = ones(length(Guess),1)*NaN;
881         System = System_definition();
882         return;
883     end
884 end
885
886 % Superheat and Subcooling
887 SC_final = propertyRH('T','P',Cond.Pout,'Q',0,refri) - ...
888     propertyRH('T','P',Cond.Pout,'H',Cond.hout,refri);
889 if SC_final ≤ 0
890     SC_final = -(Cond.hout - propertyRH(...
891         'H','P',Cond.Pout,'Q',0,refri ...
892     ))/propertyRH('C','P',Cond.Pout,'Q',0,refri);
893 end
894 Comp.Tin = propertyRH('T','P',Comp.Pin,'H',Comp.hin,refri);
895 Comp.SH =  Comp.Tin - propertyRH('T','P',Comp.Pin,'Q',1,refri);
896 Evap.SH = propertyRH('T','P',Evap.Pout,'H',Evap.hout,refri) - ...
897     propertyRH('T','P',Evap.Pout,'Q',1,refri);
898 LL_SC = propertyRH(...
```

```
899     'T','P',LiquidLine.Pout,'Q',0,refri ...
900 )-propertyRH('T','P',LiquidLine.Pout,'H',LiquidLine.hout,refri);
901 if LL_SC<0
902     LL_SC = -(LiquidLine.hout - propertyRH(...
903         'H','P',LiquidLine.Pout,'Q',0,refri ...
904     ))/propertyRH('C','P',LiquidLine.Pout,'Q',0,refri);
905 end
906
907 % storing in the system structure as output
908 System.mdot_r = Comp.mdot_r;
909
910 System.Comp.Pout = Comp.Pout;
911 System.Comp.hout = Comp.hout;
912 System.Comp.W = Comp.W;
913 System.Comp.Pin = Comp.Pin;
914 System.Comp.hin = Comp.hin;
915
916 System.HotGas.Charge = HotGas.Charge;
917 System.HotGas.Pin = HotGas.Pin;
918 System.HotGas.Pout = HotGas.Pout;
919 System.HotGas.rho_avg = HotGas.rho_avg;
920
921 System.Cond.Tain = Cond.Tain;
922 System.Cond.wain = Cond.wain;
923 System.Cond.Pin = Cond.Pin;
924 System.Cond.hin = Cond.hin;
```

```
925 System.Cond.Pout = Cond.Pout;
926 System.Cond.hout = Cond.hout;
927 System.Cond.Charge = Cond.Charge;
928 System.Cond.OtherOutput = Cond.OtherOutput;
929 System.Cond.LumpedInput = Cond.LumpedInput;
930 System.Cond.FanPower = Cond.FanPower;
931 System.Cond.Taout = Cond.Taout;
932 System.Cond.mdot_a = Cond.mdot_a;
933 System.Cond.Q = Cond.Q;
934 System.Cond.rho_avg = Cond.rho_avg;
935
936 System.LiquidLine.Charge = LiquidLine.Charge;
937 System.LiquidLine.hin = LiquidLine.hin;
938 System.LiquidLine.Pin = LiquidLine.Pin;
939 System.LiquidLine.Pout = LiquidLine.Pout;
940 System.LiquidLine.hout = LiquidLine.hout;
941 System.LiquidLine.rho_avg = LiquidLine.rho_avg;
942
943 System.EXV.Pin = EXV.Pin;
944 System.EXV.hin = EXV.hin;
945 System.EXV.mdot_r = EXV.mdot_r;
946 System.EXV.Pout = Evap.Pin;
947 System.EXV.hout = Evap.hin;
948 System.EXV.OpeningStep = EXV.OpeningStep;
949
950 System.Evap.Tain = Evap.Tain;
```

