Cooling System Design Tool for Rapid Development and Analysis of Chilled Water Systems aboard U.S. Navy Surface Ships

By
Amiel B. Sanfiorenzo

Bachelor of Science in Computer Engineering
Penn State University, 2005

Master of Business Administration
Charleston Southern University, 2008

Submitted to the Department of Mechanical Engineering
In Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer

and

Master of Science in Mechanical Engineering

at the
Massachusetts Institute of Technology
June 2013

Signature of Author____________________________________

Amiel B. Sanfiorenzo
Department of Mechanical Engineering
May 13, 2013

Certified by____________________________________

Chryssostomos Chryssostomidis
Thesis Supervisor
Professor of Mechanical and Ocean Engineering

Accepted by____________________________________

David Hardt
Chairman, Committee for Graduate Students
Department of Mechanical Engineering

(THIS PAGE INTENTIONALLY LEFT BLANK)

Center for Ocean Engineering
Naval Construction & Engineering Program
Department of Mechanical Engineering

Massachusetts Institute of Technology
77-massachusetts Avenue, Building 5-317
Cambridge, Massachusetts 02139–4307

Cooling System Design Tool for Rapid Development and Analysis of Chilled Water Systems aboard U.S. Navy Surface Ships

Amiel B. Sanfiorenzo

Submitted to the Department of Mechanical Engineering on May 14, 2013
in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer
and
Master of Science in Mechanical Engineering

Abstract

Over the last several decades, there has been a dramatic increase in the complexity and power requirements of radars and other combat systems equipment aboard naval combatants and this trend is expected to continue for the foreseeable future. This increase in the power demand has a direct effect on the amount of heat which has to be removed by the cooling systems, with future combatants expected to require 5-10 times the cooling capacity currently installed on naval combatants (McGillan, Perotti, McCunney, & McGovern). In the past, the cooling system could be designed and integrated into the ship towards the later stages of the ship design process; however, this is no longer possible. The growing complexity and size of the cooling systems needed require preliminary design and integration in the early-stages of the ship design process. To design and integrate cooling systems several tools are available to the naval architect, but vary in complexity and usefulness depending on the design stage considered.

The focus of this thesis is on the early-stage design of cooling systems aboard U.S. Navy surface ships utilizing the principles of naval architecture and mechanical engineering concepts. The intent was to study the heat transfer process within the chilled water system and the auxiliary seawater system and develop a Cooling System Design Tool (CSDT) based on the thermodynamic laws that govern heat transfer as well as the hydrodynamic principles that govern fluid flow, specifically the incorporation of flow network analysis (FNA). The key purposes of the CSDT are to provide rapid visualization and analysis of the cooling system to test overall feasibility and performance of the system.

The framework of the model was built using Matlab in conjunction with Excel. The program interacts with the user primarily through the command window, guiding the user through the design process. Some visualization is provided as the design progresses, allowing the user to quickly determine and correct errors in the design. The CSDT also displays important results of various analyses that can be performed on the data, including a weight summary, a static temperature distribution, and a temperature distribution that captures transients in space and time. The program interaction, chilled water plots and analyses output enables the user with the ability to quickly visualize, develop and analyze cooling systems aboard naval vessels.

Thesis Supervisor: Chryssostomos Chryssostomidis
Title: Professor of Mechanical and Ocean Engineering

(THIS PAGE INTENTIONALLY LEFT BLANK)

Table of Contents

Massachusetts Institute of Technology
77-massachusetts Avenue, Building 5-317
Cambridge, Massachusetts 02139–4307

List of Figures

List of Tables

Massachusetts Institute of Technology
77-massachusetts Avenue, Building 5-317
Cambridge, Massachusetts 02139–4307

Biographical Note

Lieutenant Ben Sanfiorenzo began his military career as an enlisted soldier in the United States Army. He enlisted in 1997 and following boot camp at Fort Jackson, SC, continued on at Fort Gordon, GA to complete his Advanced Individual Training as a Network Switching Systems Operator/Maintainer (31F). Upon completion of his training, he was stationed at Fort Richardson, AK as a team chief of a Small Extension Node. LT Sanfiorenzo earned his Associates of the Arts degree from the University of Alaska, Anchorage in 2001.

Following active duty, LT Sanfiorenzo joined the Pennsylvania Army National Guard in 2001 while concurrently earning his Bachelor's degree in Computer Engineering from Penn State University. In 2004, LT Sanfiorenzo returned to active duty through the Navy NUPOC program. LT Sanfiorenzo earned his Bachelor's degree in 2005.

Upon completion of Officer Indoctrination School in Newport, RI in 2006, LT Sanfiorenzo reported to Naval Nuclear Power Training Command where he was an instructor in the Enlisted Mathematics Department, the Enlisted Reactor Principles Department, and the Division Director of the Enlisted Mathematics Department. While at NNPTC, LT Sanfiorenzo earned his Master of Business Administration from Charleston Southern University in 2009.

After being selected for lateral transfer into the Engineering Duty Officer community, LT Sanfiorenzo began his training by pursuing a Naval Engineers degree and a Master of Science in Mechanical Engineering degree from the Massachusetts Institute of Technology. Upon completion of his technical training, LT Sanfiorenzo will serve on a submarine to earn his warfare qualification ED dolphin pin.

Lieutenant Sanfiorenzo's awards include the Navy Commendation Medal and the Army Achievement Medal (two awards).

Acknowledgements

First, and foremost, I would like to thank my wife for the support she has given me over the past year while working on this thesis and over the past few years while pursuing my degrees here at MIT. She is very supportive of my dedication to my work and understands all too well the sacrifice in time necessary to accomplish anything worthwhile. I would also like to thank my children, Joseph, Elle, and Zoe for their love and patience and hope I have instilled in them the importance of hard work and perseverance.

I would also like to thank my thesis supervisor, Prof. Chrysostomosis Chryssostimidis for his guidance and feedback while designing the CSDT and writing this thesis. I would also like to thank Dr. Julie Chalfant for her support and vast knowledge of ship systems which has helped me out tremendously. Lastly, I would like to thank Prof. George Karniadakis for his guidance and expertise in heat transfer which enabled me to overcome many difficulties in the design of the CSDT.

To the Navy, I give special thanks for allowing me the opportunity to attend MIT, which had been a dream of mine since I was a child.

Lastly, I would like to mention this work is supported by the U.S. Department of Defense, Office of Naval Research Award Number N00014-08-1-0080, ESRDC Consortium, and MIT Sea Grant College Program under NOAA Grant Number NA06OAR4170019, MT SG Project Number 2008-ESRDC-01-LEV.

1.0 Chapter 1: Introduction

The focus of this thesis is on the early-stage design of cooling systems aboard U.S. Navy surface ships utilizing the principles of naval architecture and mechanical engineering concepts. The intent was to study the heat transfer process within the chilled water system, the seawater system and the electronic cooling water system and develop a Cooling System Design Tool (CSDT) based on the thermodynamic laws that govern heat transfer as well as the hydrodynamic principles that govern fluid flow, specifically the incorporation of flow network analysis (FNA). The key purposes of the CSDT are to provide rapid visualization and analysis of the cooling system to test overall feasibility and performance of the system.

1.1 Organization of Thesis

This thesis contains five chapters (Introduction, Design Tool Fundamentals, Design Tool Architecture, Simulation & Results, and Conclusions) and two appendices. The Introduction provides background information and fundamental concepts pertaining to chilled water systems, seawater systems and electronic cooling water systems. It also provides a brief discussion pertaining to the motivation behind the CSDT and the intent of this thesis. Design Tool Fundamentals provides the theory to which the CSDT algorithm was based upon. This includes: fundamental heat transfer concepts, pipe characteristics, flow network analysis, pump and valve characteristics, head loss associated with flow configurations and junctions, heat exchanger and cooling coil characteristics, expansion tank design concepts, and A/C unit characteristics. Design Tool Fundamentals also provides assumptions made pertaining to the theory behind the CSDT as well as validation of those assumptions wherever possible. Design Tool Architecture describes the layout of the CSDT, in particular the user inputs and outputs provided by the CSDT, and an in-depth explanation of the CSDT algorithm. Design Tool Architecture also explains the program requirements and the user pre-requisites, and guidelines in designing a chilled water system. Simulation & Results discusses in detail an example of a cooling system modeled using the CSDT, including pertinent analyses of the cooling system. The modeled cooling system is analyzed statically as well as dynamically. Several scenarios are explored to study the effects of the thermal transients. Lastly, Simulation & Results also contains validation of the CSDT transient analysis through the use of analytic comparison. The final chapter, Conclusions, the major benefits and drawbacks of the CSDT are discussed, as well as areas of future research. The attached appendices include the notional heat loads used in the simulation and refrigerant characteristics.

1.2 Topic Motivation

Over the last several decades, there has been a dramatic increase in the complexity and power requirements of radars and other combat systems equipment aboard naval combatants and this trend is expected to continue for the foreseeable future. This increase in the power demand has a direct effect on the amount of heat which has to be removed by the cooling systems, with future combatants expected to require 5-10 times the cooling capacity currently installed on naval combatants (McGillan, Perotti, McCunney, & McGovern). In the past, the cooling system could be designed and integrated into the ship towards the later stages of the ship design process; however, this is no longer possible. The growing complexity and size of the cooling systems needed require preliminary design and integration in the early-stages of the ship design process. To design and integrate cooling systems several tools are available to the naval architect, but vary in complexity and usefulness depending on the design stage considered.

For early-stage design, ASSET and Rhinoceros may be used. ASSET provides the naval architect with the basic idea of a ship based on relatively few input parameters. This is often the start to a new (or modified) ship design, and it offers much in return pertaining to weights, electric loads and general hydrostatic analyses. This information can then be used in conjunction with other tools such as POSSE or Rhinoceros for further development in other specific areas such as intact and/or damaged stability and 2-D/3-D arrangement drawings of the ship. ASSET does provide output pertaining to the cooling system such as weight and power requirements. However, this is based on historical data of older surface ships. ASSET offers very little in the design of the cooling system, only allowing the user to specify weight, center of gravity, area and power through the use of the Payloads and Adjustments table.

Rhinoceros is a CAD tool that can be used to design the internal and external arrangements of a ship. This may be used in the design of a cooling system, but only gives the naval architect the ability to visualize the layout of the cooling system if the design is already known. Rhinoceros offers no capability to analyze the cooling system, other than visualization.

For mid-stage design, Paramarine can be used. The tool offers much capability in analyzing the cooling system, including visualization of the piping structure, weight analysis and flow analysis. The major drawback of using Paramarine is the complexity of the tool. There is a very steep learning curve associated with Paramarine and much time has to be invested in order to become proficient and take advantage of what Paramarine has to offer.

Finally for late-stage design, commercially available tools such as Flowmaster®, PIPE-FLO®, and FluidFlow® may be used. These tools are useful in solving for flow and pressure within the piping network, and have the capability to integrate several systems together such as the HVAC and chilled water systems, but require an in-depth model of the ship and piping structure.

A previous MIT 2N student, Ethan Fiedel, recognized this need for an early-stage cooling system design tool that is easier to use than Paramarine and which does not require an in-depth model of the ship. Fiedel's version of the CSDT (CSDT v1.0) provided much insight into the design of the chilled water system and provided an interface with Paramarine for further analysis. However, a drawback to CSDT v1.0 was the use of rules of thumb for determining flow within the piping network (Fiedel, 2011). In contrast to CSDT v1.0, this version of the CSDT (CSDT v2.0[1]) focuses on designing the cooling system through the use of hydrodynamic and thermodynamic principles beginning with the projected heat loads of the ship and the location of these loads.

[1] This paper refers to CSDT v2.0 simply as CSDT. When referring to Fiedel's version, the v1.0 is explicitly stated.

1.3 Description of Cooling Systems

There are many different types of cooling systems aboard U.S. Navy surface ships. This paper focuses on three cooling systems including the chilled water system, the seawater system and the electronic cooling water system. All three cooling systems provide similar functions in providing cooling to various electronic components but vary in cooling water temperature and purity.

The chilled water system is only one of many freshwater systems aboard U.S. Navy vessels. The purpose of the chilled water system is to provide cooling for electronic cooling water heat exchangers for electronic components requiring demineralized water below a certain temperature and for other electronic equipment requiring cooling water. The A/C cooling coils use a significant amount of chilled water, accounting for as much as 75% of the heat load serviced by the chilled water system. Other components requiring chilled water may include SQS-53 (surface sonar), SLQ-32 (surface electronic warfare system), SPY Antenna (surface radar), A/C Unit Lube Oil Cooler, among other electronics equipment and coolers (Frank & Helmick, 2007).

The seawater system provides a low cost solution in removing waste heat offering a lower weight and smaller footprint than that of the chilled water system, but the cooling fluid temperature is generally higher (Johnson, West, Miller, & Zouridakis, 2004). Also, if used directly to cool electronic equipment, fouling of the channels may take place. Therefore, a flat plate heat exchanger is typically used to transfer heat between the seawater loop and a demineralized water loop as seen in Figure 1 below.

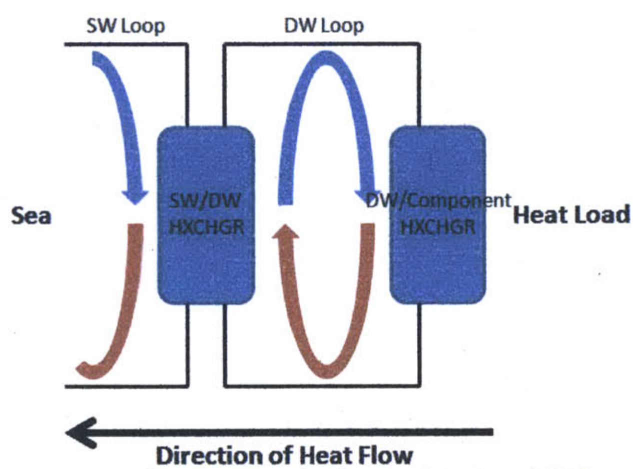


Figure 1: Heat flow from heat load to sea via DW, and SW loops

The electronic cooling water system is a system specifically designed to remove heat from electronic equipment by supplying necessary quantities of conditioned coolant water. The electronic cooling water system can be broken down further into three distinct cooling water systems based upon the cooling water temperature required by the electronic equipment.

1. For high temperature limit applications: The seawater primary cooling system supplies cooling water for electronic components requiring cooling water in excess of the highest expected seawater temperatures.
2. For low temperature limit applications: The chilled water primary cooling system supplies chilled water for electronic components requiring cooler cooling water.
3. For mid temperature limit applications: If the required cooling water temperature is close to that of the maximum expected seawater temperature, then a hybrid approach may be taken. The electronic cooling water system is cooled by chilled water when the seawater temperature is high, but can be cooled by seawater if the seawater temperature is low enough.

Each of these three configurations utilize a heat exchanger to transfer heat from either the seawater or chilled water loop to the demineralized water within the electronic cooling water system.

1.3.1 Description of Chilled Water Systems

The chilled water system may be composed of several chilled water plants. Each chilled water plant is made up of several major components, including: an air conditioning chilled water plant (a chiller), chilled water pumps (historically centrifugal pumps), a chilled water expansion tank, a chilled water supply and return header, and various instruments and controls. The chilled water system is usually broken up into several zones within the ship. Each zone contains a chilled water plant and branch piping which serve to provide a closed looped system capable of circulating chilled water within the loop and provide cooling for all equipment within that zone. The chilled water supply and return piping have components which run longitudinally along the majority of the ship's length (chilled water mains) and vertical components (chilled water risers) which connect the chillers to the chilled water mains. The chilled water branches are typically smaller diameter piping which branches off of the supply header and provides cooling to the heat loads. The branch piping reconnects downstream to the return header, forming a closed loop. Cross-connections provide connections athwartships between chilled water mains. A diagram showing the interconnections of the major components is shown in Figure 2.

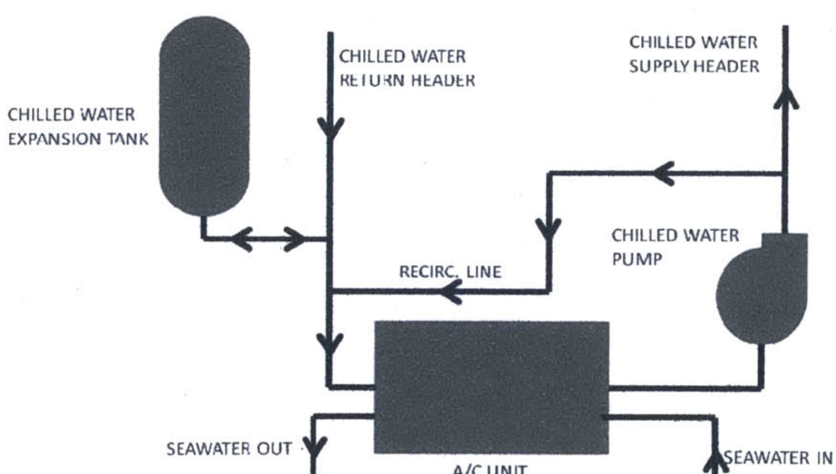


Figure 2: Schematic of chilled water plant (valves not shown)

The chilled water plant can be configured in several different ways. The simplest configuration consists of a single freshwater chiller, a single chilled water circulating pump and a single chilled water expansion tank. The chiller takes the hotter fluid returning from the branch piping and return header and cools it to approximately 6.6°C (Pruske & Kiehne). The cooler fluid is pumped by the circulating pump and is discharged into the supply header, where it diverges into the branch piping. Connected to the return header, the expansion tank provides an expansion volume when the chilled water is secured and the temperature of the water rises. In addition, the expansion tank provides a source of make-up water. Other configurations of chilled water plants consists of two chillers with two pumps operating within a single zone and sharing a single supply and return header. This increases the cooling capacity within that zone. It is also possible to have a single chiller and pump in two different zones, each with their own supply and return headers with the two zones having a cross-connection. This provides flexibility in separating the two zones by shutting the cross-connect valve; however, the cross-connect valve could be opened if one system is down, allowing the other chilled water plant to supply chilled water to both zones.

