

University of Illinois at Urbana-Champaign



Air Conditioning and Refrigeration Center

A National Science Foundation/University Cooperative Research Center

Multiple Heat Exchangers Simulation Within the Newton-Raphson Framework

X. Liu and C. W. Bullard

ACRC TR-194

March 2002

For additional information:

Air Conditioning and Refrigeration Center
University of Illinois
Mechanical & Industrial Engineering Dept.
1206 West Green Street
Urbana, IL 61801

(217) 333-3115

*Prepared as part of ACRC Project #69
Development of Modular A/C and Refrigerator Simulation Program
C. W. Bullard, Principal Investigator*

The Air Conditioning and Refrigeration Center was founded in 1988 with a grant from the estate of Richard W. Kritzer, the founder of Peerless of America Inc. A State of Illinois Technology Challenge Grant helped build the laboratory facilities. The ACRC receives continuing support from the Richard W. Kritzer Endowment and the National Science Foundation. The following organizations have also become sponsors of the Center.

Alcan Aluminum Corporation
Amana Refrigeration, Inc.
Arçelik A. S.
Brazeway, Inc.
Carrier Corporation
Copeland Corporation
Dacor
Daikin Industries, Ltd.
Delphi Harrison Thermal Systems
General Motors Corporation
Hill PHOENIX
Honeywell, Inc.
Hydro Aluminum Adrian, Inc.
Ingersoll-Rand Company
Kelon Electrical Holdings Co., Ltd.
Lennox International, Inc.
LG Electronics, Inc.
Modine Manufacturing Co.
Parker Hannifin Corporation
Peerless of America, Inc.
Samsung Electronics Co., Ltd.
Tecumseh Products Company
The Trane Company
Valeo, Inc.
Visteon Automotive Systems
Wolverine Tube, Inc.
York International, Inc.

For additional information:

*Air Conditioning & Refrigeration Center
Mechanical & Industrial Engineering Dept.
University of Illinois
1206 West Green Street
Urbana, IL 61801*

217 333 3115

Abstract

A general framework is proposed for simulating complex heat exchanger geometries in a manner suitable for sequential solution of the refrigerant- and air-side equations for mass, momentum and energy. The sequential solution enables the algorithm to be applied to a single module of a complex heat exchanger, and then integrated with other modules within a simultaneous equation solver employing a Newton-Raphson approach. This report also describes the integration of component subroutines into system simulation models for air conditioners and refrigerators. The modular approach is illustrated by describing its application to a dual-evaporator refrigerator simulation.

Table of Contents

	Page
Abstract	iii
List of Figures	vi
List of Tables	viii
Chapter 1: Introduction	1
1.1 Background.....	1
Chapter 2: General Model Description	3
2.1 Model structure	4
2.1.1 Main program.....	6
2.1.2 Sequential subroutines	7
2.2 Communications.....	10
2.2.1 Communication between main program and sequential subroutine	10
2.2.2 Communication between main program and supportive files	13
2.2.3 Communication between sequential components	14
Chapter 3: Heat Exchanger Algorithms	16
3.1 Description.....	16
3.2 Heat exchanger configurations.....	17
3.2.1 Parallel refrigerant flow	18
3.2.2 Serial refrigerant flow.....	21
3.3 Complex component analysis and simulation.....	24
3.3.1 Division	24
3.3.2 Conquer.....	25
3.3.3 Combination	27
Chapter 4: Dual Evaporator System Modeling.....	32
4.1 Introduction.....	32
4.2 Sequential simulation.....	33
4.2.1 Compressor.....	33
4.2.2 Condenser.....	34
4.2.3 Captube-suction line heat exchanger.....	36
4.2.4 Evaporator.....	37
References.....	39
Appendix A: Capillary Tube Suction-Line Heat Exchanger Design Model.....	40

A.1 Design operating condition	40
A.1.1 Effect of low condensing pressure	41
A.2 Capillary Tube-Suction Line Heat Exchanger Model user's reference	42
A.2.1 Definition description for variables and parameters	42
A.2.2 XK file, variables and parameters definition	44
A.2.3 Sample instruction file	46
A.2.4 Solver setting file	46
Appendix B: Dual Evaporator Simultaneous System Modeling.....	48
B.1 Introduction	48
B.2 Simultaneous system	48
Appendix C: Residential A/C System Modeling.....	53
C.1 Introduction	53
C.2 Model description	53
C.2.1 Condenser.....	53
C.2.2 Expansion device	60
C.2.3 Compressor	60
C.2.4 Evaporator	60
C.3 Residual equations	69

List of Figures

	Page
Figure 2.1 General model structure	4
Figure 2.2 New general model structure.....	6
Figure 2.3 Evaporator subroutine structure.....	8
Figure 2.4 Condenser parallel flow configurations.....	9
Figure 2.4 main program/sequential subroutine communications.....	11
Figure 2.5 Jacobian matrix	12
Figure 2.6 Subroutine Subsets Description.....	13
Figure 2.7 Components connection configuration.....	14
Figure 2.8 System residual equations for component connection.....	15
Figure 3.1 Evaporator component configurations.....	17
Figure 3.2 Parallel heat exchanger configuration.....	18
Figure 3.3 Serial heat exchanger configuration.....	18
Figure 3.4 Cross/parallel flow configuration	19
Figure 3.5 Parallel/parallel flow configuration.....	19
Figure 3.6 Counter/parallel flow configuration	19
Figure 3.7 Residual equations of cross/parallel configuration.....	20
Figure 3.8 Equations of counter/parallel and parallel/parallel configurations	21
Figure 3.9 Residual equations describing connection.....	21
Figure 3.10 Cross/serial flow configuration.....	22
Figure 3.11 Parallel/serial flow configuration	22
Figure 3.12 Counter/serial flow configuration.....	22
Figure 3.13 Residual equations describing cross/serial configuration.....	23
Figure 3.14 Residual equations describing parallel/serial configuration	23
Figure 3.15 Second level elements	24
Figure 3.16 Third level modules	25
Figure 3.17 Overall structures of heat exchangers subroutines	26
Figure 3.18 Downstreaming and upstreaming equations interface	27
Figure 3.19 Residual equations groups simulating P1	28
Figure 3.20 Residual equations simulating P ₂	29
Figure 3.21 Residual equations simulating P ₃	30
Figure 3.22 Combination equations for parallel modules.....	31
Figure 3.23 Airside connection equations for cross flow configuration	31
Figure 4.1 New model structure of dual evaporator system.....	33
Figure 4.2 interface of residual equations and condenser sequential subroutine.....	35
Figure 4.3 condenser sequential subroutine variable categories.....	35
Figure 4.4 interface of sequential subroutine, residual equations and variables.....	37
Figure 4.5 residual equations associated with evaporator component.....	38
Figure A.1 Divisions of capillary-tube suction line heat exchanger.....	42

Figure B.1 Freezer component simulation: no air flows into fresh food compartment	49
Figure B.2 Fresh food compartment simulation	51
Figure B.3 stable refrigerant states of dual evaporator system	51
Figure B.4 Refrigerant states of single fresh food evaporator case.....	52
Figure B.5 Refrigerant states at connection points of freezer simulation	52
Figure C.1 Condenser geometry	54
Figure C.2 Condenser module structure	54
Figure C.3 Flow chart for condenser subroutine	56
Figure C.4 Process of determining surface condition.....	62
Figure C.5 Evaporator subroutine flow chart	65
Figure C.6 System residual equations.....	71

List of Tables

	Page
Table A.1 Calculated results for several different cases	40
Table A.2 Calculated length for different diameters to meet design mass flow	41
Table A.3 Effectiveness of low condensing pressure	41
Table.A.4 Sample XK initialization file	44
Table A.5 Instruction file Sample	46

Chapter 1: Introduction

1.1 Background

Earlier versions of the ACRC refrigerator and a/c simulation models consisted of an equation solver and a set of governing equations that simulated the components. The ACRC solver utilized a Newton-Raphson algorithm to simultaneously solve the system governing equations. The biggest advantage of the solver is that it allows the input parameters and output variables to be interchangeable without the need to reprogram the model. The room air conditioning simulation model developed by Bridges and Bullard (1995), was summarized by Mullen *et al.* (1998), and its development and validation are described in Mullen, Bridges and Bullard (1998), Kirkwood and Bullard (1999). Woodall and Bullard (1996) developed the RFSIM simulation model. Kirkwood and Bullard (1999) developed the split system model with microchannel heat exchangers, and Stott and Bullard (1999) validated it. Recently, Stein, Bullard and Newell (2000) began to develop the dual evaporator simulation model and Gerlach and Newell (2000) finalized it.

Although the system simulation model was proved an accurate and sophisticated design tool, the program had two prominent limitations. The NR algorithm requires the user to provide accurate initial guesses for all output variables in order to ensure convergence. For many obscure output variables, this proved difficult and frustrating. Reducing the number of required initial guesses was therefore identified as an important goal in the development of the next generation of simulation programs. That approach was described by Harshbarger and Bullard (2000) and included another desirable feature: the ability to simulate modern heat exchanger designs, particularly exchangers having complex circuiting, or consisting of multiple slabs in the airflow direction. The geometry of each heat exchanger module may be the same or different from others.

Another limitation of the original version was that it could only simulate one system with specified geometry. To simulate different systems, it was necessary to rewrite the governing equations. This led to a proliferation of distinct models, with the need to keep updating all of them. This report describes the implementation of the modular system simulation models, which also accommodates systems having multiple evaporators or multiple condensers systems (e.g. dual evaporator refrigerator and minivan system).

A new model structure was created to implement the modular simulation approach. Finite element solutions of the heat exchangers were developed for the condenser and evaporator, allowing simulation of complex geometries that were not possible with conventional methods. The finite element solutions were integrated into the system model in a manner that reduced the number of required initial guesses and therefore, the burden on the user. In this modular structure, each component in the system simulated by a stand-alone subroutine, solved sequentially in a self-contained manner. Therefore, each component simulation can easily be isolated and/or integrated into a simultaneous set of system-level equations.

This kind of modeling technique allows the simulation of complex heat exchanger designs while maintaining the interchangeability of the inputs and outputs; and also reduces the burden on the user to provide many initial guesses. A further advantage of this algorithm is that the initial guesses are restricted to readily known quantities. The enhanced algorithm uses a novel approach by simultaneously employing a NR solver for the system

and a sequential simulation for each component. This kind of modeling technique also gives us a easier way to build general structured framework for simulating multi-evaporator or multi-condenser systems.

Chapter 2 of this report details the general framework of the modular simulation model. The mathematics of the new algorithm is discussed in general terms, along with the process of interfacing a sequential simulation within a Newton-Raphson solution. Chapter 3 explains the mechanics and implementation of the new algorithm for multiple heat exchangers. Chapter 4 describes the modular structure of the dual evaporator simulation model. Appendix A explains the captube-suction line heat exchanger model design. Appendix B explains how one could use the single evaporator simulation model to simulate a dual evaporator refrigerator. Appendix C concisely describes the a/c modular simulation model, and provides details of component subroutines and algorithms to simulate the complex heat exchangers.

Chapter 2: General Model Description

The structure of the ACRC refrigerator and air-conditioner models is versatile and the models are accurate for simulating various types of systems and components. The principal advantage is that the structure is independent of the user's selection of dependent and independent variables. This is unlike conventional models employing the method of successive substitution in which the model structure is tied uniquely to an a priori selection of input and output variables.

At the same time, the limitations of this kind of structure are obvious and frustrating. The biggest disadvantage is that user and programmer need to provide a set of internally consistent and reasonably accurate initial guess values for all unknown variables. Otherwise, the system model and its Newton-Raphson solution algorithm will not converge to a solution. Practically, it has proved to be very difficult and a big burden to programmers and users, for example when switching refrigerants and therefore needing to alter the initial guess values of enthalpies, subcooled areas, etc. Now, reducing the number of system initial guess values is an important goal, because heat exchanger geometries are becoming more complex. A single heat exchanger can have very complex circuiting or many slabs, or each component can have multi-exchangers. A good design model structure is needed to simulate the more complicated heat exchangers and systems.

Harshbarger and Bullard (2000) have suggested a new model structure to simulate the a/c systems. In order to address the limitations described above, finite element solutions of the heat exchangers were developed for the condenser and evaporator. The finite element structure allows the simulation of complex geometries that were very hard within the conventional N-R framework. The finite element solutions were integrated into the system model in a manner that reduced the number of required initial guesses and therefore, the burden on the user. To further the capabilities of the model, a modular structure was adopted. Using a structure similar as TRNSYS (Klein et al., 1976), each component in the system is solved in a self-contained subroutine. Therefore, each component simulation can easily be isolated and/or integrated into a simultaneous set of system-level equations.

This kind of modeling technique allows the simulation of complex heat exchanger designs while maintaining the interchangeability of the inputs and outputs, because the core part of original model, NR solver, is still used in the new model simulation. It also reduces the burden on the user to provide many initial guesses; the user can restrict initial guesses to a subset of variables that can be easily known or measured.

The new model structure uses a novel approach by simultaneously employing a NR solver and a series of sequential simulations. Newton-Raphson solver is still the core in the main program, which simultaneously solves all the residual equations simulating the system. NR solver also provides new updated guess values of unknown (X) variables for the next iteration until the system converges to the final solution. A finite-element approach is used in sequential component to simulate the complex heat exchangers.

All the component-specific sequential subroutines can stand alone, providing more flexibility for programmers to integrate different stand-alone subroutines into the model. They also facilitate simulation of multi-heat exchanger systems, which are becoming more common. The corresponding subroutines can be integrated into the system without rewriting any code. A detailed discussion of mechanics, structures and implementations will be introduced in the following sections.

2.1 Model structure

In the original Newton-Raphson simulation model developed by Mullen *et al.* (1998), all the residual equations are stored in a single file. The new model uses sequential subroutines to simulate each system component. In the hx subroutines, a finite-element approach is introduced to simulate more complicated geometries. This technique greatly reduces the number of residual equations in the main program, which connects the components together to define the system. The sequential subroutine transmits the calculated output variables back to the main program, so the Newton-Raphson solver can solve the residual equations simultaneously. From the user perspective, only the component models are visible. The NR equations and solver operate in the background, returning after each iteration a set of updated inputs to each of the component subroutines.

The user initiates the simulation after selecting and providing values for the known independent variables (K's), unknown output variables (X's), and known parameters (P's). The internal relationships within the model complete the simulation. The calculated values of the output variables (X's), and the informative variables (C's) are returned to the user, along with the known parameters (K's) and the input parameters P that the user supplied to help specify components.

Different systems may have different condenser or evaporator geometries (e.g. finned tube; microchannel; wire-on-tube, etc). If we have already built self-contained subroutines for those types of geometries, the main program can just call the subroutines for components that are used in the simulated system, without necessity to rewrite the codes. The same compressor subroutine is used in different systems, but different compressor maps (curve fits) are chosen to calculate the mass flow rate and power consumption. The new model structure is shown below.

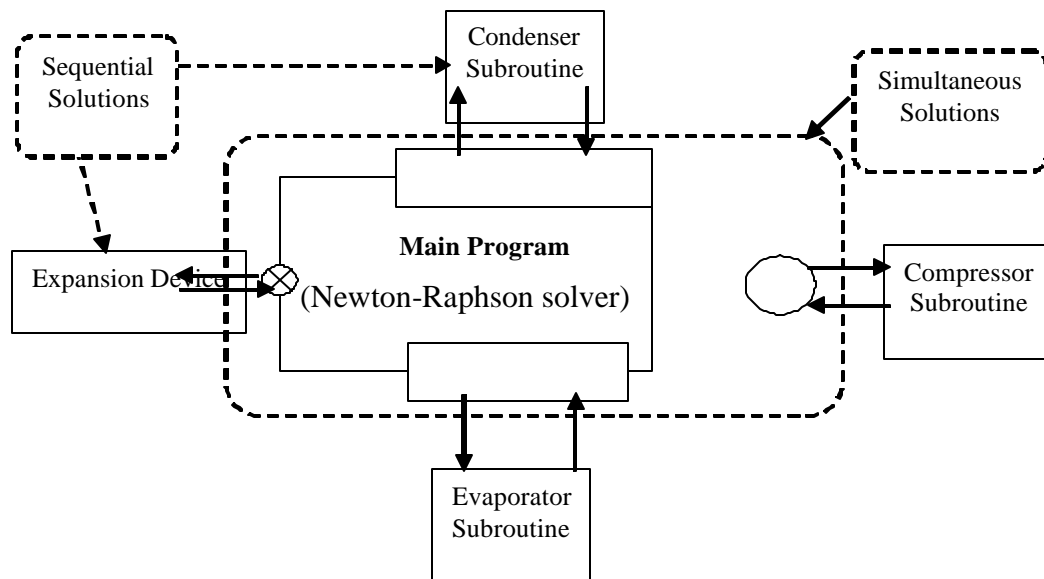


Figure 2.1 General model structure

In the general simulation model structure, which includes main program (NR solver is the core part), system components are simulated within their sequential subroutine. This new model structure has two kinds of

calculations: simultaneous and sequential. In main program, with the returns of calculated variables from sequential subroutines, NR solver simultaneously solves all residual equations, and in every iteration submits updated inputs unknown (X) variables to sequential subroutines.

The internal structure of the subroutines was defined by the need to solve sequentially. To accomplish this, the programmer pre-selected a subset of variables to be subroutine inputs. This subroutine structure is transparent to the user, who is free to switch independent and dependent (X's and K's) at the overall structure's user interface. Whenever one of the subroutine inputs is "unknown" to the user and the main program, the current or initial guess value for that variable is supplied as the input to the subroutine.

Inside the sequential subroutine, the dependent and independent variable inputs (X's and K's) and parameters (P's) are provided by main program. A finite-element approach is used to sequentially simulate the heat exchangers and capillary tube based upon the geometry, running condition and inlet states. The outputs from subroutine include two parts: 1) calculated variables suffixed with '_calc' and returned to the main program and 2) calculated variables (Cs) that are not needed by the NR solver, but provide valuable information to the user and to programmers who wish to know about the system running condition. For example, in HX subroutine, the total heat transfers are decided by two phase part, therefore the input or output enthalpies. So the calculated heat transfer or enthalpy variables can not give us too much information, but subcooled or two-phase area ratios can tell us what is going on inside the subroutines, which is very useful in debugging procedure. So generally, the whole calculation procedure involves simultaneous and sequential simulation.

Not only the heat exchanger geometries are becoming more and more complicated, so also are a/c and refrigerator systems, as multiple evaporators are being served by a single condensing unit. We are employing a finite-element approach to match the needs for individual heat exchangers. For systems having multiple evaporators or compressors, a more general idea about the model structure is needed. Figure 2.2 shows the more general structures used in simulating cases where evaporator or condenser component is actually a combination of serial and parallel heat exchangers. This may include a/c heat exchangers having multiple layers (slabs) or a refrigerator with freezer and food compartment evaporators in series. We need to pay attention to the mass flow rates and inlet states of each heat exchanger as well as air flow directions because air flow directions decide which algorithm we should call to simulate the HX. For serial cases, the outlet states values are equal to the inlet values of the coming heat exchanger with the equal mass flow rate. But for parallel ones, each heat exchanger has the same inlet state values, but these mass flow rates are not necessary equal although their sum should be equal to the whole system mass flow rate. More details about inlet states and mass flow rate distribution will be provided in the later discussion.

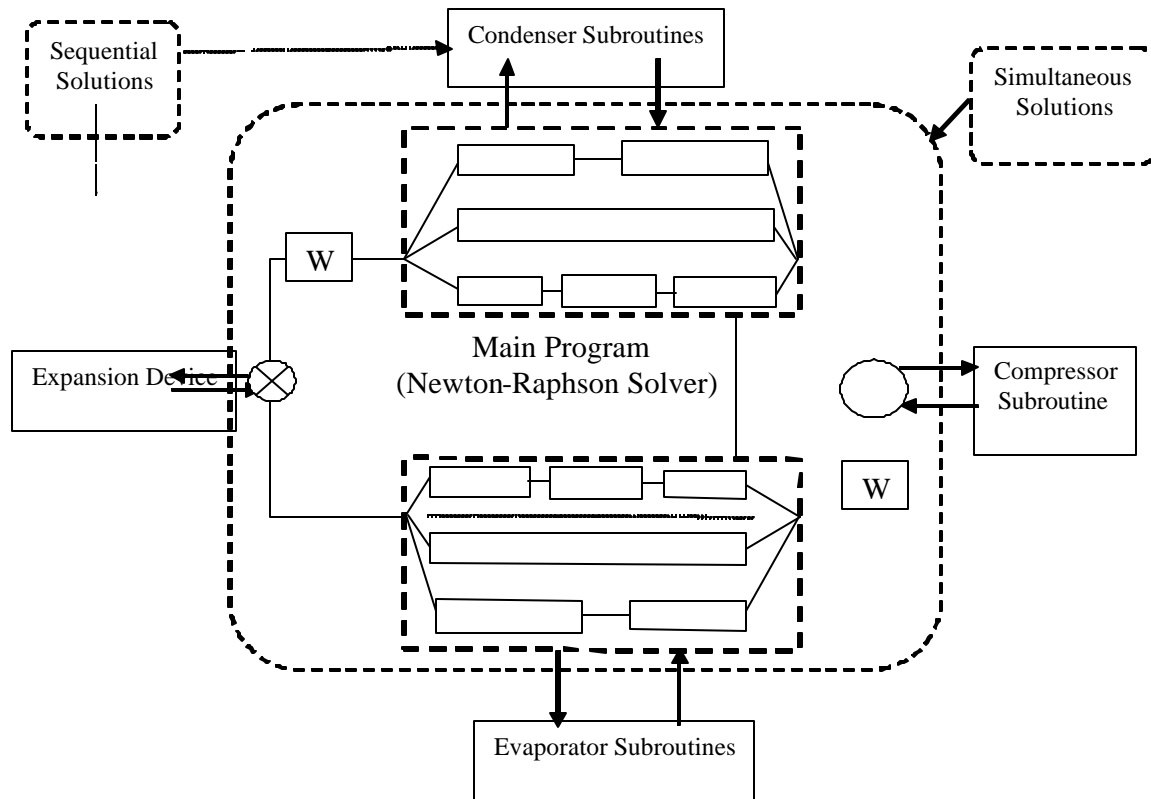


Figure 2.2 New general model structure

2.1.1 Main program

In main program, the NR solver is the essential part. It operates in the same manner as a traditional NR algorithm. The solver simultaneously determines the solution to a given set of N equations, the same number as system variables (X_s in XK file). The algorithm starts with an initial guess value for the ‘ X ’ variables from the user interface. By utilizing first order derivative information, the NR solver iteratively improves the guess values until the solver converges to a solution. During the solution of the simultaneous equations, the solver communicates with the sequential subroutines, submitting subroutine inputs, receiving calculated variables suffixed with ‘_calc’. This communication is internal to the model and is transparent to the user.

Firstly, the user initializes the system with providing values for parameters P , independent variables K and initial guesses for the dependent variables X . The main program transmits these initial guess values and parameters needed by sequential subroutine. Based on the inputs, subroutine sequentially calculates the output results: the output-calculated variables marked with “_calc” and the informative variables (C_s). C_s are sent back to the user interface and ‘_calc’ variables are sent back to system equations that are solved simultaneously by Newton-Raphson Solver.

The residual equations and the connections equations between major components are listed in the main program. After the main program calls each sequential subroutine, which return component output “_calc” variables which are then used by NR solver to simultaneously solve residual governing equations, one associated with each “_calc” output from the component subroutines.

If there is no X variable in inputs of subroutines, the simulation will be finished just in one iteration. Because of the interchangeability between Xs and Ks at the user interface, Xs can be inputs to the subroutine in some cases. Then, NR updates the input X variables with new calculated values in each iteration. Main program iteratively calls components subroutines with new updates inputs until the NR solver converges to a solution.

2.1.2 Sequential subroutines

The sequential subroutine contains all the information needed to simulate a component, solving all the governing equations. Without the force of NR solver, the subroutine will only solve a component for a certain set of specified input variables, parameters and geometry. The inputs and outputs of the sequentially solved subroutine are not interchangeable.

Harshbarger and Bullard (2000) built a new concept to divide the global set of subroutine inputs and outputs into four subsets: Set I, Set P, Set C and Set O, which are very helpful for our discussion. Both Set I and Set P comprise the subroutine inputs, while outputs include Set C and Set O. Set I is a subset of the interchangeable variables (X's and K's) from the main program. Usually the sequential subroutine requires the inlet refrigerant and air states, mass flow rates for the air and refrigerant, and a set of variables that describe the heat exchanger geometry. Unless the user happens to specify all these subroutine inputs as independent variables (K's), the whole calculation needs more than one iteration. Any Xs contained in the Set I is improved by NR solver in every iteration. Set P is a subset of Ps needed by this subroutine, which are specified in XK file and not changed during the whole calculation.

The subroutine outputs include a new category of variable, denoted a 'calc' variable. The subroutine does not output actual value of the interchangeable variables. Instead, the subroutine outputs 'calc' variables that represent the same quantities as interchangeable variables. The 'calc' variables are suffixed with 'calc' in order to distinguish them from their corresponding interchangeable variables. All interchangeable variables (X's and K's) not input to the subroutine correspond to subroutine outputs, which are included in set O. The 'calc' variables correspond to the interchangeable variables in Set O. Together sets O and I include all M interchangeable variables (X's and K's), that is $I \cup O = X \cup K = M$ (Harshbarger and Bullard, 2000). Generally, the sequential structure of the subroutine involves solving for the heat exchanger outlet refrigerant and air states, performance variables (heat transfer, pressure drop and mass charge), and the remaining geometry values that simulate and describe the running conditions.

Each sequential subroutine is called from the main program for simulating the associated component. Iteratively using new updated inputs from main program, subroutines calculate outputs suffixed with 'calc', and return them to main program. NR solver uses the new calculated outputs from subroutines to simultaneously solve the governing equations until converging to a solution. If inputs to subroutines are all Ks and include no X variables, the calculation will be finished in one iteration because the inputs and outputs are specified and not interchangeable. Otherwise, more iterations may be needed.