```
951 System.Evap.wain = Evap.wain;
952 System.Evap.Pin = Evap.Pin;
953 System.Evap.hin = Evap.hin;
954 System.Evap.Pout = Evap.Pout;
955 System.Evap.hout = Evap.hout;
956 System.Evap.Charge = Evap.Charge;
957 System.Evap.OtherOutput = Evap.OtherOutput;
958 System.Evap.LumpedInput = Evap.LumpedInput;
959 System.Evap.FanPower = Evap.FanPower;
960 System.Evap.Taout = Evap.Taout;
961 System.Evap.waout = Evap.waout;
962 System.Evap.mdot_a = Evap.mdot_a;
963 System.Evap.Q = Evap.Q;
964 System.Evap.SHR = Evap.SHR;
965 System.Evap.rho_avg = Evap.rho_avg;
966
967 System.SuctionLine.Charge = SuctionLine.Charge;
968 System.SuctionLine.hin = SuctionLine.hin;
969 System.SuctionLine.Pin = SuctionLine.Pin;
970 System.SuctionLine.Pout = SuctionLine.Pout;
971 System.SuctionLine.hout = SuctionLine.hout;
972 System.SuctionLine.rho_avg = SuctionLine.rho_avg;
973
974 System.Charge = System.SuctionLine.Charge + ...
975     System.Evap.Charge + System.LiquidLine.Charge + ...
976     System.Cond.Charge + System.HotGas.Charge;
```

```
977 System.Charge_tuned = System.Charge;
978 System.mdot_r = Comp.mdot_r;
979 System.CondenserSC = SC_final;
980 System.LiquidLineSC = LL_SC;
981 System.EvaporatorSH = Evap.SH;
982 System.CompressorSH = Comp.SH;
983
984 System.Heatloss = Comp.W + Evap.Q - Cond.Q;
985
986 System.Charge_calculate.Cond.wsh = System.Cond.OtherOutput(5);
987 System.Charge_calculate.Cond.wtp = System.Cond.OtherOutput(6);
988 System.Charge_calculate.Cond.wsc = System.Cond.OtherOutput(7);
989 System.Charge_calculate.Cond.rho_sh = Cond.rho_sh;
990 System.Charge_calculate.Cond.rho_tp = Cond.rho_tp;
991 System.Charge_calculate.Cond.rho_sc = Cond.rho_sc;
992 System.Charge_calculate.Evap.wsh = System.Evap.OtherOutput(4);
993 System.Charge_calculate.Evap.wtp = System.Evap.OtherOutput(5);
994 System.Charge_calculate.Evap.rho_sh = Evap.rho_sh;
995 System.Charge_calculate.Evap.rho_tp = Evap.rho_tp;
996 System.Charge_calculate.LiquidLine.rho_LL = 0;
997
998 System.ValveLeakage = 0;
999 System.LL_restriction = 0;
1000 System.Non_condensable = 0;
1001 System.ValveLeakFlow = 0; % new entry
1002 System.LL_extra_ΔP = 0; % new entry
```

```
System.NC_amount = 0; % new entry

% calculate residuals
residual(1,1) = (Evap.hout-SuctionLine.hin)/propertyRH(...
    'C','T',Evap.Tain,'Q',1,refri ...
)/10.0;
if strcmp(SCSH.System_name, 'TM_040_R410A_packaged_TXV_Recip')
    residual(2,1) = (SCSH.CompSH-Comp.SH)/10.0;
else
    residual(2,1) = (EXV.mdot_r - Comp.mdot_r)/Cond.para.m_r;
end
if SCSH.Accumulator == 0
    residual(3,1) = (propertyRH(...
        'D','T',HotGas.Tin,'P',HotGas.Pin,refri ...
    ) - SCSH.CompDout)/SCSH.CompDout;
else
    residual(3,1) = (Comp.hin - propertyRH(...
        'H','P',Comp.Pin,'Q',1,refri ...
    ))/propertyRH('C','T',Evap.Tain,'Q',1,refri)/10.0;
end
residual(4,1) = (residual_Comp)/10.0;
residual(5,1) = (mdot_r - Comp.mdot_r)/Cond.para.m_r;
end

function System = System_definition()
% to define the system definition for assignment
```

```
System.mdot_r = 0;

System.Comp.Pout = 0;
System.Comp.hout = 0;
System.Comp.W = 0;
System.Comp.Pin = 0;
System.Comp.hin = 0;

System.HotGas.Charge = 0;
System.HotGas.Pin = 0;
System.HotGas.Pout = 0;
System.HotGas.rho_avg = 0;