Within each zone, the heat loads can be broken up into vital and non-vital loads. Vital loads consist of machinery space services, electronic equipment, and vital air conditioning cooling coils. Non-vital loads contain all services not classified vital. An example of a vital load branch of the chilled water system is shown in Figure 3.

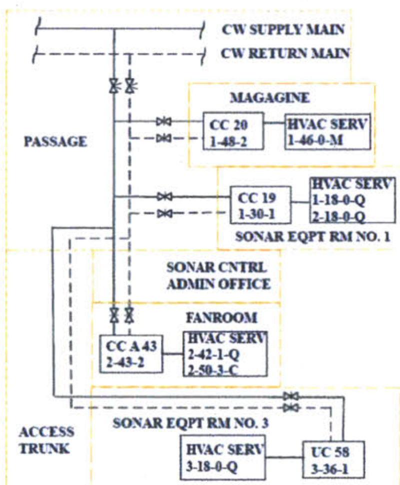


Figure 3: Example of chilled water vital load branch - DDG51 1W (Fang, Jiang, Khan, & Dougal)

1.3.2 Description of the Seawater System

The seawater system provides seawater to the ship through the use of the main and auxiliary seawater systems. The auxiliary seawater system is of primary importance since this is the system used for A/C unit heat rejection. The auxiliary seawater system is composed of several SW pumps which pump seawater from the sea chests through a seawater piping system. The seawater can be used to transport waste heat from various locations such as the condensing coils within the A/C plant or the seawater side of FW/SW heat exchangers. A drawback to the seawater system is that the temperature of the sea has to be accounted for. Also, the impurity of the sea does not allow the seawater to be used directly to cool components in most applications. However, the main benefit of the seawater system is the plentiful source of water it provides and the relatively low cost of the seawater system, making it an attractive option for cooling systems. In fact, the use of a FW/SW cooling system is used wherever possible due to the lower cost over the chilled water system and the lower footprint required in implementing a FW/SW cooling system (Johnson, West, Miller, & Zouridakis, 2004).

1.3.3 Description of Electronic Cooling Water Systems

The electronic cooling water system is a closed system that works in conjunction with either a chilled water loop or a seawater loop or both. As stated above, this is dependent on the cooling water temperature needed within the electronic cooling water system.

The most desirable type would be a seawater cooling system-electronic cooling water system configuration since this is the lowest cost solution. However, this configuration is only possible if the cooling water needed is 5-10°F above the maximum seawater temperature. The electronic components transfer their heat to the electronic cooling water via a heat exchanger, possibly through the use of a cold plate with very thin channels. The warmer electronic cooling water then transfers heat to the seawater within the seawater loop via a seawater/demineralized water heat exchanger. The warmer seawater is then discharged overboard and cooler seawater is pumped in the seawater inlet.

Another configuration of the electronic cooling water system would be that of the chilled water cooling system-electronic cooling water system. This configuration is necessary when the electrical components require a high level of cooling water purity and a low temperature for the cooling water. The configuration is similar to that described above in that the electronic cooling water system comprises a closed loop that transfers heat via a heat exchanger. The heat exchanger transfers the heat from the warmer demineralized cooling water to the cooler chilled water. This cools down the demineralized water within the electronic cooling water system and this cooler water is circulated through the channels of the electronic component heat exchangers. The chilled water then rejects heat to the sea via the condenser[2] within the A/C unit.

The last configuration of the electronic cooling water system is the seawater/chilled water cooling system-electronic cooling water system. This configuration is used when the electronic components require a cooling water temperature between the two ranges discussed above. This configuration incorporates two heat exchangers, a SW/DW heat exchanger and a CW/DW heat exchanger. Seawater can be used as the primary heat sink. When the seawater inlet temperature is low enough, the heat is

[2] There is actually an additional closed loop within the A/C unit. The warmer chilled water transfers heat to the cooler refrigerant within the A/C unit. The refrigerant is compressed causing a rise in temperature. The hot refrigerant transfers heat to cool seawater. The warmer seawater is then discharged overboard. This is discussed in greater detail in Section 2.10.

transferred to the seawater loop. However, if the seawater temperature is too great, the heat from the electronic cooling water system is transferred to the chilled water system via the CW/DW heat exchanger.

A diagram of the heat flow of the electronic cooling water system, the chilled water system and the seawater system and its interfaces are shown below in Figure 4.

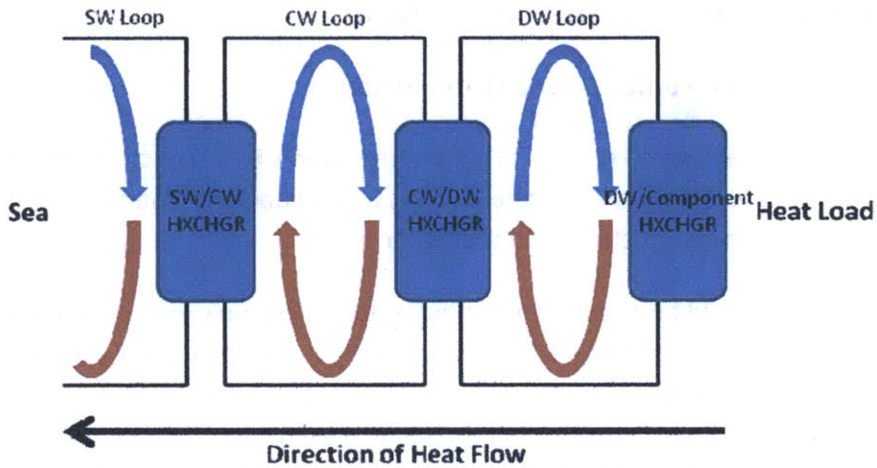


Figure 4: Heat flow from heat load to sea via DW, CW, and SW loops

1.4 Thesis Intent

The intent of this thesis is to provide a more refined CSDT that can be used by Naval Architects, students training to become Naval Architects, Technical Warrant Holders and practicing engineers. This includes modeling the CSDT from thermodynamic and hydrodynamic principles. The framework of the model was built using Matlab in conjunction with Excel. The program interacts with the user primarily through the command window, guiding the user through the design process. Some visualization is provided as the design progresses, allowing the user to quickly determine and correct errors in the design. The CSDT also displays important results of various analyses that can be performed on the data, including a weight summary, a static temperature distribution, and a temperature distribution that captures transients in space and time. The program interaction, chilled water plots and analyses output enables the user with the ability to quickly visualize, develop and analyze cooling systems aboard naval vessels.

2.0 Chapter 2: Design Tool Fundamentals

Thermodynamic laws and equations and hydrodynamic principles form the basis of the CSDT. This is the most fundamental difference between CSDT v1.0 and the version discussed in this paper. Where CSDT v1.0 incorporated rules of thumb to determine the pipe characteristics (e.g., diameter) and flow characteristics (e.g., velocity and mass flow rate), the current CSDT version uses thermodynamic and hydrodynamic principles to determine these characteristics (Fiedel, 2011).

2.1 Heat Transfer Fundamentals

The major components that comprise the chilled water system include: valves, pumps, heat exchangers, expansion tanks, and the pipes that connect these components together. To determine the pipe dimensions it is necessary to explore the heat transfer processes involved within the chilled water system.

2.1.1 Modes of Heat Transfer

Conduction and radiation are the two modes of heat transfer; however, convection is also often thought as a separate and distinct mode of heat transfer. The main difference between conduction and radiation is the mean free path of the energy carriers. Conduction can be described as the transfer of energy between molecular elements with a short mean free path between interactions. Radiation is similar, but the mean free path is much larger. On the other hand, convective heat transfer can be described as the process of heat transfer between a solid and a moving fluid, an efficient way to transfer heat since thermal energy is transported due to fluid motion (Mills, 1999). This paper focuses on the heat transfer processes involving conduction and convection. The basic equations used to compute the rate of heat transfer for convection and conduction are:

$$\dot{Q} = \dot{m} c_p \Delta T_{conv}$$

Equation 1 (Mills, 1999)

and

$$\dot{Q} = UA\Delta T_{cond}$$

Equation 2 (Mills, 1999)

respectively, where $\dot{Q}$ is the rate of heat transfer [W], $\dot{m}$ is the mass flow rate of the fluid [kg/s], c_p is the specific heat capacity of the fluid [J/kg-K], ΔT_{conv} is the differential temperature of the fluid undergoing convection [K], ΔT_{cond} is the differential temperature across the boundary/medium [K], U is the overall heat transfer coefficient [W/m^2-K], and A is the area of the surface in which the heat transfer occurs [m^2].

2.1.2 Types of Flow

In addition to the modes of heat transfer, it is also important to distinguish between the types of flow that exist for convective heat transfer. Flow can be laminar or turbulent, forced or natural, internal or external.

2.1.2.1 Laminar vs. Turbulent

When hydrodynamically fully developed, laminar flow within a cylindrical tube has a parabolic velocity profile consistent with Poiseuille flow. For turbulent flow, there is greater mixing of the fluid within the center of the channel (tending to flatten out the velocity profile towards the center of the channel), and therefore, there are greater rates of heat transfer and higher convective heat transfer coefficients. The flow regime can be determined by the Reynold's number:

$$Re = \frac{VD}{\nu}$$

Equation 3 (Mills, 1999)

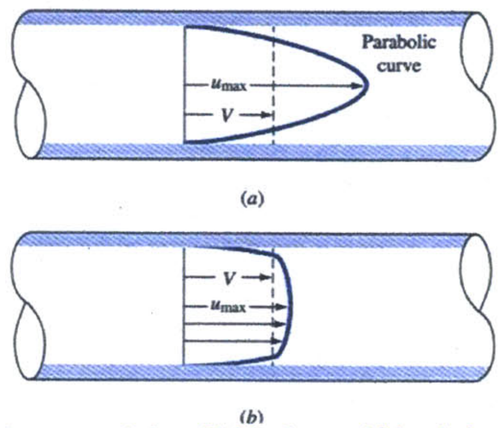


Figure 5: Depiction of (a) laminar and (b) turbulent flow (Sellens)

where Re is the Reynolds number (dimensionless), V is the velocity of the fluid [m/s], $D$is the characteristic dimension of length [m], which in this case is the diameter of the pipe, and ν is the kinematic viscosity [m^2/s] (Mills, 1999). Laminar flow generally forms with Re < 2,300, while fully turbulent flow forms with Re > 10,000. There is a critical zone that exists for Re between 2,300-5000 and a transition zone that depends on the Re number and the relative roughness of the pipe (Mills, 1999). A profile of flow within a channel is shown in Figure 5 which depicts laminar flow and turbulent flow in a cylindrical pipe.

2.1.2.2 Forced Convection vs. Natural Convection

For convective heat transfer, the main methods of heat removal are through forced convection (air), forced convection (liquid), natural convection (air), and natural convection (liquid). The difference between natural convection and forced convection is that in forced convection the fluid (either air or liquid) is propelled by some external force, usually a fan or a pump. With natural convection, the fluid circulates due to differences in density caused by differences in temperature. The hotter, less dense fluid rises and the cooler, denser fluid falls. This can result in circulation of the fluid with gravity as the force which sustains the flow of the fluid. The method of heat removal plays a crucial role in the efficiency of heat transfer between the heat source and the heat sink. Typical ranges of the average convective heat transfer coefficients of air and water undergoing forced convection and natural convection are summarized in Table 1 below. The average heat transfer coefficient is dependent on the geometry of the system, the fluid velocity, and the fluid thermal conductivity.

Flow and Fluid	$\bar{h}_c$ [W/m^2-K]
Natural convection, air	3-25
Natural convection, water	15-1,000
Forced convection, air	10-200
Forced convection, water	50-10,000

Table 1: Range of average convective heat transfer coefficients for various flow and fluid (Mills, 1999)

Higher average convective heat transfer coefficients will result in smaller differential temperatures needed for the same rate of heat transfer. Because of this, forced convection is generally used in chiller systems; however, many systems aboard naval vessels use forced convection (air) to cool electrical components, which is not as efficient as direct contact with water as discussed in the paper *Thermal-Electric Co-Simulation of Power Conversion Systems aboard an All-Electric Ship* (Pruske & Kiehne). To increase the surface area of the electrical components, fins are generally used, which results in higher heat transfer coefficients. Some examples of fins used in standard integrated circuits packages can be seen in Figure 6. In addition, fins can be attached to the outer surface of the chilled water piping in contact with the hot flowing air. This increases the surface area in contact with the air, thus increasing the heat transfer efficiency. However, even with the use of fins both on the electrical components and on the chilled water piping, the growing trend of increased heat generation and thermal loads may be too great as the Navy shifts towards larger and more powerful electrical systems and the all electric ship. With this in mind, other methods of thermal management should be explored such as direct contact of fluid with electrical components along with more exotic methods such as two-phase flow and jet spray methods.

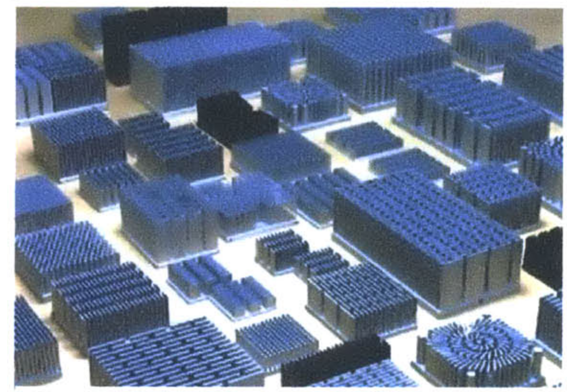


Figure 6: Examples of fins used in cooling electrical components (Alpha Novatech, 2007)

2.1.2.3 Internal vs. External Flow

Internal flow describes the flow of chilled water within the cooling system. The velocity profile for internal flow is shown above in Figure 5. External flow is a bit more complicated and is as equally important to the chilled water system because within the heat exchangers, forced air passes across the external surface of the pipe cylinders[3]. Figure 7 shows the flow pattern for flow across a cylinder for different regimes.

[3] This is assuming the heat exchanger is similar to that of a cooling coil. For a flat plate heat exchanger, a cold plate heat exchanger, or a more exotic heat exchanger, the heat transfer mechanism on the secondary side differs.

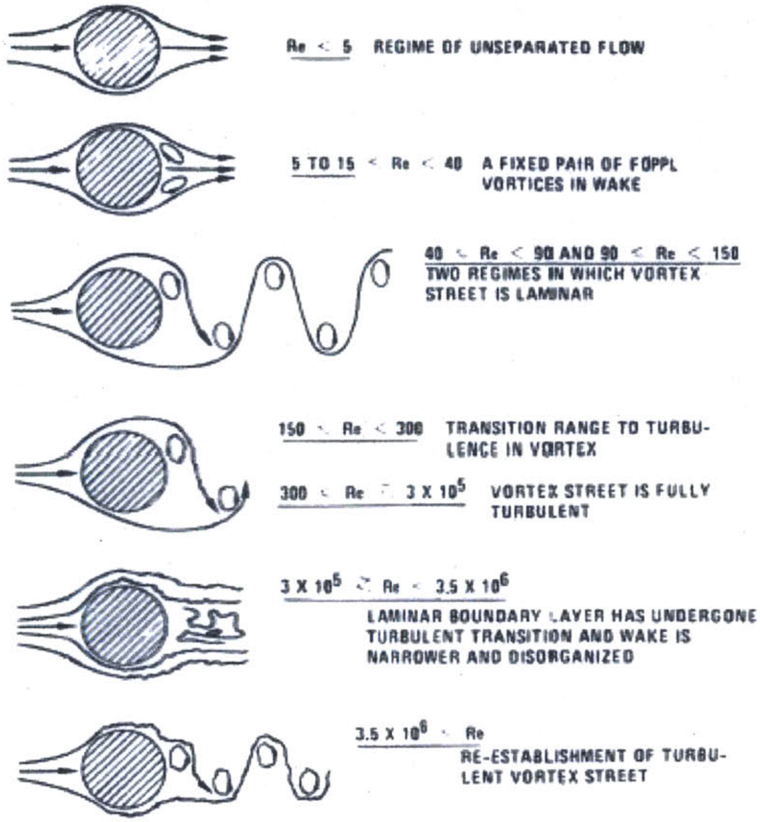


Figure 7: Flow across a cylinder for different flow regimes (Sunden, 2011)

Most of the heat sources identified within the library of the CSDT have associated heat transfer coefficients; however, if not specified, a set of empirical equations can be used to determine the average Nusselt number. The equations suggested by Churchill and Bernstein are shown below.

$$\overline{Nu}_D = 0.3 + \frac{0.62 Re_D^{\frac{1}{2}} Pr^{\frac{1}{3}}}{\left[1 + \left(\frac{0.4}{Pr}\right)^{\frac{2}{3}}\right]^{\frac{1}{4}}} \quad for\ Re_D < 10^4$$

Equation 4 (Mills, 1999)

$$\overline{Nu}_D = 0.3 + \frac{0.62 Re_D^{\frac{1}{2}} Pr^{\frac{1}{3}}}{\left[1 + \left(\frac{0.4}{Pr}\right)^{\frac{2}{3}}\right]^{\frac{1}{4}}} \left[1 + \left(\frac{Re_D}{282{,}000}\right)^{\frac{1}{2}}\right] \quad for\ 2x10^4 < Re_D < 4x10^5$$

Equation 5 (Mills, 1999)

$$\overline{Nu}_D = 0.3 + \frac{0.62 Re_D^{\frac{1}{2}} Pr^{\frac{1}{3}}}{\left[1 + \left(\frac{0.4}{Pr}\right)^{\frac{2}{3}}\right]^{\frac{1}{4}}} \left[1 + \left(\frac{Re_D}{282{,}000}\right)^{\frac{1}{2}}\right]^{\frac{4}{5}} \quad for\ 4x10^5 < Re_D < 5x10^6$$

Equation 6 (Mills, 1999)

where $\overline{Nu}_D$ is the average Nusselt number (dimensionless), Re_D is the Reynolds number (dimensionless), and Pr is the Prandtl number (dimensionless). These equations should be used with caution, as they represent external flow over a cylindrical pipe. If the geometry is more complex, including bends, fins, cross-flow, etc., then the above equations should not be used and the convective heat transfer coefficient should be determined experimentally.