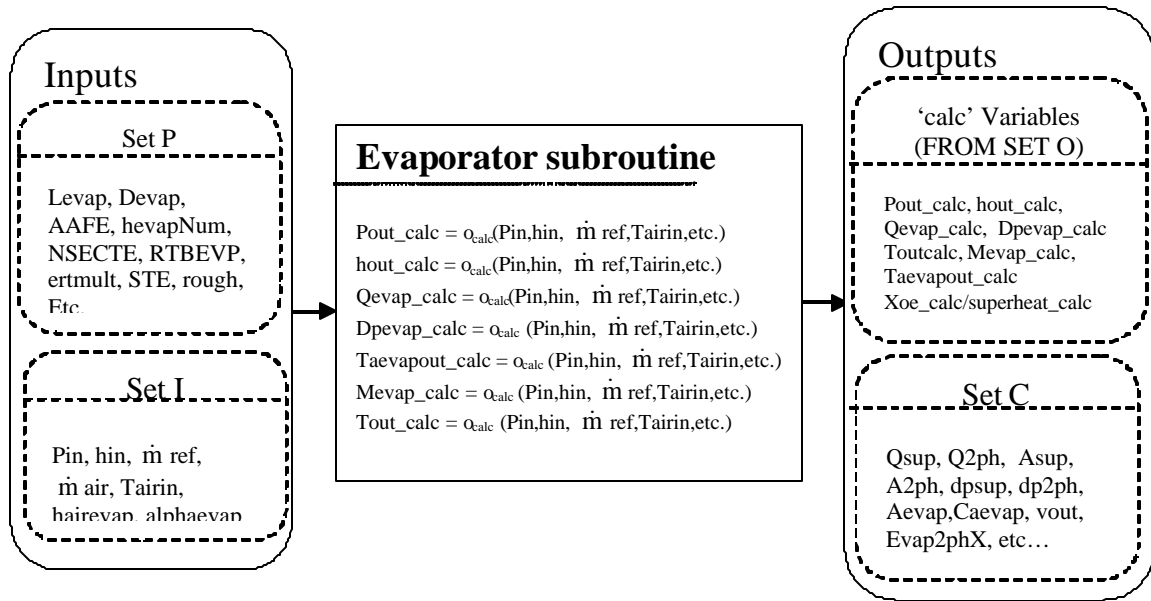


Figure 2.3 Evaporator subroutine structure

Figure 2.3 shows the structure of a simple dry evaporator sequential subroutine: the inputs (Set I and Set P) and the outputs (Set C and Set O). All the outputs are functions of the inputs and can be sequentially calculated.

In the subroutine, a finite-element approach is introduced to simulate heat exchangers having different geometries. Within the family of cross-flow heat exchangers, there are two possible configurations: the cross-parallel flow case where the bulk refrigerant flow is travelling in the same direction as the air and the cross-counter flow case where the bulk refrigerant travels in the opposite direction of the airflow. When the air and refrigerant streams flow in the same direction, the outlet variables for air and refrigerant are calculated for each element. By marching downstream in the refrigerant flow direction, the states of both refrigerant and air are exactly determined. The outlet states from former element are the incoming states of the latter element. The algorithm always marches in the downwind direction, to successively calculate the air temperatures. When the bulk refrigerant and air flow in opposite directions, the algorithm again marches downwind, but in the upstream direction relative to the refrigerant flow. In this (cross-counterflow) case, the subroutine inputs must include the refrigerant outlet states.

For all kinds of cross-flow heat exchangers, there are three kinds of regions we need to consider: superheated, two-phase and subcooled refrigerant states, respectively. Where a phase transition occurs inside a single finite element, the algorithms are able to handle this situation by solving an implicit equation to break the element into two parts. Three logic flags variables are defined: supheat, twoph and subcool. When they are true, the refrigerant is superheated, two-phase or subcooled, respectively. At one time, only one of them can be true. Along the flow direction of refrigerant, the heat exchanger is divided into the specified number of small elements. Serially, each element is simulated by a group of sequentially solved governing equations.

Because all elements are solved in numerical order and the outputs of one element are the inputs of the next element, only a few of them have state transitions and most of them just keep the state. If the refrigerant is in the superheated state, we only need to watch for the transition into two-phase. Whenever its enthalpy is less than

saturated enthalpy calculated by library function with current pressure and quality equal to 1.0 ($h_{px}(p_{local}, 1.0d0)$), this element changes its state from superheated into two-phase. Then different governing equations are used to simulate the following elements. When the refrigerant is in two phases, we need to watch for the transitions to both subcooled and superheat. If the enthalpy decreases, we watch for it to become less than the calculated enthalpy with local pressure and quality equal to 0 ($h_{px}(p, 0.0d0)$), where the element is changing to a subcooled state. If the local enthalpy is greater or equal to the calculated enthalpy, $h_{px}(p_{local}, 1.0d0)$, the following element will be in superheat states. Each element is solved by using the governing equations associated with the appropriate states. Within the subroutine, arrays with same number as elements are defined to store local refrigerant heat transfer, pressure drop and refrigerant mass. Based on the overall energy balance, we also can calculate the air inlet or outlet temperature of the heat exchanger. Finally, the arrays containing local heat transfer, pressure drop and refrigerant mass are used to calculate the subroutine output variables. Then these calculated variables (Set O) are returned back to residual equations in main program.

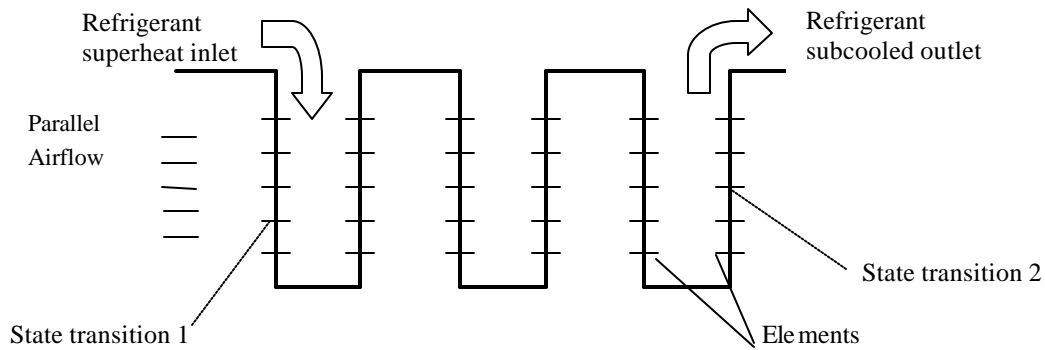


Figure 2.4 Condenser parallelflow configurations

For cross-flow case, we have two kinds of configurations: parallel airflow and counter airflow. Harshbarger and Bullard (2000) have developed two algorithms, respectively: downstreaming for the cross-parallel configuration, and upstreaming for the cross-counterflow configuration. The above Figure 2.4 shows the transitions in condenser parallel flow case. In either algorithm, the calculation begins from the inlet end of the air. There are two transition points in the heat exchanger along the flow direction of the refrigerant: one is from superheat to two-phase, the other one is from two-phase to subcooled. In total, five conditions may exist for any given element. They are superheated vapor, two-phase refrigerant, subcooled liquid, transition between superheated vapor and two-phase refrigerant, and transition between two-phase refrigerant and subcooled liquid. Each element is capable of deciding which conditions it is in and uses exact governing equations to simulate the element. The upstreaming algorithm is similar to the downstreaming algorithm. The only difference is the inputs: downstreaming needs refrigerant inlet states (inlet pressure and inlet enthalpy) as inputs and calculates the outlet states (outlet pressure and outlet enthalpy); upstreaming algorithm needs refrigerant outlet states (outlet pressure and outlet enthalpy) as inputs and calculates refrigerant inlet states as outputs. This allows both algorithms to begin the calculation with the inlet air.

2.2 Communications

Because the new model structure divides the whole system into individual stand-alone components, the communications between main program and sequential subroutines are very important issues. There are three main kinds of communications in the new model structure: 1) communication between main program (NR solver) and each sequential subroutine; 2) communication between main program and supportive files; and 3) communication between sequential subroutines. The communication between sequential subroutines also includes two kinds of situations: serial connection or parallel connection. More details will be provided in the following section.

2.2.1 Communication between main program and sequential subroutine

Each subroutine uses the same way to communicate with main program: getting inputs from main program and transmitting calculated variables back to main program. All calculated variables are functions of the subroutine inputs, so they can be sequentially calculated. There is a set of residual equations in the main program, associated with each component simulated by the sequential subroutine. In a system simulation, NR solver simultaneously solves the residual equations corresponding to all the system components, using the calculated variables from all the sequential subroutines. The same general structure can be used to simulate individual components.

For component simulations the NR main program is smaller because its residual equations correspond to the “calc” output variables from only one component subroutine. Main calls the subroutine while allowing the user to interchange X’s and K’s. As an example, Figure 2.4 shows the communication configurations between main program and refrigerator evaporator subroutine.

The NR solver performs several iterations in determining the solution to a set of equations. For the simple evaporator (component) simulation, the NR solver solves a set of seven simultaneous residual equations. The solver simultaneously forces the values of each of the seven equations, written in residual format, to zero. The number of simultaneous equations is equal to the number of ‘calc’ variables from the evaporator sequential subroutine. The seven NR residual equations are shown within the NR solver box. Each equation equates an interchangeable variable with its corresponding subroutine output variable.

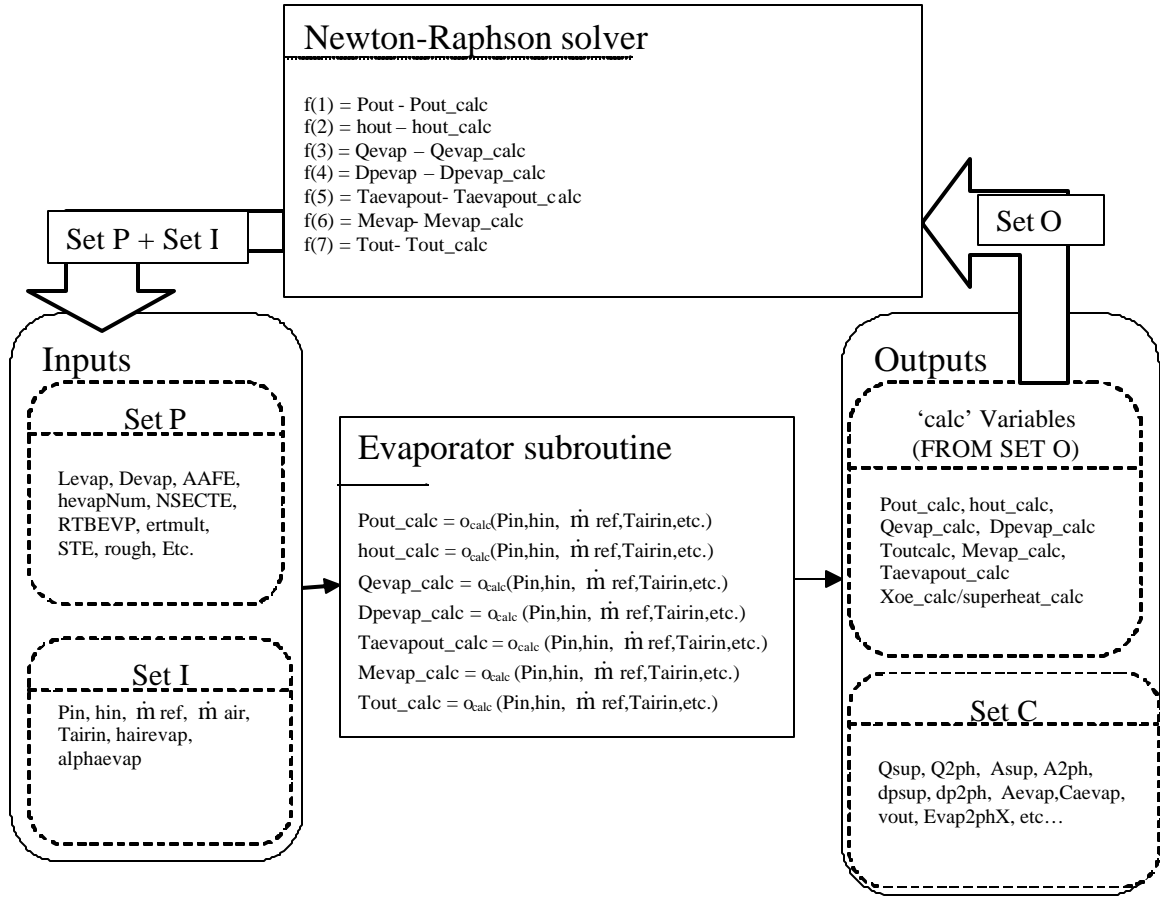


Figure 2.4 main program/sequential subroutine communications

At each iteration of the NR solver, the simultaneous equation set in Figure 2.4 is solved for new improved values of the 'X' variables. The process involves solving Equation 2.1, where $[J]$ is the Jacobian matrix, $\{f\}$ is the vector of NR residual equation values, and $\{\Delta X\}$ is the vector used to update the values of the 'X' variables (Harshbarger and Bullard, 2000).

$$\{\Delta X\} = [J]^{-1} \{f\} \quad (2.1)$$

The Jacobian, shown in Figure 2.5, consists of derivatives of the NR equations, f , with respect to the 'X' variables. The derivatives in the Jacobian are approximated numerically, using Equation 2.2. The first step in this process is to evaluate the NR equations using the known parameters (K 's) and the current iteration's guess values for the unknown variables (X 's). The results are seven scalar values for the NR residuals. These values are nonzero for each iteration until a solution is achieved. The next step is to slightly alter the value of an individual 'X' value and recalculate the values of the seven NR equations. The derivative is then approximated by the change of the NR residual equation divided by the change in the altered 'X' variable. This process is repeated for each 'X' variable.

$$[J] = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \frac{\partial f_1}{\partial X_2} & \frac{\partial f_1}{\partial X_3} & \dots \\ \frac{\partial f_2}{\partial X_1} & \frac{\partial f_2}{\partial X_2} & \frac{\partial f_2}{\partial X_3} & \dots \\ \frac{\partial f_3}{\partial X_1} & \frac{\partial f_3}{\partial X_2} & \frac{\partial f_3}{\partial X_3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Figure 2.5 Jacobian matrix

$$\frac{\partial f}{\partial X} \approx \frac{\delta f}{\delta X} \quad (2.2)$$

Indirectly, the evaporator sequential subroutine is used to calculate the Jacobian. For each evaluation of the NR equations, the condenser subroutine is solved for new values of the ‘calc’ variables. The new values of the ‘calc’ variables alter the value of the NR residuals for the next iteration. More details about the fundamental mathematical algorithm can be found in Harshbarger and Bullard (2000).

If none of the inputs to the sequential subroutine were designated unknown (X) variables by the user, the calculation will be finished in one iteration. Otherwise, if any X variables appear among the subroutine inputs, more than one iteration can be needed to finish the calculation. In every iteration, initial guesses and current values of X variables are improved based upon the Jacobian Matrix by Eq 2.1.

All variables and parameters in XK file are categorized into four groups, marked with X, K, P and C, respectively. Xs are unknown interchangeable variables that are to be calculated by the model. All the initial values are guesses provided by the user. The number of the Xs should be equal to the number of the residual equations. Ks are the subset of interchangeable variables that are specified by user, whose values are not changed during the calculation. Ks are interchangeable with Xs because in different simulations, users want to calculate different variables. Ps are parameters that are always known by users, including the flags to select heat transfer and pressure drop correlations, provide values such as ambient air pressure or other parameters describing the refrigerant and air. Ps are noninterchangeable and cannot be changed during the calculation. Cs are informative variables calculated by the sequential subroutines, based on the inputs (Ps, Xs and Ks). Cs are not essential variables needed by the simulation system, but they are helpful for understanding the system or transmitting information to user and programmer.

Based on the variable categories, it is easier to understand the communication between main program and sequential subroutines. Generally speaking, the system communication is the transmission of the system variables. The input and outputs of the sequential subroutine are divided into four sets: Set I, Set P, Set O and Set C. The following Figure 2.6 shows the relationship among these four subsets.

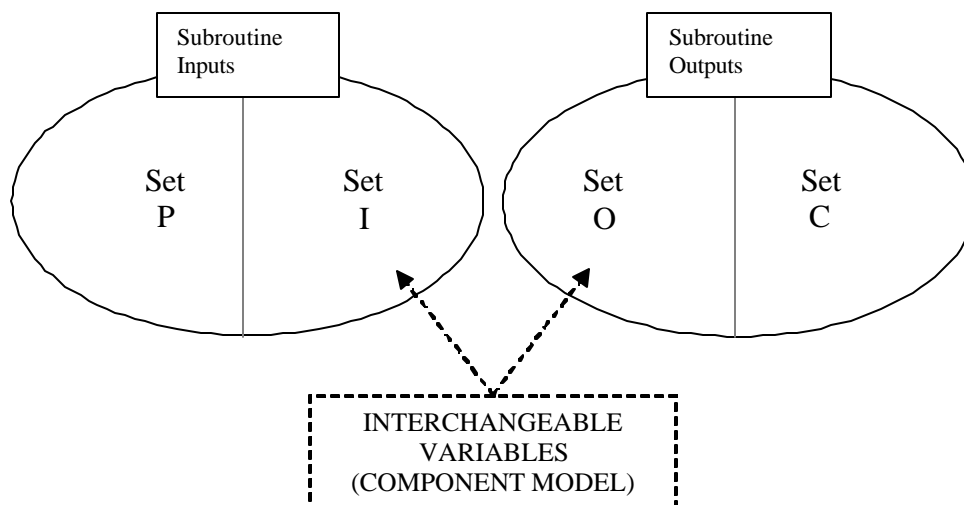


Figure 2.6 Subroutine Subsets Description

It is important to understand the implications of each variable subset prior to sorting the variables. Inputs include Set P and Set I. Set P contains only known parameters. But Set I can include both Ks and Xs, both of them are interchangeable variables. In outputs, Set C only includes Cs variables, which are not used by NR solver. Just like set I, set O includes both Xs and Ks. The interchangeable variables that are inputs to the subroutine (set I) are not present in any simultaneous equation for a single component simulation. However, each interchangeable variable in set O does appear in a NR equation. For each NR equation there is one 'X' variable that requires an initial guess. Therefore, the number of variables in set O will be the number of required initial guesses in the final simulation. By placing vaguely known variables into sets P and C, initial guesses can be limited to readily known quantities.

2.2.2 Communication between main program and supportive files

The communication between main program and supportive files is also an important issue, which is very helpful for running the simulation model correctly. Besides the NR solver and sequential subroutines simulating the system components, the model contains separate supportive files and subroutines for model initialization, checking and solution output. Although the checking files can be used as pre- or post-processors, their primary purpose is to provide a means of checking the values of variables and parameters before or after the solution.

The checking that takes place before solving is used to set logical flags that are used within the list of governing equations and subroutines. For example, the "before" checking will determine, based upon the parameters and the initial guesses of the variables, which kind of expansion device or heat exchanger is used, and a logical flag will be set accordingly. This flag will cause the NR solver to evaluate the correct set of governing equations related the correct device. For an example of "before" checking, the logical flag, CTS LHXSIM, indicates whether or not the capillary tube-suction line heat exchanger (ct-slhx) model is going to be used in the simulation. If the XKflag of CaptubeModel is a "K", then the ct-slhx model will be used and CTS LHXSIM is given a value of "true". The XKflag of the effectiveness of the ct-slhx (ectslhx) is given a value of "C" since it will be calculated in the subroutine. If the XKflag of CaptubeModel is an "X", then the ct-slhx model will not be used in the system and

CTSLHXSIM is given a value of “false”. The XKflag for the effectiveness of the ct-slhx is given a value of “K”, and the value of the effectiveness is entered.

The “after” checking is used to see if the values of certain variables are within allowable ranges (e.g. evaporating and condensing temperatures for the compressor maps).

In the model directory, a file named “XK” serves as the input for the model, providing the desired values of the parameters and the initial guess values for all the variables in this file. All the variables are global variables. Their memory addresses can be reached by all the system files during the calculation, to update the values of the X variables and calculated variables. During the NR iterations, the user-specified Ks and Ps remain constant, but Xs and Cs will be updated in each iteration by sequential subroutines and NR solver.

Other files allow the user to control the operation of the Newton-Raphson solver, and the overall operation of the program. The files “SLVERSET” and “INSTR” specify the solver options and the type of model run desired, respectively. The file SLVERSET contains the settings for various Newton-Raphson parameters such as the convergence criteria and maximum number of iterations, and it also contains information specifying the type of model output. The INSTR file tells the NR solver whether to perform a “SINGLE”, “MULTIPLE”, “SENSITIVITY”, or “UNCERTAINTY” analysis, and it also contains new specified K values different from the values in XK file. Output file name and compressor map used by the model are also indicated in the INSTR file.

2.2.3 Communication between sequential components

The basic simulation system is divided into four main components connected in the order of compressor, condenser, expansion device and evaporator. To clarify and understand the connection between components is very important. Compared with traditional systems, condenser and evaporator in modern systems may be the combination of multiple heat exchangers in serial or parallel format. The detailed communications inside the condenser or evaporator will be described in a later section.

In a multi-component system, certain variables describe the communication between components. These links between subsystems must be identified and included in a specific manner. Refrigerant is the link between components in the system. The pressure and enthalpy of the refrigerant are picked to describe the connections between components. Within the simulation system, each component is described by a subset of the residual governing equations. There are also sets of residual equations linking components in the main program. The following Figure 2.7 shows the connections of two serial components in the simulation system. One is simulating the evaporator and the other one is simulating the compressor.

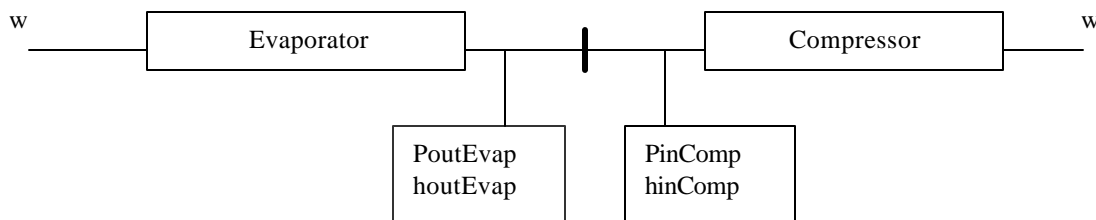


Figure 2.7 Components connection configuration

The links between these two components are the outlet pressure and enthalpy of evaporator and the inlet pressure and enthalpy of compressor. An essential aspect of the relationship between components is the definition of one state point by two variables. In this example, the outlet pressure of the evaporator is one variable while the inlet pressure to the compressor is another. While these variables are physically the same pressure, the mathematics of the system simulation dictate that two variables be used. Variables representing the same state point will be equated in a NR equation at the system level (Harshbarger and Bullard, 2000).

For a full system simulation, both components must be solved together by the NR solver. The NR equations from each individual component are included in the full set of NR equations. Additionally, NR equations that equate the linking variables are included. Figure 2.8 shows the full set of NR equations for the example system. These equations ensure that each component converges to a solution consistent with the remaining components.

By introducing two variables that describe a single state point, several advantages are obtained. The main advantage is the system simulation is allowed to converge. Another advantage is the user can easily understand the significance to a variable based upon its name. Introducing variables in this manner allows each component to be contained within its own modular solution. Because of the modular construction, components can be linked in various combinations with minimal reprogramming.

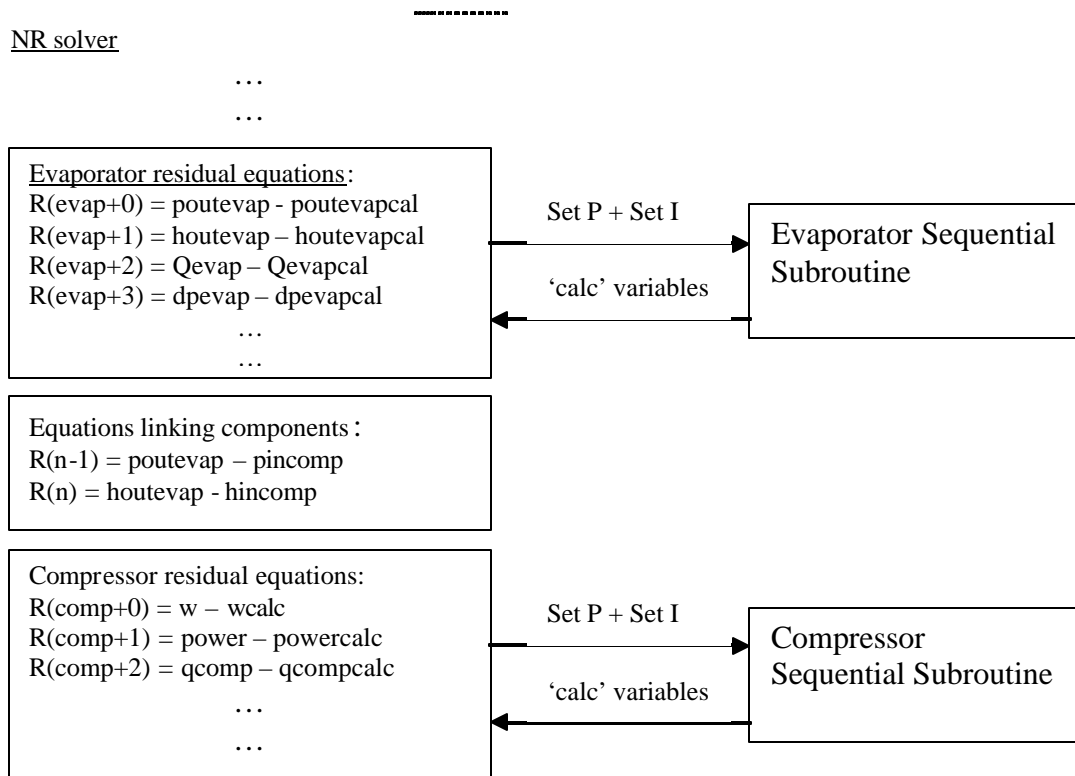


Figure 2.8 System residual equations for component connection

Chapter 3: Heat Exchanger Algorithms

3.1 Description

Harshbarger and Bullard (2000) employed a “module” algorithm to correctly simulate more complex individual heat exchanger geometries in finite-element method. “Module” is defined generally as portion, or sub-heat exchanger, part of a larger complex heat exchanger. The essential quantities defining a module are the refrigerant flow configuration and the number of tubes within the module. A heat exchanger can be defined by any number of modules.

Recently, the condenser and evaporator components are becoming more and more complicated geometrically and companies are showing more interest in simulating multi-heat exchanger systems cases, such as dual evaporator refrigerators and a/c systems, including minivan air-conditioning systems. An algorithm dealing with multiple complex condensers or evaporators in single simulation system is needed now. An algorithm, “divide-and-conquer”, is proposed here to simulate modern complex systems, which have multiple heat exchangers in serial or parallel. The name of the algorithm is borrowed from computer science. This approach divides the problem into several modules that are smaller but similar to the original one, solves the modules recursively, and then combines these solutions to create a solution to the original problem. The divide-and-conquer paradigm involves three steps.

The first step is to **divide** the whole component into a number of modules. Just like electronic circuit analysis, the whole evaporator or condenser component is a complex combination of parallel and serial configurations. Parallel and serial configurations are basic structures of the simulated system. When we start the division from the original component (first level), the number of modules is the number of the parallel modules at this level and each of these modules might consist of several serial smaller modules (second level). The smaller modules of second level can be serially divided into a number of modules. At each level, the structure can be recursively divided into many levels until each module is single stand-alone heat exchanger. In Harshbarger and Bullard (2000), this smallest element was called a “module”, and each of those modules was simulated using a finite element algorithm.