System.Cond.Tain = 0;
System.Cond.wain = 0;
System.Cond.Pin = 0;
System.Cond.hin = 0;
System.Cond.Pout = 0;
System.Cond.hout = 0;
System.Cond.Charge = 0;
System.Cond.OtherOutput = 0;
System.Cond.LumpedInput = 0;
System.Cond.FanPower = 0;
System.Cond.Taout = 0;
System.Cond.mdot_a = 0;
System.Cond.Q = 0;
```

```
1055 System.Cond.rho_avg = 0;
1056
1057 System.LiquidLine.Charge = 0;
1058 System.LiquidLine.hin = 0;
1059 System.LiquidLine.Pin = 0;
1060 System.LiquidLine.Pout = 0;
1061 System.LiquidLine.hin = 0;
1062 System.LiquidLine.rho_avg = 0;
1063
1064 System.EXV.Pin = 0;
1065 System.EXV.hin = 0;
1066 System.EXV.mdot_r = 0;
1067 System.EXV.Pout = 0;
1068 System.EXV.hout = 0;
1069 System.EXV.OpeningStep = 0;
1070
1071 System.Evap.Tain = 0;
1072 System.Evap.wain = 0;
1073 System.Evap.Pin = 0;
1074 System.Evap.hin = 0;
1075 System.Evap.Pout = 0;
1076 System.Evap.hout = 0;
1077 System.Evap.Charge = 0;
1078 System.Evap.OtherOutput = 0;
1079 System.Evap.LumpedInput = 0;
1080 System.Evap.FanPower = 0;
```

```
System.Evap.Taout = 0;
System.Evap.waout = 0;
System.Evap.mdot_a = 0;
System.Evap.Q = 0;
System.Evap.SHR = 0;
System.Evap.rho_avg = 0;

System.SuctionLine.Charge = 0;
System.SuctionLine.hin = 0;
System.SuctionLine.Pin = 0;
System.SuctionLine.Pout = 0;
System.SuctionLine.hout = 0;
System.SuctionLine.rho_avg = 0;

System.Charge = 0;
System.Charge_tuned = 0;
System.mdot_r = 0;
System.CondenserSC = 0;
System.LiquidLineSC = 0;
System.EvaporatorSH = 0;
System.CompressorSH = 0;

System.Heatloss = 0;

System.Charge_calculate.Cond.wsh = 0;
System.Charge_calculate.Cond.wtp = 0;
```

```
System.Charge_calculate.Cond.wsc = 0;
System.Charge_calculate.Cond.rho_sh = 0;
System.Charge_calculate.Cond.rho_tp = 0;
System.Charge_calculate.Cond.rho_sc = 0;
System.Charge_calculate.Evap.wsh = 0;
System.Charge_calculate.Evap.wtp = 0;
System.Charge_calculate.Evap.rho_sh = 0;
System.Charge_calculate.Evap.rho_tp = 0;
System.Charge_calculate.LiquidLine.rho_LL = 0;

System.ValveLeakage = 0;
System.LL_restriction = 0;
System.Non_condensable = 0;
System.ValveLeakFlow = 0; % new entry
System.LL_extra_ΔP = 0; % new entry
System.NC_amount = 0; % new entry
end
```

N.11 Function for Charge Tuning

```
function [Charge_mea Charge_mea_w Charge_mea_UA_oil ...
    Charge_predicted_w b_w ...
    Charge_predicted_UA_oil b_UA_oil UA_oil_range r2 b_w_para ...
    b_UA_oil_para] = charge_tuning_v075(System, flag, data, ...
    data_sim, index, Q_evap, Q_evap_ref, SHR, SHR_ref, W_comp, ...
```

```
    Q_cond, Q_cond_ref, SC, SC_EEV, SH_comp, Q_evap_sim, ...
    SHR_sim, W_comp_sim, Q_cond_sim, SC_sim, SC_EEV_sim, ...
    SH_comp_sim, version, residual, fault, refri)