2.1.3 Temperature Profile

The main purpose of the chilled water system is to cool electrical equipment such that the system and component levels of electrical equipment stay below a certain temperature threshold. If this threshold is surpassed, then failure of electrical systems and/or components will follow. With this in mind, a maximum temperature threshold is established for each group of equipment cooled by the chiller system. By default, it was assumed that the electrical components could not exceed a temperature of 100°C. Through the use of forced convection of air (or some liquid), the electrical components are cooled through the use of a fan blowing over the surface of the components (or recirculation pump in the case of a liquid). The hotter air (liquid) then passes over the surface of the piping of the chilled water system (the tube bundles within the heat exchanger). The surface temperature of the chilled water system piping is much cooler and thus cools the hot air (liquid), which is then recirculated back to the electrical components. The surface of the piping is heated up by the hot air (liquid) and heat is transferred through conduction across the outer wall of the pipe to the inner wall of the pipe. The piping holds the chilled water which flows at some velocity. The forced convection of water within the pipe removes the heat generated by the electrical components and transfers the heat to the chiller unit. In steady state, the heat generated by the heat source is equivalent to the rate of heat transfer across each boundary, as well as the rate of heat transfer from inlet to outlet[4]. The cross-sectional view of the pipe and its associated temperature profile for steady-state heat transfer is shown below in Figure 8.

[4] This assumes the loss into the surrounding air is negligible. In reality, some of the heat load will be dissipated into the surrounding air through the boundaries of the component, such as the cabinet walls which house electronic equipment. The CSDT makes the assumption that the heat load provided by the user is not the total heat generated by the component, but rather the portion of that heat load which is to be removed by the chilled water.

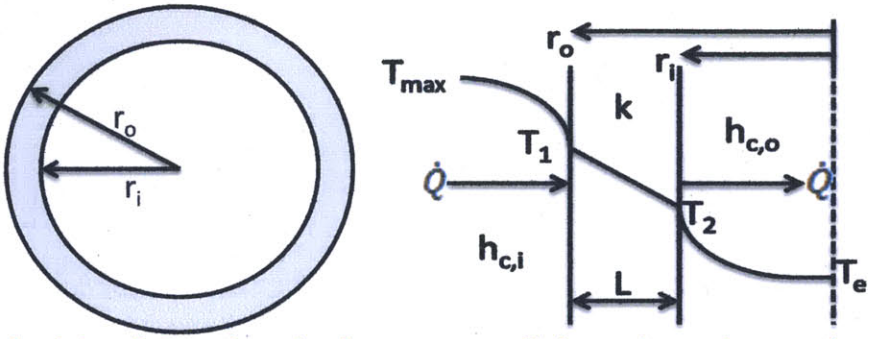


Figure 8: Cross-sectional view of pipe and associated temperature profile for steady-state heat transfer across the pipe wall

The temperature profile shows the rate of heat transfer from the hotter fluid through the pipe wall and into the fluid within the pipe with the distance varying radially from the center of the pipe. When in steady-state, the rate of heat transfer will be equal across each boundary and will be equivalent to the rate of heat generation of the heat source (electrical waste heat). The variable T_{max} corresponds to the temperature of the hotter fluid being blown across the surface of the electrical components. This hot fluid comes in contact with the surface of the outer pipe wall. The surface temperature of the outer pipe wall is T_1. The temperature drops linearly through the pipe wall by conduction. Lastly, the temperature drops throughout the flowing fluid within the pipe, with the center of the pipe having a temperature of T_e. The surface temperature of the inner pipe wall is T_2. Each layer also has specific thermal properties described by the variables $h_{c,i}$, $h_{c,o}$, and k. The two fluids undergoing forced convection have associated heat transfer coefficients $h_{c,i}$ and $h_{c,o}$. The pipe has a certain thickness, L, and a thermal conductivity, k, which is dependent on the material composition.

This heat transfer process can be depicted using an electrical diagram. The difference in temperature from the heat source to the free stream fluid flowing in the pipe can be thought of as a voltage potential. Each boundary also has some resistance to the flow of heat and can be thought of as a resistor. The flow of heat from the heat source to the heat sink (the fluid in the pipe) can be thought of as current. Figure 9 is a thermal circuit showing the heat transfer process.

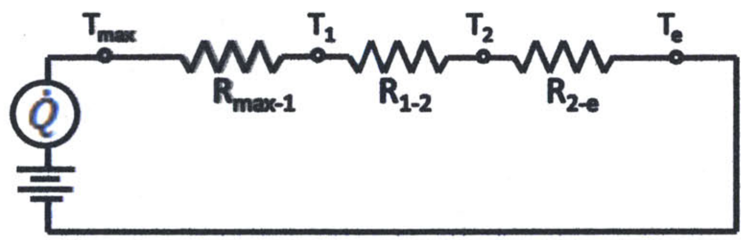


Figure 9: Electrical analogy to heat flow (thermal circuit)

Each resistance can be calculated if the properties of the medium are known. Going from the electrical components to the outer wall of the piping, the following equation was used to determine the resistance to heat flow, where A_i is the area of contact (the inner surface area of the pipe):

$$R_{max-1} = \frac{1}{h_{c,i}A_i} = \frac{1}{h_{c,i}2\pi r_o}$$

Equation 7 (Mills, 1999)

To determine the resistance to heat flow across the piping wall, the following equation was used:

$$R_{1-2} = \frac{ln\left(\frac{r_o}{r_i}\right)}{2\pi kL}$$

Equation 8 (Mills, 1999)

The above equation had to take into account the curvature of the pipe, which is why there is a logarithmic term in the numerator as opposed to a linear term as is the case for a slab. Lastly, to determine the resistance to heat flow from the inner wall to the fluid in the center of the pipe, the following equation was used:

$$R_{2-e} = \frac{1}{h_{c,o}A_o} = \frac{1}{h_{c,o}2\pi r_i}$$

Equation 9 (Mills, 1999)

Using the equations of resistance (Equations 7-9) along with the analogy of Ohm's law, the temperature values at each node can be determined as shown in the equation below.

$$T_{max} = T_e + \frac{\dot{Q}}{unit\ length}\sum R = T_e + \frac{\dot{Q}}{unit\ length}\left[\frac{1}{h_{c,i}2\pi r_o} + \frac{ln\left(\frac{r_o}{r_i}\right)}{2\pi kL} + \frac{1}{h_{c,o}2\pi r_i}\right]$$

Equation 10 (Mills, 1999)

2.1.4 Convective Heat Transfer Coefficient

An important parameter to be calculated is the convective heat transfer coefficient. To determine the convective heat transfer coefficients $h_{c,i}$ and $h_{c,o}$ the flow regime must be known for the two fluids. For the case of laminar flow, the convective heat transfer coefficient can be computed using the following equation:

$$h_c = 3.66\frac{k}{D}$$

Equation 11 (Mills, 1999)

where $k$is the fluid thermal conductivity [W/m-K] and D is the diameter of the pipe [m]. This equation assumes that the temperature along the pipe wall is constant and that the point of interest is far from the entrance of the pipe, where there may be some fluctuations in h_c due to vortices and a step-change in heat exchange across the pipe length at the pipe entrance.

For the case of fully turbulent flow (Re_D > 10,000) and $Pr > 0.5$, the convective heat transfer coefficient can be computed using the following equation:

$$h_c = 0.023\frac{V^{0.8}k^{0.6}(\rho c_p)^{0.4}}{D^{0.2}\nu^{0.4}}$$

Equation 12 (Mills, 1999)

where V is the velocity of the fluid [m/s], k is the fluid thermal conductivity [W/m-K], ρ is the density of the fluid [kg/m^3], c_p is the specific heat capacity of the fluid [J/kg-K], D is the diameter of the pipe [m], and ν is the kinematic viscosity [m^2/s]. Again, it is assumed that the temperature along the pipe wall is constant and that the point of interest is far from the entrance of the pipe. This equation can be rewritten using dimensionless parameters as follows:

$$Nu_D = 0.023(Re_D)^{0.8}(Pr)^{0.4}$$

Equation 13 (Mills, 1999)

where Nu_D is the Nusselt number and Pr is the Prandtl number defined as:

$$Nu_D = \frac{h_c L_D}{k}$$

Equation 14 (Mills, 1999)

and

$$Pr = \frac{c_p \mu}{k}$$

Equation 15 (Mills, 1999)

respectively, where μ is the dynamic viscosity [kg/m-s].

Initially, the convective heat transfer coefficient for turbulent flow is calculated using the above equation; however, the equation is not valid for Re_D within the transition zone and only provides an approximation for the convective heat transfer coefficient. Once the pipe diameter and velocity have been estimated, a more refined approximation of the convective heat transfer coefficient can be obtained using Gnielinski's formula:

$$Nu_D = \frac{\left(\frac{f}{8}\right)(Re_D - 1000)Pr}{1 + 12.7\left(\frac{f}{8}\right)^{\frac{1}{2}}\left(Pr^{\frac{2}{3}} - 1\right)}$$

Equation 16 (Mills, 1999)

This equation provides a more accurate value for the convective heat transfer coefficient, and is valid for thermally fully developed flow with $Pr > 0.5$ and $3{,}000 < Re_D < 10^6$, although there is greater uncertainty with $Re_D < 10^4$ due to intermittent turbulence with error reaching up to 20% (Mills, 1999).

2.1.5 Assumptions

Some assumptions were made in order to simplify the equations involved in determining the necessary pipe diameter and fluid velocity within the pipe. This included:

- A constant temperature of 6.6°C was assumed along the length of the supply header and at the inlet of each branch during the first iteration of computation involving pipe sizing and determination of head loss. However, the second iteration did not include this assumption, with the calculated head loss from the first iteration used in determining the associated inlet temperatures for each branch. These inlet temperatures were subsequently used in resizing the various branch diameters and header diameter.
- The effect of radiation is negligible.
- The effect of natural convection is negligible.
- The temperature at a particular length of piping is only dependent on the radial component, r.
- The liquid is incompressible, with a constant ρ (during operation of the chilled water system).
- Changes in fluid properties are negligible, including: k, ν, and c_p (during operation of the chilled water system).
- Representative values for valve loss coefficient were chosen for gate, globe and check valves when loss coefficients were not known.
- Only gate, globe and check valves were modeled within the CSDT.
- The equations provided by Churchill and Berstein were assumed adequate in calculating the average Nusselt number for heat exchangers (with the exception of flat plate heat exchangers) that did not have an associated heat transfer coefficient within the CSDT library. The equations do not take into account specific arrangement of the cylindrical tubes or fin geometry, if present.
- The radius of curvature for pipe bends was assumed to be three times the inner pipe diameter. This value can be modified by the user within the CSDT.
- The radius of pipe entrance/exit curvature was assumed to be 0.1 times the inner branch pipe diameter. This value can be modified by the user within the CSDT.

2.2 Pipe Characteristics

As mentioned earlier, the pipe material plays a role in the heat transfer from the heat source to the heat sink. The two types of piping material used include: copper-nickel alloy 90-10 (copper alloy number 715) and copper-nickel alloy 70-30 (copper alloy number 706). The thermal conductivity of copper-nickel alloys range from 10-50 W/m-K with copper alloy number 715 having a thermal conductivity of 50 W/m-K and copper alloy number 706 having a thermal conductivity of 29 W/m-K (Copper Development Association, Inc., 2012).

As specified in MIL-T-16420K, there are specific tube diameters and thicknesses used aboard naval vessels. These thicknesses depend on the copper alloy number and the class to which the pipe belongs.

There are six classes covered in the document, of which five are discussed here. They include: Class 200, Class 700, Class 1650, Class 3300, and Class 6000. The Class denotes the maximum working pressure in lb/in^2. Below, Table 2 summarizes the various diameters and thicknesses of pipes for each class of copper alloy number 715 tube.

Outside Diameter	Class 200		Class 700		Class 1650		Class 3300		Class 6000	
	Thickness	Wt/ft	Thickness	Wt/ft	Thickness	Wt/ft	Thickness	Wt/ft	Thickness	Wt/ft
in.	in	lbs	in	lbs	in	lbs	in	lbs	in	lbs
0.125	-	-	-	-	-	-	0.028	0.033	0.028	0.033
0.250	0.035	0.092	-	-	-	-	0.035	0.092	0.058	0.136
0.375	-	-	-	-	-	-	0.049	0.194	0.083	0.295
0.405	-	-	-	-	-	-	0.058	0.245	0.095	0.359
0.500	0.035	0.198	0.065	0.344	0.035	0.198	0.072	0.375	0.120	0.555
0.540	0.065	0.376	0.065	0.376	0.042	0.255	0.072	0.410	0.120	0.614
0.675	0.065	0.483	0.072	0.529	0.049	0.373	0.095	0.671	0.148	0.950
0.750	-	-	-	-	0.058	0.489	0.109	0.851	0.165	1.18
0.840	0.065	0.614	0.072	0.673	0.058	0.552	0.120	1.05	0.203	1.57
1.000	-	-	-	-	0.072	0.814	0.134	1.41	0.220	2.09
1.050	0.065	0.780	0.083	0.977	0.083	0.977	0.148	1.63	0.238	2.35
1.250	-	-	-	-	0.095	1.34	0.165	2.18	0.284	3.34
1.315	0.065	0.990	0.095	1.41	0.095	1.41	0.180	2.49	0.300	3.71
1.500	-	-	-	-	0.109	1.85	0.203	3.21	0.340	4.80
1.660	0.072	1.39	0.095	1.81	0.120	2.25	0.220	3.86	0.380	5.92
1.900	0.072	0.16	0.109	2.38	0.134	2.88	0.250	5.02	0.425	7.63
2.000	-	-	-	-	0.148	3.34	0.284	5.93	0.454	8.55
2.375	0.083	2.32	0.120	3.30	0.165	4.44	0.340	8.43	0.520	11.7
2.500	-	-	-	-	0.180	5.09	0.340	8.94	0.547	13.0
2.875	0.083	2.82	0.134	4.47	0.203	6.60	0.380	11.5	0.630	17.2
3.500	0.095	3.94	0.165	6.70	0.250	9.89	0.458	17.0	0.760	25.3
4.000	0.095	4.52	0.180	8.37	0.284	12.8	-	-	-	-
4.500	0.109	5.83	0.203	10.6	0.340	17.2	-	-	-	-
5.000	0.120	7.13	0.203	11.9	0.380	21.4	-	-	-	-
5.563	0.125	8.28	0.220	14.1	0.425	26.6	-	-	-	-
6.625	0.134	10.6	0.259	20.1	0.457	34.3	-	-	-	-
7.625	0.134	12.2	0.284	25.4	0.526	45.5	-	-	-	-
8.625	0.148	15.3	0.340	34.3	0.595	58.2	-	-	-	-
9.625	0.187	21.5	0.340	38.4	0.664	72.5	-	-	-	-
10.750	0.187	24.1	0.380	48.0	0.741	90.3	-	-	-	-
12.750	0.250	38.1	0.454	68.0	0.879	127	-	-	-	-
14.000	-	-	0.473	77.9	-	-	-	-	-	-
15.000	-	-	0.503	88.8	-	-	-	-	-	-
16.000	-	-	0.534	101	-	-	-	-	-	-

Table 2: Dimensions and weights of copper alloy number 715 tube (MIL-T-16420K-1, 1978)

Table 3 summarizes the various diameters and thicknesses of pipes for class 200 copper alloy number 706 tube.