The second step is to **conquer** the modules by solving them sequentially, starting with the smallest modules. Harshbarger and Bullard (2000) showed how to solve complex single heat exchanger using finite element algorithms. This step is finalized in the sequential subroutines, calling finite element algorithms.

The third and final step is to **combine** the module solutions into the solution for the upper level component, until the original component solution is obtained. It reverses the dividing process. Newton-Raphson residual equations in the main program describe the connections among the modules. If the current modules are in series, the outlet states of the refrigerant from former module are the inlet states to the latter module. Each module shares the same mass flow rate but may have different air and refrigerant states. Otherwise if the current modules are parallel, refrigerant mass flow rates through each module must be determined by solving the equations simultaneously, by setting their exit pressures equal. For each module, running condition and heat exchanger geometry potentially decide the mass flow rate, and the sum of air and refrigerant flow rates of each parallel module should be equal to

the next upper level mass flow rate. The direction of combining is just opposite to the dividing process. More details about the combining process are provided in later sections of this chapter.

Finite element algorithm gives us a way to simulate complex geometry of heat exchangers, but the divide-and-conquer algorithm gives us a way to deal with modern complex system with any number and combination of heat exchangers. Figure 3.1 shows the general structure in evaporator component with multiple heat exchangers. All the heat exchangers are serially or parallel arranged. Details are provided to describe how we simulate this complex structure in three steps according to divide-and-conquer algorithm.

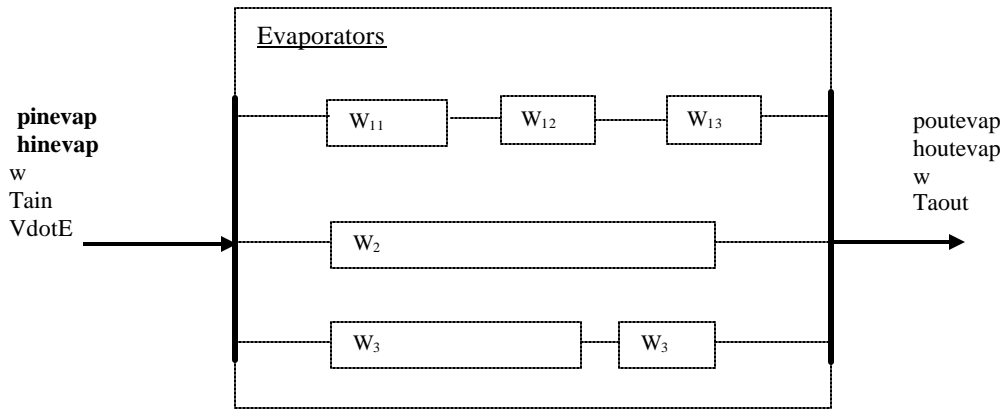


Figure 3.1 Evaporator component configurations

3.2 Heat exchanger configurations

The modern evaporator and condenser components may be the combination of serial and parallel heat exchanger geometries. There are several issues that we should pay attention to: mass flow rate distribution, air and refrigerant states, air flow direction over each heat exchanger, connections between serial and parallel heat exchangers (this connection still includes air-side), and special geometries of heat exchangers. The advantage that each heat exchanger is stand-alone gives us great flexibility to model complex combinations of serial and parallel heat exchangers. We only need to indicate correct flags to call already-built sequentially-solved subroutines that reside in the system library. In main program, residual equations that connect each stand-alone heat exchanger are built to finalize the simulation of multiple-heat exchanger system.

Parallel and serial configurations are two fundamental configurations, which are defined by the refrigerant flow direction. Any complex system can be divided into these two fundamental configurations. Parallel configuration consists of two or more heat exchangers at the same level, having identical input states but maybe different output states, depending on geometries and air-side input states. Each parallel heat exchanger does not necessarily have the exactly same geometry for general cases. If each parallel heat exchanger has the same geometry and the same air and refrigerant inlet states, they will carry the same mass flow rate fraction. Otherwise, these heat exchangers may carry different mass flow fractions because of different geometries or air-side input states or both, which will be simultaneously determined by the Newton-Raphson solver in the main program. The Figure 3.2 and Figure 3.3 show examples of the parallel and serial heat exchanger configuration, respectively.

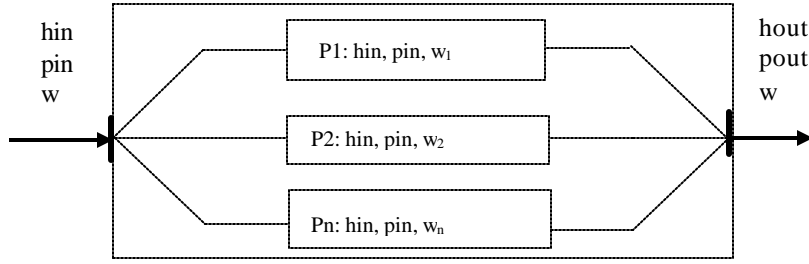


Figure 3.2 Parallel heat exchanger configuration

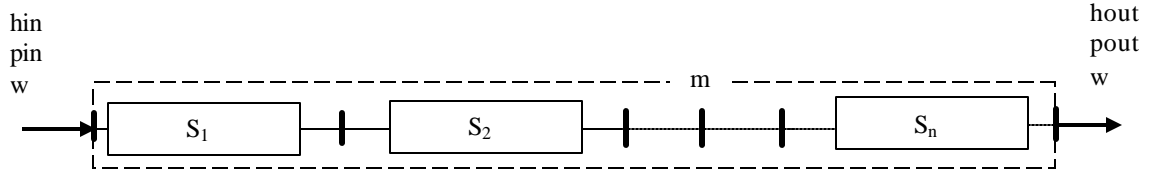


Figure 3.3 Serial heat exchanger configuration

Note that P_i or S_i can be a stand-alone heat exchanger module (smallest unit under discussion here), or they can be combinations of several heat exchangers, since we recursively divide every complex component into many different levels in order to simulate it.

Before we turn into complex structure, these two basic elements, parallel and serial configurations, need to be clearly understood. In either configuration, mass flow rate of refrigerant is the most important issue because we define these two configurations based on their refrigerant flow patterns. At the same time, the air-side flow determines the different kinds of algorithms we use to simulate the heat exchanger: e.g. counter flow, parallel flow or cross-flow, which describe the heat transfer based on airflow directions.

3.2.1 Parallel refrigerant flow

For parallel configuration, the whole mass flow rate of refrigerant is divided into the number of parallel modules, with a fraction of the mass flowing through each corresponding parallel module. Each heat exchanger may have different geometries and different air and refrigerant states might be specified. The mass flow fractions can then be calculated; they may not be equal. The following mass, momentum energy equations must apply to the parallel configuration:

$$\dot{m} = \sum \dot{m}_i \quad (i=1, 2 \dots N) \quad \text{Eq. 3.1}$$

$$\dot{m} * h_{out} = \sum \dot{m}_i * h_{out_i} \quad (i=1, 2 \dots N) \quad \text{Eq. 3.2}$$

$$P_i = P_j \quad \forall i, j \quad (i, j=1, 2 \dots N) \quad \text{Eq. 3.3}$$

Where \dot{m}_i is the mass flow rate through each of the N parallel modules; \dot{m} is the sum of each mass flow rate through all modules; h_{out_i} is the outlet enthalpy of the i^{th} module; h_{out} is the mixed outlet enthalpy from all modules; P_i and P_j are outlet pressures of i^{th} and j^{th} modules, respectively. The mass flowing through each circuit may have a different experience, as it encounters different geometries and heat transfer. At the starting point, all refrigerant has the same inlet thermodynamics state. But at outlet point of the configuration, flows from each circuit, each with a

potentially different flow rate and heat transfer experience, combine together as shown in Figure 3.2. The outlet states detected by the component outside of the configuration, h_{out} and p_{out} , result from the combinations of these mass flows, shown in the equations above.

Generally, a parallel refrigerant configuration can have three air flow patterns, as shown in the Figures 3.4, Figure 3.5 and Figure 3.6, respectively: cross/parallel, parallel/parallel and counter/parallel configurations, each requiring a different algorithm to simulate. Actually the governing equations for Figure 3.5 and Figure 3.6 are identical. However, they require two different solution algorithms. Both march downwind, but one requires the refrigerant inlet states as input, while the other requires the refrigerant outlet states in order to begin the sequential finite element algorithm.

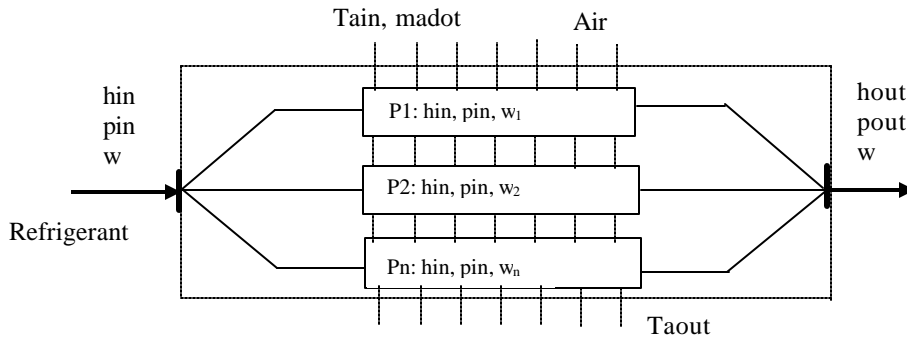


Figure 3.4 Cross/parallel flow configuration

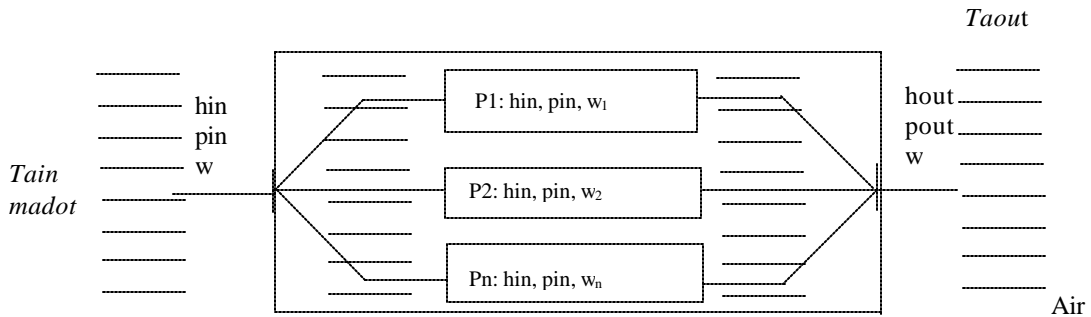


Figure 3.5 Parallel/parallel flow configuration

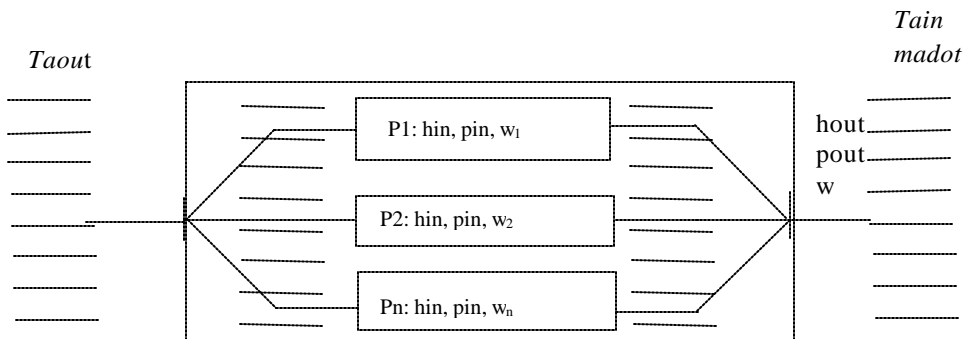


Figure 3.6 Counter/parallel flow configuration

For the cross/parallel flow configuration, air flows across each parallel module one-by-one. Each module encounters the same airflow rate, but with a different inlet temperature. The outlet temperature from former module serves as the inlet temperature for the next one. The following residual equations are included in the main program to specify the connections among the modules on both air and refrigerant sides. The equations in the main program, which includes the subroutine calls, are solved simultaneously solved by the Newton-Raphson solver.

<p>Air-side equations:</p> $R(1) = T_{ain_1} - T_{ain}$ $R(2) = T_{ain_2} - T_{aoutcalc_1}$ $R(3) = T_{ain_3} - T_{aoutcalc_2}$ <p>.</p> $R(n) = T_{aoutcalc_n} - T_{aout}$	<p>Refrigerant side equations:</p> $R(p+1) = Q_{tot} - \sum Q_{calc_i}$ $R(p+2) = M_{tot} - \sum M_{calc_i}$ $R(p+3) = w \cdot h_{out} - \sum w_i \cdot h_{outcalc_i}$ $R(p+4) = p_{out} - \sum p_{outcalc_i}$ $R(p+5) = w - \sum w_{calc_i}$
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.7 Residual equations of cross/parallel configuration

For parallel/parallel and counter/parallel configurations, each module may not encounter the whole air mass flow rate. They may have equal or unequal air flow fractions, and the sum of their fractions should be equal to the whole airflow rate. They are calculated from the frontal areas of each heat exchanger, where the current version of the model assumes that the air mass flux and pressure drop are identical for each air flow fraction. Just like refrigerant outlet thermodynamics states, the outlet enthalpy of the air results from the mixture of each mass flow fraction through each heat exchanger. The residual equations in Figure 3.8 are used to simulate these configurations, where ‘madot’ is the sum of air flow, madot_i is air fraction flowing through the ith module, haout is mixed outlet enthalpy of the combined air flows; haout_i is the outlet enthalpy of the air fraction flowing through the ith module; Afr_i is the frontal area of the ith module and Afr is the total frontal area of the whole configuration.

The outputs of the finite element subroutines return the results from each module to the main program, where the module’s governing equations and the “connection equations” describing the serial or parallel configurations are calculated simultaneously by the Newton-Raphson solver. Generally, the model can not converge in only one iteration because we can not specify the mass fractions or the input air temperatures for all modules in cross/parallel configuration except the first one. There is no difference between parallel/parallel and counter/parallel flows since they share the same residual connection equations.

<p>Air-side equations:</p> $R(1) = \dot{m}_{aout} - \sum \dot{m}_{aout_i}$ $R(2) = \dot{m}_{aout} - \sum \dot{m}_{aout_i}$ $R(3) = \dot{m}_{aout_i} - (A_{fr_i}/A_{fr}) \dot{m}_{aout}$	<p>Refrigerant side equations:</p> $R(p+1) = \dot{Q}_{tot} - \sum \dot{Q}_{calc_i}$ $R(p+2) = \dot{M}_{tot} - \sum \dot{M}_{calc_i}$ $R(p+3) = \dot{w} - \sum \dot{w}_{i_out}$ $R(p+4) = \dot{p}_{out} - \sum \dot{p}_{out_i}$ $R(p+5) = \dot{w} - \sum \dot{w}_{calc_i}$
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.8 Equations of counter/parallel and parallel/parallel configurations

3.2.2 Serial refrigerant flow

Compared to parallel configuration, it is easier to analyze and simulate the serial configuration because all the refrigerant has the same experience, as it flows serially through all modules in this configuration. The main issue we need to consider here is the sequence in which the modules are connected. States of connection points between any two serial modules (for example, numbered m and $m+1$) are described by two groups of variables: one is the outlet state from the former module and the other one is the inlet state of the latter one. Corresponding residual equations that are created to describe the connections are shown in Figure 3.9. Details about connections between parallel and serial configurations will be provided in the specified example in the following section.

$$R(n) = h_{out_m} - h_{in_m+1}$$

$$R(n+1) = p_{out_m} - p_{in_m+1}$$

$$R(n+2) = w_m - w_{m+1}$$

Figure 3.9 Residual equations describing connection

As in the case of parallel configurations, we have equations describing connection of refrigerant states between serial modules. We also need to pay attention to the airside flow situations. We also have three kinds of configurations: cross/serial flow, parallel/serial flow and counter/serial flow, shown in Figures 3.10, 3.11 and 3.12, respectively.

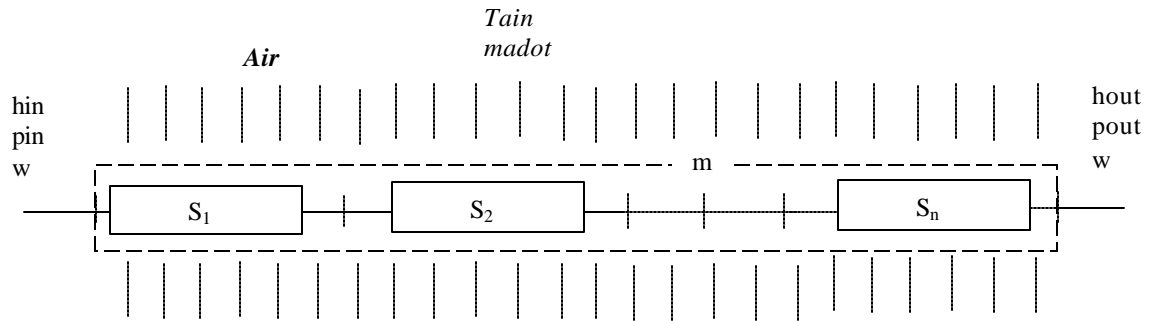


Figure 3.10 Cross/serial flow configuration

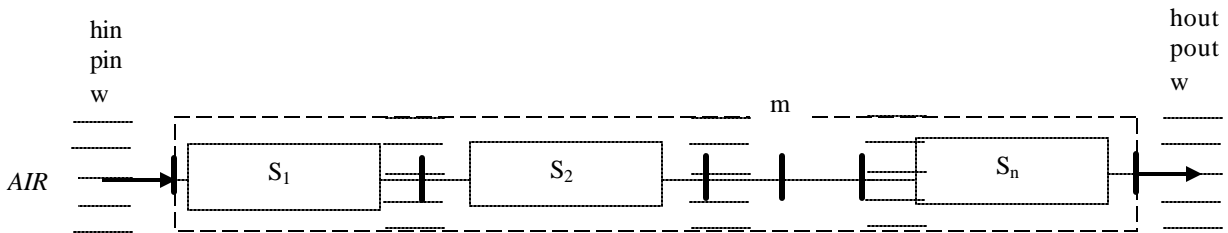


Figure 3.11 Parallel/serial flow configuration

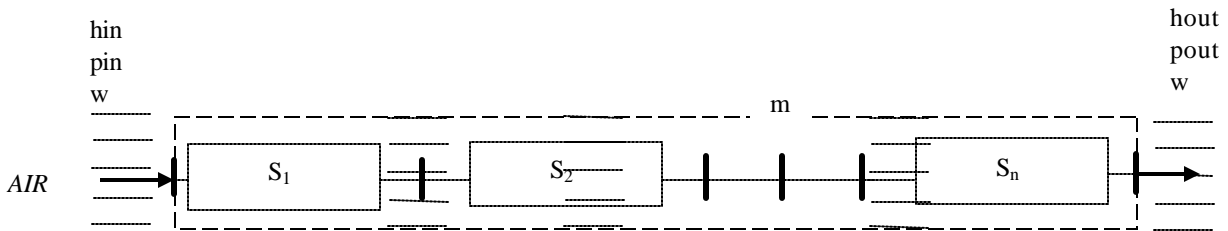


Figure 3.12 Counter/serial flow configuration

For the cross/serial flow case, all the modules have the same inlet air states, but maybe different air mass fractions if they do not have the exactly same geometries. The following residual equations in Figure 3.13 simulate this configuration.

<p><i>Air-side equations:</i></p> $R(1) = m_a \text{ haout} - \sum m_{a_i} \text{ haout_i}$ $R(2) = m_a - \sum m_{a_i}$ $R(3) = m_{a_i} - (\text{Afr_i}/\text{Afr}) m_a$ $R(4) = \text{Taout_i} - \text{Taoutcalc_i}$	<p><i>Refrigerant-side Equations:</i></p> $R(n) = Q_i - Q_icalc$ $R(n+1) = M_i - M_icalc$ $R(n+2) = \text{hout_i} - \text{hout_icalc}$ $R(n+3) = \text{Pout_i} - \text{Pout_icalc}$ <p>...</p> $R(n+m) = \text{hout_i} - \text{hin_i+1}$ $R(n+m+1) = \text{pout_i} - \text{pin_i+1}$ $R(n+m+2) = w_i - w$ $R(n+m+3) = \text{hout} - \text{hout_ncalc}$ $R(n+m+4) = \text{pout} - \text{pout_ncalc}$ $R(n+m+5) = M_{\text{tot}} - \sum M_i$
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.13 Residual equations describing cross/serial configuration

For the counter/serial and parallel/serial configurations, refrigerant side has the same residual equations as cross flow configuration, and has easier air side equations because all the air passes each heat exchanger sequentially, and undergoes the same heat transfer experience. Both configurations are governed by almost the same residual equations in the main program, but the sequentially -solved finite element subroutines for each module require specification of inlet refrigerant enthalpy and pressure for the parallel/serial configuration, and the refrigerant outlet state for the counter/serial configuration. The boldface equations in Figures 3.14 highlight the difference between these two configurations.

<p><i>Air-side equations:</i></p> $R(1) = \text{Taout} - \text{Taout_n}$ $R(2) = m_a - m_{a_i}$ $R(3) = \text{Taout_i} - \text{Tain_i} + 1$ $R(4) = \text{Tain_1} - \text{Tain}$ $R(5) = \text{Taout_i} - \text{Taoutcalc_i}$	<p><i>Refrigerant-side Equations:</i></p> $R(n) = Q_i - Q_icalc$ $R(n+1) = M_i - M_icalc$ $\mathbf{R(n+2) = \text{hout_i} - \text{hout_icalc}}$ $\mathbf{R(n+3) = \text{Pout_i} - \text{Pout_icalc}}$ <p>...</p> $R(n+m) = \text{hout_i} - \text{hin_i+1}$ $R(n+m+1) = \text{pout_i} - \text{pin_i+1}$ $R(n+m+2) = w_i - w$ $\mathbf{R(n+m+3) = \text{hout} - \text{hout_ncalc}}$ $\mathbf{R(n+m+4) = \text{pout} - \text{pout_ncalc}}$ $R(n+m+5) = M_{\text{tot}} - \sum M_i$
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.14 Residual equations describing parallel/serial configuration

3.3 Complex component analysis and simulation

An example, which shows a general structure of a complex component, is used to illustrate the use of the divide-and-conquer algorithm, which is designed to deal with the multiple-heat exchanger component. As described above, three steps are needed to simulate the complex structure.

3.3.1 Division

Division is the first step of the divide-and-conquer algorithm and we start from the original complex components consisting of multiple condensers or evaporators. The division will continue at consecutive levels until every module is an individual stand-alone heat exchanger, where the finite element method can be used for its simulation. Residual equations describing the connections and combinations among different heat exchangers are also listed in main program. The following paragraphs describe the procedure of dividing the complex structure shown in Figure 3.1.

Division is executed at different levels and the original component is the first level. The original component consists of three parallel heat exchangers, which are second level elements. At each level, we focus first on any parallel configuration. Without parallel elements, we then turn to serial configurations. A detailed view of the second level is shown below:

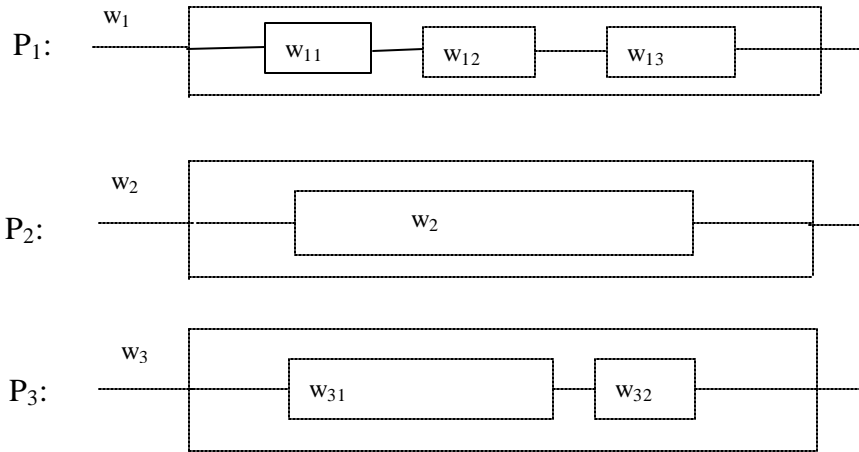


Figure 3.15 Second level elements

$W = w_1 + w_2 + w_3$, where w is the mass flow rate of refrigerant of the whole system and w_1 , w_2 and w_3 are the mass flow rates of refrigerant through P₁, P₂ and P₃, respectively. At the same time, w_1 is equal to w_{11} , w_{12} , w_{13} and w_3 is equal to w_{31} , w_{32} since the modules are serial. w_{11} , w_{12} and w_{13} are mass flow rates through the three serial modules in P₁ and w_{31} , w_{32} are mass flow rates through the two serial modules in P₃.

Two of the second level heat exchangers P_{*i*} (*i*=1,2,3) are serial. Therefore we continue the division because there are still heat exchangers that are not single modules. In this example, the first heat exchanger P₁ includes three smaller modules, which are serially connected and share the same mass flow rate of refrigerant, w_1 . P₃ has two serial modules, which share the mass flow rate of refrigerant, w_3 .

P_1 is divided into three smaller modules at the third level:

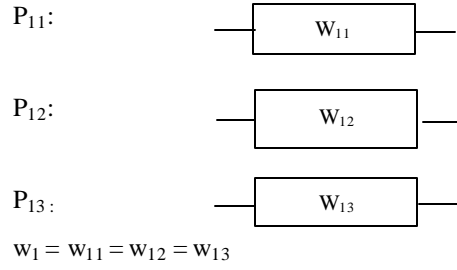
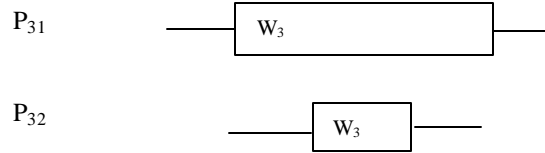


Figure 3.16 Third level modules

P_{11} , P_{12} , P_{13} are all individual stand-alone modules, which are the smallest units of this algorithm, and can not be divided any more. Similarly, P_3 also can be divided into two third-level modules:



where P_{31} and P_{32} share the same mass flow rate w_3 since they are serially connected stand-alone heat exchangers and do not need further division any more.

Now all the modules are individual stand-alone heat exchangers, which can be simulated by calling sequential finite element subroutines. Many such stand-alone subroutines for simulating different geometries are already stored in the main library of the simulation system. In the main program, one just needs to select the right subroutine names and flags for different heat exchangers. These kinds of stand-alone subroutines give us great flexibility in simulating different heat exchangers without rewriting codes and redesigning algorithms.