% For charge tuning

i = 1;
% for i = 1:m
m = length(flag);
i = 1;
while i ≤ m
    if ¬isnan(sum(residual(i,:))) & ((norm(residual(i,:))<2.5e-3))
        q = 1+1;
    else
        data(i,:) = [];
        data_sim(i,:) = [];
        System(i) = [];
        residual(i,:) = [];
        Q_evap(i,:) = [];
        Q_evap_ref(i,:) = [];
        SHR(i,:) = [];
        SHR_ref(i,:) = [];
        W_comp(i,:) = [];
        Q_cond(i,:) = [];
        Q_cond_ref(i,:) = [];
        SC(i,:) = [];
```

```
32          SC_EEV(i,:) = [];
33          SH_comp(i,:) = [];
34          Q_evap_sim(i,:) = [];
35          SHR_sim(i,:) = [];
36          W_comp_sim(i,:) = [];
37          Q_cond_sim(i,:) = [];
38          SC_sim(i,:) = [];
39          SC_EEV_sim(i,:) = [];
40          SH_comp_sim(i,:) = [];
41          fault(i,:) = [];
42          m = m - 1;
43          i = i - 1;
44      end
45      i = i + 1;
46  end
47  Q_gain = Q_evap - Q_cond + W_comp;
48
49  % obtain est. charge, heat transfer and SHR values
50  % remove the ones with accumulator in action
51  qq = m;
52  i = 1;
53  j = 1;
54  % Charge_mea_ori = data(:,31);
55  while i ≤ qq
56          Tain(i,1) = System(j).Evap.Tain;
57          wain(i,1) = System(j).Evap.wain;
```

```
58          Tamb(i,1) = System(j).Cond.Tain;
59          Tdiff(i,1) = abs(Tain(i,1) - Tamb(i,1));
60  %     end
61      i = i + 1;
62      j = j + 1;
63  end
64
65  r2.r2_tuned_full = NaN;
66  r2.r2_tuned_UA_tp = NaN;
67
68  % obtain liquid line length and x-value for regression
69  qq = m;
70  i = 1;
71  j = 1;
72  clear X;
73  % y-value
74  Charge_mea = data(:,31);
75  Cond_V = data(:,29);
76  Evap_V = data(:,30);
77  while i ≤ qq
78      if System(j).Accumulator_stat == 1 & System(j).CompressorSH ≤ 0
79          Cond_V(i,:) = [];
80          Evap_V(i,:) = [];
81          Charge_mea(i,:) = [];
82          i = i - 1;
83          qq = qq - 1;
```

```
84      else
85          x_evap_in = (System(j).Evap.hin - propertyRH(...
86              'H','P',System(j).Evap.Pin,'Q',0,refri ...
87          ))/(propertyRH(...
88              'H','P',System(j).Evap.Pin,'Q',1,refri ...
89          )-propertyRH('H','P',System(j).Evap.Pin,'Q',0,refri));
90          rho_high_vapor = propertyRH(...
91              'D','P',System(j).Cond.Pin,'Q',1,refri ...
92          );
93          X(i,1) = System(j).Cond.OtherOutput(7);
94          X(i,2) = System(j).Cond.OtherOutput(5);
95          X(i,3) = System(j).Evap.OtherOutput(4);
96          UA_oil_index(i,1) = j;
97          w_cond_sc(i,1) = System(j).Cond.OtherOutput(7);
98          w_evap_sh(i,1) = System(j).Evap.OtherOutput(4);
99          Charge(i,1) = System(j).Charge;
100     end
101     i = i + 1;
102     j = j + 1;
103 end
104 X_ori = X;
105
106 % extra adjustment
107 index_rated = find(w_cond_sc>0.01&w_evap_sh>0.01);
108 if isempty(index_rated)
109     index_rated = 1;
```