Outside diameter	Class 200	
	Wall thickness	Wt/ft
Inches	in	lbs
0.250	0.035	0.092
0.500	0.035	0.198
0.540	0.065	0.376
0.675	0.065	0.483
0.840	0.065	0.613
1.050	0.065	0.779
1.315	0.065	0.989
1.660	0.072	1.39
1.900	0.072	1.60
2.375	0.083	2.32
2.875	0.083	2.82
3.500	0.095	3.94
4.000	0.095	4.51
4.500	0.109	5.83
5.000	0.120	7.12
5.563	0.125	8.28
6.625	0.134	10.6
7.625	0.140	12.2
8.625	0.151	15.3
9.625	0.187	21.5
10.750	0.187	24.0
12.750	0.250	38.0

Table 3: Dimensions and weights of copper alloy number 706 tube (MIL-T-16420K-1, 1978)

2.3 Flow Network Analysis

Flow network analysis is a method that can be used to determine the velocities at every location of a pipe network simultaneously. It is important to use flow network analysis because each component of the network depends on every other component of the network. Branch velocities cannot be accurately solved in isolation. The analogy to flow network analysis would be solving for currents and voltages in an electrical circuit using Kirchoff's current law (KCL) and Kirchoff's voltage law (KVL).

The chilled water piping system is a network of interconnected pipes. The system is composed of two different pipe types, the header and branch pipes. The header pipes branch out into parallel segments which are the portions that come in contact with the heat sources. Each branch will vary in diameter, length and other characteristics such as bends, tees, and valves which will affect the mass flow rate within that branch, and ultimately the mass flow rate in the header piping. The fundamental equations that govern how fluid will flow within the network of pipes are based on the conservation of mass, momentum and energy.

Intuitively, the mass flow rate at the inlet of a branch segment is equal to the mass flow rate at the outlet of the segment (conservation of mass), and:

$$\dot{m} = \rho A V$$

Equation 17 (Rennels & Hudson, 2012)

where $\dot{m}$ is the mass flow rate [kg/s], ρ is the density of the fluid [kg/m^3], A is the cross-sectional area of the pipe [m^2] and V is the average velocity of the fluid [m/s]. Therefore:

$$(\rho AV)_1 = (\rho AV)_2$$

Equation 18 (Rennels & Hudson, 2012)

where 1 denotes the branch pipe inlet and 2 denotes the branch pipe outlet.

Conservation of momentum states that the sum of the forces acting on a control volume is equal to the change in momentum of the fluid. This can be shown in the equation below for a pipe with flow along the x-axis.

$$F_x = (PA)_1 - (PA)_2 + \dot{m}(V_1 - V_2)_x$$

Equation 19 (Rennels & Hudson, 2012)

where F_x is the apparent force acting on the control volume due to frictional resistance and/or difference in pressure across the control volume along the x-axis.

Lastly, the conservation of energy is used to derive the general energy equation. Neglecting forms of energy such as electrical, atomic or chemical, which are not germane to the flow problem pertaining to the chilled water system, the general energy equation takes the form:

$$\frac{P_1}{\rho_1 g} + \frac{\varphi_1 V_1^2}{2g} + Z_1 + \frac{JU_1}{g} + \frac{JQ_1}{\dot{m}g} + \frac{E_p}{\dot{m}g} = \frac{P_2}{\rho_2 g} + \frac{\varphi_2 V_2^2}{2g} + Z_2 + \frac{JU_2}{g} + \frac{JQ_2}{\dot{m}g} + \frac{E_T}{\dot{m}g}$$

Equation 20 (Rennels & Hudson, 2012)

where P is the pressure [N/m],g is the acceleration due to gravity [m/s^2], φ is the kinetic energy correction factor, Z is the relative height with respect to some reference height [m], J is a conversion factor used to convert heat units to specific work units [N-m/kcal], Q is heat flux [kcal/s], E_p is the mechanical work done on the fluid by a pump [N-m/s], and E_T is the work done by the fluid on a turbine [N-m/s]. Some of these parameters are not relevant to the chilled water system, such as E_T, but are included above for completeness

Even though there are great temperature differences from the heat source to the bulk fluid, within the closed system of the chilled water, the temperature differences are within a few degrees. This does not contribute significantly to changes in density (pressure changes also have little impact on the density of the chilled water); therefore, the above equation can be simplified for the case of the chilled water system to the equation below:

$$\left(\frac{P_1 - P_2}{\rho g}\right) + \left(\frac{\varphi_1 V_1^2 - \varphi_2 V_2^2}{2g}\right) + (Z_1 - Z_2) + \frac{E_p}{\dot{m}g} = H_L$$

Equation 21 (Rennels & Hudson, 2012)

where H_L is head loss [m]. Two main sources of head loss include: losses due to surface friction and losses due to induced turbulence. Whenever two mediums are in direct contact with one another and have a net difference in velocity, surface friction will be present. Flow regime plays a significant role in determining the head loss attributed to surface friction. The Hagen-Poiseuille law can be used to determine head loss due to surface friction for laminar flow. The Darcy-Weisbach equation can be used to calculate head loss for turbulent flow. The Hagen-Poiseuille law is shown below:

$$H_L = \frac{32\mu LV}{D^2\rho g}$$

Equation 22 (Rennels & Hudson, 2012)

The Darcy-Weisbach equation is shown below:

$$H_L = f\frac{LV^2}{2Dg} = K\frac{V^2}{2g}$$

Equation 23 (Rennels & Hudson, 2012)

where K is the loss coefficient (dimensionless) and is defined as:

$$K = f\frac{L}{D}$$

Equation 24 (Rennels & Hudson, 2012)

The loss coefficient can be found for any component that contributes to head loss. Examples of these include: pipe bends, valves, pipe expansions, pipe contractions, pipe orifices, pipe entrances, pipe exits and the intersection of pipes that form a tee. These pipe elements will be discussed in greater detail in the proceeding sections. The Darcy friction factor can easily be determined for laminar flow by combining the Hagen-Poiseuille law and the Darcy-Weisbach equation to obtain:

$$f = \frac{64}{Re}$$

Equation 25 (Rennels & Hudson, 2012)

For turbulent flow, the Colebrook-White equation can be used, which is valid even in the transition zone (2,100 < Re < 5000). The use of the Colebrook-White equation lends itself better to a computer program than does the Moody chart, which provides a visual representation of the equation to determine the Darcy friction factor. The Colebrook-White equation is shown below:

$$f = \left(-2\log\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right)\right)^{-2}$$

Equation 26 (Rennels & Hudson, 2012)

This requires an iterative approach as can be seen in the equation. An initial guess of f = 0.02 is assumed and plugged into the equation. This process is repeated 2-3 times with the Darcy friction factor converging quickly. For Cu-Ni alloy pipes, the surface roughness (ε) is 0.05mm (Norsok Standard Fifth Edition, 2006).

The second contributor to head loss is induced turbulence. The Borda-Carnot equation can be used to determine the head loss due to induced turbulence caused by a sudden expansion in pipe diameter. The Borda-Carnot equation is:

$$H_L = \frac{V_1^2}{2g}\left(1 - \frac{A_1}{A_2}\right)^2$$

Equation 27 (Rennels & Hudson, 2012)

Using the equations described above, the flow network of the chilled water system can be analyzed. Network analysis can be divided into three types of flow: series flow, parallel flow and branch flow. The next three sections explain each of these flows in greater detail.

2.3.1 Series Flow

Series flow takes in to account several elements of a pipe that are aligned with one another such that the mass flow rates of each element are equal. An example of this would be a straight pipe connected to a gate valve followed by a segment of straight pipe, a 90° bend, straight pipe, a flow reducer, and a last segment of straight pipe. For this case, all elements are in series with one another with the outlet of one segment connected to the inlet of the following element. With the exclusion of a pump, there will be a pressure drop along the length of the pipe[5], with each element contributing to the overall loss of pressure due to the associated surface friction losses and induced turbulence losses. Since the overall pressure loss is the sum of the individual pressure losses, the loss coefficients of the elements can be summed together as long as the cross-sectional area of each component is factored in. The overall head loss for series flow with N elements is:

$$(H_L)_{Oa} = \frac{\dot{m}^2}{2g\rho^2}\sum_{i=1}^{N}\frac{K_i}{A_i^2}$$

Equation 28 (Rennels & Hudson, 2012)

In addition, the overall pressure loss can be found using:

$$(\Delta P)_{Oa} = \frac{\dot{m}^2}{2g\rho}\sum_{i=1}^{N}\frac{K_i}{A_i^2}$$

Equation 29 (Rennels & Hudson, 2012)

[5] This will not always be the case. It is possible for the pressure along a length of pipe to go up due to the decrease in velocity. The pressure will go up if the velocity head which is converted to pressure head is greater than the pressure drop associated with friction along the pipe length.

2.3.2 Parallel Flow

Parallel flow pertains to flow coming from a central source which diverges into two or more paths and then converges back somewhere downstream. Applying the conservation of energy and the conservation of mass principles, the following equation can be found:

$$\left(\frac{K}{A^2}\right)_{Oa} = \left[\sum_{i=1}^{N} \left(\frac{K}{A^2}\right)_i^{-0.5}\right]^{-2}$$

Equation 30 (Rennels & Hudson, 2012)

Afterwards, the solution to this equation can be inserted into the following equation to solve for the individual mass flow rates for the parallel branches:

$$\dot{m}_i = \dot{m}_{Total} \sqrt{\frac{\left(\frac{K}{A^2}\right)_{Oa}}{\left(\frac{K}{A^2}\right)_i}}$$

Equation 31 (Rennels & Hudson, 2012)

where the total mass flow rate is equal to the sum of the individual mass flow rates. Hence:

$$\dot{m}_{Total} = \sum_{i=1}^{N} \dot{m}_i$$

Equation 32 (Rennels & Hudson, 2012)

2.3.3 Branch Flow

Branch flow is the combination of series flow and parallel flow, but may be more complicated since the parallel branches do not necessarily converge downstream. However, for chilled water systems, the branches do converge into the header pipe, and thus, the application of the equations for series flow and parallel flow will suffice for solving the branch flow problem that this particular system presents.

An example of a branch flow network can be modeled using an electrical circuit analogy. Figure 10 shows a diagram of a segment of a cooling system. With the various sources of head loss modeled as a resistive component, the flow through the various pipe branches can be determined given a differential pressure or an inlet mass flow rate. Figure 11 shows an electrical circuit analogy to the chilled water system diagram.

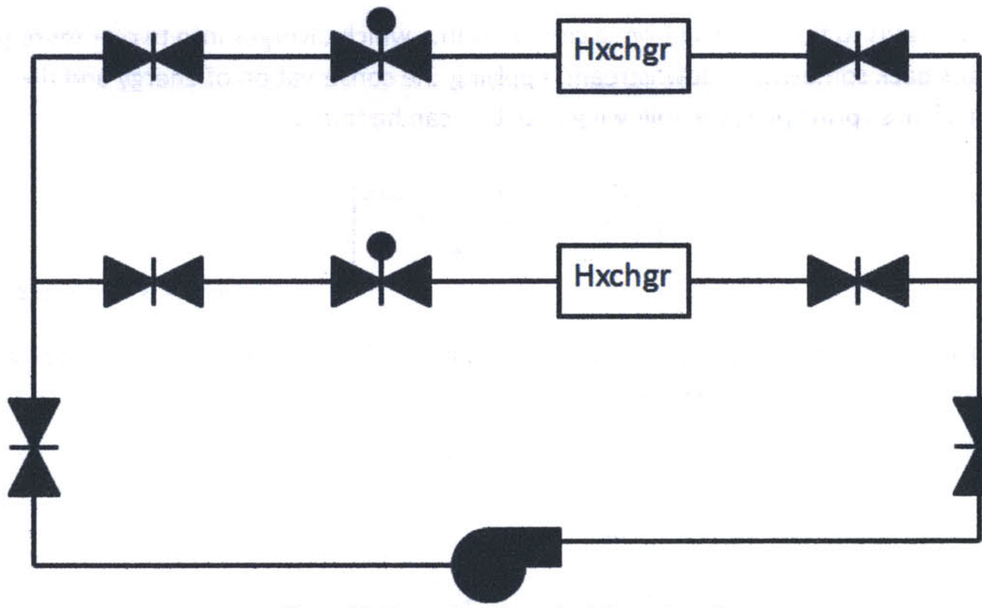


Figure 10: Example of branch piping network

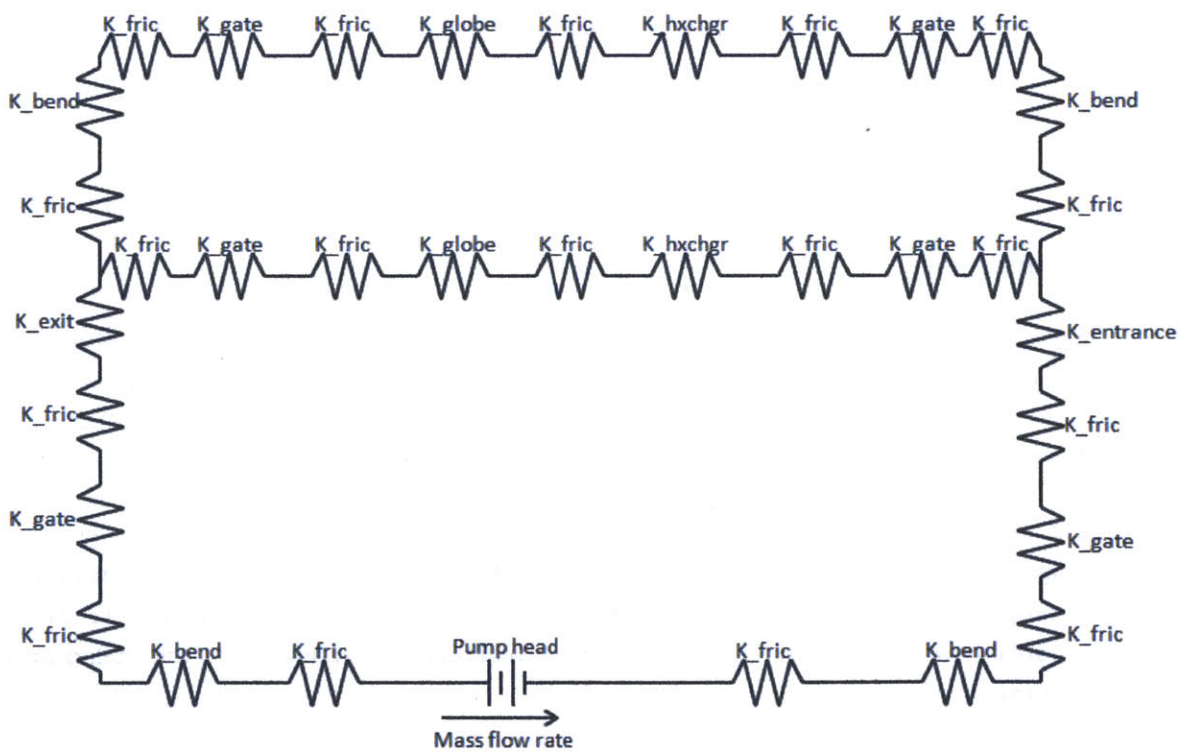


Figure 11: Electrical network analogy to branch piping network

2.4 Pump Characteristics

The chilled water system has a circulating pump, typically a motor driven centrifugal pump, which provides the pump head needed to circulate the fluid within the chilled water system at the necessary flow rate. The sizing of these pumps depends on three factors: the required pump capacity, the pump head and the operating speed of the pump. An example of a centrifugal pump is shown in Figure 12 below.

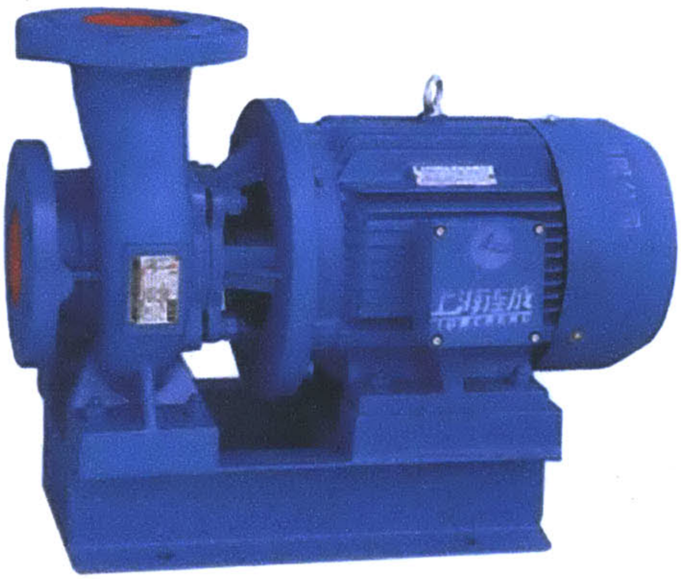


Figure 12: Example of a centrifugal pump (ThomasNet, 2013)

2.4.1 System and Pump Curves

To properly size a pump, the system curve of the pump and the pump curve must be considered. The system curve shows the system head as a function of flow rate and is comprised of the static head in the system and the head loss associated with major and minor losses. Figure 13 below shows an example of the system curve along with how the curve shifts with changes in head loss (e.g., shutting or opening valves) (System Curve and Pump Performance Curve).