Briefly, all parallel configurations share the same inlet refrigerant states, and their outlet pressures must be identical, but not necessarily their refrigerant outlet enthalpies, or their air inlet and outlet states. As discussed above, the outlet pressure and enthalpy result from the combination of all parallel outlets. All serial configurations share same mass flow rates of refrigerant. That is the criterion we use to divide the complex components. On the air side, we need to pick up different residual equations to simulate the connections among heat exchangers defined by different airflow directions. In order to simulate a single stand-alone heat exchanger, airflow directions still define corresponding algorithms we should use in simulation. The details will be provided in the latter sections of this chapter.

3.3.2 Conquer

Conquer is the second step of the divide-and-conquer algorithm, whose purpose is to simulate every smallest module, stand-alone single heat exchanger. Sequential stand-alone subroutines have been built and are available within ACRC for many different geometries. These subroutines are stored in library of the simulation system.

Within conquer process, we focus on simulating every single heat exchanger with different geometries. Harshbarger and Bullard (2000) already have developed “module” algorithms to simulate complex single heat exchangers, even multi-slab counterflow and parallel flow designs such as those shown in Figure 3.17.

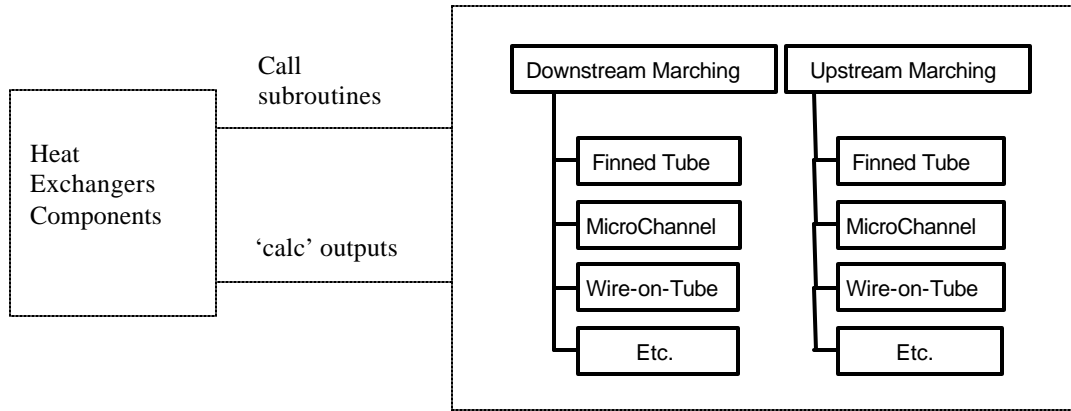


Figure 3.17 Overall structures of heat exchangers subroutines

Figure 3.17 shows the structure of heat exchangers subroutines that can be called to simulate different heat exchanger geometries without re-compiling source codes. These subroutines are sequential stand-alone procedures that simulate one heat exchanger having a particular geometry. The subroutines can be shared by different systems with the heat exchangers of same geometries and they can be called recursively when simulating a more complex system, which has multiple heat exchangers.

Based on the relationship between the flow directions of refrigerant and air, there are two kinds of algorithms, upstreaming and downstreaming, referring to the refrigerant flow direction. Simulations always proceed downwind, so the upstreaming and downstreaming algorithms apply to overall counter-flow and parallel-flow heat exchanger configurations, respectively. To enable sequential solution of the upstreaming subroutine, refrigerant outlet pressure and enthalpy are needed as input because the calculation is started from air inlet (refrigerant outlet) point, and the calculated inlets of pressure and enthalpy are returned. For the downstreaming subroutines, because the air and refrigerant are flowing in the same direction, each of their states is assumed to be known. Then the highest and lowest temperatures needed by ϵ -NTU method are easily decided for each small element, as the calculation starts from inlet point and marches downwind and towards the refrigerant outlet. Calculated outlet pressure and enthalpy of refrigerant are the returned values. Therefore the residual equations simulating these heat exchangers are different. The Figure 3.18 shows the residual equation groups simulating downstreaming and upstreaming algorithms for the single stand-alone heat exchanger, respectively. The most important difference is in the first two boldface equations shown in both boxes.

<u>Downstreaming Equations:</u>	<u>Upstreaming Equations:</u>
R(n) = hout – hout_calc	R(n) = hin – hin_calc
R(n+1) = pout – pout_calc	R(n+1) = pin – pin_calc
R(n+2) = dp – dp_calc	R(n+2) = dp – dp_calc
R(n+3) = M – M_calc	R(n+3) = M – M_calc
R(n+4) = Q – Q_calc	R(n+4) = Q – Q_calc
R(n+5) = Tairout – Tairout_calc	R(n+5) = Tairout – Tairout_calc
R(n+6) = Tout – Tout_calc	R(n+6) = Tout – Tout_calc

Figure 3.18 Downstreaming and upstreaming equations interface

A user-selected flag is set to select either the downstream or the upstream marching algorithm, corresponding to overall parallel or counterflow, respectively. A separate flag is used to select the type of heat exchanger in use. This structure allows the locations of the governing equations to be logically organized within the source code. Because of the sequential nature of the subroutines, the solution method and the assumptions are more apparent and understandable to users.

3.3.3 Combination

Combination is the final step of the divide-and-conquer algorithm, which reverses the reverse process of division. The programmer reunites the divided modules. That is, we start from the lowest level of the modules (single stand-alone heat exchanger), build the connections and combinations until the highest level to finish the simulation of complex component. The combinations and connections issues among multiple heat exchangers are the most important points we need to understand for our simulations. The connection and combination equations are written in residual format, and are listed in the main program and simultaneously solved by Newton-Raphson solver.

Harshbarger and Bullard (2000) have provided details about complex heat exchanger using finite element algorithms. Here we focus on the connections and combinations among multiple heat exchangers in the same component. Residual equations describing the combination of the former example in ‘division’ step are grouped in the later section.

At the third level, P_{11} , P_{12} and P_{13} are the smallest units, which are serially connected. The links between any two serial modules are the inlet and outlet pressures and enthalpies of refrigerant flowing through the whole serial structure. According to their geometries, finite element subroutines are called to simulate them. At the same time, each module has a group of residual equations describing the heat transfer performance. Figure 3.18 gives the equations of individual module for both upstreaming and downstreaming configurations. Here we focus on the combination of these three modules. Figure 3.19 shows the residual equations and connection equations listed in the main program, for the case where air flows parallel to the refrigerant, so the downstreaming algorithm is used to simulate the individual modules.

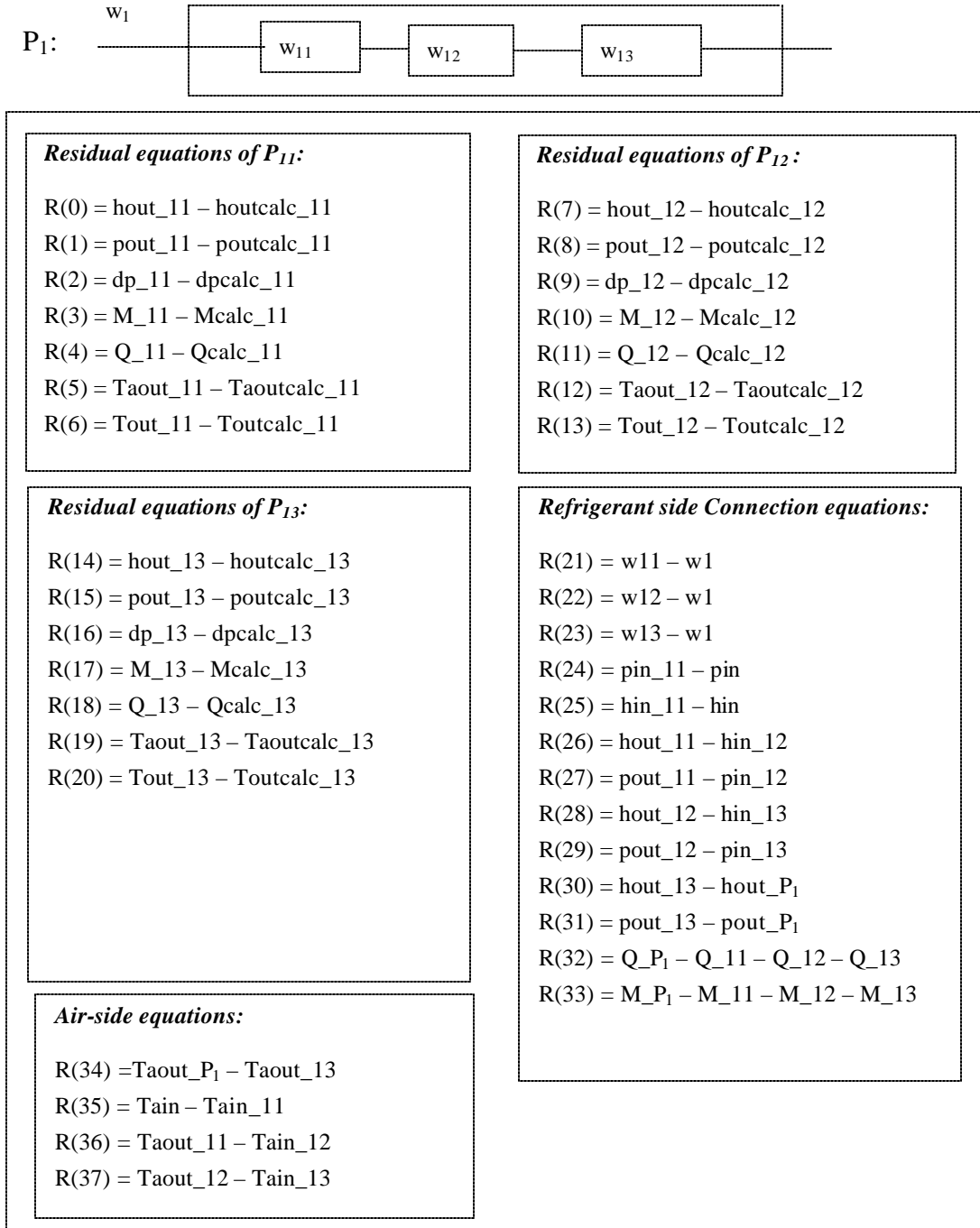


Figure 3.19 Residual equations groups simulating P1

In order to classify the variables, we append the number to them, such as hout_11, hout_12, which are outlet enthalpies of P_{11} and P_{12} , respectively. Pout_11 and Pout_12 are outlet pressures of P_{11} and P_{12} . Hout_P1 and pout_P1 are the outlet enthalpy and pressure of the whole module, P_1 . We can explain other variables in the same way.

Similarly, P_3 has two serial stand-alone modules, P_{31} and P_{32} , which can be simulated by using the same groups of residual equations and the connection equations used for P_1 , as shown in Figure 3.21. In order to simplify this example, we suppose air flows parallel to the refrigerant in all the modules. P_2 is the smallest unit and can be simulated as a single module, as shown in Figure 3.20.

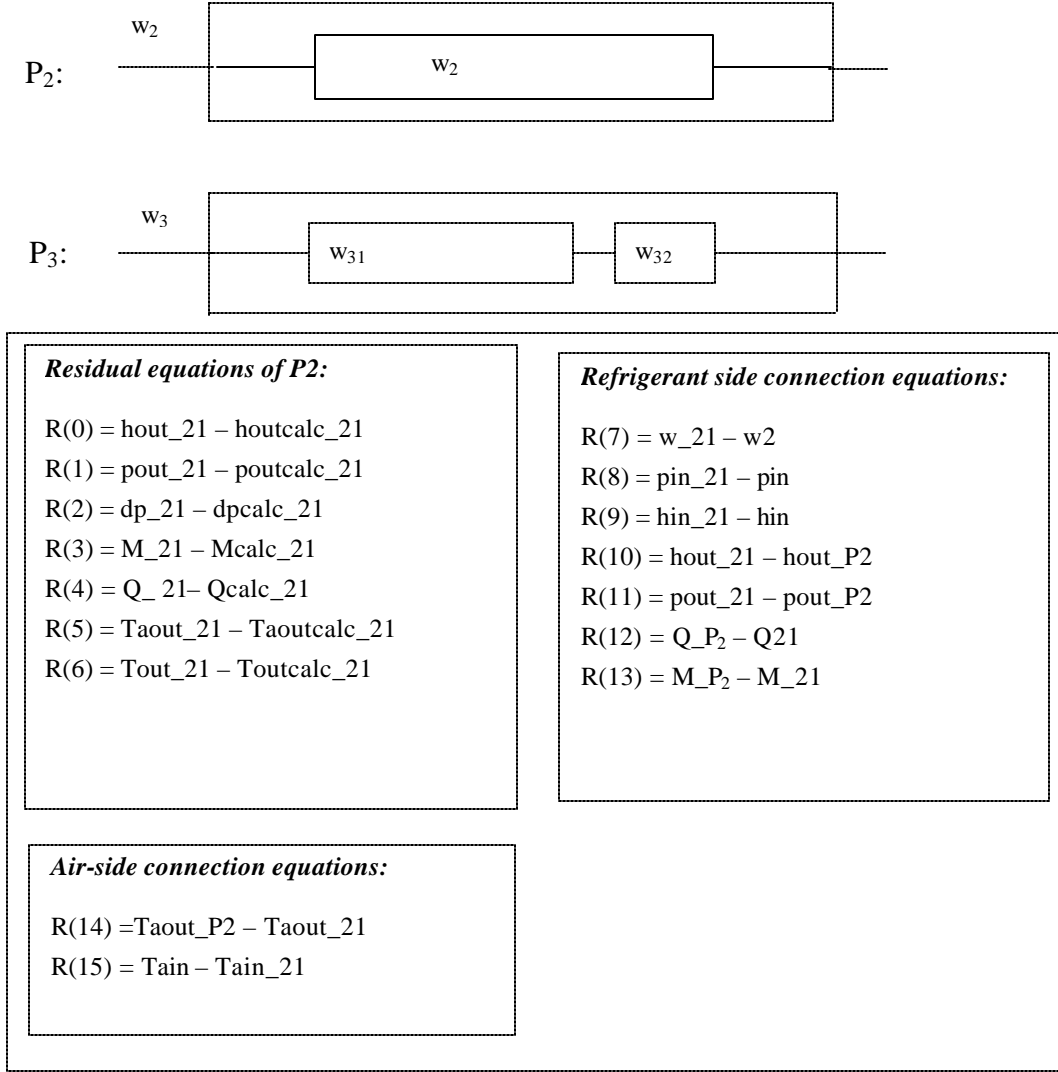


Figure 3.20 Residual equations simulating P_2

<p>Residual equations of P_{31}:</p> <p> $R(0) = h_{out_31} - h_{outcalc_31}$ $R(1) = p_{out_31} - p_{outcalc_31}$ $R(2) = dp_31 - dp_{calc_31}$ $R(3) = M_31 - M_{calc_31}$ $R(4) = Q_31 - Q_{calc_31}$ $R(5) = Ta_{out_31} - Ta_{outcalc_31}$ $R(6) = Tout_31 - Tout_{calc_31}$ </p>	<p>Residual equations of P_{32}:</p> <p> $R(7) = h_{out_32} - h_{outcalc_32}$ $R(8) = p_{out_32} - p_{outcalc_32}$ $R(9) = dp_32 - dp_{calc_32}$ $R(10) = M_32 - M_{calc_32}$ $R(11) = Q_32 - Q_{calc_32}$ $R(12) = Ta_{out_32} - Ta_{outcalc_32}$ $R(13) = Tout_32 - Tout_{calc_32}$ </p>
<p>Air-side connection equations:</p> <p> $R(14) = Ta_{in_32} - Ta_{out_31}$ $R(15) = Ta_{in} - Ta_{in_31}$ $R(16) = Ta_{out_32} - Ta_{out_P3}$ </p>	<p>Refrigerant side connection equations:</p> <p> $R(17) = w_31 - w_3$ $R(18) = pin_31 - pin$ $R(19) = hin_31 - hin$ $R(20) = h_{out_32} - h_{out_P2}$ $R(21) = p_{out_32} - p_{out_P2}$ $R(22) = w_32 - w_3$ $R(23) = h_{out_31} - hin_32$ $R(24) = p_{out_31} - pin_32$ $R(25) = Q_{P3} - Q_{31} - Q_{32}$ $R(26) = M_{P3} - M_{31} - M_{32}$ </p>

Figure 3.21 Residual equations simulating P_3

We built residual equations to simulate these three parallel modules, P_1 , P_2 and P_3 . We need to turn to the combination of these parallels to finalize the combination of the complex component. Figure 3.22 contains both the refrigerant side and air side combination equations. Total heat transfer and refrigerant charge, Q and M , are calculated as well as each air mass fractions, where it is assumed that the air mass flux is identical across the frontal areas. Refrigerant outlet enthalpy results from the mixture of the parallel circuits.

<p>Refrigerant side connection equations:</p> $R(0) = w - w_1 - w_2 - w_3$ $R(1) = w \cdot h_{out} - w_1 \cdot h_{out_P1} - w_2 \cdot h_{out_P2} - w_3 \cdot h_{out_P3}$ $R(2) = p_{out_P1} - p_{out}$ $R(3) = p_{out_P2} - p_{out}$ $R(4) = p_{out_P3} - p_{out}$ $R(5) = Q - Q_{P1} - Q_{P2} - Q_{P3}$ $R(6) = M - M_{P1} - M_{P2} - M_{P3}$	<p>Air side connection equations:</p> $R(7) = V_{adot} - V_{adot_P1} - V_{adot_P2} - V_{adot_P3}$ $R(8) = V_{adot} \cdot ha_{out} - V_{adot_P1} \cdot ha_{out_P1} - V_{adot_P2} \cdot ha_{out_P2} - V_{adot_P3} \cdot ha_{out_P3}$ $R(9) = V_{adot_P1} - V_{adot} \cdot (Afr_P1/Afr)$ $R(10) = V_{adot_P2} - V_{adot} \cdot (Afr_P2/Afr)$ $R(11) = V_{adot_P3} - V_{adot} \cdot (Afr_P3/Afr)$
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.22 Combination equations for parallel modules

If the air side performance is not parallel but cross flow, the refrigerant side connection equations are the same, but air side equations need to be changed. Figure 3.23 shows the air side connection equations for the cross flow configuration.

<p>Air-side equations:</p> $R(1) = Ta_{out} - Ta_{out_P3}$ $R(2) = m_a - m_{a_P1}$ $R(3) = m_a - m_{a_P2}$ $R(4) = m_a - m_{a_P3}$ $R(5) = Ta_{out_P1} - T_{ain_P2}$ $R(6) = Ta_{out_P2} - T_{ain_P3}$ $R(7) = T_{ain_P1} - T_{ain}$

Figure 3.23 Airside connection equations for cross flow configuration

By ‘divide-and-conquer’ algorithm, any kind of complex component with multiple heat exchangers with any kind of geometries can be simulated. A simple example, dual-evaporator refrigerator simulation model where two evaporators are serially connected, is provided in the Chapter 4.

Chapter 4: Dual Evaporator System Modeling

4.1 Introduction

Dual evaporator refrigerators were modeled using a computer simulation. Modifications were made to the previously developed code in order to simulate better two evaporators arranged in series, served by a single condenser and compressor.

The model was initially developed for the study of single evaporator refrigerator-freezers at the Air Conditioning and Refrigeration Center (ACRC) at the University of Illinois at Urbana-Champaign. It consists of a general Newton-Raphson solver linked to a series of equations and functions that describe the particular refrigeration system being modeled (Mullen and Bullard (1994) and Mullen *et al.* (1998)). The simulation model for refrigerators is called RFSIM. The model assumes a steady state operation and the single evaporator version is described in more detail by Woodall and Bullard (1996). RFSIM was modified (Stein *et al.* (1999)) for dual evaporator refrigerators by adding a second evaporator in the fresh food section and eliminating air exchange between the compartments. The fresh food evaporator is modeled as a two-phase region and the freezer evaporator includes both a two-phase region and a single-phase superheated region. Additional modifications were needed to accurately represent the prototypes tested. In response to a manufacture's request, we tried to simulate a parallel-configured dual evaporator system by using the single-evaporator version of RFSIM. One evaporator was simulated, supposing the other one idle. Only a few variables and residual equations need to be modified to simulate such a dual evaporator system. More details can be found in the Appendix B.

A simulation model with all equations solved simultaneously built by Stein *et al.* (1999) and Gerlach and Newell (2000), for a serially-configured dual-evaporator system. This chapter describes a new simulation model with a modular structure. Every system component has an associated sequential subroutine describing the component. The number of initial guess values is thereby decreased substantially, from 144 to 67. Moreover, structured and independent sequential subroutines can be easily embedded in the simulation system without recompiling and changing codes.

The nomenclature used in Stein *et al.* (1999) and continued here is that the freezer compartment variables are written simply such as "tevap." The fresh food variables have an "f" added, e.g. tevapf. Alternatively, a freezer variable is denoted with a "z" added, e.g. tevapz. In the new model structure, the variable names are kept unchanged, but all the variables are categorized into four categories as discussed above and by Harshbarger and Bullard (2000).

4.2 Sequential simulation

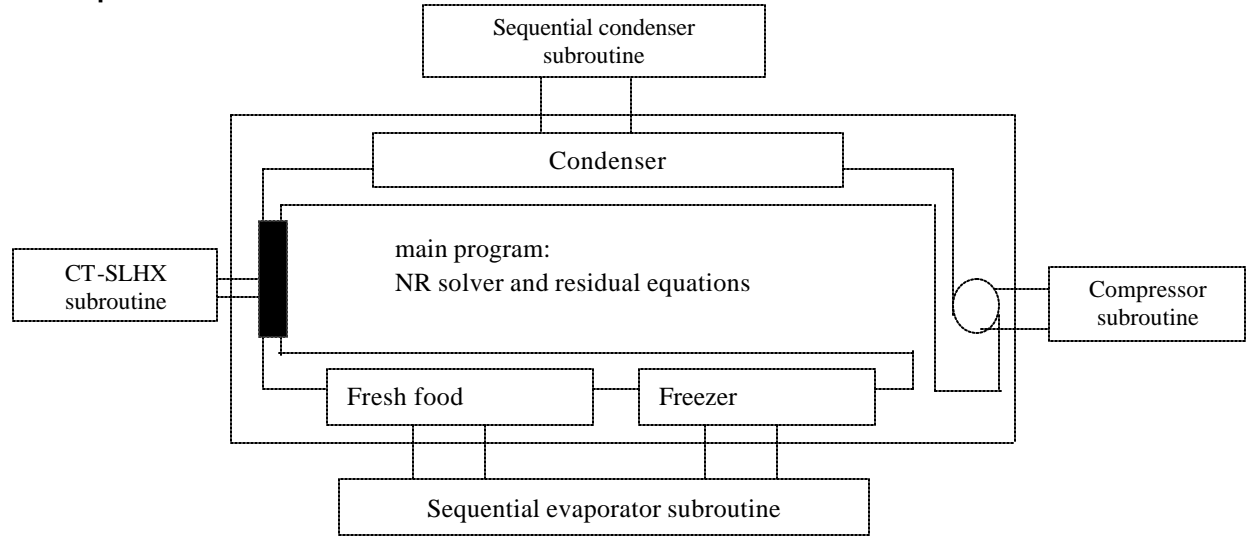


Figure 4.1 New model structure of dual evaporator system

In the new model structure, the dual-evaporator system is divided into several components: compressor, condenser, ct-slhx and evaporator. The evaporator component includes two serial evaporators: fresh food evaporator and freezer evaporator. Each component is associated with a sequential subroutine describing the component. Evaporator component will call sequential subroutine twice, with flags changed and different values describing the heat exchanger geometries and inlet conditions. All the components are serial, so the residual equations simulating the connections among the components are created. The evaporator component only has two serial heat exchangers, or subcomponents, which are “connected” by defining the intermediate states in the main program. The connection issue is the main issue to be considered at this point we do not need to be concerned about combining flows among the subcomponents since there are no parallel heat exchangers. More details are provided below.

4.2.1 Compressor

The compressor subroutine used by the dual evaporator system is based on the manufacturer’s performance map specified in the ‘instr.base’ instruction file. Using these inputs, the compressor subroutine calculates mass flow rate through the compressor, power consumed by the compressor; refrigerant-side energy balance about the compressor; air-side energy balance about the compressor; and a rate equation describing the heat transfer from the compressor shell to the air stream. The mass flow rate through the compressor and the power consumed by the compressor are described by compressor map stored in the system library. The mass flow rate and power consumption are calculated as functions of the saturation temperatures corresponding to the inlet and outlet pressures of the compressor. These relations or data necessary to make them are available from the manufacturers. In compressor subroutine, two equations involving the compressor mass flow rate and power consumption appear as follows:

$$R(\text{comp}+0) = \text{beta_Wmap} * \text{wf}(\text{tsatoutcomp}, \text{tsatincomp}, \text{CompNum}) - w \quad (\text{Eq. 4.1})$$

$$R(\text{comp}+1) = \text{beta_Pmap} * \text{pcompf}(\text{tsatoutcomp}, \text{tsatincomp}, \text{CompNum}) - \text{powercomp} \quad (\text{Eq.4.2})$$

Refrigerant-side energy balance equation about the compressor is a classic application of the first law of thermodynamics for a control volume:

$$R(\text{comp}+2)=\text{BTU}(\text{powercomp})-w*(\text{houtcomp}-\text{hincomp})-q_{\text{comp}} \quad (\text{Eq. 4.3})$$

Similarly, the following is the residual equation to simulate the air-side energy balance:

$$R(\text{comp}+3)=\text{mdotacond} * (\text{ha}(\text{tacondfanin}) - \text{ha}(\text{tacondout})) - q_{\text{comp}} \quad (\text{Eq. 4.4})$$

The rate of heat transfer from the compressor can also be described through the use of a convection heat transfer relation, and an empirical relationship also obtained from the manufacturer's compressor data, expressing shell temperature as a linear function of discharge temperature (see Kim and Bullard, 2000).

$$T_s = -3.4407 + 0.88355 * t_0$$

$$R(\text{comp}+4)=hA_{\text{comp}} * (T_s - \text{tacondout}) - q_{\text{comp}} \quad (\text{Eq. 4.5})$$

Beta_Wmap, beta_Pmap and CompNum are the compressor inputs from system XK file specified by the user. The basic purpose of beta_Wmap and beta_Pmap is to scale the compressor maps to simulate the effect of a change in compressor speed or compressor size. Tsat0 and tsat11 are variables calculated by the subroutine based on pressure inputs: inlet pressure and outlet pressure of the compressor p11 and p0. Wf and pcompf are functions to calculate power and mass flow rate stored in library file. H0, mdotacond and Ts are calculated variables. Tacondout and h11 are inputs variables of compressor subroutine. Qcomp, w, , powercomp, taconfanin and t0 are output variables of the subroutine.

4.2.2 Condenser

The condenser is modeled as a cross-flow heat exchanger, using a finite-element method in the new model structure. All stand-alone sequential subroutines simulating different geometries (e.g. wire-on-tube) are stored in system library. The special flags and subroutine name are used to call the corresponding sequential subroutine, returning the expected calculated outputs needed by the subset of the system residual equations dealing with that component.