```
end
    function r2_charge = tune_charge(qqq)
        X = X_ori;
        Charge_mea_rated = data(index_rated(qqq),31);
        if length(Charge_mea_rated) > 1
            dummy = Charge_mea_rated(1);
            clear Charge_mea_rated;
            Charge_mea_rated = dummy;
        end
        if mean(Tamb) < mean(Tain)
            Heating = 1;
            Charge_diff_rated = Charge_mea_rated - ...
                Charge(index_rated(qqq),1);
        else
            Heating = 0;
            Charge_diff_rated = Charge_mea_rated - ...
                Charge(index_rated(qqq),1);
        end
        Y = Charge_mea-Charge_diff_rated-Charge;
        Charge_temp = Charge;
        for i = 1:size(X,2)
            X_adj(i,1) = mean(X(index_rated(qqq),i));
            X(:,i) = X(:,i) - X_adj(i,1);
        end
        UA_oil_range.maxi = max(X);
        UA_oil_range.mini = min(X);
```

```
136
137         % regression
138         if(index > 0)
139             b = X\Y;
140             if max(abs(b))/min(abs(b)) > 5.e4
141                 b = zeros(length(X(1,:)),1);
142             end
143             % repeat
144             repeat_back = 1;
145             kk = size(X,2);
146             se_b = zeros(length(b),1);
147             rm_index = zeros(length(b),1); % removal index
148             alpha = 0.05; % Prob I for hypothesis testing
149             % t statistic of each coefficient
150             t_b = zeros(length(b),1);
151             % difference in t statistics
152             t_diff = ones(length(b),1)*9999;
153             while repeat_back == 1
154                 % eliminate b with column of X == 0
155                 for i = 1:size(X,2)
156                     if rm_index(i,1) > 0
157                         b(i,1) = 0;
158                     end
159                 end
160                 options = optimset(...
161                     'MaxFunEvals',2000,'display','off',...
```

```
                    'GradObj','on','Algorithm','interior-point',...
                    'MaxIter',1000,'TolFun',1.e-8,'TolX',1.e-8 ...
                );
                [b_UA_oil,resnorm,flag,output_message] = fminunc(...
                    @(b) ChargeTuning(...
                        b,X,Y,Charge_mea,Evap_V,Cond_V, 1 ...
                    ), b, options ...
                );
                [SS_E, dummy, weigh] = ChargeTuning(...
                    b_UA_oil,X,Y,Charge_mea,Evap_V,Cond_V, 0 ...
                );
                weigh_total = (1./weigh)*ones(size(weigh,2),1);
                dof = size(X,1) - (kk + 1);
                MS_E = SS_E/dof;
                X_prime = [];
                for i = 1:qq
                    X_prime(i,:) = (X(i,:)'.*sqrt(...
                        weigh_total(i,1) ...
                    )./Charge_mea(i,1))';
                end
                index_zero = [];
                j = 1;
                for i = 1:size(X,2)
                    if sum(X_prime(:,i)) == 0
                        index_zero(j,1) = i;
                        j = j + 1;
```

```
188                     end
189                 end
190                 if isempty(index_zero)
191                     C = (X_prime'*X_prime)/MS_E;
192                 else
193                     X_prime(:,index_zero) = [];
194                     Cov = (X_prime'*X_prime)/MS_E;
195                     [ppp qqq] = size(Cov);
196                     i = 1;
197                     j = 1;
198                     while i ≤ size(X,2)
199                         if i ≠ index_zero
200                             C(i,i) = Cov(j,j);
201                             i = i + 1;
202                             j = j + 1;
203                         else
204                             C(i,i) = 0.0; % to be ignored
205                             i = i + 1;
206                         end
207                     end
208                 end
209                 t_stat_neg = tinv(alpha/2.0, dof);
210                 t_stat_pos = tinv(1-alpha/2.0, dof);
211                 max_tuning_index = zeros(size(X,2),1);
212                 for i = 1:size(X,2)
213                     if C(i,i) > 0.0
```

```
                    se_b(i,1) = sqrt(C(i,i));
                end
                if rm_index(i,1) > 0
                    t_b(i,1) = 0;
                else
                    t_b(i,1) = b_UA_oil(i,1)/se_b(i,1);
                end
                if t_b(i,1) > 0 && rm_index(i,1) == 0
                    t_diff(i,1) = t_b(i,1) - t_stat_pos;
                elseif rm_index(i,1) == 0
                    t_diff(i,1) = t_stat_neg - t_b(i,1);
                else
                    t_diff(i,1) = 9999;
                end
                max_tuning = abs(max(b_UA_oil(i,1).*X(:,i)));
                if max_tuning > Charge_mea_rated
                    max_tuning_index(i,1) = ...
                        max_tuning/Charge_mea_rated;
                end
            end
            if true
                repeat_back = 0;
                for i = 1:size(X,2)
                    if rm_index(i,1) > 0
                        b_UA_oil(i,1) = 0;
                    end
```