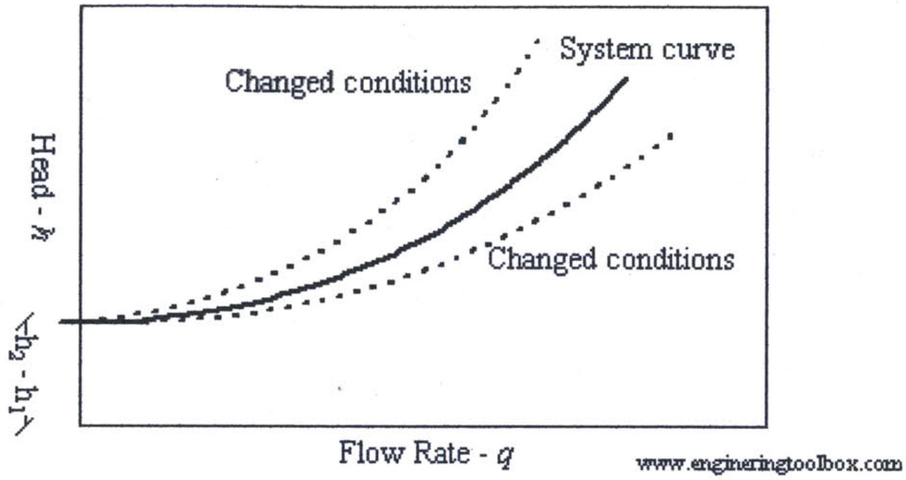


Figure 13: System curve (System Curve and Pump Performance Curve)

The pump performance curve depends on the specific pump considered and provides the head of the pump as a function of flow rate. An example of pump performance curves for a pump with impeller diameters of 6 in, 8 in, and 10 in is shown in Figure 14 below (System Curve and Pump Performance Curve).

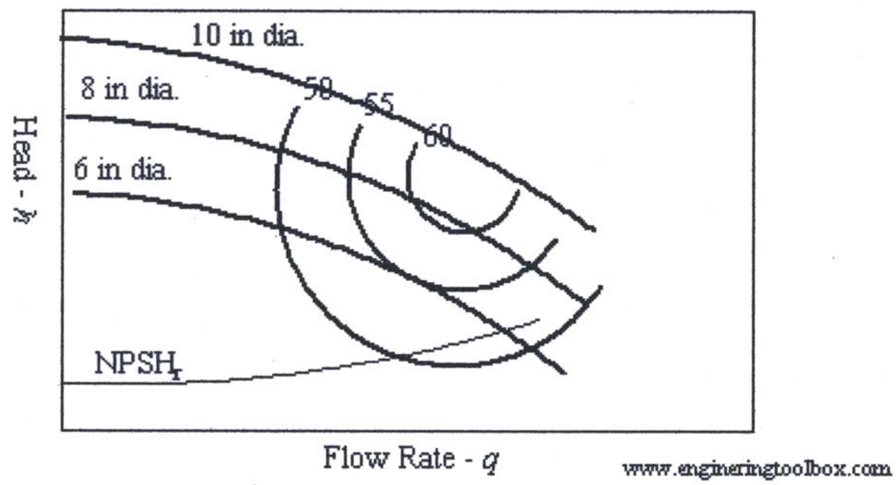


Figure 14: Pump performance curve (System Curve and Pump Performance Curve)

Superimposing the system curve and the pump curve will yield the operating point, the point at which the two curves intersect. The operating point specifies the head in the system along with the flow rate which will be expected for that specific system and selected pump. Figure 15 below shows an example of the operating point (System Curve and Pump Performance Curve).

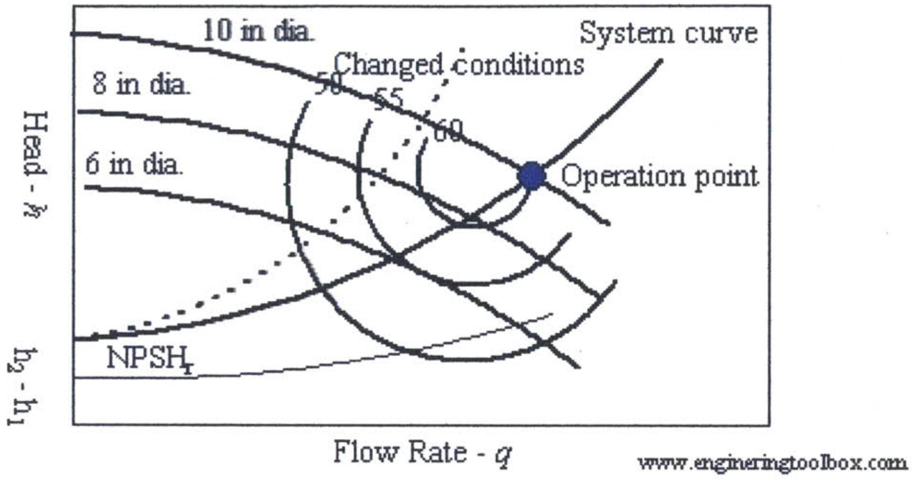


Figure 15: Operating point (System Curve and Pump Performance Curve)

Typically, the pump selected should have the operating point coincide with the best efficiency point (BEP) (System Curve and Pump Performance Curve).

2.4.2 Head loss

The head loss accounts for friction losses, load losses and regulating fitting losses. The total pump head can be found using the equation:

$$H_P = HL_F + HL_L + HL_{RF}$$

Equation 33

The pump capacity was found by determining the mass flow rates through each branch and the subsequent mass flow rate through the supply header. This is a somewhat complex process utilizing flow network analysis and dependent on the heat loads and the electronic component heat exchanger geometry.

As stated in Section 2.1.5, it was assumed that the temperature inlet for each branch did not vary and was equal to the inlet temperature of the supply header of 6.6°C. This assumption was validated by calculating the associated temperature rise along the length of the supply header due to head loss. The equation for the temperature rise ΔT due to head loss is shown below.

$$\Delta T = \frac{H_L}{C_1 c_p}$$

Equation 34 (Rennels & Hudson, 2012)

where C_1 is a conversion factor equal to 778.169262 [ft-lbf/Btu]. A simulation was conducted that contained 180 heat loads with a branch for each load. The branch pipe diameter was calculated, along with the header pipe diameter and various flow velocities through the header and each branch. The greatest rise in temperature would be seen in the branch furthest downstream. The rise in temperature along the length of the supply header was on the order of 10^{-5} °C. This is due to the relatively low velocities encountered within the chilled water system. Appreciable rises in temperature due to head loss is not seen until velocities approach sonic speeds. Therefore, neglecting the rise in temperature associated with head loss is reasonable. The heating up of the chilled water due to the environment is of greater concern with temperature rises on the order of 10^{-3} °C calculated.

2.4.3 Pump Selection

Due to the endless supply of pumps available, the approach used within CSDT v1.0 was also used, considering the 1510 series pump manufactured by Bell & Gossett (Fiedel, 2011). The 1510 series pumps which operate at 60 Hz can be operated at slow, medium, and high speed with speeds of 1150 rpm, 1750 rpm and 3500 rpm, respectively (Bell & Gossett, 1998). Figure 16 shows the envelope of operation for the 1510 series pumps based on speed.

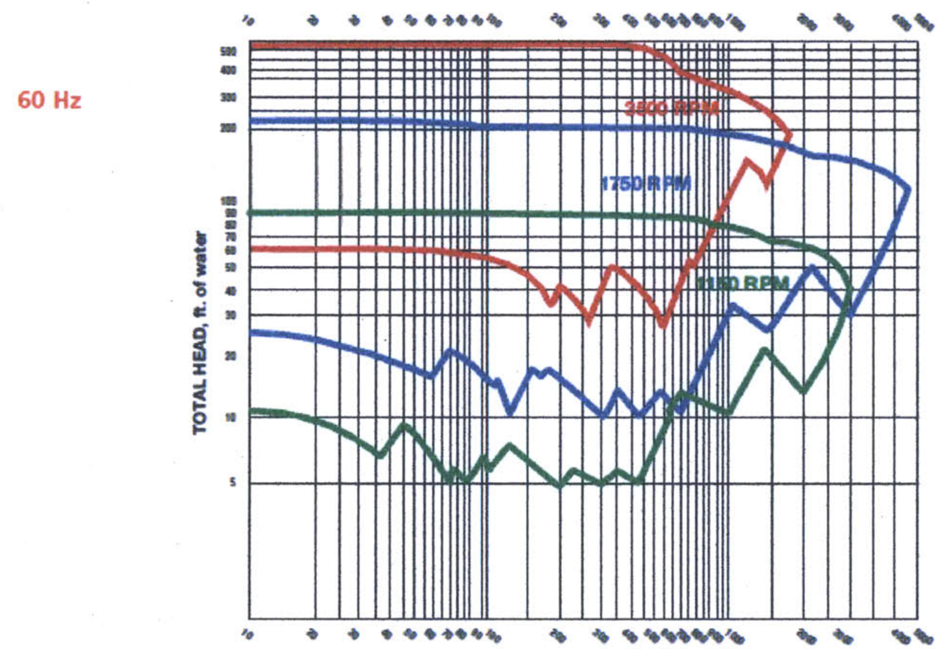


Figure 16: Envelope of operation of Bell & Gossett 1510 series pump based on pump speed (Bell & Gossett, 1998)

The CSDT only considers Bell & Gossett 1510 series pumps operating at 1750 rpm. The 1510 series performance curves operating at 1750 rpm is shown in Figure 17 below.

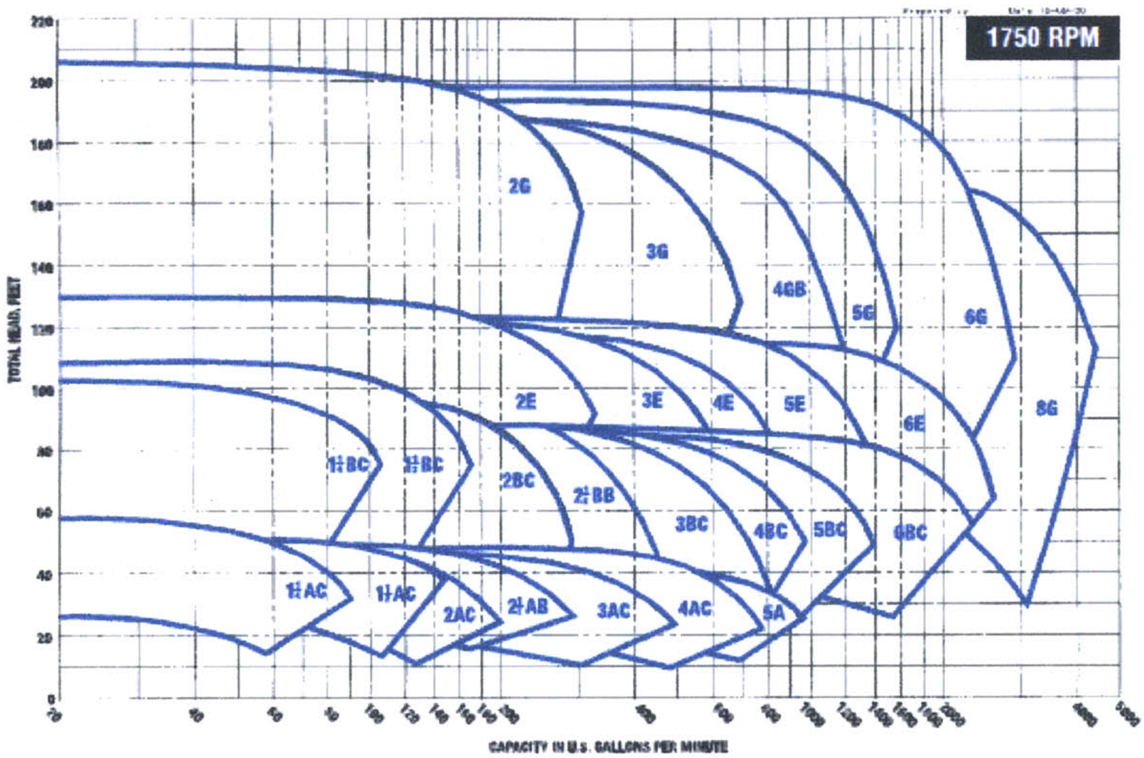


Figure 17: Bell & Gossett 1510 series performance curves operating at 1750 rpm (Bell & Gossett, 1998)

The pump selection process begins with the head loss of the system for a specific A/C unit line-up and operating condition (e.g., shore, design, cruise, battle). The mass flow rate can also be found based on the specific A/C unit line-up and operating condition. With this information, the intersection of head and mass flow rate yields the optimal pump for that A/C unit configuration.

A difficulty arises in that the head loss of the system and the requisite mass flow rate differs depending on the A/C unit line-up and operating condition. To select the pump, the design condition is used, but with many different options available for A/C unit line-up, there may be different optimal pumps considered. A solution to this problem may be the selection of a variable speed pump which operates efficiently at different speeds depending on the A/C unit line-up. A second solution may be selecting a pump with a high efficiency over a wide range of mass flow rates and heads.

For the development of the CSDT several points follow:

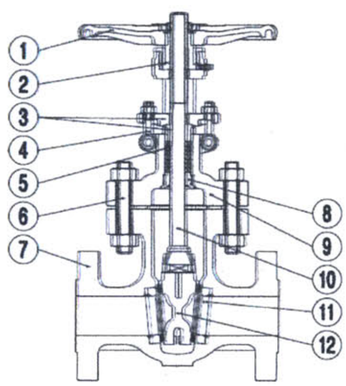
- The pump selected provides a solution but does not guarantee the optimal solution.
- Only pumps of the Bell & Gossett 1510 series were considered. Other manufacturers and series would provide greater available options for pump selection.
- Impeller diameters were not considered.
- An average weight of 1200 kg was used for all pumps selected.

2.5 Valve Characteristics

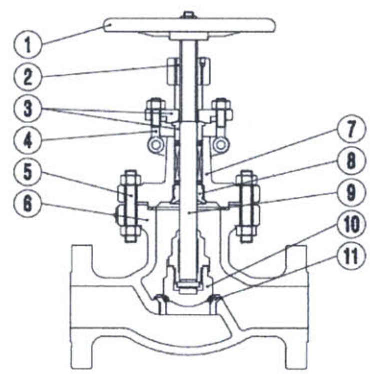
Valves are used for a variety of reasons. Some are used for isolating a segment of the system such as a gate valve. Others are used for controlling the flow through the system such as a control valve or a globe valve. Yet, others are used for ensuring flow in a specific direction such as a check valve.

It is assumed that there is a gate valve at either end of the branch for branch isolation. Also, a control valve is assumed to be at the outlet of each branch to control the flow depending on temperature. For the header branch, it is assumed there is a gate valve on the supply header and on the return header. Lastly, it is assumed there is a check valve downstream of each chilled water pump.

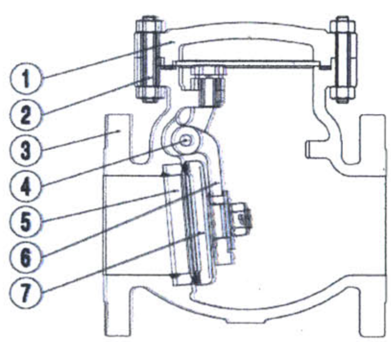
Since valve geometry and size vary greatly, there is no explicit formula that can be used to calculate the loss coefficient of the specific valve accurately. The pressure drop must be specified by the manufacturer and included as an input into the CSDT program. Schematics of a gate valve and a globe valve can be seen in Figure 18. As can be seen in the schematics, the flow path is much more tortuous for the globe valve, resulting in a higher loss coefficient and greater head loss. If no manufacturer data is available for the specific valve used in the chilled water system, a nominal value for the valve loss coefficient was used. The nominal values chosen for the valve loss coefficients can be seen in Table 4.



IT.	DESCRIPTION
1	HANDWHEEL
2	YOKE SLEEVE
3	GLAND FLANGE
4	GLAND EYE BOLT
5	PACKING
6	BONNET BOLT
7	BODY
8	BONNET
9	BACK SEAT
10	STEM
11	SEAT RING
12	WEDGE



IT.	DESCRIPTION
1	HANDWHEEL
2	YOKE SLEEVE
3	GLAND FLANGE
4	GLAND EYE BOLT
5	BONNET BOLT
6	BODY
7	BONNET
8	BACK SEAT
9	STEM
10	DISC
11	SEAT RING



IT.	DESCRIPTION
1	COVER
2	BOLT
3	BODY
4	HINGE PIN
5	SEAT RING
6	HINGE
7	DISC

Figure 18: Schematics of gate valve, globe valve and check valve (Bonney Forge, 2012)

Valve Type	Notional Loss Coefficient Value (dimensionless)
Gate – Full Port	0.2
Globe – Standard	3.5
Globe – Angle	4
Check – Swing	1.5

Table 4: Notional valve loss coefficient values (Rennels & Hudson, 2012)

2.6 Flow Configurations

As mentioned earlier, the specific elements of the flow need to be taken into account as they all contribute to the pressure drop across the pipe. Specifically, pipe bends and tees contribute to the head loss within the chilled water system.

2.6.1 Bends

Bends in pipes contribute to the head loss that takes place within the chilled water system. For the design of the chilled water system, it was assumed that all bends constituted a 90° angle and that all bends were smooth.

An empirical equation used to calculate the loss coefficient due to a bend in a pipe was used. The equation is:

$$K = f\alpha\frac{r}{d} + (0.10 + 2.4f)sin\left(\frac{\alpha}{2}\right) + \frac{6.6f\left[\sqrt{sin\left(\frac{\alpha}{2}\right)} + sin\left(\frac{\alpha}{2}\right)\right]}{\left(\frac{r}{d}\right)^{\frac{4\alpha}{\pi}}}$$

Equation 35 (Rennels & Hudson, 2012)

where α is the bend angle in radians (0-π), r is the radius of curvature of the pipe measured from the centerline of the pipe [m], and d is the pipe diameter [m]. This equation is valid for smooth pipe bends. The loss coefficient for miter bends can be computed using a different empirical equation, but was not considered in the CSDT.