The Figure 4.2 shows schematically the interface between the NR solver and the sequential condenser subroutine. The seven NR residual equations are shown within the NR solver box. Each equation equates an interchangeable variable with its corresponding subroutine output variable. The NR solver performs several iterations in determining the solution to a set of equations. For the condenser (component) simulation, the NR solver solves a set of seven simultaneous equations. The solver simultaneously forces the values of each of the seven equations, written in residual format, to zero. The number of simultaneous equations is equal to N, the number of 'calc' variables.

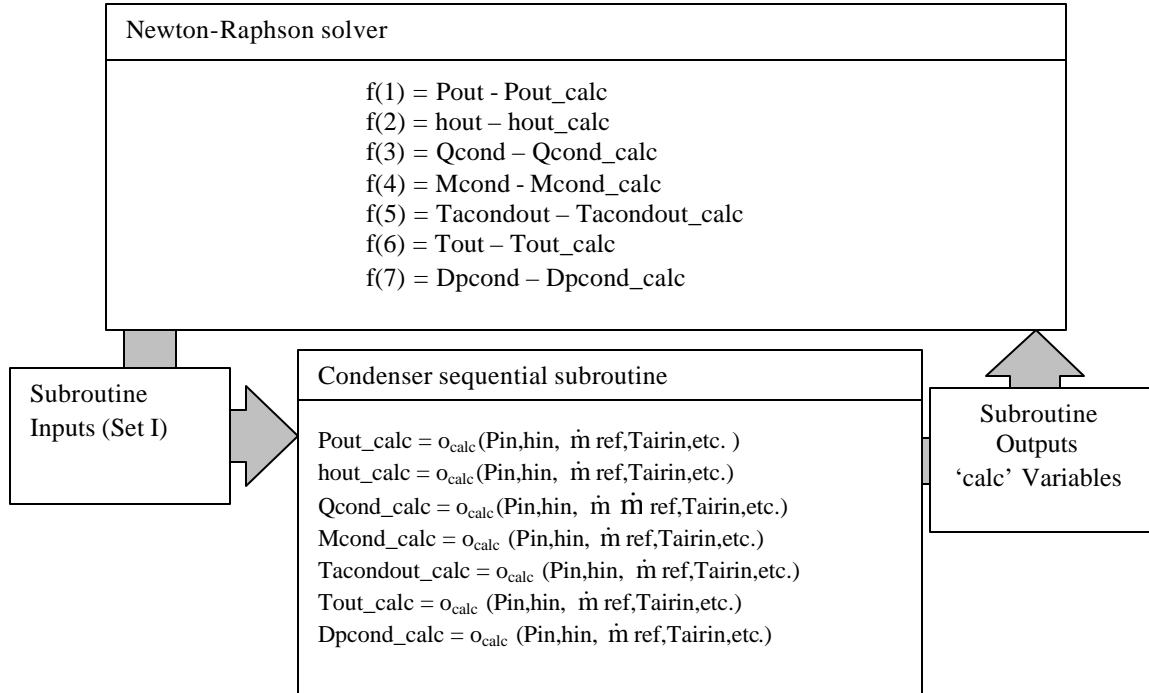


Figure 4.2 interface of residual equations and condenser sequential subroutine

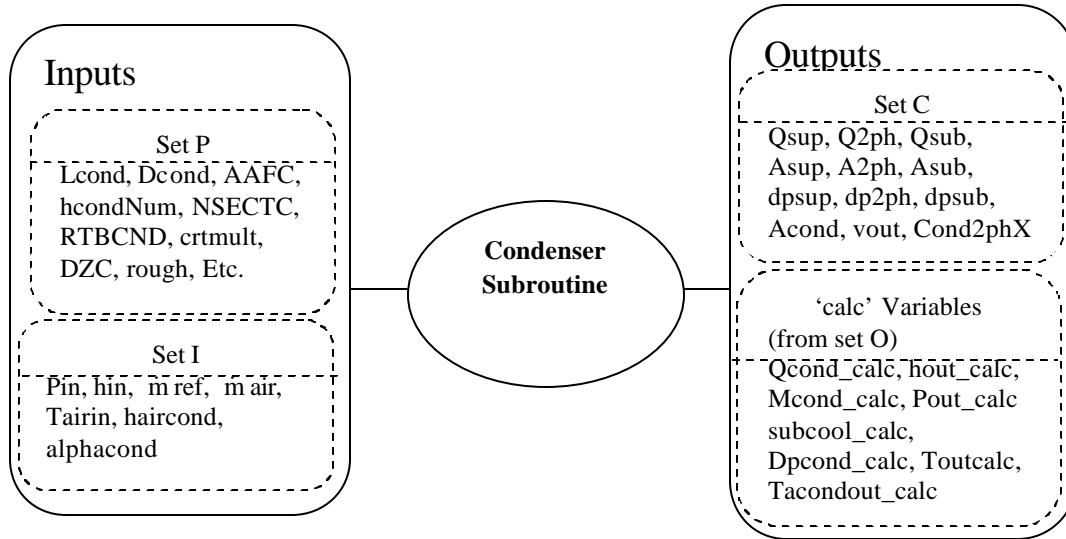


Figure 4.3 condenser sequential subroutine variable categories

Each 'calc' output of subroutine is the function of inlet state, mass flow rate, inlet conditions and geometry. Iteratively, NR solver updates the X variables associated with this component until the system converges to a series of solution. If there are no X variables among the inputs to the subroutine, the calculation will finish in only one iteration. Otherwise, iterations will be necessary, based on good initial guess values for the unknown X variables.

Figure 4.3 shows the inputs and outputs of the condenser finite-element subroutine. Variable names are defined in Appendix E of TM22. The inputs include heat exchanger geometry, inlet conditions, mass flow rate of refrigerant through the heat exchanger, air mass flow rate and inlet temperature, plus some other necessary XK

variables needed by the subroutine to simulate the condenser. Usually, haircond is calculated in subroutine by library function. But in this version of the model, haircond is a user-specified input to the subroutine. The outputs include two main parts: Set C and Set O. The set C consists of calculated variables, which are directly returned back to user interface, but not to the NR solver. The Set O are calculated X variables, needed by NR solver to simultaneously solve the set of residual equations associated with the X variables. From the viewpoint of sequential subroutine, the inputs and outputs are not interchangeable. But at the main program level, the XK variables are interchangeable, which is the big advantage of the NR solver. The number of unknown X variables must be kept equal to the number of residual equations.

4.2.3 Captube-suction line heat exchanger

The residual equations that describe the behavior of the capillary tube-suction line heat exchanger (ct-slhx) are substantially different from the other groups of residual equations. There are actually two different sets of equations, or submodels, that can be used to model the CT-SLHX. One sub-model is based upon a finite-difference solution of the governing equations for refrigerant flow through it. This method calculates directly the mass flow rate and heat transfer that takes place within the component based on published correlations. The other sub-model is a simple method that relies on a user-specified heat transfer effectiveness of the CT-SLHX, instead of performing geometry-specific calculation. In this simple submodel, only two residual equations describe the capillary tube-suction line heat exchanger. The first one predicts the amount of the heat transfer from the hot refrigerant in the capillary tube to the colder refrigerant in the suction line based upon the user-supplied value of the effectiveness: ectslhx, one variable from XK:

$$R(\text{cap}+0) = \text{ectslhx} * (\text{hpt}(\text{pincomp}, \text{tinexp}) - \text{houtE}) - (\text{hincomp} - \text{houtE}) \quad (\text{Eq. 4.6})$$

Where $(\text{hincomp} - \text{houtE})$ represents the actual heat transfer and $(\text{hpt}(\text{pincomp}, \text{tinexp}) - \text{houtE})$ represents the maximum heat transfer that could occur when the refrigerant at the suction line outlet reaches the temperature of the refrigerant at the capillary tube inlet.

The second residual equation that describes the capillary tube – suction line heat exchanger in this case is the refrigerant-to-refrigerant energy balance for the component. It is assumed that there is no heat transfer from the capillary tube or suction line to the environment:

$$R(\text{cap}+1) = (\text{hinexp} - \text{hinE}) - (\text{hincomp} - \text{houtE}) \quad (\text{Eq. 4.7})$$

This residual equation sets the change in enthalpy across the capillary tube $(\text{hinexp} - \text{hinE})$ equal to the change in enthalpy across the suction line $(\text{hincomp} - \text{houtE})$.

The other residual equations that are used to simulate the CT-SLHX when the effectiveness-based sub-model is used are shown below:

$$R(\text{cap}+2) = \text{CaptubeModel} - 1.0 \quad (\text{Eq. 4.8})$$

$$R(\text{cap} + 3) = ((0.75 * \text{Lin} / (\text{Lin} + \text{Lhx} + \text{Lout})) * (\text{pinexp} - \text{pcrit}) / \text{numDPin} - \text{Dpin}) \quad (\text{Eq. 4.9})$$

$$R(\text{cap}+4) = (\text{tincomp} - \text{toutE}) / \text{numDTsl} - \text{DTsl} \quad (\text{Eq. 4.11})$$

$$R(\text{cap}+5) = ((2.5d0 * \text{Lout} / (\text{Lin} + \text{Lhx} + \text{Lout})) * (\text{pinexp} - \text{pcrit}) / \text{numDPout} - \text{Dpout}) \quad (\text{Eq. 4.12})$$

$$R(\text{cap}+6) = \text{pinE} - 10.0d0 - \text{pcrit} \quad (\text{Eq. 4.13})$$

4.2.4 Evaporator

Similarly, the evaporator is modeled locally as a cross-flow heat exchanger, using a finite-element method. There are two serial evaporators in this component, so the sequential subroutine is called twice from the main program with different values to respectively simulate the fresh food evaporator and freezer evaporator. Figure 4.4 shows the interface of the subroutine and system residual equations as well as variables categories involving in the evaporator component.

The NR solver will serially call the sequential subroutine, with different geometry and different inlet states. Because these two evaporators are serial, the mass flow rates through the evaporators are equal and the outlet calculated states from the fresh food evaporator are the input states for the freezer evaporator. Additional connection equations describing the connection force the refrigerant states to equate at the connection point.

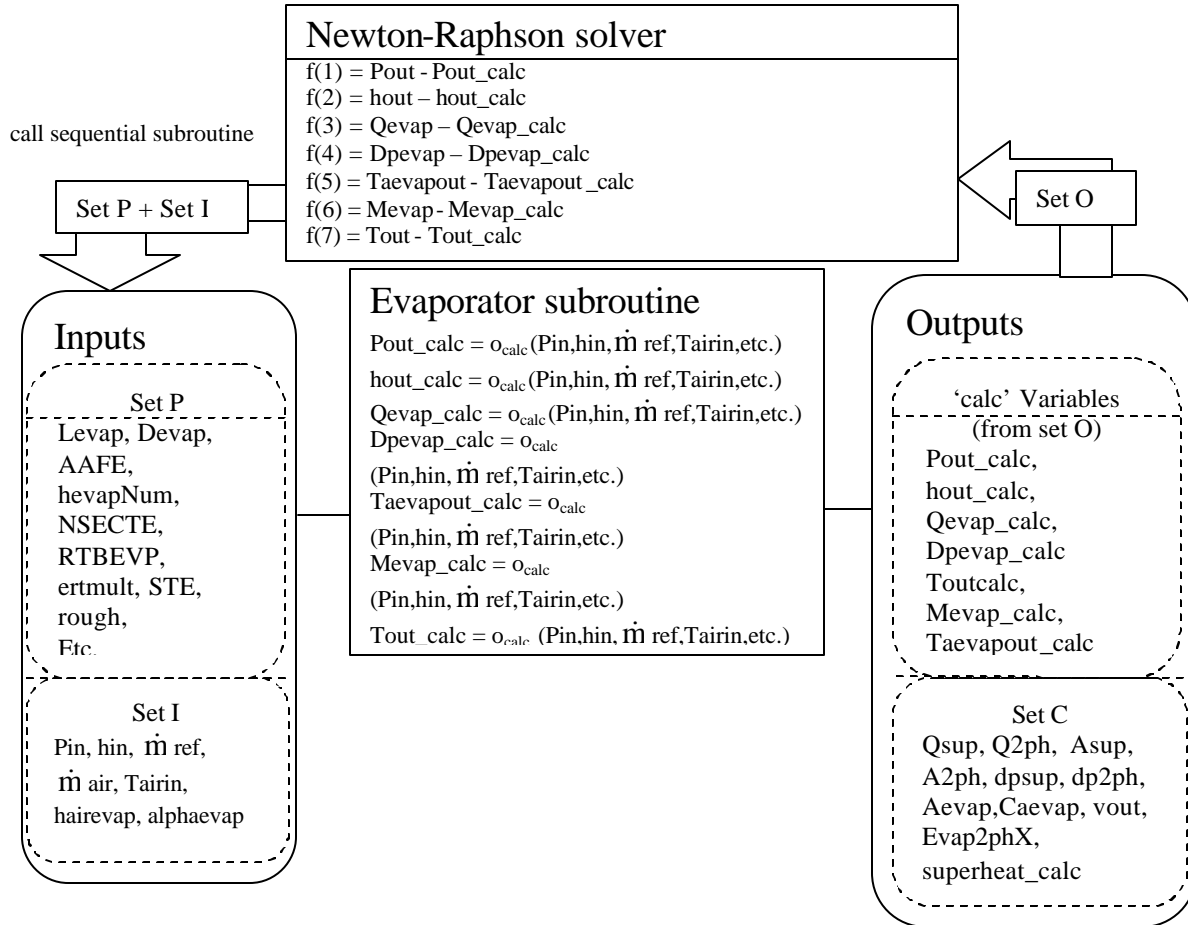


Figure 4.4 interface of sequential subroutine, residual equations and variables

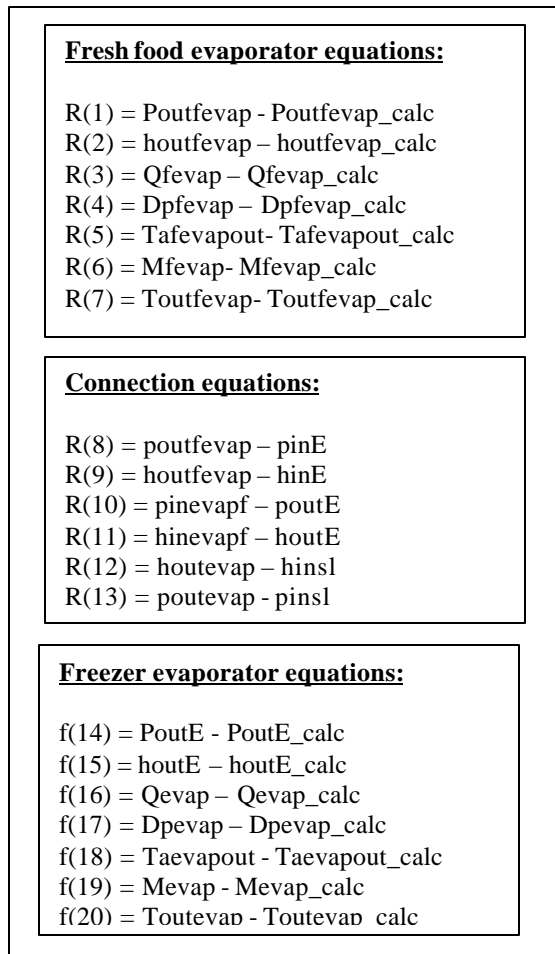


Figure 4.5 residual equations associated with evaporator component

References

- Andrade, M.A. and C.W. Bullard, "Controlling Indoor Humidity Using Variable Speed Compressors and Blowers", *University of Illinois at Urbana-Champaign*, ACRC TR 151, 1999.
- American Society of Heating, Refrigeration and Air-conditioning Engineers, "Handbook of Fundamentals", *ASHRAE*, 1993.
- Bridges, B.D. and Bullard, C.W., *Unpublished Manuscript*, University of Illinois Air Conditioning & Refrigeration Center, 1994.
- Harshbarger, D.S. and Bullard, C.W., "Finite Element Heat Exchanger Simulation within a Newton-Raphson Framework", August 2000
- Incropera, F. P. and DeWitt, D. P., "Fundamentals of Heat and Mass Transfer", Fourth edition, JOHN WILEY & SONS Corp. 1991
- Kirkwood, A.C. and Bullard, C.W., "Modeling, Design, and Testing of a Microchannel Split-System Air Conditioner", *University of Illinois at Urbana-Champaign*, ACRC TR-149, 1999.
- Kirby, E.S., Bullard C.W. and Dunn, W.E., "Effect of Airflow Nonuniformity on Evaporator Performance", *ASHRAE Transactions*, vol. 104, no. 2, pp. 755-762, 1998.
- Klein, S.A., Duffie, J.A. and Beckman, W.A., " TRNSYS - A Transient Simulation Program", *ASHRAE Transactions*, vol. 82, pp. 623-631, 1976.
- Mullen, C.E. et al., "Development and Validation of a Room Air-Conditioning Simulation Model", *ASHRAE Transactions*, vol. 104, no. 2, pp. 389-397, 1998.
- Stoecker, W.F., "Design of Thermal Systems", third edition, McGraw-Hill, Inc. 1989
- Woodall, R.J. and Bullard, C.W., "Development, Validation, and Application of a Refrigerator Simulation Model", ACRC TR-99, June 1996

Appendix A: Capillary Tube Suction-Line Heat Exchanger Design Model

A.1 Design operating condition

In this section, several capillary tube-suction line heat exchanger (ct-slhx) geometries are simulated to determine mass flow rate and several other performance indicators at the “design” operating condition. Refrigerator is tested at a 90° F, with $T_e = 12.4^\circ\text{F}$ and $T_c = 105^\circ\text{F}$, corresponding to high and low side pressures of 149.7 psia and 15.7 psia, respectively. To maximize performance of evaporator and condenser, superheat degree and subcooling degree are usually set at 5° F and 3° F. The inside diameter of the suction line is 0.375 inches. At these conditions the design calls for a mass flow rate $w = 12.4$ lbm/hr. The following summary Table (Table A.1) shows only L_{ct} , L_{in} , L_{hx} , w , T_{suc} and X_{crit} . However, T_{suc} is the indicator of effectiveness of the ct-slhx.

In order to investigate how subcooling affects the mass flow rate and other performance indicators, the CTSLHX model is run at 1°, 3° and 6° F subcooling, respectively.

Simulations were performed for several different total lengths of capillary tube, where possible, the inlet length of capillary tube is changed while keeping the heat exchanger and outlet lengths unchanged at 68 inches and 6 inches, respectively. For each diameter (0.038, 0.036, 0.034 and 0.032 inches, respectively), from the property profile printed on the computer screen, it was found that re-condensation occurred in the following cases: when length decreases to 95 inches and 85 inches with $\Delta T_{sub} = 6^\circ\text{F}$, there is re-condensation in the heat exchanger part, and flashing occurs again in the outlet part; It also occurs in the case where $D_{ct} = 0.034$ inches, and $L_{ct} = 90$ inches with 6° F subcooling; For $D_{ct} = 0.032$ inches with only 1° F subcooling, re-condensation also can be found. However, if length decrease to 70 inches with 6° F subcooling degree, the whole capillary tube is almost filled with single-phase liquid refrigerant, and there is only one flash point which is located in the adiabatic outlet section.

The following Table summarizes the results for all geometry simulated:

Table A.1 Calculated results for several different cases

Geometry				$\Delta T_{sub} = 1^\circ\text{F}$			$\Delta T_{sub} = 3^\circ\text{F}$			$\Delta T_{sub} = 6^\circ\text{F}$		
L_{ct} [inch]	L_{in} [inch]	D_{ct} [inch]	L_{hx} [inch]	w Lbm/hr	P_{crit} [psia]	T_{suc} [F]	w Lbm/hr	P_{crit} [psia]	T_{suc} [F]	w Lbm/hr	P_{crit} [psia]	T_{suc} [F]
130	56	0.038	68	15.6	29	69	16.5	30	69	17.7	31	69
120	46	0.038	68	16.6	30	70	17.6	32	70	19.0	33	70
110	36	0.038	68	17.9	32	72	19.0	34	71	20.7	36	70
105	31	0.036	68	16.2	32	73	17.2	34	73	18.8	36	72
95	21	0.036	68	18.0	36	74	19.2	37	74	21.1	40	73
85	11	0.036	68	20.6	40	76	22.2	43	75	24.8	47	74
110	36	0.034	68	13.4	30	73	14.2	31	73	15.4	34	72
100	26	0.034	68	14.6	33	75	15.6	34	74	17.1	37	73
90	16	0.034	68	16.5	36	76	17.7	38	75	19.5	41	74
80	6	0.032	68	15.2	40	79	16.6	43	78	17.6	44	77
70	10	0.032	54	15.1	40	71	16.3	43	71	18.1	47	69

From the above summary table, it is obvious that all the calculated mass flow rates are greater than the design target of 12.4 lbm/hr. If the mass flow rate should meet the design value, the length of capillary tube should be extended. The ct-slhx model was then used to calculate the adiabatic inlet length required to meet the 12.4 lbm/hr mass flow rate for all capillary diameters situation at 1F, 3F, 6F subcooling.

Table A.2 Calculated length for different diameters to meet design mass flow

DTsubcooling=1° F			DTsubcooling=3° F			DTsubcooling=6° F		
Tsuc [F]	Dct [inch]	Lin [inch]	Tsuc [F]	Dct [inch]	Lin [inch]	Tsuc [F]	Dct [inch]	Lin [inch]
65	0.038	104	64	0.038	117	63	0.038	137
69	0.036	71.2	68	0.036	82.0	66	0.036	97.2
73	0.034	45.7	71	0.034	54	70	0.034	66.0
78	0.032	18.8	76	0.032	24.0	74	0.032	32.4

The results suggest that a diameter equal to 0.032 inches would have several advantages. At Lct=92.8 inches, the length is the shortest, and it also transfers the most heat from the capillary tube to the suction line, as evidenced by the suction inlet temperature. That should maximize the EER increase contributed by the ct-slhx.

A.1.1 Effect of low condensing pressure

It may be possible that the captube exit becomes subsonic at low ambient temperature and ΔT_{sub} may be larger. The following Table shows us the results. The subcooling degrees shown in the table are the maximal values corresponding to different diameters and lengths, which can be used to calculate the critical pressure by ct-slhx model. From the result table, critical pressure is always greater than the evaporator pressure in these special situations, so in every case there is no subsonic.

Table A.3 Effectiveness of low condensing pressure

Dct=0.038 [inch]			Dct=0.036 [inch]			Dct=0.034 [inch]			Dct=0.032 [inch]		
Lin [inch]	Pcirt [psia]	Tsub [F]	Lin [inch]	Pcirt [psia]	Tsub [F]	Lin [inch]	Pcirt [psia]	Tsub [F]	Lin [inch]	Pcirt [psia]	Tsub [F]
56	34.1	10	31	40.1	10	36	36.5	10	6	40.1	1
56	36.5	15	31	41.6	12.5	36	39.6	15	6	42.3	2.5
56	38.8	20	31	43.0	15.5	36	40.5	17.5	6	44.9	5.0
46	36.4	10	21	40.3	5.0	26	38.3	7.5	10	45.9	5
46	39.2	15	21	44.4	10.0	26	40.3	10	10	47.8	8.5
46	41.3	19	21	45.5	12.0	26	43.0	13.5	10	47.9	9.2
36	39.3	10	11	43.2	2.5	16	40.8	5	---	---	---
36	42.7	15	11	46.9	6.0	16	43.4	7.5	---	---	---
36	42.9	16	11	47.8	7.5	16	44.8	9.0	---	---	---

For the Lct=70 inches case with Dct=0.032 inches, the length of capillary tube is not long enough to keep the length of heat exchanger at 68 inches, so the inlet length is set at 10 inches, and deduced the heat exchanger part to 54 inches.

A.2 Capillary Tube-Suction Line Heat Exchanger Model user's reference

The purpose of this section is to acquaint any potential users with the operation of the capillary tube-suction line heat exchanger model. The ct-slhx model calculates the mass flow rate through the capillary tube, the temperature rise in the suction line and the exit pressure and quality of the capillary tube. By using the cold liquid refrigerant in the suction line to decrease the enthalpy of the refrigerant in the capillary tube, this equipment can increase system capacity with a modest increase in the compressor power. At the same time, it can make sure that it is pure vapor refrigerant to enter into the compressor. Generally speaking, for the modeling purpose, the capillary tube is divided into three sections: adiabatic inlet section, heat exchanger section and the adiabatic outlet section.

A.2.1 Definition description for variables and parameters

In the capillary tube-suction line heat exchanger model, the capillary tube is divided into three different sections: adiabatic inlet section, followed by heat exchanger section, and the last one is adiabatic outlet section. The following figure (Woodall and Bullard, 1996) defines the main variables and parameters appearing in the XK initialization file. The flashing point is located in the inlet section, however, it can occur in either of the other two sections. At the same time, the refrigerant may re-condense in the heat exchanger section and then re-flash occur in the following adiabatic outlet region, which is not shown in the following figure.

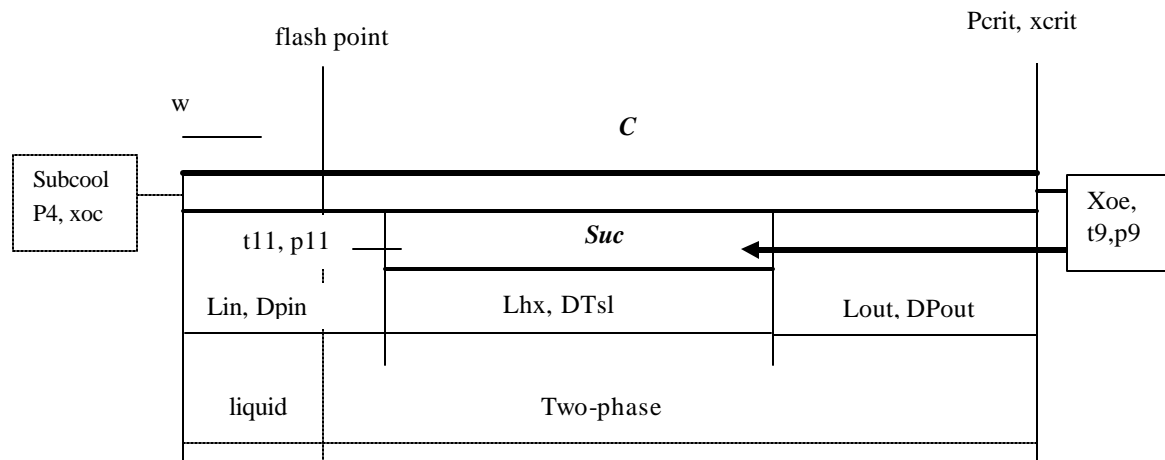


Figure A.1 Divisions of capillary-tube suction line heat exchanger

Definition description for the variables and parameters:

Dct	Diameter of the capillary tube
Dsuct	Diameter of suction line
Lin	Adiabatic inlet section length of capillary tube
Lhx	Heat exchanger length of capillary tube
Lout	Adiabatic outlet section length of capillary tube
w	Mass flow rate through the capillary tube
subcool	Degree of subcooling inlet the capillary tube
t9	Temperature at inlet of suction line
t11	Temperature at the exit of suction line

p4	Pressure at the exit of condenser
p9	Pressure at the inlet of suction line
p11	Pressure at the outlet of suction line
Pcrit	Critical pressure at the exit of capillary tube
Xoc	Quality at the exit of condenser
Xoe	Quality at the exit of capillary tube
Xcrit	Quality at the exit of capillary tube(choked flow)
Dpin	Pressure step in the inlet section of capillary tube
DTsl	Temperature step in the suction line heat exchanger section
Dpout	Pressure step in the outlet section of capillary tube
superheat	Degree of superheat at the exit of evaporator

The procedure that refrigerant flows through capillary tube-suction line heat exchanger is very complicated, so this component is a very difficult one to simulate. Depending on different locations in capillary tube and different design conditions, there will be several different processes occurring in the capillary tube. Generally, the total capillary tube is divided into three consecutive three sections, just like the above plot.