```
                    end
                else % eliminate backwards
                    index_new = find(t_diff==min(t_diff));
                    rm_index(index_new,1) = 1;
                    kk = kk - 1;
                    X(:,index_new) = zeros(size(X,1),1);
                    b_UA_oil(index_new,1) = 0;
                end
            end
        else
            b_UA_oil = [1;0;0;0;0];
            resnorm = 1;
        end

        if (min(Charge_mea) - max(Charge_mea))/max(Charge_mea) > -0.01
            b_UA_oil = zeros(length(b_UA_oil),1);
            Charge_diff_rated = mean(Charge_mea - Charge_temp);
        end

        % statistics
        if(index > 0)
            Charge_predicted_UA_oil = X*b_UA_oil + Charge_temp + ...
                Charge_diff_rated;
            b_UA_oil(1:length(b_UA_oil)+1+length(X_adj),1) = [...
                Charge_diff_rated; b_UA_oil; X_adj ...
            ];
```

```
            r2.r2_tuned_UA_oil = 1 - sum((...
                Charge_mea-Charge_predicted_UA_oil ...
            ).^2)/sum((Charge_mea-mean(Charge_mea)).^2);
            b_UA_oil_para.Cov_X = covariance_adj(X);
            b_UA_oil_para.X_limit = max(diag(...
                X*b_UA_oil_para.Cov_X*X' ...
            ));
        else
            Charge_predicted_UA_oil = Charge;
            r2.r2_tuned_UA_oil = NaN;
            b_UA_oil_para.Cov_X = 0;
            b_UA_oil_para.X_limit = Inf;
        end
        Charge_mea_UA_oil = Charge_mea;

        r2.r2_tuned_UA = NaN;
        r2_charge = r2.r2_tuned_UA_oil;
    end

r2_max = -Inf;
qqq_ans = 1;
for qqq = 1:length(index_rated)
    r2_charge = tune_charge(qqq);
    if r2_charge > r2_max;
        r2_max = r2_charge;
        qqq_ans = qqq;
```

```
292     end
293 end
294 tune_charge(qqq_ans);
295 
296 % obtain liquid line length and x-value for regression
297 qq = m;
298 i = 1;
299 j = 1;
300 clear X; clear Y;
301 % y-value
302 Charge_mea_w = data(:,31);
303 Y = Charge_mea_w;
304 Charge_temp = zeros(length(Y),1);
305 Cond_V = data(:,29);
306 Evap_V = data(:,30);
307 while i ≤ qq
308     if System(j).Accumulator_stat == 1 & System(j).CompressorSH ≤ 0
309         Y(i,:) = [];
310         Cond_V(i,:) = [];
311         Evap_V(i,:) = [];
312         Charge_mea_w(i,:) = [];
313         Charge_temp(i,:) = [];
314         i = i - 1;
315         qq = qq - 1;
316     else
317         X(i,1) = 1;
```

```
318         X(i,2) = System(j).Cond.OtherOutput(7);
319         w_index(i,1) = j;
320     end
321     i = i + 1;
322     j = j + 1;
323 end
324
325 % regression
326 if(index > 0)
327     b = X\Y;
328     if max(abs(b))/min(abs(b)) > 5.e4
329         b = zeros(length(X(1,:)),1);
330     end
331     options = optimset(...
332         'MaxFunEvals',2000,'display','off',...
333         'GradObj','on','LargeScale','on' ...
334     );
335     A = [0 -1 0 0; 0 0 1 0; 0 0 0 1;];
336     bb = zeros(3,1);
337     A = [];
338     bb = [];
339     [b_w,resnorm] = fminunc(...
340         @(b) ChargeTuning(...
341             b,X,Y,Charge_mea_w,Evap_V,Cond_V, 1 ...
342         ), b, options ...
343     );
```