A picture of a smooth, circular bend and a miter bend is shown below in Figure 19.

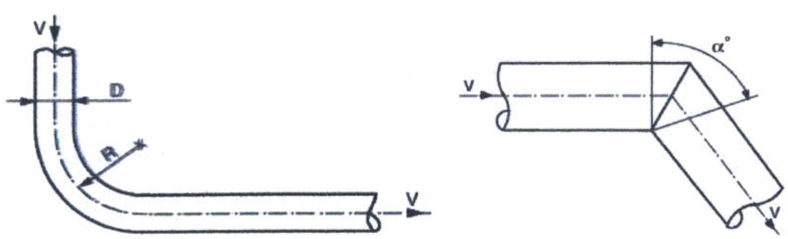


Figure 19: Figure of smooth, circular bend and miter bend (Cross-Flooding area, 2004)

In the design of the CSDT, it was assumed that the spacing between the bends were sufficiently long such that coupling effects can be ignored. In addition, the radius of curvature was assumed to be equal to 3D, or 3 times the pipe diameter. In industry, the radius of curvature varies from a short bend (one pipe diameter), to a long bend (1.5 times the pipe diameter), to a bend that is 3, 5 or 10 times the diameter (3D, 5D and 10D respectively). However, according to MIL-STD-1627B(SH), the minimum bend radius allowed within piping systems is 2D, thus short bends and long bends are not allowed without special permission (MIL-STD-1627B(SH), 1981). The default value of 3D within the CSDT can be modified by the user to other values such as 2D, 5D or 10D.

2.6.2 Tees

An important source of head loss in the chilled water system is the convergence and divergence of flow. The most common angle of convergence and divergence is 90°, forming a T shape, i.e. tee. The four specific types of tee configurations used within the chilled water system are: the divergence of flow through the header, the divergence of flow through the branch, the convergence of flow through the header, and the convergence of flow through the branch. Figure 20 shows the four configurations of converging and diverging flow.

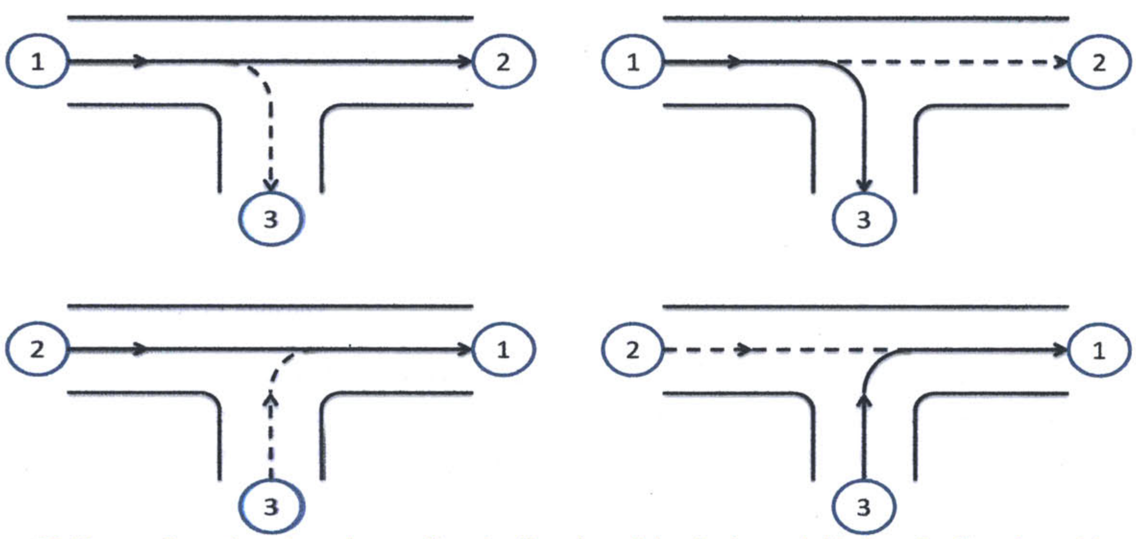


Figure 20: Flow configurations through tees: diverging flow through header (upper left), diverging flow through branch (upper right), converging flow through header (lower left), converging flow through branch (lower right) (Rennels & Hudson, 2012)

Entrance effects cause disruption in flow and tend to increase the rate of heat transfer at localized areas. For turbulent flow, fully developed hydrodynamic flow can exist 10-15 pipe diameters from the entrance of the pipe assuming no large scale eddies are present. The hydrodynamic entrance length[6] (L_{ef}) may be as high as 20-40 pipe diameters if large scale eddies are present. The thermal entrance

[6] The hydrodynamic entrance length is the distance required for the friction factor (f) to decrease within 5% of the fully developed value of the friction factor (f_∞).

length[7] (L_{eh}) is somewhat lower for high or low Pr, with 5 pipe diameters sufficient for fully developed thermal flow.

For diverging flow through the header, the loss coefficient can be found using the equations below:

$$K_{12_1} = 0.36 - 0.98\frac{\dot{m}_1}{\dot{m}_2} + 0.62\left(\frac{\dot{m}_1}{\dot{m}_2}\right)^2 + 0.03\left(\frac{\dot{m}_2}{\dot{m}_1}\right)^6$$

Equation 36 (Rennels & Hudson, 2012)

and

$$K_{12_2} = 0.62 - 0.98\frac{\dot{m}_1}{\dot{m}_2} + 0.36\left(\frac{\dot{m}_1}{\dot{m}_2}\right)^2 + 0.03\left(\frac{\dot{m}_2}{\dot{m}_1}\right)^6$$

Equation 37 (Rennels & Hudson, 2012)

where K_{12_1} and K_{12_2} are loss coefficients.

For diverging flow through the branch, the loss coefficient can be found using the equations below:

$$K_{13_1} = 1.00 - 1.13\frac{\dot{m}_3}{\dot{m}_1} + \left[0.81 + \left(1.12\frac{d_3}{d_1} - 1.08\frac{d_3^3}{d_1^3} + K_{Eq}\right)\frac{d_1^4}{d_3^4}\right]\frac{\dot{m}_3^2}{\dot{m}_1^2}$$

Equation 38 (Rennels & Hudson, 2012)

and

$$K_{13_3} = \left(0.81 - 1.13\frac{\dot{m}_1}{\dot{m}_3} + \frac{\dot{m}_1^2}{\dot{m}_3^2}\right)\frac{d_3^4}{d_1^4} + 1.12\frac{d_3}{d_1} - 1.08\frac{d_3^3}{d_1^3} + K_{Eq}$$

Equation 39 (Rennels & Hudson, 2012)

where

$$K_{Eq} = 0.57 - 1.07\left(\frac{r}{d_3}\right)^{\frac{1}{2}} - 2.13\left(\frac{r}{d_3}\right) + 8.24\left(\frac{r}{d_3}\right)^{\frac{3}{2}} - 8.48\left(\frac{r}{d_3}\right)^2 + 2.90\left(\frac{r}{d_3}\right)^{\frac{5}{2}}$$

Equation 40 (Rennels & Hudson, 2012)

For converging flow through the header, the loss coefficient can be found using the equation below:

$$K_{21_1} = 1 - 0.95\frac{\dot{m}_2^2}{\dot{m}_1^2} - 2C_{xC}\left(\frac{\dot{m}_2}{\dot{m}_1} - \frac{\dot{m}_2^2}{\dot{m}_1^2}\right) - 2C_M\left(1 - \frac{\dot{m}_2}{\dot{m}_1}\right)$$

Equation 41 (Rennels & Hudson, 2012)

and

[7] The thermal entrance length is the distance required for the Nusselt number to decrease within 5% of Nu_∞.

$$K_{21_2} = \frac{\dot{m}_1^2}{\dot{m}_2^2} - 0.95 - 2C_{xC}\left(\frac{\dot{m}_1}{\dot{m}_2} - 1\right) - 2C_M\left(\frac{\dot{m}_1^2}{\dot{m}_2^2} - \frac{\dot{m}_1}{\dot{m}_2}\right)$$

Equation 42 (Rennels & Hudson, 2012)

where

$$C_M = 0.23 + 1.46\left(\frac{r}{d_3}\right) - 2.75\left(\frac{r}{d_3}\right)^2 + 1.65\left(\frac{r}{d_3}\right)^3$$

Equation 43 (Rennels & Hudson, 2012)

and

$$C_{xC} = 0.08 + 0.56\left(\frac{r}{d_3}\right) - 1.75\left(\frac{r}{d_3}\right)^2 + 1.83\left(\frac{r}{d_3}\right)^3$$

Equation 44 (Rennels & Hudson, 2012)

For converging flow through the branch, the loss coefficient can be found using the equation below:

$$K_{31_1} = -1 + 2(2 - C_{xC} - C_M)\frac{\dot{m}_3}{\dot{m}_1} + \left[(2C_{yC} - 1)\frac{d_1^4}{d_3^4} + 2(C_{xC} - 1)\right]\frac{\dot{m}_3^2}{\dot{m}_1^2}$$

Equation 45 (Rennels & Hudson, 2012)

and

$$K_{31_3} = 2C_{yC} - 1 + \frac{d_3^4}{d_1^4}\left[2(C_{xC} - 1) + 2(2 - C_{xC} - C_M)\frac{\dot{m}_1}{\dot{m}_3} - 0.92\frac{\dot{m}_1^2}{\dot{m}_3^2}\right]$$

Equation 46 (Rennels & Hudson, 2012)

where

$$C_{yC} = 1 - 0.25\left(\frac{d_3}{d_1}\right)^{1.3} - \left[0.11\left(\frac{r}{d_3}\right) - 0.65\left(\frac{r}{d_3}\right)^2 + 0.83\left(\frac{r}{d_3}\right)^3\right]\frac{d_3^2}{d_1^2}$$

Equation 47 (Rennels & Hudson, 2012)

2.7 Expansion Tank

During normal operation of the chilled water system, the rise in temperature across the system is very small, on the order of 5-10°C. This will not result in an appreciable increase in volume due to changes in density; however, there would be an appreciable increase in volume due to a rise in temperature if the system is not in operation and the temperature within the pipes rises to ambient temperatures, or worse yet, if the heat loads are still present, causing even greater rises in temperature of the chilled water. This volume expansion is accounted for through the use of an expansion tank. The expansion tank is connected to the chilled water system through the return header and can be isolated by use of a gate isolation valve. Each A/C unit-chilled water pump combination must have its own expansion tank.

The expansion tank serves several purposes. The first is to serve as an expansion volume to mitigate the effects of pressure due to changes in chilled water temperature. The second purpose of the expansion tank is to collect air entrained in the system. The third purpose of the expansion tank is to provide a source of makeup water to replace water lost due to leaks within the system. Lastly, the expansion tank is to provide some predetermined pumping capacity for the chilled water pump. To determine the operating water capacity of the expansion tank, we multiply an assumed time by the pump flow rate:

$$V_O = t_r Q_{CW}$$

Equation 48

where V_O is the operating water capacity of the expansion tank [gal], t_r is the assumed duration of time the expansion tank is required to supply water to the chilled water pumps [s], and Q_{CW} is the capacity of the pump [gal/min]. The default value for t_r is 30 seconds, but can be changed by the user. The capacity of the pump was determined by the method described in Section 2.4.

To ensure air does not enter the chilled water system with a leak present, the system is operated at a minimum pressure, P_O, of 5 psi under all conditions. To maintain this pressure, the expansion tank must be maintained at a pressure greater than this. The expansion tank charging pressure can be found using the equation:

$$P_C = P_O + \rho_w H_T$$

Equation 49

where P_C is the expansion tank charging pressure [psi] and H_T is the vertical distance between the expansion tank and the highest point [ft].

Including a 10% safety factor, the total expansion tank capacity was determined using the equation:

$$V_{T_1} = 1.1 V_O \left(1 + \frac{P_{ATM}}{P_C}\right)$$

Equation 50

where P_{ATM} is atmospheric pressure [psi].

A second method to compute the expansion tank volume is to determine the volume needed to account for the expanding fluid within the system from a rise in temperature from 32°F to 120°F.

The expanded volume can be calculated fairly easily since the pipe dimensions are known as well as the change in density occurring due to the rise in temperature. The density of pure water at 6.6°C is 999.41 kg/m^3, which is the target temperature within the supply header. A more conservative approach is taken, using pure water at 0°C, which has a density, ρ_c, of 1000 kg/m^3. The assumed temperature rise in sizing the expansion tank is 120°F, which is equal to 48.89°C and has an associated density, ρ_h, of 988.31 kg/m^3. Therefore, the volume expansion due to a rise in temperature from 32°F to 120°F is:

$$V_E = \left(\frac{\rho_c}{\rho_h} - 1\right)(V_P + V_O)$$

Equation 51

where V_P is the volume of water in the piping [gal]. Again, including a 10% safety factor, the total expansion tank volume needed is then found using the equation:

$$V_{T_2} = 1.1(V_E + V_O)$$

Equation 52

The larger of the two values, V_{T_1} or V_{T_2}, is then used as the expansion tank volume.

The thickness of the expansion tank was calculated assuming the pressure vessel is thin-walled. With this assumption the radial stress is negligible in comparison to the tangential stress and the tangential stress can be assumed to be uniform across the wall. Summing the forces and rearranging yields the equation:

$$t = \frac{Pr}{\sigma}$$

Equation 53 (Storage Tank Thickness Determination, 2013)

where P is the design pressure [psi], r is the tank inner radius [m], and $\sigma$is the maximum allowable stress of the material [ksi]. To account for the weld, a weld joint factor is added to the equation. The equation is then:

$$t = \frac{Pr}{\sigma E - 0.1P}$$

Equation 54 (Storage Tank Thickness Determination, 2013)

where E is the weld joint factor. The weld joint factor was assumed to be 1.00 which is a recommended value for butt welds undergoing pressure loading (Conversion Factor of Weld Joint). The design pressure was assumed to be twice that of the operating pressure. With a maximum expected operating pressure of 100 psi, the design pressure is 200 psi. Using stainless steel to construct the tank, an allowable stress of 4900 psi was used. This yields an expansion tank thickness of 0.76 mm for a tank with a radius of 0.4 m. The minimum thickness of the tank was assumed to be the greater of the calculated value or 4 mm.

To calculate the dimensions of the expansion tank (radius and height), the surface area of a right circular cylinder was minimized for a given volume (calculated using the above method) with the ratio of the radius to height is equal to 0.2. Within the CSDT a maximum height of 2 m was allowed. Therefore, if a larger tank was needed, the right circular cylinder would not retain the optimal ratio between radius and height. A single tank with a non-optimal radius-to-height ratio will still have less surface area than multiple tanks with optimal radius-to-height ratios; therefore the program constructs a single expansion tank per A/C unit.

2.8 Heat Exchangers

There are several types of heat exchangers available, but the concept is similar in all cases. The heat exchanger provides a way for heat to be transferred from one medium to another. The various heat exchangers vary based on the geometry of the flow configuration, the type of heat transfer surface and the construction materials. Some of the basic types of heat exchangers include: single stream, two-stream parallel flow, two-stream counter flow, two-stream cross-flow with zero one or both streams either mixed or unmixed, two-stream cross-counter flow, and two-stream multi-pass. Some examples of these heat exchangers can be seen in Figure 21 and Figure 22 below.

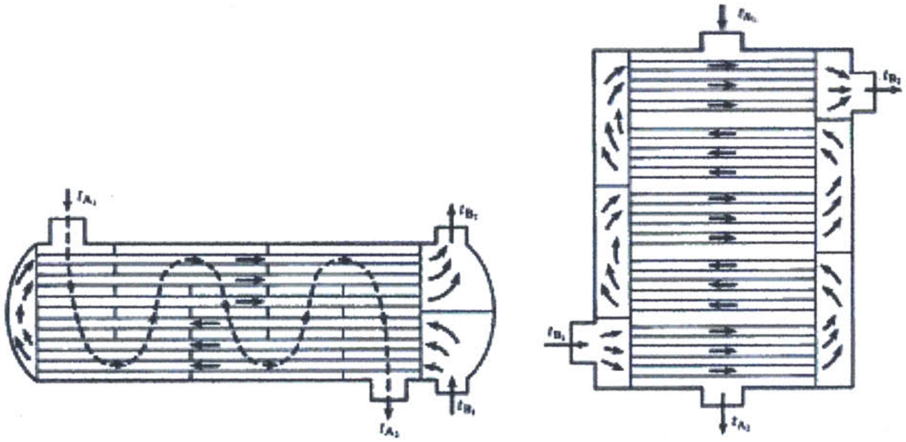


Figure 21: Schematic of multi-pass cross counter flow shell and tube heat exchangers (Adam, 2004)

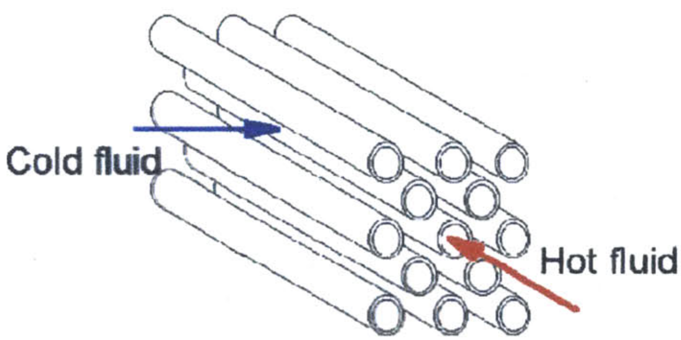


Figure 22: Depiction of flow for two-stream cross-flow (Travkin, 2001)

The type most encountered in chilled water systems is the shell-and tube type heat exchanger which is a two-stream multi-pass configuration. This is a more complicated heat exchanger design than most mentioned above, but is necessary to achieve compactness. The simpler heat exchangers such as a two stream parallel flow would require a very long section of piping in order to achieve the surface area contact between the heat source and heat sink and would not be practical for large heat loads. The shell and tube design heat exchanger provides the tube bundles which have greater surface area and the multiple passes the air flow makes with the tubes allows the heat exchanger the more compact form.