Now, the first adiabatic inlet section is considered. The entering refrigerant usually has two states: pure subcooled liquid or two-phase mixture. If pure subcooled liquid, the pressure will decrease without changing temperature before it reaches the saturation pressure. However, its pressure decreases when it flows through the inlet section. At the saturation pressure, liquid refrigerant begins to vaporize. We mark this location as flash point, and the remainder length of adiabatic inlet could be modeled as two-phase mixture. The subcooling degree is large, and the inlet section length is not long enough for refrigerant to vaporize and flash. The total inlet section is single-liquid flow. If the entering refrigerant is two-phase mixture, the total capillary tube can be model as two-phase condition. The pressure decreases with the increase of the quality when it flows through the inlet section.

Leaving the inlet section, the refrigerant enters into heat exchanger section. However, the entering state is subcooled liquid or two-phase mixture, which is decided by the operating condition and the geometry of inlet section. During flowing through this heat exchanger section, the heat transfer loss will cause decrease of the quality and the temperature. At the same time, the flow friction causes the drop of the pressure. Unfortunately, the pressure drop and the heat loss cause opposite changing on the refrigerant. The pressure drop of the refrigerant tends to increase the quality, however, the heat loss will increase the subcooling tendency or decrease refrigerant's quality. Under such complicated circumstances, the state transition of refrigerant in the heat exchanger will be unpredictable and complicated. Under the effect of the two opposite mechanisms, there will be five different possible scenarios in this part (R.J.Woodall and C.W.Bullard,1996):

- (1). The refrigerant enters as two-phase mixture and it stays two-phase;
- (2). The refrigerant enters as two-phase mixture and it recondenses and exits as subcooled liquid;
- (3). The refrigerant enters as subcooled liquid and it stays subcooled;
- (4). The refrigerant enters as subcooled liquid and it flashes and exits as a two-phase mixture;
- (5). The refrigerant enters as subcooled liquid and it flashes and then recondenses downstream and exits as a subcooled liquid;

The last scenario only can occur in a special situation, where the effect of pressure drop of the refrigerant is dominant in the upper stream, however the effect of heat transfer loss is dominant in the downstream of the heat exchanger section.

The last part is adiabatic outlet section. Just like adiabatic inlet section, there is no heat transfer, only pressure drop. This will also have the same two possible entering states: pure subcooled liquid or two-phase mixture. The situation should be the same as inlet section, except that the exiting state of outlet section should always be two-phase mixture at the choked flow condition. So if the entering state is pure liquid, the refrigerant subcooled degree should decrease until the saturation point, then vaporize and increase quality as two-phase mixture. For two-phase entering condition, the refrigerant pressure drops with the increase of the quality under the effect of the flow friction. When the quality increases, the specific volume of the mixture will increase, too. Since the mass flow rate is constant, the increase of the specific volume will cause the increase of the refrigerant flow speed. The velocity of the refrigerant will increase until critical flow is reached at the exit. At a prescribed condenser pressure, further reductions of the evaporator pressure below this point will not increase the mass flow rate of the refrigerant. Therefore, the condition of choked flow at the exit of the outlet section is assumed. If refrigerant is choked at the exit, there will be a discontinuity between the critical pressure at the capillary tube exit and the pressure at the inlet to the evaporator (R.J.Woodall and C.W.Bullard, 1996).

A.2.2 XK file, variables and parameters definition

The XK file is the primary bridge for communication between the user and the ct-slhx model. In the XK file, variables and parameters can be changed for every case, and the corresponding output calculated by the model is written to the output file. Now, the output name in the ct-slhx model is “s.derek”, whose name can be changed in the instruction file. In the XK file, variables are marked with “X” flags and parameters are marked with “K” flags. At the same time, X’s and K’s can be switched without recompiling. The primary storage location for variables and parameters is the XK array, where every element is made “equivalent” to Fortran variables and parameters, which appear in the governing equations in the “equation.f” file. For example, in the XK file of ct-slhx model, XK (1) and Dpin variable are “equivalent” and they can be used interchangeably. In other words, variables and parameters are usually referred to by XK# in the solver, while the governing equations and subroutines refer to them by their names. The XK array and “equivalent” Fortran variables and parameters are declared and put in the common block in “EQUIVLNT.INC” file, which is usually included in different subroutines and files.

The following is the XK interface, which can be used to change parameter’s values or switch X’s, and K’s. If variable or parameter is needed to appear in the written file, the “output flag” value should be set to 1, otherwise, it should be 0. Parameters are flagged with “K”, which need user-specified values. However, variables are flagged with “X”, which need ideal initial guess value for model’s calculation, and are updated by the model latest results appearing in the written file.

Table.A.4 Sample XK initialization file

```

** XK initialization file: initializes variable guesses and parameter values.
** Output Flag specifies if variable is printed to spreadsheet readable file.
** (1 = Print, 0 = Don't Print )

```

** Parameters are flagged with "K" and variables are flagged with "X."
 ** The units are delimited with '[]'.
 ** The last number signifies the number of decimal places (0-10).
 ** The ORDER of the input lines CANNOT CHANGE without program modification.

Output Flag	Name	XK#	Value	Units	# of digit
*****	DO NOT DELETE THESE FIRST NINE LINES!				*****
0 X	DPin	= XK(1) =	4.840	[psia]	3
0 X	DPout	= XK(2) =	4.056	[psia]	3
0 X	DTsl	= XK(3) =	5.453	[F]	3
1 X	pcrit	= XK(4) =	42.35	[psia]	3
1 X	w	= XK(5) =	10.856	[lb m/hr]	3
1 X	xcrit	= XK(6) =	0.1025	[]	4
0 K	CaptubeOutput	= XK(7) =	2.0	[]	1
1 K	Dct	= XK(8) =	0.002750	[ft]	6
0 K	Dsuctline	= XK(9) =	0.02604	[ft]	5
1 K	Lin	= XK(10) =	4.250	[ft]	3
1 K	Lhx	= XK(11) =	5.167	[ft]	3
1 K	Lout	= XK(12) =	0.667	[ft]	3
0 K	numDPin	= XK(13) =	4.	[]	0
0 K	numDPout	= XK(14) =	5.	[]	0
0 K	numDTsl	= XK(15) =	6.	[]	0
1 K	p4	= XK(16) =	12 0.400	[psia]	3
1 K	p9	= XK(17) =	42.320	[psia]	3
1 K	subcool	= XK(18) =	7.625	[F]	3
1 K	t9	= XK(19) =	37.050	[F]	3
1 C	superheat	= XK(20) =	10.000	[F]	3
0 K	xoc	= XK(21) =	0.000	[]	3
0 K	xoe	= XK(22) =	1.000	[]	3
1 C	t11	= XK(23) =	69.766	[F]	3
0 K	Cap_clog	= XK(24) =	1.	[]	0
0 K	absR	= XK(25) =	0.0000009708	[ft]	10

The next part is the description for the parameter-variable switch. Just like interface above, the w(mass flow rate) is the variable, and Dct is the parameter. If w is wanted to be a specified parameter value, and Dct is required to be calculated by the ct-slhx model with an initial guess, only the two flags are required to be changed in the above XK initialization file. The corresponding two lines

1 X	w	= XK(5) =	10.856	[lbm/hr]	3
1 K	Dct	= XK(8) =	0.002750	[ft]	6

would have to be changed to

1 K	w	= XK(5) =	10.856	[lbm/hr]	3
1 X	Dct	= XK(8) =	0.002750	[ft]	6

After forgoing operation, the model will run according to the above description. When swapping variables and parameters, caution must be exercised to ensure that the equations are not made singular or non-independent because of the swapping operation. Since the ct-slhx mode may become more sensitive to initial guess values because of the swapping, it is recommended that a solution for original equations be found before swapping, and this solution can be set as the initial guess value for the new XK file with parameters and variables swapping. As far as we know, some equations are very sensitive to a particular parameter, a larger change in the particular parameter

may make it difficult to get solution. It maybe is a good idea to achieve the designed change using a series of intermediate steps, with each intermediate solution used as updated initial guess value for the XK file.

Parameters marked with “C” are constant values specified by ct-slhx model. The flags can not be changed without program modification, and in the output file, these parameters can get the results calculated by ct-slhx model.

A.2.3 Sample instruction file

For different cases, different parameter values may be needed, then the instruction file is needed. The instruction file contains a list of parameters, which are needed to have their values modified, as well as the list of values for these parameters. On the other hand, the number of intermediate steps can also be specified to take between solutions. If the step number is one or more, then instruction file can direct the Newton-Raphson solver to linearly interpolate the intermediate solution between the previous specified parameter and the next specified parameters. This method can ensure a smooth transition from previous parameter to the designed value, when the model is very sensitive to this particular parameter’s initial guess value. According to the user’s choice in the XK file, the final results will be written into the specified file, however, the intermediate solutions will not be saved.

The following is the example for the instruction file:

Table A.5 Instruction file Sample

Items	Description
MULTIPLE	Solve for multiple sets of parameters
XK	Name of XK initialization file
XK.richard	Name of output file
1,4,8	#of runs, #of parameters to modify, #of intermediate steps
8 9 10 11	List of parameters to be modified in XK interface file
0.28 0.03 2.1 5.7	List of specified values for parameters’ modification

A.2.4 Solver setting file

The file “SLVERSET” contains settings for various solver parameters, convergence criteria and tolerances, and output options. The following is the sample SLVERSET file. Some of the output settings, such as printing initial, intermediate and final XK and R values, and other settings, are primarily useful for debugging a model, and normally the solver setting need not be changed. However, according to the design value for different case, these setting can be changed to meet the design need.

```
*****
***** NEWTON-RAPHSON SETTINGS *****
Instruction file name           : INSTR
Step factor for partial derivatives : .0001
Maximum allowable NR iterations : 15
Convergence criteria 1(Maximum residual) : 1.0e-3
Convergence criteria 2 (RMS residual) : 1.0e-4
Selected convergence criteria (1 or 2) : 2
NR step relaxation parameter    : 1.0
Use sparse matrix techniques?   : .TRUE.
Update guesses between runs?    : .TRUE.
```

***** GENERAL OUTPUT SETTINGS *****

Send general output to screen?	: .TRUE.
Send general output to a file?	: .FALSE.
Print abbreviated solver settings?	: .TRUE.
Print initial XK values?	: .FALSE.
Print initial residual values?	: .FALSE.
Print iteration summaries?	: .TRUE.
Print intermediate XK values?	: .FALSE.
Print intermediate residual values?	: .FALSE.
Print final XK values?	: .FALSE.
Print final residual values?	: .FALSE.
Print a final summary	: .TRUE.

***** SOLUTION OUTPUT SETTINGS *****

Save XK values in input file format?	: .FALSE.
Save XK values in spreadsheet format?	: .TRUE.
Output digits 0-10 (-1 = as in XK file)	: -1

Using the solver setting file, the operating conditions and desired formation can be set. According to different need, special flags can realize different solution output appearances. At the same time, special values can be input to change model's initial information. For example, if file format like XK formation is needed, then logic value in "SOLUTION OUTPUT SETTINGS" section should be switched as "TRUE", otherwise, it should be "FALSE".

Appendix B: Dual Evaporator Simultaneous System Modeling

B.1 Introduction

Dual evaporator refrigerators are modeled most accurately using the modular system simulation approach described in Chapter 4. This Appendix describes a crude intermediate method for using a single-evaporator model to simulate a dual evaporator system. This model is now obsolete, having been superceded by the version described in Chapter 4. However, it is documented here for archival purpose. Modifications were made to the previously developed single-evaporator code in order to simulate better the refrigerators in this study.

The ACRC refrigerator simulation model was initially developed for the study of single evaporator refrigerator-freezers at the Air Conditioning and Refrigeration Center (ACRC) at the University of Illinois at Urbana-Champaign. It consists of a general Newton-Raphson solver linked to a series of equations and functions that describe the particular refrigeration system being modeled (Mullen (1994) and Woodall (1996)). The simulation model for refrigerators is called RFSIM. The model assumes a steady state operation. The single evaporator model is described in more detail in Woodall and Bullard (1996). RFSIM was modified (Stein, 1999) and refined by Gerlach (2000) for dual evaporator refrigerators by adding a second evaporator in the fresh food section and eliminating the equation describing air exchange between the compartments. The fresh food evaporator was modeled as a two-phase region and the freezer evaporator includes both a two-phase region and a single-phase superheated region.

Additional modifications were needed to accurately represent the prototypes tested. Based upon the request of the sponsor, we tried to simulate the dual evaporator system separately by running the single-evaporator model alternately, first simulating one evaporator, supposing the other one idle. A few variables and residual equations were modified to simulate the dual evaporator system and its separate refrigerated compartment. This approximate approach is not recommended, but is described in Sec.B.2 for those who prefers to use the simultaneous instead of the modular version of the RFSIM.

Chapter 4 describes the simulation model with new modular structure for simulating dual-evaporator refrigerators. Every system component has an associated sequential subroutine describing the component. The number of initial guess values substantially decreases, also structured and independent sequential subroutines were easily embedded in the simulation system without recompiling and changing codes.

The nomenclature used in Stein (1999) and continued here is that the freezer compartment variables are written simply such as “tevap.” The fresh food variables have an “f” added, e.g. tevapf. Alternatively, a freezer variable is denoted with a “z” added, e.g. tevapz. In the new model structure, the variables name are kept unchanged, but all the variables are categorized into four categories as discussed in Harshbarger and Bullard (2000).

B.2 Simultaneous system

This simultaneous system is built upon the original RFSIM, using one-evaporator system to simulate dual-evaporator system. Suppose we run the dual evaporators separately, that means one time we run fresh food compartment, and the other time we run the freezer compartment. In the single evaporator RFSIM model, we ignore

the heat transfer between these two compartments through the mullion, but in dual evaporator model, it is an issue we need to pay attention to. The UA of the mullion is a new variable added into the XK variable list.

We simulate fresh food and freezer compartments concurrently in single evaporator system. Variables 'fz' and 'runtime' are describing the running conditions. In order to simulate dual-evaporator system using the single-evaporator approach, we need to focus on one compartment at one time. Variable 'fz' is a specified K value, which is switched between 0 and 1 for fresh food case and freezer case, respectively. For fresh food case, fz is specified to zero, meaning that no air flows into freezer compartment; On the other hand, fz is equal to one in the freezer case, where all air flows into freezer compartment instead.

For freezer simulation, the Figure 4.1 shows us the specified inputs for freezer compartment case, which is a little different from the original RFSIM evaporator system.

Governing equations for the freezer case are shown below. They have been modified from the original RFSIM system model since new heat transfer boundary is considered in the modified simulation model.

The following residual equation simulates the total heat transfer balance of the freezer compartment. The mullion heat transfer between fresh food and freezer compartments has been added. The heat transfer sources include heat transfer between ambient and freezer compartment, heat transfer from heater, heat transfer between fresh food and freezer compartments and heat transfer of liquid line (post-condenser loop for heating the door gasket areas), all of which is assumed to enter the freezer compartment.

$$R(\text{cab}+0) = \text{BTU}(\text{UA}_z) * (\text{tamb} - \text{tafrez}) + \text{BTU}(\text{FrezHeater}) + \text{qliqline} + \text{BTU}(\text{UAm}) * (\text{tafrig} - \text{tafrez}) - \text{Qfrez} \quad (\text{Eq. B.1})$$

$$Q = \text{UA}_z * (\text{Tamb} - \text{Tafrez})$$

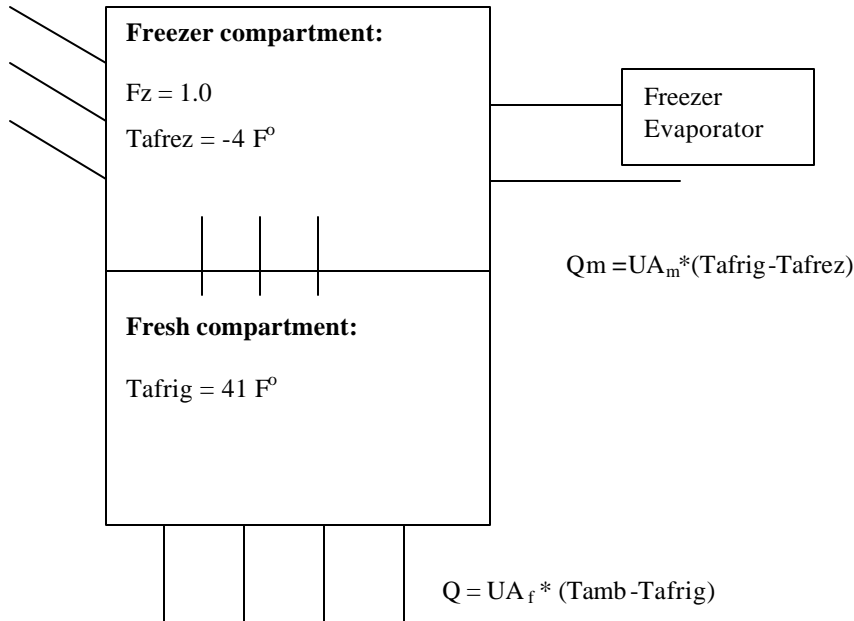


Figure B.1 Freezer component simulation: no air flows into fresh food compartment

The second residual equation describes the total heat transfer balance of the fresh food compartment without air flowing through since Fz is specified 1.0. This is the running condition we suppose. The heat transfer sources include heat transfer between two compartments, heat transfer from ambient and heater.

$$R(cab+1) = BTU(UAf)*(tamb - tafrig) + BTU(FrigHeater) - BTU(UAm)*(tafrig - tafrez) - Qfrig \quad (Eq. B.2)$$

The following residual equation is used to calculate the time that the system has to run to remove all the heat added to the freezer compartment in order to keep the constant temperature in the compartment.

$$R(cab+2) = (mdotaevap*fz*(ha(tafrez)-ha(taevapfanout)))*Runtime - Qfrez \quad (Eq. B.3)$$

where the fz is equal to 1.0. When simulating the freezer, the other residual equation that is modified to simulate the running condition is the one calculating the fresh food compartment heat transfer 'Qfrig'. Since we know there is no air flowing into this compartment, variable 'Qfrig' is forced to be equal to zero all the time in this case.

$$R(cab+3) = Qfrig - 0.0 \quad (Eq. B.4)$$

The residual equation that is not listed above but in the original system is still kept to help the simulation.

We should keep in mind that during the whole simulation, the fresh food compartment temperature should constantly keep 41 °F, so the heat transfer between ambient and fresh food compartment should be equal to the heat transfer from fresh food compartment to freezer part. In order to make the equation valid, we need to keep UAf as a 'X' variable. Similarly, when we simulate the fresh food compartment, we need exchange UA_z and UA_f: UA_f is K parameter, but UA_z is allowed to float to some artificial value. We can do the simulation in another way, keeping the variable fz float to some artificial value instead and specifying the UA_z or UA_f alternately. In our current model, we specify and switch fz between 1 and 0, keeping UA_z or UA_f float in order to balance the heat transfer in fresh and freezer compartments.

We use similar way to simulate the fresh food compartment, but this time fresh food evaporator is active and total air flows into fresh food compartment. Variable fz is specified to zero. During the simulation, freezer compartment temperature, -4 °F, is constantly kept. At the same time, just as discussed about, variable UA_z is allowed to float in order to satisfy the heat transfer between freezer compartment, fresh food compartment and ambient. The Figure B.2 shows us the relationship around the fresh food compartment.

Similarly, some residual equations as below are modified to simulate the running condition from the original RFSIM residual equations.

The first residual equation is to describe the heat transfer balance of the freezer compartment. Heat transfer between ambient and freezer compartment, heat transfer from heater and liquid line and heat transfer between these two compartments are considered in this equation.

$$R(cab+0) = BTU(UA_z)*(tamb - tafrez) + BTU(FrezHeater) + qliqline + BTU(UA_m)*(tafrig - tafrez) - Qfrez \quad (Eq B.5)$$

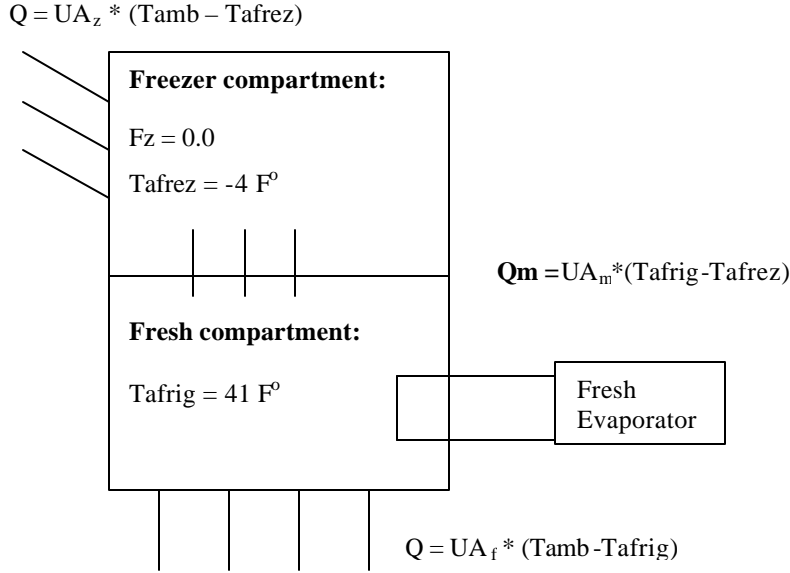


Figure B.2 Fresh food compartment simulation

The second residual equation modified is the equation simulating the heat transfer balance of the fresh food compartment.

$$R(\text{cab}+1) = \text{BTU}(\text{UAf}) * (\text{tamb} - \text{tafrig}) + \text{BTU}(\text{FrigHeater}) - \text{BTU}(\text{UAm}) * (\text{tafrig} - \text{tafrez}) - \text{Qfrig} \quad (\text{Eq. B.6})$$

The following is the residual equation used to calculate the fraction time the system uses to remove the heat added to this compartment in order to keep temperature balanced.

$$R(\text{cab}+2) = (\text{mdotaevap} * (1.0 - \text{fz}) * (\text{ha}(\text{tafrig}) - \text{ha}(\text{taevapfanout}))) * \text{RunTime} - \text{Qfrig} \quad (\text{Eq. B.7})$$

The last residual equation needing attention is to specify the heat transfer balance of the freezer compartment. The variable 'Qfrez' is forced to zero since there is no air flowing in and the evaporator is idle now.

$$R(\text{cab}+3) = \text{Qfrez} - 0.0 \quad (\text{Eq. B.8})$$

In conclusion, we can use single evaporator system model to simulate the dual evaporator refrigerator components separately, but the simulation method can not provide exact results for us. Previously, when this dual evaporator system is stable, the Figure B.3 shows the states of refrigerant at different connection points.

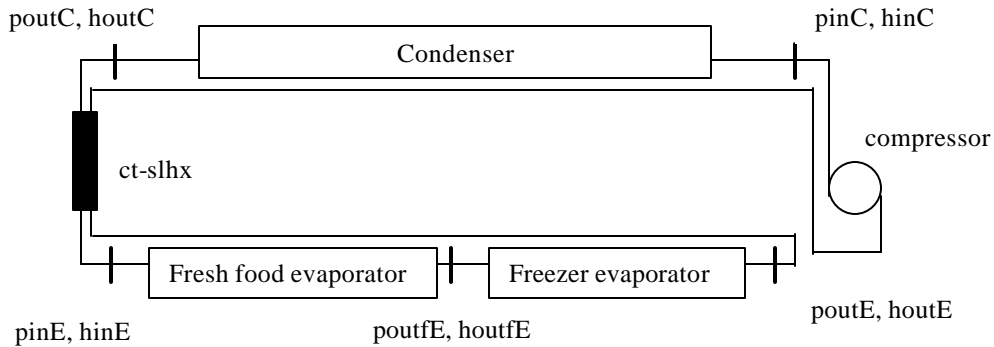


Figure B.3 stable refrigerant states of dual evaporator system

For fresh food evaporator, the inlet states of refrigerant are $hinE$ and $pinE$ from expansion device. The outlet states of refrigerant are $poutfE$ and $houtfE$. Since these two evaporators are serial, $poutfE$ and $houtfE$ are inlet states of the freezer evaporator with the outlet states of $poutE$ and $houtE$. The other main state points are marked above.

In our running cases, we simulate the dual evaporators separately, using the single evaporator system. When we simulate the fresh food evaporator case, the refrigerant states are described below in figure B.4.

The combination of compressor, condenser and expansion device is the same as the figure above, but the freezer evaporator is removed from the dual-serial-evaporator system and assumed idle during the simulation. Several steps are used to analyze the process. We supposed that the combination of compressor, condenser and expansion device provides the same inlets and outlets states of refrigerant as above, then fresh food evaporator has the inlet states, $pinE$ and $hinE$. Secondly, the evaporator provides the same outlet states as before, $houtfE$ and $poutfE$. Now $poutfE$ and $houtfE$ are inlet states of the compressor, but not $poutE$ and $houtE$ any more. With the different inputs, the combination changes its outputs to fresh food evaporator in the next round, different from $hinE$ and $pinE$. At this time, the whole system is under unstable state. When the whole system becomes stable again, the system has different refrigerant states at the connection points in the figure. The fresh food evaporator has input states, $hinE_1$ and $pinE_1$, and output states, $poutE_1$ and $houtE_1$. With the same specified variables, the system definitely has different performance now, not as expected in the original figure B.3.