```
344 else
345     b_w = [1;0];
346     resnorm = 1;
347 end
348
349 % statistics
350 if(index > 0)
351     Charge_predicted_w = X*b_w + Charge_temp;
352     r2.r2_tuned_w = 1 - sum(...
353         (Charge_mea_w-Charge_predicted_w).^2 ...
354     )/sum(...
355         (Charge_mea_w-mean(Charge_mea_w)).^2 ...
356     );
357     b_w_para.Cov_X = inv(X'*X);
358     b_w_para.X_limit = max(diag(X/(X'*X)*X'));
359 else
360     Charge_predicted_w = mean(Charge_mea_w);
361     r2.r2_tuned_w = NaN;
362     b_w_para.Cov_X = 0;
363     b_w_para.X_limit = Inf;
364 end
365
366 r2.r2_tuned_oil = NaN;
367
368 r2.r2_Q = 1 - sum((Q_evap-Q_evap_sim).^2)/sum(...
369     (Q_evap-mean(Q_evap)).^2 ...
```

```
);
r2.r2_SHR = 1 - sum((SHR-SHR_sim).^2)/sum(...
    (SHR-mean(SHR)).^2 ...
);
r2.r2_Q_cond = 1 - sum((Q_cond-Q_cond_sim).^2)/ ...
    sum((Q_cond-mean(Q_cond)).^2);
r2.r2_W = 1 - sum((W_comp-W_comp_sim).^2)/sum(...
    (W_comp-mean(W_comp)).^2 ...
);

r2.r2_Q_ref = 1 - sum((Q_evap_ref-Q_evap_sim).^2)/ ...
    sum((Q_evap_ref-mean(Q_evap_ref)).^2);
r2.r2_SHR_ref = 1 - sum((SHR_ref-SHR_sim).^2)/sum(...
    (SHR_ref-mean(SHR_ref)).^2 ...
);
r2.r2_Q_cond_ref = 1 - sum((Q_cond_ref-Q_cond_sim).^2)/ ...
    sum((Q_cond_ref-mean(Q_cond_ref)).^2);

end

function [resid, g, weigh] = ChargeTuning(...
    b,X,Y,Charge_mea,Evap_V,Cond_V, return_g ...
)
m = length(Y);
bin_num = 5;
weigh = weighing_function_v000(Charge_mea, bin_num);
```

```
396 weigh(:,2) = weighing_function_v000(Evap_V, bin_num);
397 weigh(:,3) = weighing_function_v000(Cond_V, bin_num);
398 o = length(b);
399 [p q] = size(weigh);
400 resid = X*b - Y;
401 resid = sqrt((1./weigh)*ones(q,1))./Charge_mea.*resid;
402 resid = resid'*resid;
403 if return_g > 0
404     g = diffChargeTuning(b,X,Y,Charge_mea,Evap_V,Cond_V);
405 else
406     g = 0;
407 end
408 end
409
410 function diff = diffChargeTuning(b,X,Y,Charge_mea,Evap_V,Cond_V)
411 bAD = myMatrixAD(b);
412 outAD = ChargeTuning(bAD,X,Y,Charge_mea,Evap_V,Cond_V, 0);
413 diff = getderivs(outAD)';
414 end
415
416 function [resid, g, weigh] = WeightedRegression(...
417     b,X,Y,Charge_mea,Evap_V,Cond_V, return_g ...
418 )
419 m = length(Y);
420 bin_num = 5;
421 weigh = weighing_function_v000(Charge_mea, bin_num);
```