As can be assumed, heat exchanger selection greatly affects the thermal efficiency of the system and greatly contributes to head loss due to the number of bends encountered by the flow, the entrance and exit losses, and the greater surface area within the heat exchanger necessary for greater heat transfer from one medium to the other. Because of these factors, it will be difficult to determine the associated head loss of the heat exchanger and the rate of heat transfer across the heat exchanger based solely on geometry. Therefore, it is crucial that the CSDT have reliable, accurate and complete information pertaining to the parameters associated with head loss and heat transfer for each heat exchanger used within the chilled water system. Otherwise, the accuracy of the CSDT will diminish greatly, but a rough approximation for the rate of heat transfer across the heat exchanger and the head loss attributed to the heat exchanger can be determined using the fundamental equations described above, in particular, the equations used to determine the average Nusselt number given by Churchill and Bernstein. With the average Nusselt number, the average convective heat transfer coefficient and the rate of heat transfer can be determined.

The geometry of the heat exchanger can be very complicated, and the above method will only provide an approximate solution. The tubes of the heat exchanger may be staggered or aligned, which will affect the flow of air that passes external to the tubes. In addition, fins may be present on the outer surface of the tubes in order to increase the surface area in contact with the hotter air. This will affect the convective heat transfer coefficient, but is not considered in the determination of the average Nusselt number which introduces a source of error.

For the electronic cooling water system, there is an interface between the system and the heat sink (either the chilled water system and/or the seawater system). The interface is the heat exchanger between the demineralized water loop and the chilled water and/or seawater loop. The type of heat exchanger typically used for the seawater/demineralized water heat exchanger is a titanium flat plate heat exchanger. The type of heat exchanger typically used for the chilled water/demineralized water heat exchanger is a shell and straight tube heat exchanger with double tube sheet construction. The demineralized water flows through the shell side and the chilled water flows through the tube side. Figure 23 shows a schematic of a flat plate heat exchanger.

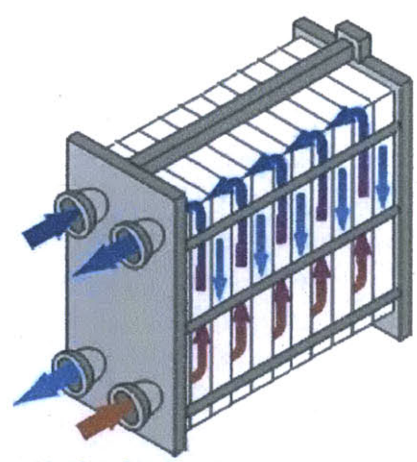


Figure 23: Flat plate heat exchanger (Energy-Film)

2.8.1 Notional Flat Plate Heat Exchanger Design

A notional flat plate heat exchanger is provided in the Excel spreadsheet used in conjunction with the Matlab program. The flat plate heat exchanger was designed starting from the fluid type on the secondary side, along with the expected inlet and outlet temperatures of the secondary fluid. The process in designing the flat plate heat exchanger was based on a similar example found in *Fundamentals of Heat and Mass Transfer 7th ed.* (Incropera & DeWitt, 2002).

The following bullets summarize the assumptions made in designing the notional flat plate heat exchanger:

- The notional flat plate heat exchanger considers demineralized water on the secondary side with an inlet temperature of 30°C and an outlet temperature of 18°C.
- The mass flow rate of the demineralized water was assumed to be 0.5 kg/s. The resulting heat load was calculated to be 25.08 kW.
- The mass flow rate of the chilled water was assumed to be 3.6 gpm/ton, which is equivalently 1.6197 kg/s.
- The inlet chilled water temperature was assumed to be 7.2°C and the outlet chilled water temperature was assumed to be 10.9°C.
- Cross flow was assumed.
- The dimensions of the heat exchanger (length, width, height) were assumed to be identical.
- 60 gaps were assumed within the heat exchanger.
- A plate thickness of 0.5 mm was assumed.

With the inlet and outlet temperatures on the primary side and the secondary side defined, the log mean temperature difference was found using the equation:

$$\Delta T_{log-mean} = \frac{(T_{DW,in} - T_{CW,out}) - (T_{DW,out} - T_{CW,in})}{\ln[(T_{DW,in} - T_{CW,out})/(T_{DW,out} - T_{CW,in})]}$$

Equation 55 (Incropera & DeWitt, 2002)

The log mean temperature was calculated to be 14.5°C.

Assuming fully-developed laminar flow between the heat exchanger plates, the Nusselt number was determined to be (Incropera & DeWitt, 2002):

$$Nu = \frac{h_c D_h}{k} = 7.54$$

Equation 56 (Incropera & DeWitt, 2002)

which is valid for rectangular channels of infinite length (the thickness of the channel is much smaller than the length of the channel) and the surface temperature is uniform.

With this, the convective heat transfer coefficients for the primary and secondary sides were found to be:

$$h_{c_{pri}} = 4.28 \frac{W}{m-K}\frac{N}{L}$$

and

$$h_{c_{pri}} = 4.54 \frac{W}{m-K}\frac{N}{L}$$

where N is the number of gaps, and L is the length of the heat exchanger. The length of the heat exchanger was then computed to be 0.2218 m.

Assuming a plate thickness of 0.5 mm, the gap thickness was found to be 3.2 mm. With copper plates, the dry weight of the heat exchanger was calculated to be 12.75 kg. Assuming a factor of 1.5 for casing, inlet and outlet plenums, the weight was estimated at 19.13 kg. The wet weight accounts for half the gaps filled with chilled water and the other gaps filled with demineralized water. The heat exchanger wet weight was calculated at 28.57 kg.

With a hydraulic diameter of 6.4 mm, the mean chilled water velocity and the mean demineralized water velocity were calculated as 0.0658 m/s and 0.0204 m/s, respectively. The corresponding Reynolds numbers are 336.896 and 135.592, respectively. The assumption that laminar flow existed for the chilled water side and the demineralized water side was valid.

Additional flat plate heat exchangers could be modeled by copying the notional flat plate heat exchanger and modifying the following parameters:

- Secondary fluid
- Secondary fluid specific heat capacity (taken at the mean temperature)
- The design inlet temperature of the secondary fluid
- The mass flow rate of the secondary fluid
- The design outlet temperature of the secondary fluid (this can be calculated and entered if the heat load is known)
- The convective heat transfer coefficient on the secondary side – This will most likely be the most challenging variable to determine. If the flow is laminar and fully developed, then the same approach above using the Nusselt number can be used.
- The number of gaps
- The thermal conductivity of the plates

The calculation of the weight assumes copper as the material used in constructing the plate. If the user wishes to modify this, then the weight will also have to be entered manually along with the thermal conductivity of the plate material.

2.9 Air Conditioning Sizing

The chilled water system provides cooling to the Heating, Ventilation, and Air Conditioning (HVAC) system through the air conditioning cooling coils. To properly size the chilled water system, it is necessary to accurately model the HVAC system and size the air conditioning cooling coils. Similar to breaking up the ship's chilled water plants into zones, the HVAC system is also broken up into zones.

2.9.1 Air Conditioning Cooling Coils

Two typical air conditioning cooling coil configurations used aboard older U.S. Navy ships are the double-serpentine coils and the single-serpentine coils. The differences between the two configurations are the number of passes and circuits in each type of cooling coil. The single serpentine cooling coil has the same number of rows and the same number of tubes per row, but has half the number of circuits and twice the number of passes per circuit than the double serpentine cooling coil. The two air conditioning coil configurations are shown below in Figure 24.

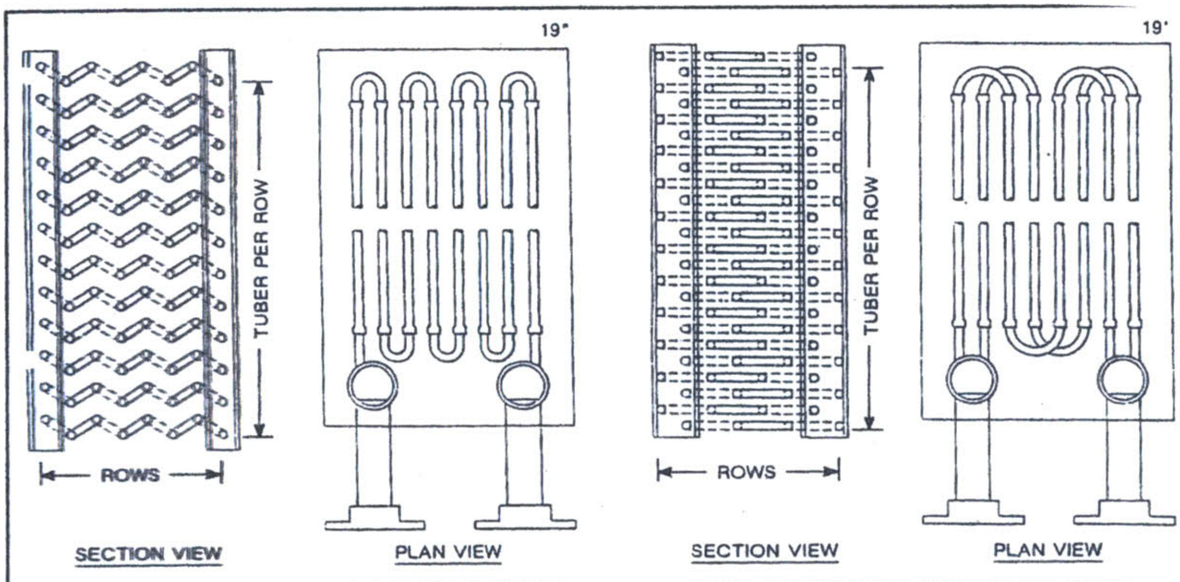


Figure 24: Single and double serpentine cooling coil flow configurations (Foltz, 1990)

As mentioned, the two serpentine cooling coils are an old design which may only exist on older ships. The double serpentine cooling coil (50 series cooling coil) has been replaced by the 1.5 serpentine cooling coil (60 series cooling coil). However, since the 50 series cooling coils may still be used on older ships, they were included in the heat exchanger database. In addition, unit coolers which are based off of the 50 series cooling coils have also been included within the heat exchanger database. The characteristics of the 50 series cooling coils, the 60 series cooling coils and the unit coolers have been included in Tables 6-8 below. Figures 16-18 show a photo of the unit cooler and the 60 series cooling coil.

Sizes	Capacity (BTU/hr)	Airflow (CFM)	Air Velocity (ft/min)	Leaving Air Temperatures DB/WB (F)	Water Pressure Drop (ft H_2O)	Air Pressure Drop (in H_2O)
61	9020	280	491	58.6/56.8	0.30	0.60
62	16470	450	500	56.9/55.3	0.90	0.70
63	27260	670	496	55.3/53.8	2.20	0.70
64	39970	975	488	55.2/53.7	2.40	0.70
65	63440	1450	485	53.1/52.5	2.40	0.80
66	112.20	2500	500	52.7/52.1	4.00	0.80
67	183.60	3800	507	51.1/50.8	4.00	0.80
68	240.70	5000	500	51.1/50.9	3.50	0.95

Table 5: 60 series cooling coil characteristics (MIL-PRF-2939G, 2001) (Frank & Helmick, 2007)

Sizes	Capacity (BTU/hr)	Airflow (CFM)	Air Velocity (ft/min)	Dimensions W"xH"xD"	Dry Weight (lbs)	Wet Weight (lbs)
51	14	280	491	26-1/2x12-1/8x15	152	157
52	23	450	500	28-3/4x14-3/8x15	176	183
53	34	670	496	35-3/4x14-3/8x15	225	236
54	50	975	488	40-1/2x16-7/8x15	301	317
55	65	1500	483	47x18-7/8x15	390	414
56	121	2500	500	55x23-3/8x15	562	602
57	190	3750	507	56-3/8x36-7/8x17-5/8	975	1040
58	234	5000	500	56-3/8x45-7/8x17-5/8	1225	1310

Table 6: 50 series cooling coil characteristics (DRS Technologies, 2011)

Sizes	Capacity (BTU/hr)	Air Flow (CFM)	Water flow (gpm)	Frame Size L"xW"xD"	Dry Weight (lbs)	Wet Weight (lbs)
51	13,500	215	4.0	23x12-1/8x38-7/8	202	207
52	22,200	340	7.0	25-1/4x14-3/8x38-5/8	236	239
53	33,500	510	10.0	32-1/4x14-3/8x40-3/8	315	326
54	49,300	750	15.0	37-1/4x16-5/8x40-7/8	411	427
55	62,400	1120	19.0	43-1/2x18-7/8x43-7/8	510	534

Table 7: Unit cooler characteristics (MIL-C-2939E(SH), 1984) (DRS Technologies, 2011)

Type DW61-68
Cooling Coils (60 Series)

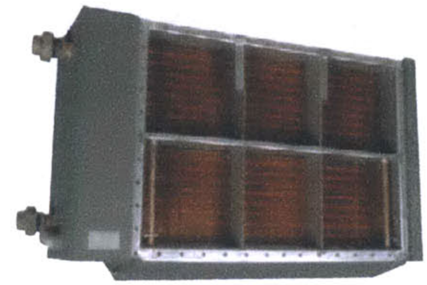


Figure 25: 60 series cooling coil (DRS Technologies, 2011)

Type DW51-58
Cooling Coils (50 Series)

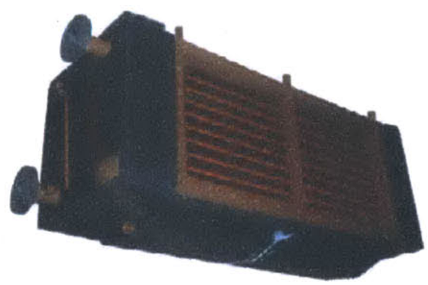


Figure 26: 50 series cooling coil (DRS Technologies, 2011)

Type UW51-55
Unit Coolers

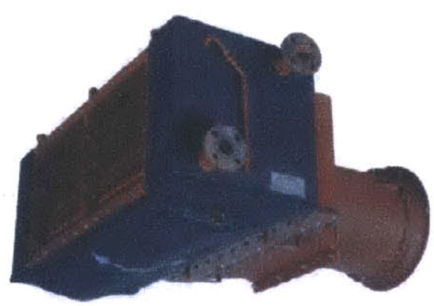


Figure 27: Unit cooler (DRS Technologies, 2011)

The head loss across the cooling coils will affect the flow into the branch containing the cooling coils. If the head loss is high, then there is a greater resistance to flow and thus more flow will be diverted into other parallel branches of the chilled water system. To account for this, the loss coefficient and head loss across the cooling coils must be known or calculated. The loss coefficient is composed of the losses due to friction, which is a factor of length, diameter and friction factor; and the losses due to 180° bends. There will also be losses associated with entrance and exit effects. The CSDT can compute these losses; however, the difficulty arises when determining the head loss on the secondary side. This will be discussed in greater detail in Section 3.1.1.2.

2.10 Air Conditioning Plants

The air conditioning plants are the mechanisms used to lower the temperature of the warmer water within the return header to the 6.6°C inlet temperature of the supply header[8]. There are various types of A/C units such as centrifugal type, screw type, and reciprocating type, but they all operate using the same underlying principles. An example of a specific A/C unit is the R-114 centrifugal A/C plant. The R-114 air conditioning plant utilizes a vapor compression system using centrifugal compressors.

Figure 28 below shows a schematic of the refrigeration cycle internal to the A/C unit. The A/C unit contains a closed loop containing a refrigerant such as R134a. The refrigerant runs through two different heat exchangers, one being a heat exchanger involving the chilled water, where heat is absorbed from the chilled water, and the other being a heat exchanger involving seawater, where heat is discharged to the seawater.

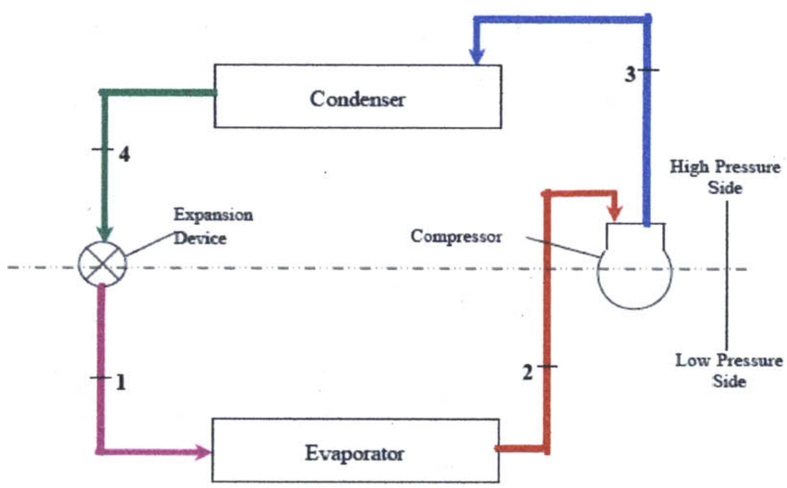


Figure 28: Vapor-compression refrigeration cycle diagram (enggcyclopedia)

The refrigeration cycle starts with a cool refrigerant such as R134a. The cool refrigerant is heated up by the warmer chilled water. With a low boiling point, the rise in temperature causes the refrigerant to change states and become a vapor. The vapor is then compressed by a centrifugal compressor, a screw compressor or some other mechanism. The refrigerant rises in pressure and temperature. The hot refrigerant enters a condenser (a heat exchanger), where heat is transferred from the hot refrigerant to seawater. The cooler refrigerant then enters an expansion valve. This reduces the pressure and temperature of the refrigerant. The process then repeats itself (Cloutier).