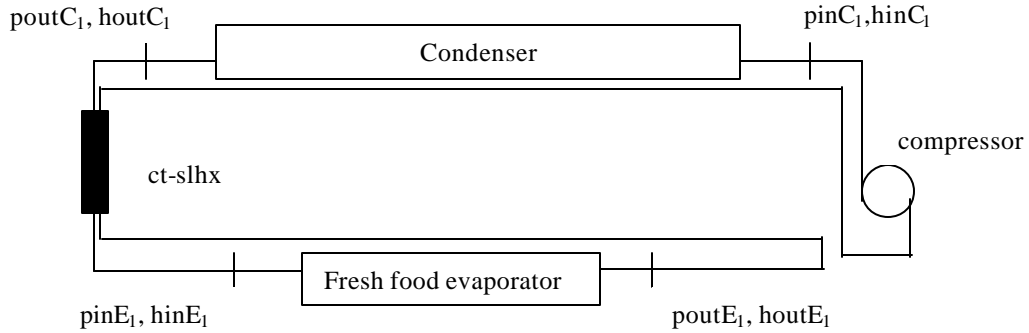


Figure B.4 Refrigerant states of single fresh food evaporator case

Similarly, in the freezer evaporator simulation, the evaporator gets the inlet states, $hinE_2$ and $pinE_2$, and outlet states, $houtE_2$ and $poutE_2$, which are different from expected values: $hinE$, $pinE$, $poutE$ and $houtE$, respectively.

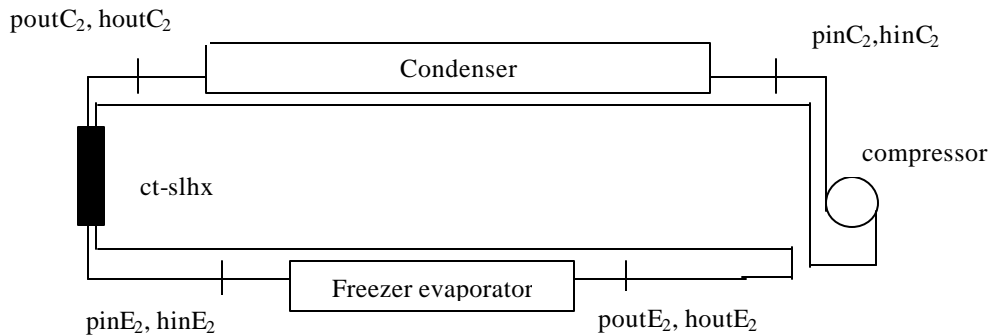


Figure B.5 Refrigerant states at connection points of freezer simulation

Appendix C: Residential A/C System Modeling

C.1 Introduction

The ACRC air conditioner system simulation model was developed by Mullen *et al.* (1998). Instead of solving the equations with a successive substitution algorithm, the ACRC solver utilizes a Newton-Raphson algorithm to solve the governing equations simultaneously. The solver allowed the input parameters and output variables to be interchanged without the need to reprogram the model.

Recently, the system simulation model was improved by Andrade and Bullard (1999). Equations were added that allowed the simulation of split type a/c units in addition to the window units. Improvements were made to the evaporator heat and mass transfer equations, implementing a study by Kirby, Bullard and Dunn (1998). Equations simulating sensible and latent loads of a house were also added, to simulate the system's ability to reduce indoor humidity for a given set of outdoor conditions and air infiltration rates.

The system simulation model has proved an accurate and sophisticated design tool. However, the program had two prominent limitations. Modern and future heat exchanger designs were exposing the limitations with the conventional modeling techniques. This became apparent when Kirkwood and Bullard (1999) modified the model to simulate microchannel heat exchanger geometries in a multizone framework, where a finite element approach would have been more appropriate. Additionally, for the ACRC solver to calculate a solution, accurate initial guesses must be known for each output variable. Many initial guesses were required, sometimes for obscure values, which caused great burden on the user. Furthermore, sponsoring companies were expressing interest in individual component models. With a large set of interrelated equations, component simulations were difficult to isolate and export from the system simulation program.

To address these limitations, Harshbarger and Bullard (2000) have developed a new structure to be implemented into the system simulation model. Finite element solutions of the heat exchangers were developed for the condenser and evaporator. The finite element structure allows the simulation of complex geometries that were not possible with conventional methods. The finite element solutions were integrated into the system model in a manner that reduced the number of required initial guesses and therefore, the burden on the user.

To further the capabilities of the model, a modular structure was adopted. Using a structure similar as TRNSYS (Klein *et al.*, 1976), each component in the system is solved in a self-contained manner. Therefore, each component simulation can easily be isolated and/or integrated into a simultaneous set of system-level equations.

C.2 Model description

The RACMOD system model consists of components models for the condenser, evaporator, compressor and capillary tube, as well as simulation equations for system, component connections and charge calculation. We built sequential subroutines for each component and the details describing the modeling strategies and algorithms used in the subroutines are provided.

C.2.1 Condenser

The air conditioner condenser is modeled as a crossflow heat exchanger with uniform inlet air temperature and velocity. A stand-alone sequential subroutine is used to simulate this component, where the finite element

algorithm is used. The first part of the condenser geometry has two refrigerant circuits and both are defined as identical and parallel modules. We simulate one module, and multiply by the number of parallel modules to calculate the total heat transfer performance and areas for different heat transfer zones.

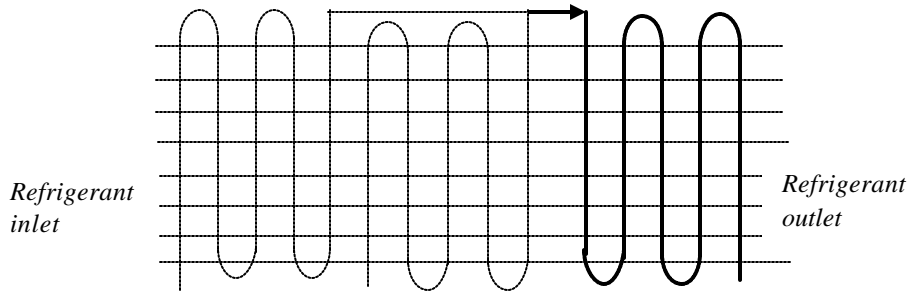


Figure C.1 Condenser geometry

Along the refrigerant flow direction, the circuit is equally divided into many small elements. Each small element has the same length, so they have the same air-side and refrigerant-side heat transfer areas during simulation. This is a cross flow configuration, where each element has the same air input temperature and we assume air mass flux is identical everywhere. As shown above in Figure C.1, air flows vertically into page from the outside. The difference is the input refrigerant state of each element since they are divided along the refrigerant flow.

The condenser includes two parts: the first part has two identical parallel circuits, which join together in the middle. The second part only has one circuit. In old 3-zone modeling framework, the circuit number is assumed 1.5 to calculate an approximate mass flux for all three zones. However, in the new simulation model, each small element uses the exact circuit number and mass flux for the calculation. Mass flux is calculated locally as well as heat transfer area, pressure drop, refrigerant and air properties and local heat transfer.

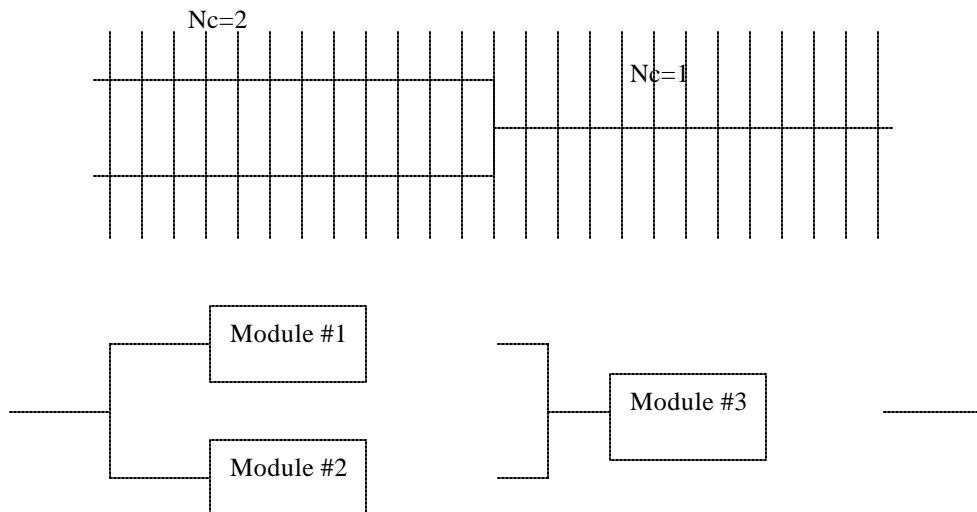


Figure C.2 Condenser module structure

In Figure C.2, we suppose we have two identical parallel modules in the first part; the second part only has one module. Finite element algorithm is used to simulate each module. Heat transfer correlations and pressure drop correlations are also calculated in each small element in order to exactly simulate the real heat transfer. Each element in three heat transfer areas – superheat, two-phase and subcooling – is modeled using effectiveness-NTU heat transfer rate equations.

At the end of the subroutine, we accumulate the heat transfer, mass charge, heat transfer areas, pressure drop of the refrigerant, and calculate the air temperature. The Figure C.3 shows the flow chart of the condenser subroutine. Because the difference of the circuit numbers in simultaneous and new models, we get different UA and areas for three zones as well as the mass charge in subcooling zone. However, the total heat transfer is almost the same because air-side heat transfer coefficient is always the dominant one.

C.2.1.1 State transition

There is another point we need to put attention to, which is the state transition point in the middle of each element. At the beginning of each element, we decide the current state of the refrigerant. However, there is possibility the refrigerant changes the state among superheat, two-phase and subcooling in the middle of simulation on the element. We have two methods to deal with these elements: we can divide the whole heat exchanger into small enough elements, where we suppose there is no transition. Because the element is small enough, the result is also acceptable. Another method is that we calculate the transition point inside the element, and then divide the element into two smaller elements. We calculate these two elements sequentially. In our current model, we use the latter method to decide the transition point, then call the subroutines with the exact lengths to exactly simulate the smaller elements although the difference between these two results is negligible.

C.2.1.2 Subroutine description

In the condenser main program, named condenser.f, there is a logical variable to decide which algorithm should be called to simulate it:

```

      If (downstream) then
        Call dwnstmCond(Pincond, hincond, houtC_calc, poutC_calc,
&          QtotC_calc, AtotC_calc, heightC_calc, AfrC_calc,
&          massC_calc, QsupC, Q2phC, QsubC, AsupC, A2phC, AsubC,
&          T2phin, T2phout, P2phin, P2phout, TphinC, TphoutC)

```

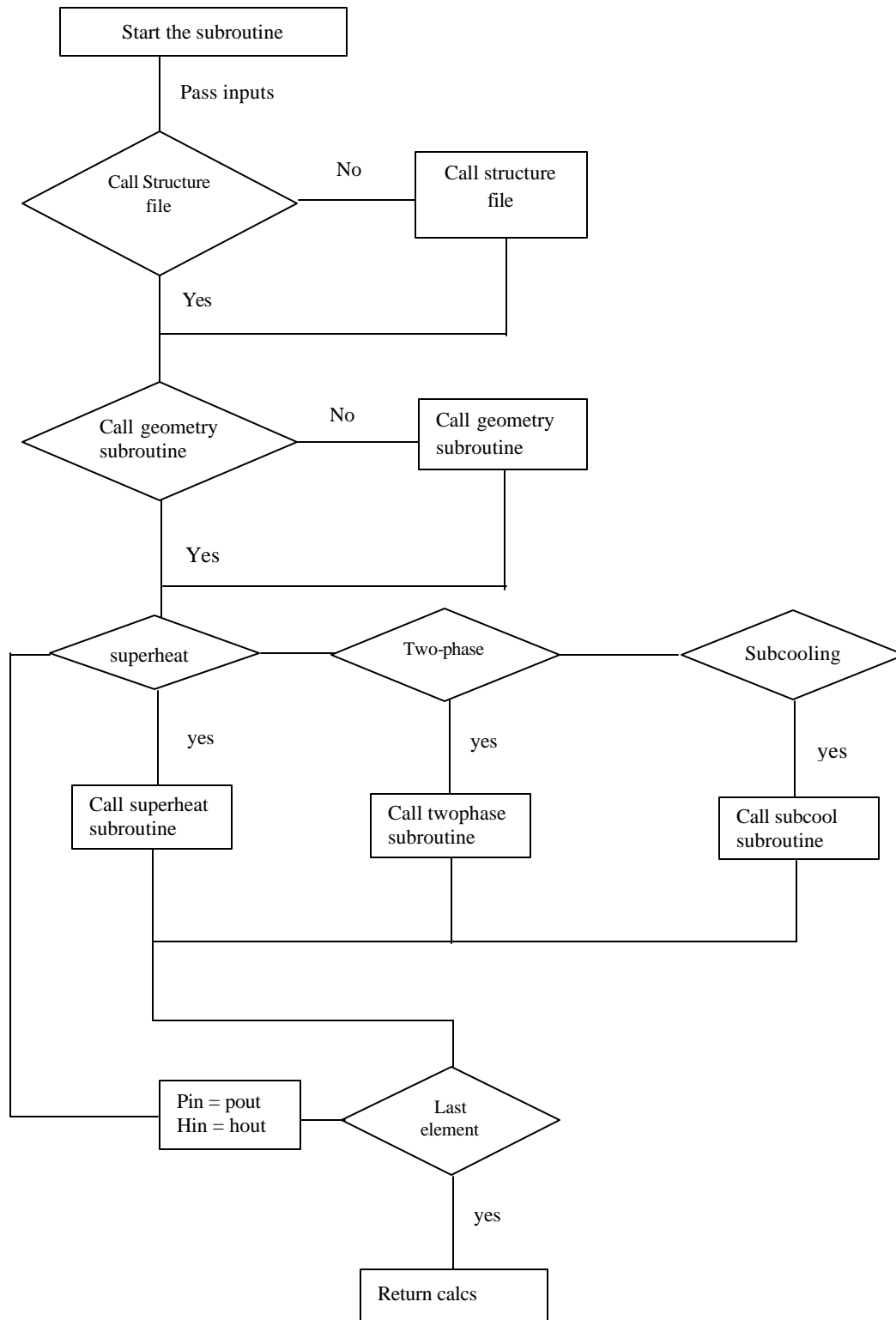


Figure C.3 Flow chart for condenser subroutine


```

Else
    Call upstmCond(Poutcond,houtcond,hinC_calc,pinC_calc,
&                QtotC_calc,AtotC_calc,heightC_calc,
&                AfrC_calc,massC_calc,QsupC,Q2phC,QsubC,AsupC,
&                A2phC,AsubC,T2phin,T2phout,P2phin,P2phout,
&                TphinC,TphoutC)
Endif

```

Consequently, two different groups of residual equations are used separately to simulate these two algorithms, which are listed in the main program:

```

If (downstream) then
10    R(cond+0) = MtotC - massC_calc
      goto 5000
20    R(cond+2) = Acond - AtotC_calc
      goto 5000
30    R(cond+1) = Qcond - QtotC_calc
      goto 5000
40    R(cond+3) = AfrC - AfrC_calc
      goto 5000
50    R(cond+4) = heightC - heightC_calc
      goto 5000
60    R(cond+5) = Poutcond - PoutC_calc
      goto 5000
70    R(cond+6) = houtcond - houtC_calc
      goto 5000
80    R(cond+7) = degsubcool - degsubcool_calc
      goto 5000
else
100   R(cond+0) = MtotC - massC_calc
      goto 5000
200   R(cond+1) = Acond - AtotC_calc
      goto 5000
300   R(cond+2) = Qcond - QtotC_calc
      goto 5000
400   R(cond+3) = AfrC - AfrC_calc
      goto 5000
500   R(cond+4) = heightC - heightC_calc
      goto 5000
600   R(cond+5) = hincond - hinC_calc
      goto 5000
700   R(cond+6) = Pincond - PinC_calc
      goto 5000
800   R(cond+7) = degsubcool - degsubcool_calc
      goto 5000
endif

```

The variables affixed with ‘calc’ are returned from the sequential subroutine called above. All the residual equations are sent to the ACRC solver, which simultaneously solves them and updates the variables in each iteration until the final solution is reached.

When the sequential subroutine is called, the main program transmits the input values to it. In each component, there is a file named ‘XK.update’, shared by the system. It gives a way for the component to access the

initial guess values for X variables, specified interchangeable variables and parameters. The flow process inside the sequential subroutine is described below. First of all, there is a logical to indicate whether the element configuration has already been read from the file, 'condenser.txt', which defines the total number of the elements, the number of parallel circuits in each element, and total number of tube passes, etc. In order to avoid recompiling, the logical variable is used to indicate whether subroutine already has initialized the arrays, which store the information listed above. If this information has already been read, this input step will be skipped.

```

    if (.not. elementread) then
        call Readelem(NelpassC,ElperpassC,ModstartC,airinputC,
&                    NparamodC,NcircuitC,'condinput.txt')
        elementread = .true.
    endif

```

The logical variable, elementread, is initialized as 'False', causing the information to be loaded before the calculation. After the first iteration, elementread is set to 'true'.

The next step is to calculate the condenser geometry. Each element has the same length. Refrigerant-side and air-side heat transfer areas for each circuit are calculated as well as other variables used in further calculation in this subroutine, which is named FTcondgeom.

```

    call FTcondgeom(DinC,DoutC,NtubesperslabC,VtubedistC,WidthC,
&                  height,LRtrnBndC,FinthC,FinPtchC,
&                  TuberowsC,HtubedistC,NelpassC,ElperpassC,
&                  thickC,DLC,Afr,VRtrnBndC,DcC,AffC,AairC,
&                  coilfactC,Volume,Area,AadivAffC,CSareaC)

```

The next part is the core of the finite element simulation. We use downstream algorithm to simulate this condenser, so we start from the refrigerant inlet and the air inlet. The first step is to decide whether the current numbered element is over the boundary of the maximal element number by the 'do-while' control loop. If so, we will skip the loop to return the calculated variables to condenser main program. Otherwise, calculation will be continued until the last element.

```

    do while (element .le. (NelpassC * ElperpassC))
        ...
        ...
    enddo

```

At the beginning of each element, we decide the current inlet status of the refrigerant: superheated, two phase or subcooled based on the element inlet, pressure and enthalpy:

```

    hsatv = hpx(pin,1.0)
    hsatl = hpx(pin,0.0)
    if (hin .ge. hsatv)then
        vapor = .true.
        twoph = .false.
        liquid = .false.
    else if (hin .ge. hsatl) then
        vapor = .false.

```

```

        twoph = .true.
        liquid = .false.
    else
        vapor = .false.
        twoph = .false.
        liquid = .true.
    endif

```

We have three subroutines to deal with vapor, liquid or two-phase, respectively, which are named supeldwnstmC, tpheldwnstmC, subeldwnstmC.

```

    if (vapor) then
        call supeldwnstmC(hin,Pin,wlocal,
&      Tairout(airinputC(element)),mdot,DLC,Area,Volume,
&      AaDivAffC,AffC,coilfactC,DcC,hout,Pout,
&      Tairout(element),Qsup,Q2ph,Asup,A2ph,Vapor,twoph,
&      liquid,mass,hsupC)
    else if (twoph)then
        call tpheldwnstmC(hin,Pin,wlocal,
&      Tairout(airinputC(element)),mdot,DLC,Area,Volume,
&      AaDivAffC,AffC,coilfactC,DcC,hout,Pout,
&      Tairout(element),Qsub,Q2ph,Asub,A2ph,Vapor,twoph,
&      liquid,mass,h2phC,U2phClocal)
    else if (liquid)then
        call subeldwnstmC(hin,Pin,wlocal,
&      Tairout(airinputC(element)),mdot,DLC,Area,volume,
&      AaDivAffC,AffC,coilfactC,DcC,hout,pout,
&      Tairout(element),Qsub,Asub,mass,hsupC)
    endif

```

The local refrigerant mass flow rate, wlocal, is calculated for each circuit in the module as well as the air mass flow rate by the following equations.

$$wlocal = (w/dbl(NparamodC(element)))/dbl(NcircuitC(element))$$

$$mdot = (MdotC/AfrC)*(DLC*VtubeDistC)$$

Where NparamodC is the array to store the numbers of the parallel modules of the current element and NcircuitC is the array to store the number of circuits in the current element, which are all initialized at the beginning of the sequential subroutine by calling the text file, 'condenser.txt'.

In each subroutine simulating the small element, the traditional ϵ -NTU method is used to calculate the heat transfer of the element. The governing equations used in the finite element are the same as the simultaneous models.

After simulating each element, we accumulate the designed variables from each element as shown below:

$$Qsuptot = Qsuptot + Qsup*dbl(NcircuitC(element))$$

$$Q2phtot = Q2phtot + Q2ph*dbl(NcircuitC(element))$$

$$Qsubtot = Qsubtot + Qsub*dbl(NcircuitC(element))$$

$$Asuptot = Asuptot + Asup*dbl(NcircuitC(element))$$

$$A2phtot = A2phtot + A2ph*dbl(NcircuitC(element))$$

$$Asubtot = Asubtot + Asub*dbl(NcircuitC(element))$$

$$masstot = masstot + mass*dbl(NcircuitC(element))$$

$$mtotalC = mtotalC + mdot*dbl(NcircuitC(element))$$

When the subroutine finishes the final element, it skips out of the 'do-while' loop. The variables above are the final values describing the performance of the component, which are returned to the main program (design variables) or the user interface (calculated variables).

Within an element there may be a transition between superheat and two-phase, or between two phase and subcooled. There are two methods to deal with this problem: the first is to ignore the transition since we usually divide the heat exchanger into small enough elements that the difference is negligible; the other method is to calculate the real transition location inside the element, and then divide the current element into two elements, with each of them calling the corresponding subroutine with the real lengths. We use the latter method to deal with problem currently.

The calculated variables are returned by the sequential subroutine to the main program, where they appear in the residual equations. The residual equations are sent to ACRC solver and are simultaneously solved.

C.2.2 Expansion device

The ACRC finite difference adiabatic capillary tube model developed by Peixoto and Bullard (1994) has been implemented in RACMOD. This model is integrated into the whole RACMOD system without rewriting. A captube option is selected by appropriately setting the parameter "CapTubeSelect". Setting CapTubeSelect =1 specifies the ACRC captube and CapTubeSelect=2 specifies the ASHRAE captube. If CapTubeSelect is a negative number, then the design model is chosen and user needs to specify the amount of the evaporator superheat.

C.2.3 Compressor

A manufacturer-supplied compressor map is used to predict the compressor mass flow rate and power consumption as a function of condensing and evaporating temperatures. Bridges and Bullard (1994) provided details about this component.

C.2.4 Evaporator

Like the condenser, the residential air conditioner evaporator model assumes a crossflow heat exchanger with uniform inlet air temperature and velocity. There is also a stand-alone subroutine to simulate this heat exchanger, where the finite element algorithm is used. Inside the sequential subroutine, modular concept is used to simplify the simulation. There are six parallel circuits, and each one is considered identical module. We only need to simulate one of them, dividing the mass flow by the number of the modules. Then the total heat transfer and mass charge result from timing the number by the performance of single module. The whole module is equally divided into hundreds of small elements, and heat transfer coefficients, pressure drop, heat transfer and mass charge are also calculated locally and accumulated together to simulate the total heat transfer performance. Each element is solved by a series of heat transfer equations that utilize an ϵ -NTU method sequentially. Two regions of the heat exchanger require unique governing equations. The two regions are the superheated and two-phase refrigerant zones. With the finite element approach, a few elements will likely experience a zone change within their volume. In the model, the element is either totally two-phase or superheated and the error introduced by this assumption is negligible if the element is small enough. The inlet enthalpy of each element is checked to determine if it is in two-phase or superheated zone.

Evaporator subroutine shares the same flow chart as the condenser subroutine. The big difference is that we need to deal with dehumidification since there is water condensed from the hot air. Depending upon the circumstances of the operating condition, an evaporator may operate with totally dry surface, totally wet surface or partially dry / partially wet surface. Due to circuiting, it may happen that the refrigerant rejects heat to the air if the upwind element is colder. So if the air inlet temperature to an element is lower than the refrigerant inlet temperature, the element is assumed to be totally dry because the refrigerant is rejecting heat instead of absorbing heat.

If the refrigerant inlet temperature to an element is higher than the air inlet dew point temperature, the element is assumed to be totally dry and we do not need to calculate the mean fin temperature at the leading edge. Otherwise, the mean fin temperature at the leading edge has to be calculated to determine the surface condition.

In partial or total wet element, Log mean enthalpy method is introduced to calculate the total heat transfer, including both sensible and latent heat transfers. The mean air enthalpy difference is given by

$$Q_t = U_{o,w} A_a LMhD \quad \text{Eq. C.1}$$

Where

$$LMhD = \frac{(h_{ai} - h_{s,ri}) - (h_{ao} - h_{s,ro})}{\ln \left(\frac{h_{ai} - h_{s,ri}}{h_{ao} - h_{s,ro}} \right)}$$

We may show that

$$U_{o,w} = \frac{1}{\frac{b_R' A_a}{A_{p,r} h_r} + \frac{b_{w,m} (1 - h_{F,w})}{h_{o,w} (A_{p,r} / A_F + h_{F,w})} + \frac{b_{w,m}}{h_{o,w}}} \quad \text{Eq. C.2}$$

Where $b_{w,m}$ is evaluated at the mean surface temperature of the water film on the fin. $h_{F,w}$ is the fin efficiency for wet surface.

We have to separate the sensible and latent capacity for enthalpy potential method after the total capacity is obtained. We use the traditional ϵ -NTU method to calculate the sensible heat transfer, and then deduct it from the total heat transfer to get the latent part.

C.2.4.1 Totally dry, partially wet or fully wet

Depending upon the circumstances of the operating condition, an evaporator may operate with totally dry surface, totally wet surface or partially dry / partially wet surface. Figure C.4 describes the process we use in the current model.