```
422 weigh(:,2) = weighing_function_v000(Evap_V, bin_num);
423 weigh(:,3) = weighing_function_v000(Cond_V, bin_num);
424 o = length(b);
425 [p q] = size(weigh);
426 resid = X*b - Y;
427 resid = sqrt((1./weigh)*ones(q,1)).*resid;
428 resid = resid'*resid;
429 if return_g > 0
430     g = diffWeightedRegression(b,X,Y,Charge_mea,Evap_V,Cond_V);
431 else
432     g = 0;
433 end
434 end
435
436 function diff = diffWeightedRegression(...
437     b,X,Y,Charge_mea,Evap_V,Cond_V ...
438 )
439 bAD = myMatrixAD(b);
440 outAD = WeightedRegression( ...
441     bAD,X,Y,Charge_mea,Evap_V,Cond_V, 0 ...
442 );
443 diff = getderivs(outAD)';
444 end
```

VITA

VITA

Howard Cheung was born in Hong Kong in 1987 and entered the Hong Kong University of Science and Technology as an undergraduate student in 2005. During his four undergraduate years, he finished a summer internship in China Light and Power Co., Ltd. to analyze data from mechanical pole testing, built a small horizontal axis wind turbine with his teammates for his final year project and did a research project on the modeling of air damping on microresonator. After graduating with a Bachelor of Engineering in Mechanical Engineering and a Bachelor of Business Administration in General Business Management with first class honor and an academic achievement in 2009, he started his PhD study at Purdue University. He worked on multiple projects during his graduate study, including testing and empirical modeling of ductless heat pump systems, mechanistic modeling of multi-split ductless heat pump systems, development of low-cost virtual performance sensors of packaged air conditioning units and inverse modeling for fault impacts of vapor compression systems. He also obtained the Ward A. Lambert Fellowship in the fall semester of 2013 and was a lecturer for an undergraduate thermodynamics class in the semester.

PUBLICATIONS

Cheung, H. and Braun, J. E. (2014). Component-based, Gray-box Modeling of Ductless Multi-split Heat Pump Systems. *International Journal of Refrigeration*, 38:30-45.

Cheung, H. and Braun, J. E. (2014). Performance Mapping for Variable-speed Ductless Heat Pump Systems in Heating and Defrost Operation. *HVAC&R Research*, 20(5):545-558

Cheung, H. and Braun, J. E. (2014). Performance Comparisons for Variable-Speed Ductless and Single-Speed Ducted Residential Heat Pumps. *International Journal of Refrigeration*, Accepted.

Cheung, H. and Braun, J. E. (2014). Modeling of Fault Impacts for a Multi-split Ductless Heat Pump System. *11th IEA Heat Pump Conference 2014*, Montréal, Québec.

Cheung, H. and Braun, J. E. (2014). Virtual Power Consumption and Cooling Capacity Virtual Sensors for Rooftop Units. *15th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Yuill, D. P., Cheung, H. and Braun, J. E. (2014). Evaluating Fault Detection and Diagnostics Tools with Simulations of Multiple Vapor Compression Systems. *15th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Yuill, D. P., Cheung, H. and Braun, J. E. (2014). Validation of a Fault-Modeling Equipped Vapor Compression System Model Using a Fault Detection and Diagnostics Evaluation Tool. *15th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Cheung, H. and Braun, J. E. (2013). Simulation of Fault Impacts for Vapor Compression Systems by Inverse Modeling Part I: Component Modeling and Validation. *HVAC&R Research*, 19(7):892-906.

Cheung, H. and Braun, J. E. (2013). Simulation of Fault Impacts for Vapor Compression Systems by Inverse Modeling Part II: System Modeling and Validation. *HVAC&R Research*, 19(7):907-921.

Cheung, H., Nyika, S. and Braun, J. E. (2013). Development and Validation of a Mechanistic Model for Variable-Speed Multi-split Heat Pumps. In *2013 ASHRAE Winter Conference*, Dallas, Texas.

Cheung, H. and Braun, J. E. (2012). Inverse Modeling to Simulate Fault Impacts for Vapor Compression Equipment Part 1: Component Modeling and Validation. In *14th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Cheung, H. and Braun, J. E. (2012). Inverse Modeling to Simulate Fault Impacts for Vapor Compression Equipment Part 2: System Modeling and Validation. In *14th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Cheung, H. and Braun, J. E. (2012). Performance Mapping for Variable Ductless Heat Pump Systems in Heating, Cooling and Defrost Operation. In *14th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.

Cheung, H. and Braun, J. E. (2010). Performance Characteristics and Mapping for a Variable-Speed Ductless Heat Pump. In *13th International Refrigeration and Air Conditioning Conference at Purdue*, West Lafayette, Indiana.