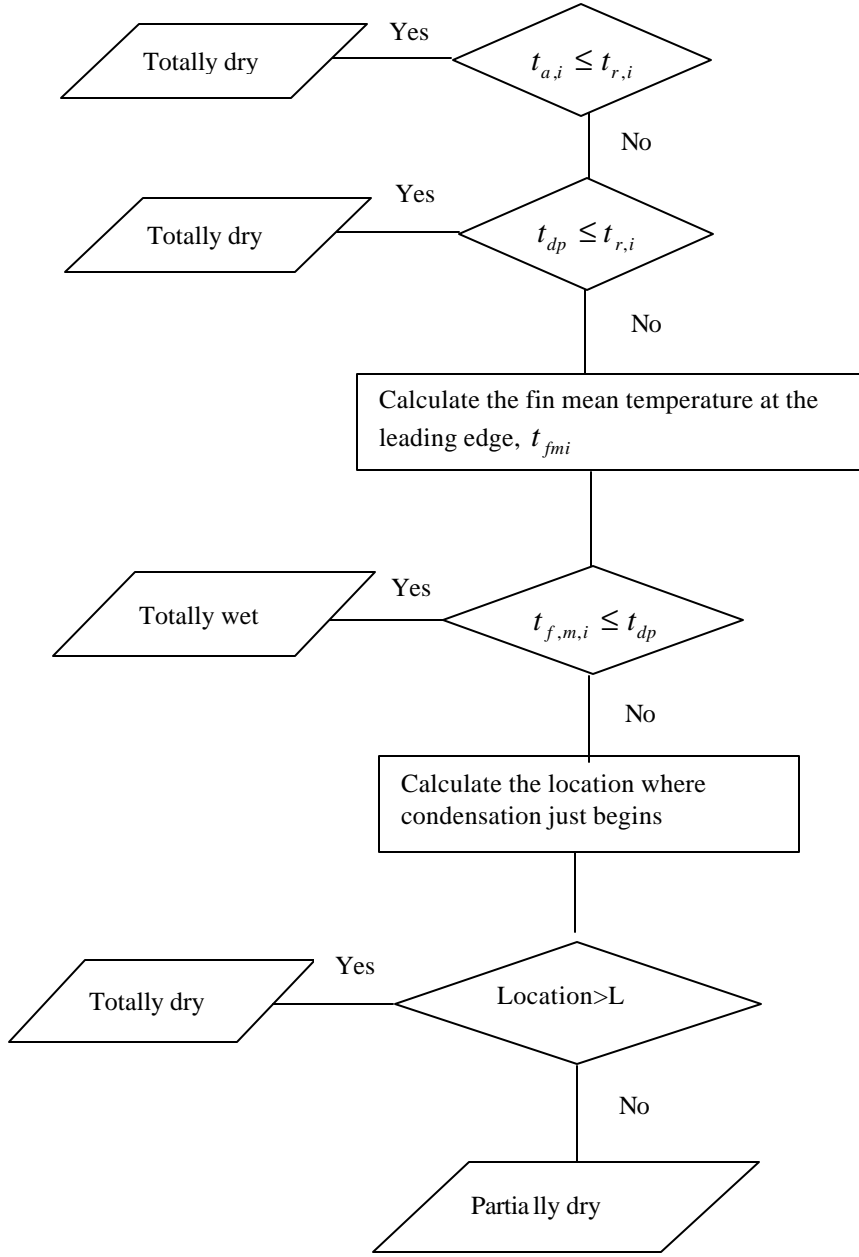


Figure C.4 Process of determining surface condition

If the refrigerant inlet temperature to an element is higher than the air inlet dew point temperature, the element is assumed to be totally dry and we do not need to calculate the mean fin temperature at the leading edge. Otherwise, the mean fin temperature at the leading edge has to be calculated to determine the surface condition.

Assuming that the surface is initially dry and the refrigerant temperature is constant on a small element, we can write the 1-D heat transfer rate equation

$$h_r A_r (t_p - t_r) = UA(t_a - t_r) \quad \text{Eq C.3}$$

Where t_p is the tube surface temperature.

$$\frac{1}{UA} = \frac{1}{h_a A_a h_{sur}} + \frac{1}{h_r A_r} \quad \text{Eq C.4}$$

We also have

$$(t_a - t_{f,m}) = h_f (t_a - t_p) \quad \text{Eq C.5}$$

Where $t_{f,m}$ is the fin mean temperature.

$$t_{f,m} = t_a - h_f \left[\left(1 - \frac{UA}{h_r A_r} \right) (t_a - t_r) \right] \quad \text{Eq C.6}$$

If the mean fin temperature at the leading edge is lower than the air inlet dew point temperature, the surface is totally wet. If the mean fin temperature at the leading edge is higher than the air inlet dew point temperature, we have to determine if the surface is totally dry or not. Recall that when the fin mean temperature is equal to the air inlet dew point temperature, condensation begins. We have

$$t_{a,o} = \frac{t_{dp} - h_f \left(1 - \frac{UA}{h_r A_r} \right) t_r}{1 - h_f \left(1 - \frac{UA}{h_r A_r} \right)} \quad \text{Eq C.7}$$

The heat transfer area needed to make mean fin temperature equal to the air inlet dew point temperature is obtained by

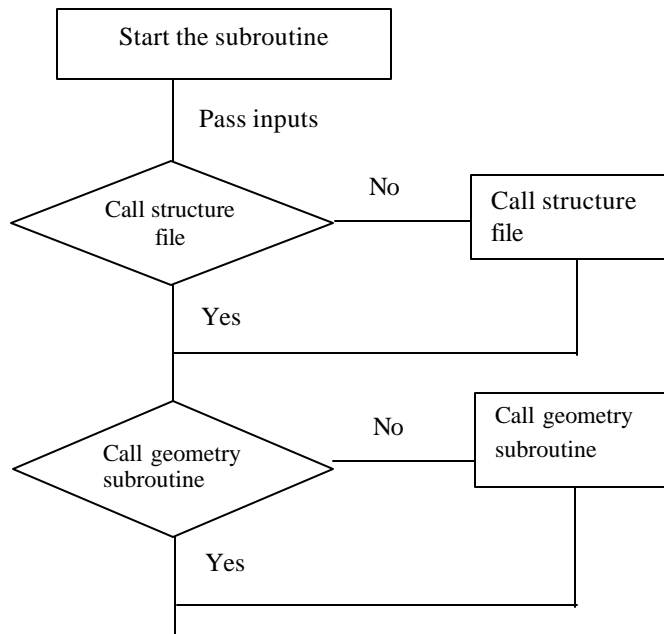
$$m_a c_{p,a} (t_{a,i} - t_{a,o}) = UA_{dry} LMTD \quad \text{Eq C.8}$$

$$\text{Where } LMTD = \frac{(t_{a,i} - t_r) - (t_{a,o} - t_r)}{\ln \frac{t_{a,i} - t_r}{t_{a,o} - t_r}} \quad \text{Eq C.9}$$

If UA_{dry} is larger than UA , the surface is totally dry. If UA_{dry} is smaller than UA , the surface is partially dry and the ratio of UA_{dry} over UA is the dry fraction of the whole heat transfer area.

C.2.4.2 Subroutine description

The Figure C.5 describes the real flow chart in the sequential subroutine called by the evaporator main program. The structure of the evaporator is very similar with the condenser, but the main difference is that we have more lines to deal with the dehumidification since some water will condense from the water air, more details are shown above.



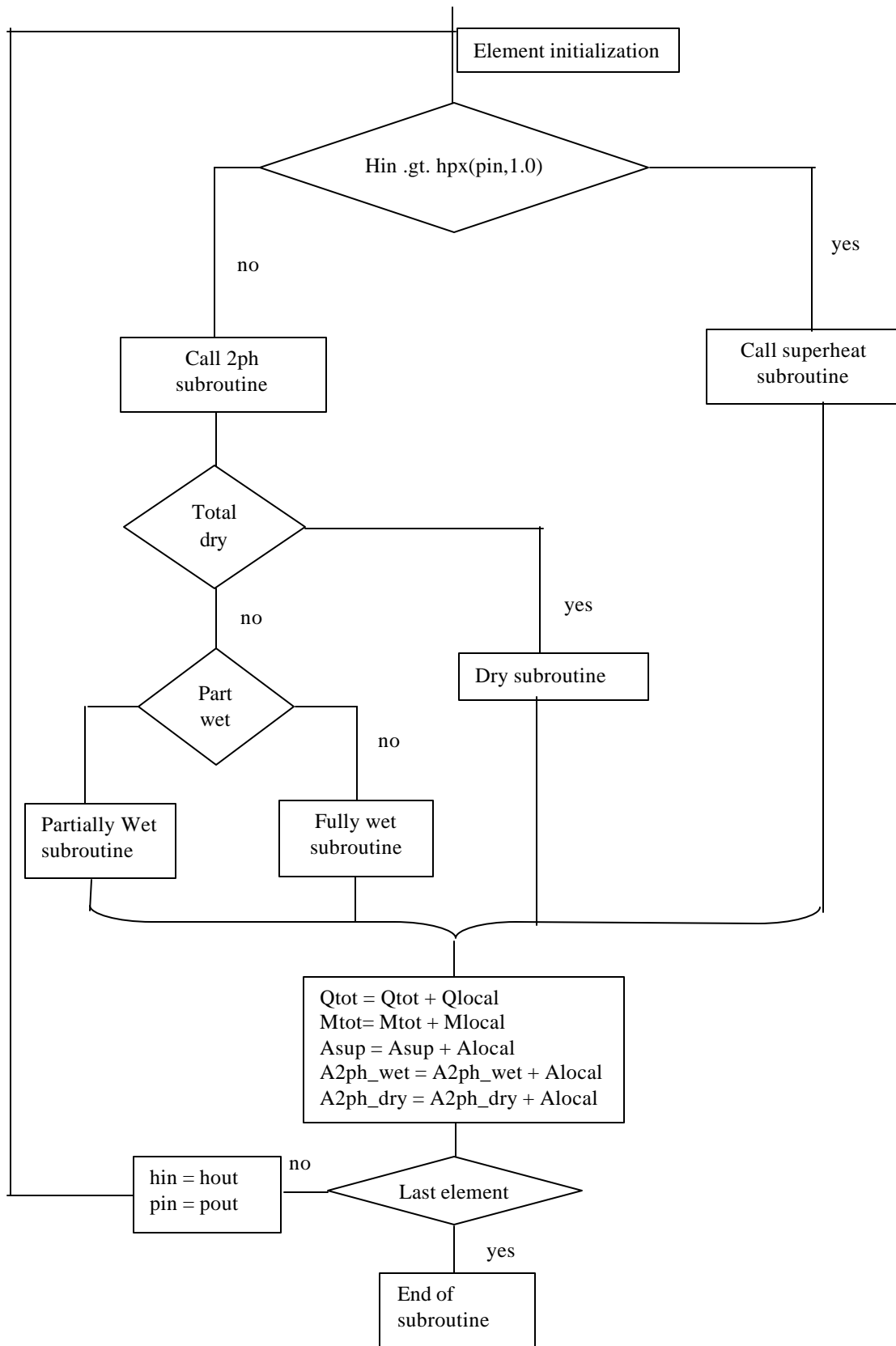


Figure C.5 Evaporator subroutine flow chart

Similar to the condenser component, we have two finite element algorithms to deal with the simulation: downstream marching algorithm and upstream marching algorithm. We have logical variable to call the right subroutine for the simulation in the evaporator main program, 'EVAPORATOR.F'. Consequentially, we have two groups of residual equations for different algorithms, shown below.

```

      If (downwind) then
        Call dwnstmevap(Pinevap, hinevap,houtE_calc,poutE_calc,
&      QtotE_calc,Qsns_calc,Qlat_calc,AtotE_calc,
&      f2phwet,A2ph,heightE_calc,AfrE_calc,massE_calc,
&      T2phout,P2phout,TphoutE,MWR_calc)

        Goto (10,20,30,40,50,60,70,80,90,100), EQNUM

10      R(evap+0) = MtotE - massE_calc
        goto 5000
20      R(evap+1) = Aevap - AtotE_calc
        goto 5000
30      R(evap+2) = Qevap - QtotE_calc
        goto 5000
40      R(evap+3) = AfrontE - AfrE_calc
        goto 5000
50      R(evap+4) = heightE - heightE_calc
        goto 5000
60      R(evap+5) = Poutevap - PoutE_calc
        goto 5000
70      R(evap+6) = houtevap - houtE_calc
        goto 5000
80      R(evap+7) = pwrfanE -PwrfanE_calc
        goto 5000
90      R(evap+8) = MWR -MWR_calc
        goto 5000
100     R(evap+9) = degsup - degsup_calc
        goto 5000
      else
        call upstmevap(Poutevap,houtevap,hinE_calc,pinE_calc,
&      QtotE_calc,Qsns_calc,Qlat_calc,AtotE_calc,
&      f2phwet,A2ph,heightE_calc,AfrE_calc,massE_calc,
&      T2phout,P2phout,TphoutE,MWR_calc)
        Goto (110,120,130,140,150,160,170,180,190,200) , EQNUM

110      R(evap+0) = MtotE - massE_calc
        goto 5000
120      R(evap+1) = Aevap - AtotE_calc
        goto 5000
130      R(evap+2) = Qevap - QtotE_calc
        goto 5000
140      R(evap+3) = AfrontE - AfrE_calc
        goto 5000
150      R(evap+4) = heightE - heightE_calc
        goto 5000
160      R(evap+5) = Pinevap - PinE_calc
        goto 5000
170      R(evap+6) = hinevap - hinE_calc

```

```

                                goto 5000
180                            R(evap+7) = pwrfanE -PwrfanE_calc
                                goto 5000
190                            R(evap+8) = MWR -MWR_calc
                                goto 5000
200                            R(evap+9) = degsup - degsup_calc
                                goto 5000
endif

```

When we call the evaporator subroutine, the inputs are passed into the subroutine. The 'XK.update' file are shared by the whole workspace, so the initial guess values for unknown variables and parameters can be accessed in any file if the 'XK.update' is included in the declaration. During the iterations, all of the residual equations are sent to the Newton-Raphson solver with the calculated variables from the sequential subroutine.

At the beginning of the sequential subroutine, there is also a logical variable, elementread, designed to indicate whether the geometry configuration has been transmitted from the text file, 'evaporator.txt', to the arrays, 'modstartE(maxmod)', 'airinputE(maxNelem)', 'NparamodE(maxNelem)', 'NcircuitE(maxNelem)', used by the subroutine for the simulation. Otherwise, the subroutine written to initialize the arrays is called.

```

if (.not. elementread) then
                                call Readelem(NelpassE,ElperpassE,ModstartE,airinputE,
&                                NparamodE,NcircuitE,'evapinput.txt')
                                elementread = .true.
endif

```

After the geometry initialization, the subroutine designed for the element geometry calculation is called. This calculation is performed for each circuit, including the refrigerant-side and air-side heat transfer area, volume of element, frontal area, frontal height, wall thickness and the ratio of air-side heat transfer area to refrigerant-side, which are used in the calculation in each element.

```

Call FTevapgeom(DinE,DoutE,NtubesperslabE,VtubedistE,WidthE,
&               height,LRtrnBndE,FinthE,FinPthE,
&               TuberowsE,HtubedistE,NelpassE,ElperpassE,
&               thicke,DLE,Afr,VRtrnBndE,DcE,AffE,AairE,
&               coilfactE,Volume,Area,AadivAffE,CSareaE,Ar)

```

The next step is to initialize the variables, including refrigerant state inputs, air state inputs and both refrigerant and air mass flow rates for local element. Then we start to simulate the heat exchanger element. A 'Do-while' loop is introduced to decide whether we are in the range of the maximal element number.

```

do while (currentelement .le. maxelement)
.....
.....
end do

```

At the beginning of each element simulation, we decide the refrigerant inlet status: two phase or superheat, then call different subroutines for the simulation.

```

hsatv = hpx(pin,1.0)
hsatl = hpx(pin,0.0)

```

```

if (hin .ge. hsatv)then
    vapor = .true.
    twoph = .false.
else if (hin .ge. hsatl) then
    vapor = .false.
    twoph = .true.
endif

```

The refrigerant and air mass flow rates are calculated using following equations:

$$w_{local} = (w/dbl(NparamodE(element)))/dbl(NcircuitE(element))$$

$$\dot{m} = (MdotE/Afr)*(DLE*VtubeDistE)$$

The subroutines are called depending on the input status:

```

if (vapor) then
    call supeldwnstmE(hin,Pin,wlocal, Tairout(airinputE(element)),
    &                  RHout(airinputE(element)),mdot,DLE,Area,Volume,
    &                  AaDivAffE,AffE,Afr,coilfactE,DcE,hout,Pout,
    &                  Tairout(element),Qsup,Asup,Vapor,twoph,mass,
    &                  RHout(element))

else if (twoph)then
    call tpheldwnstmE(hin,Pin,wlocal, Tairout(airinputE(element)),
    &                  RHout(airinputE(element)),mdot,DLE,Area,Volume,
    &                  AaDivAffE,AffE,Afr,coilfactE,DcE,Qflux,hout,Pout,
    &                  Tairout(element),RHout(element),Qdry,Qsns,Qlat,
    &                  Adry,Awet,Vapor,twoph,mass,MWRel,Ar)
endif

```

In the subroutine called to simulate the element in two-phase state, we check whether there is water condensing from the warm air onto the evaporator surface, by comparing the surface temperature and dew point temperature. As a result, there are three possibilities: totally dry surface, partially wet surface and fully wet surface.

```

If (fully dry) then
    call drysection(hin,Pin,wlocal,G,Tairin,RHin,mdot,Cair,
    &               Volume,AaDivAffE,AffE,Afr,coilfactE,DcE,Qflux,
    &               Tairout,RHout,Qdry,Area)

else if (partially wet) then
    call Pwetsection(hin,Pin,Tdew,wlocal,G,Tairout,Tairout,RHout,
    &               mdot,Cair,Volume,AaDivAffE,AffE,Afr,coilfactE,DcE,
    &               Qflux,Tairout,RHout,Qsns,Qlat,(Area-Adry),MWRel)

else if (fully wet) then
    call wetsection(hin,Pin,Ts,wlocal,G,Tairin,Tairin,RHin,
    &               mdot,Cair,Volume,AaDivAffE,AffE,Afr,coilfactE,DcE,
    &               Qflux,Tairout,RHout,Qsns,Qlat,Awet,MWRel,1,Ar)
endif

```

Inside the subroutines, the ϵ -NTU method is used to calculate the heat transfer. The governing equations used to calculate the heat transfer coefficients and pressure drop are the same as simultaneous models. The locally calculated variables are accumulated to describe the performance of the evaporator like below:

$$Q_{snstot} = Q_{snstot} + (Q_{sns} + Q_{dry}) * dbl(NcircuitE(element))$$

$$Q_{lattot} = Q_{lattot} + Q_{lat} * dbl(NcircuitE(element))$$

```

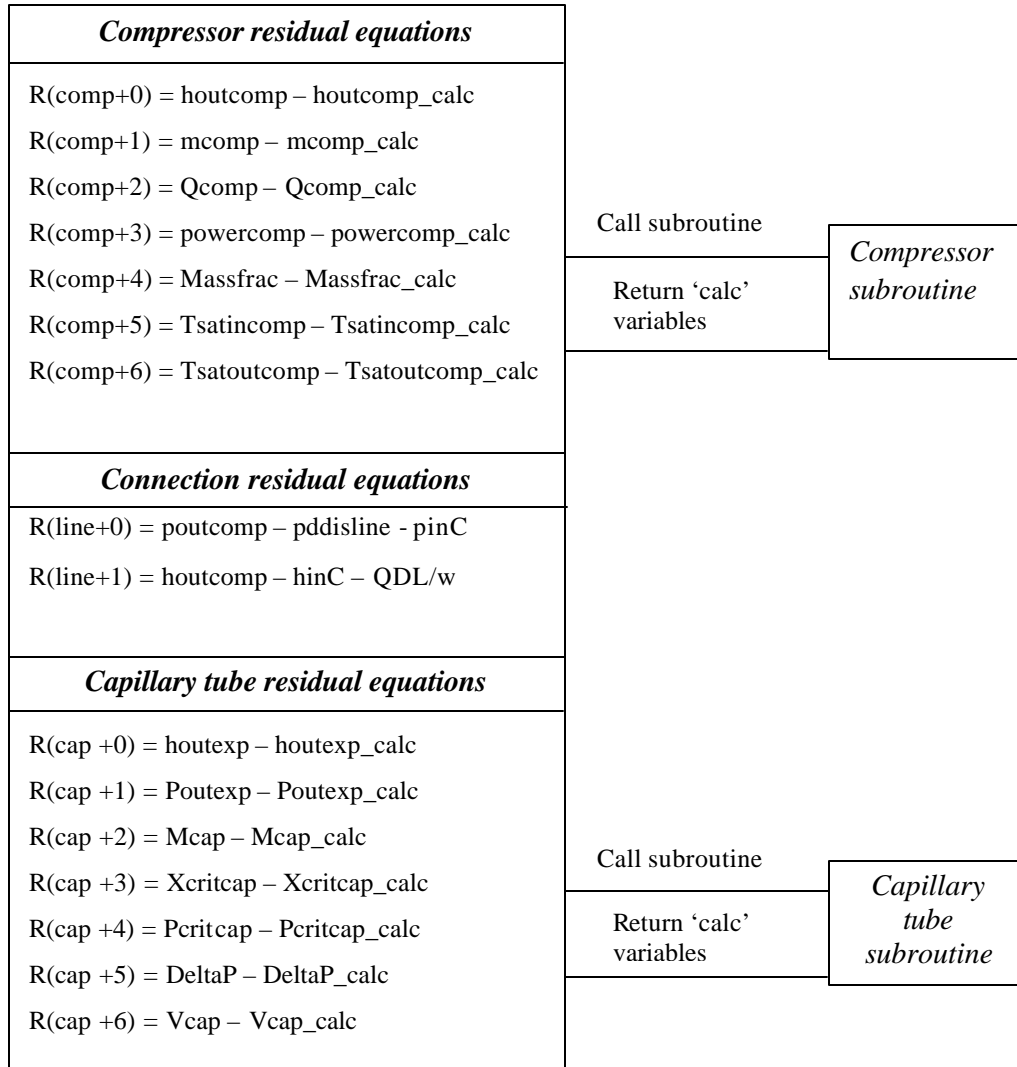
Qsuptot = Qsuptot + Qsup*dble(NcircuitE(element))
Asuptot = Asuptot + Asup*dble(NcircuitE(element))
A2phdrytot = A2phdrytot + Adry*dble(NcircuitE(element))
A2phwettot = A2phwettot + Awet*dble(NcircuitE(element))
masstot = masstot + mass*dble(NcircuitE(element))
MWRtot = MWRtot + MWRel*dble(NcircuitE(element))

```

When the calculation skips out of the loop, all the calculated variables above will be returned back to the evaporator main program used in the residual equations. If they do not match the final solution, the residual equations are sent to the Newton-Raphson solver for further iterations. Newton-Raphson solver updates the input variables for sequential subroutine, which is called until reaching the final solution.

C.3 Residual equations

We greatly reduce the number of residual equation as well as the number of the initial guess values. The Figure C.6 shows us the residual equations, associated to each component and connection point between components.



<i>Condenser residual equations</i>	
$R(\text{cond} + 0) = \text{houtC} - \text{houtC_calc}$ $R(\text{cond} + 1) = \text{PoutC} - \text{PoutC_calc}$ $R(\text{cond} + 2) = \text{Qcond} - \text{Qcond_calc}$ $R(\text{cond} + 3) = \text{TaoutC} - \text{TaoutC_calc}$ $R(\text{cond} + 4) = \text{Mcond} - \text{Mcond_calc}$ $R(\text{cond} + 5) = \text{degsub} - \text{degsub_calc}$ $R(\text{cond} + 6) = \text{Acond} - \text{Acond_calc}$	<div>Call subroutine</div> <div>Return 'calc' variables</div>
<i>Condenser residual equations</i>	
$R(\text{line} + 2) = \text{PoutC} - \text{PinLL}$ $R(\text{line} + 3) = \text{houtC} - \text{hinLL}$ $R(\text{line} + 4) = \text{hinexp} - \text{houtLL}$ $R(\text{line} + 5) = \text{pinexp} - \text{poutLL}$	
<i>Connection residual equations</i>	
$R(\text{line} + 6) = \text{Poutexp} - \text{Pinevap}$ $R(\text{line} + 7) = \text{houtexp} - \text{hinevap}$	
<i>Evaporator residual equations</i>	
$R(\text{evap} + 0) = \text{houtE} - \text{houtE_calc}$ $R(\text{evap} + 1) = \text{PoutE} - \text{PoutE_calc}$ $R(\text{evap} + 2) = \text{Mevap} - \text{Mevap_calc}$ $R(\text{evap} + 3) = \text{TaoutE} - \text{TaoutE_calc}$ $R(\text{evap} + 4) = \text{Qevap} - \text{Qevap_calc}$ $R(\text{evap} + 5) = \text{degsub} - \text{degsub_calc}$ $R(\text{evap} + 6) = \text{MWR} - \text{MWR_calc}$ $R(\text{evap} + 7) = \text{PwrfanE} - \text{PwrfanE_calc}$ $R(\text{evap} + 8) = \text{Aevap} - \text{Aevap_calc}$	<div>Call subroutine</div> <div>Return 'calc' variables</div>
<i>Evaporator residual equations</i>	
$R(\text{line} + 8) = \text{PoutE} - \text{PinSL}$ $R(\text{line} + 9) = \text{houtE} - \text{hinSL}$ $R(\text{line} + 10) = \text{hincomp} - \text{hinSL} - \text{QSL}/w$ $R(\text{line} + 11) = \text{pincomp} - \text{pinSL} + \text{pdsuctline}$	

*Condenser
subroutine*

*Evaporator
subroutine*

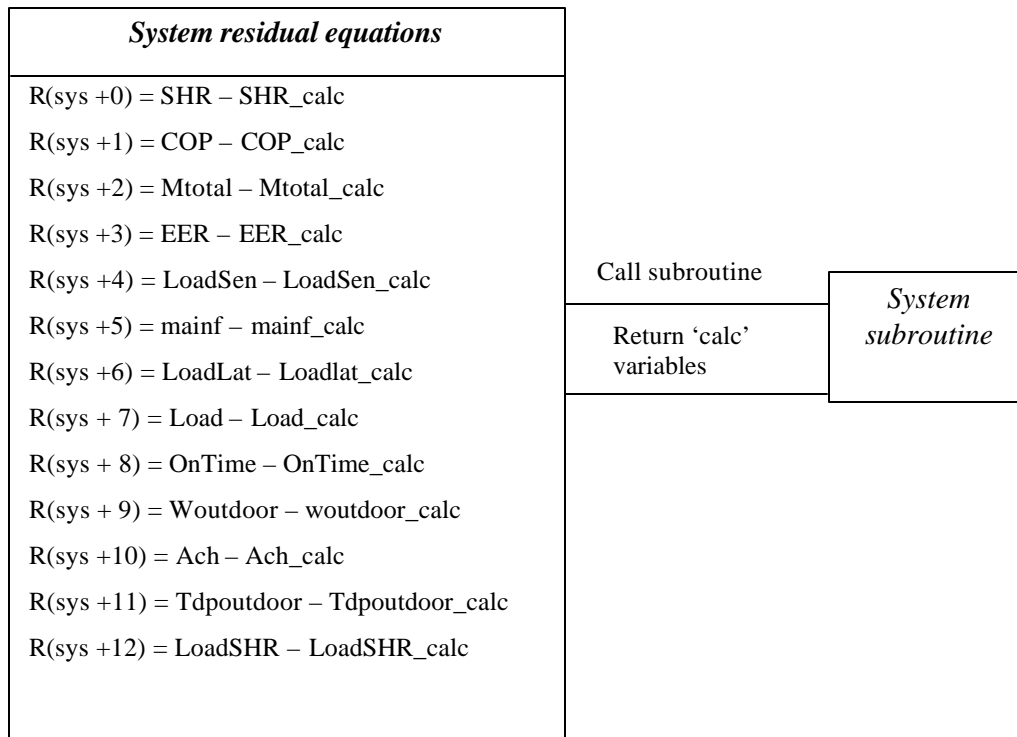


Figure C.6 System residual equations