---

[8] This assumes the total heat load serviced by the A/C unit is less than or equal to the A/C unit cooling capacity. If it is not, the outlet temperature of the A/C unit will rise.

The total plant capacity is determined by the mass flow rate of the header and the differential temperature across the chiller. The chiller that most closely satisfies the plant capacity required is then selected for that particular zone. The refrigeration cycle is modeled using notional values for:

- Evaporator outlet temperature
- Compressor inlet pressure
- Compressor outlet pressure
- Throttling inlet pressure
- Compressor efficiency
- mass flow rate

Using the heat transfer equations, the chilled water outlet temperature can be determined, along with the refrigerant compressor inlet temperature, the refrigerant compressor outlet temperature, the refrigerant throttling inlet temperature, the evaporator inlet temperature, the evaporator outlet temperature, and the seawater outlet temperature. This process is explained in greater detail in Section 3.1.1.3.

3.0 Chapter 3: Design Tool Architecture

The CSDT is comprised of Matlab code and an Excel Spreadsheet. The Matlab code consists of a geometry module, an analysis module, a modification module, and several functions. A schematic of the CSDT architecture is shown in Figure 29 below.

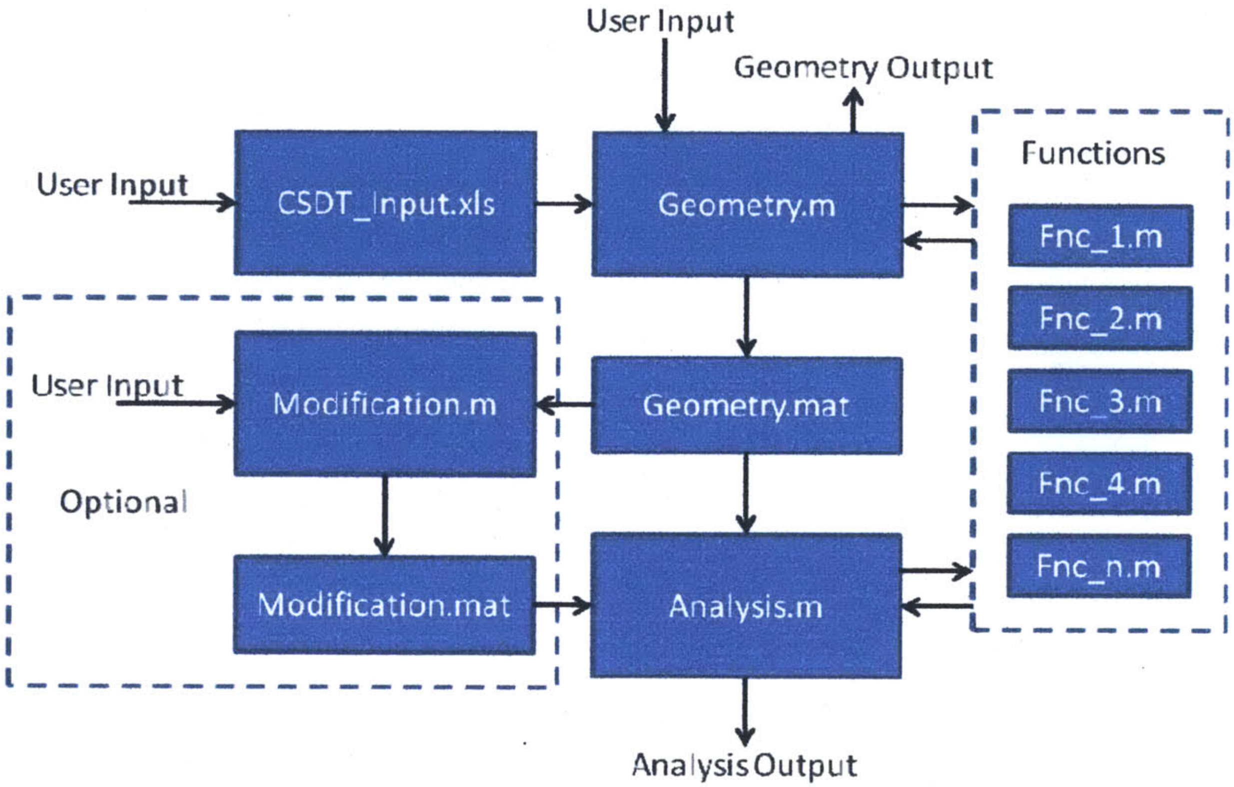


Figure 29: CSDT architecture

3.1 User Inputs

There are two major components of the CSDT, the first being the Excel spreadsheets and the second being the Matlab program. The Excel spreadsheets contain the heat load data, the heat exchanger database and the A/C unit (chiller) database. The Matlab program reads in the information provided by the spreadsheets and designs and analyzes the chilled water system with the aid of the user. A minor component of the CSDT is the Modification module which is an optional component of the CSDT.

3.1.1 Excel Spreadsheet Inputs

There are seven tabs within the Excel spreadsheet, with the first three requiring user input and the last four containing tables of refrigerant characteristics. The first tab contains data pertaining to the heat loads. The second and third tabs contain heat exchanger data and chiller data and serve as the program's database.

3.1.1.1 LoadData Tab

When first starting the design of the chilled water system, the user needs to enter the load data in the excel spreadsheet "CSDT_input" under the tab "LoadData". A screenshot of the "LoadData" spreadsheet is shown below in Figure 30.

Enter Data in Yellow Area Below!!!

* Priorities 1-2 are considered vital. All else are considered non-vital.

** Coordinate system is based on distance from midships

*** For heat exchanger, can enter either the heat exchanger type, or a specific heat exchanger.

Heat exchanger types: Cooling Coil (cc), 50 Series Cooling Coil (50cc), 60 Series Cooling Coil (60cc), Unit Cooler Cooling Coil (uc), Other Cooling Coil (oc), Flat Plate (fp), Shell and tube (st), Cold Plate (cp), Other (o)

Example of specific heat exchanger: Enter cc for the heat exchanger type and 3 for the heat exchanger to select cc3 listed within the heat exchanger database

Number of Loads	180

			Cooling Load for Various Conditions				Location**				
Load Name	Priority*	Electrical Demand	Shore	Design	Cruise	Battle	X	Y	Z	Heat Exchanger	Heat
Data String	1 = Highest	kW	kW	kW	kW	kW	m	m	m	Type***	Exchanger***
RS01	1	47.8	9.56	7.91	6.93	7.91	32.81	0.00	25.20	60cc	
RS02	1	50	3.56	20.22	20.22	20.22	33.14	0.00	22.83	60cc	
RS02_1	1	20	3.56	1.97	1.97	1.97	26.45	0.00	3.29	60cc	
RS04	1	2000	3.56	11.15	10.90	11.15	33.81	6.70	19.69	60cc	
RS0405	1	2500	0.24	13.54	12.52	13.54	33.81	0.00	19.69	60cc	
RS0405_1	1	750	45	5.31	4.99	5.31	29.12	-5.52	19.69	60cc	
RS04_1	1	250	20	6.51	6.26	6.51	23.77	6.70	19.69	60cc	
RS0405_2	1	115	9.7	3.80	3.48	3.80	29.12	5.52	19.69	60cc	
RS040505C	1	200	9.7	45.72	41.82	45.72	29.12	0.00	19.69	60cc	
RS05	1	354	0.9	10.30	10.09	10.30	33.81	-6.70	19.69	60cc	
RS05_1	1	264	1.74	6.65	6.40	6.65	23.77	-6.70	19.69	60cc	
RS07	1			5.42	4.92	5.42	-14.39	0.00	22.83	60cc	
RS07_1	1			3.62	3.13	3.62	-17.74	0.00	19.69	60cc	
RS07_2	1			8.02	7.07	8.02	-15.06	-3.85	14.05	60cc	
RS08	1			3.62	3.62	3.62	39.17	0.00	16.59	60cc	
RS12	2			6.72	6.72	6.72	33.14	0.00	14.05	60cc	
RS12_1	2			1.55	1.55	1.55	34.48	-8.54	8.51	60cc	

LoadData / HX DB / Chiller DB / R134a-superheated vapor / R134a-saturated / R404a-superheated vapor / R404a-saturated

Figure 30: LoadData tab

A heat load is defined by any piece of equipment or HVAC unit that requires chilled water cooling. As can be seen in the figure, there are several columns pertaining to data required for each heat load. The first is the heat load name.

After the name has been selected, a priority has to be assigned to the heat load. The priority ranges from 1-8 with 1 corresponding to the highest priority. This convention was retained from the previous version of the CSDT. The priority is used by the Matlab program to determine vital or non-vital status. If a heat load has a priority of 1 or 2, then the heat load is considered to be a vital load and the design pertaining to vital loads is adhered to; otherwise, the load is considered non-vital.

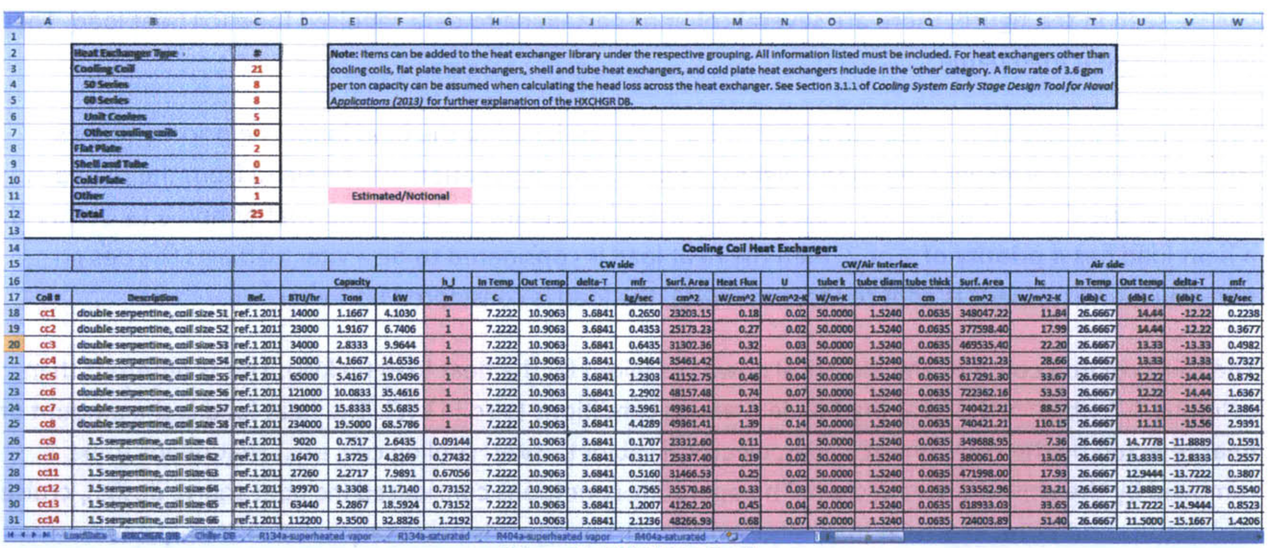
The third column contains the electrical power required by the heat load (in kW). This is different than columns 4-7 which is the heat load under various conditions (also in kW). The heat load is the amount of heat rejected by the component (radar, electrical cabinet, HVAC cooling coil, etc.) that needs to be removed by the chilled water system. The operating conditions considered include: shore, design, cruise and battle conditions. The heat load required in each condition is necessary because if only one condition was considered, the heat exchanger and branch piping associated with that load may be undersized when considering another operating condition.

A very important parameter needed by the user is the location of the center of the heat load. The heat load location is entered in columns 8-10 (in meters) with the origin (0,0,0) corresponding to midships, centerline, and baseline of the ship, respectively.

The last two columns pertain to the heat exchanger associated with each load. The heat exchanger type is tied to the second Excel spreadsheet tab "HXCHGR DB" discussed in the next section. The user selects the type of heat exchanger used as the interface between the heat load and the chilled water system. The user can either select the heat exchanger type, selecting from: cooling coil (cc), 50 series cooling coil (50cc), 60 series cooling coil (60cc), unit cooler cooling coil (uc), other cooling coil (oc), flat plate heat exchanger (fp), shell and tube heat exchanger (st), cold plate heat exchanger (cp), or other heat exchanger (o). If the user selects to enter a heat exchanger type, then the next column should remain blank. Selecting a type of heat exchanger will prompt the Matlab program to select a heat exchanger of that type properly sized for that particular heat load (or as closely sized as is possible with the heat exchanger available within the heat exchanger database). If the user wishes to select a specific heat exchanger for a particular heat load, then the user specifies the type in column 11 (cooling coil (cc), flat plate (fp), shell and tube (st), or other(o)) and the number corresponding to the specific heat exchanger as listed in the tab "HXCHGR DB". When selecting a specific heat exchanger, it is important to properly size it. In other words, the greatest heat load possible in any operating condition must be lower than the rating of the heat exchanger, otherwise flow velocities and/or temperature limits may be exceeded.

3.1.1.2 HXCHGR DB tab

The next tab in the "CSDT_input" Excel spreadsheet is the "HXCHGR DB" tab. This tab includes data for several types of heat exchangers, forming a heat exchanger database. A screenshot of the "HXCHGR DB" tab is shown in Figure 31 below.



Heat Exchanger Type	#
Cooling Coil	21
50 Series	8
60 Series	8
Unit Coolers	5
Other cooling coils	0
Flat Plate	2
Shell and Tube	0
Cold Plate	1
Other	1
Total	25

Note: Items can be added to the heat exchanger library under the respective grouping. All information listed must be included. For heat exchangers other than cooling coils, flat plate heat exchangers, shell and tube heat exchangers, and cold plate heat exchangers include in the 'other' category. A flow rate of 3.6 gpm per ton capacity can be assumed when calculating the head loss across the heat exchanger. See Section 3.1.1 of *Cooling System Early Stage Design Tool for Naval Applications (2013)* for further explanation of the HXCHGR DB.

Estimated/Notional

Cooling Coil Heat Exchangers

			Capacity			CW side								CW/Air interface			Air side					
						h_l	In Temp	Out Temp	delta-T	mfr	Surf. Area	Heat Flux	U	tube k	tube diam	tube thick	Surf. Area	hc	In Temp	Out temp	delta-T	mfr
Coil #	Description	Ref.	BTU/hr	Tons	kW	m	C	C	C	kg/sec	cm^2	W/cm^2	W/cm^2-K	W/m-K	cm	cm	cm^2	W/m^2-K	(db) C	(db) C	(db) C	kg/sec
cc1	double serpentine, coil size 51	ref.1 2011	14000	1.1667	4.1030	1	7.2222	10.9063	3.6841	0.2650	23203.15	0.18	0.02	50.0000	1.5240	0.0635	348047.22	11.84	26.6667	14.44	-12.22	0.2238
cc2	double serpentine, coil size 52	ref.1 2011	23000	1.9167	6.7406	1	7.2222	10.9063	3.6841	0.4353	25173.23	0.27	0.02	50.0000	1.5240	0.0635	377598.40	17.99	26.6667	14.44	-12.22	0.3677
cc3	double serpentine, coil size 53	ref.1 2011	34000	2.8333	9.9644	1	7.2222	10.9063	3.6841	0.6435	31302.36	0.32	0.03	50.0000	1.5240	0.0635	469535.40	22.20	26.6667	13.33	-13.33	0.4982
cc4	double serpentine, coil size 54	ref.1 2011	50000	4.1667	14.6536	1	7.2222	10.9063	3.6841	0.9464	35461.42	0.41	0.04	50.0000	1.5240	0.0635	531921.23	28.66	26.6667	13.33	-13.33	0.7327
cc5	double serpentine, coil size 55	ref.1 2011	65000	5.4167	19.0496	1	7.2222	10.9063	3.6841	1.2303	41152.75	0.46	0.04	50.0000	1.5240	0.0635	617291.30	33.67	26.6667	12.22	-14.44	0.8792
cc6	double serpentine, coil size 56	ref.1 2011	121000	10.0833	35.4616	1	7.2222	10.9063	3.6841	2.2902	48157.48	0.74	0.07	50.0000	1.5240	0.0635	722362.16	53.53	26.6667	12.22	-14.44	1.6367
cc7	double serpentine, coil size 57	ref.1 2011	190000	15.8333	55.6835	1	7.2222	10.9063	3.6841	3.5961	49361.41	1.13	0.11	50.0000	1.5240	0.0635	740421.21	88.57	26.6667	11.11	-15.56	2.3864
cc8	double serpentine, coil size 58	ref.1 2011	234000	19.5000	68.5786	1	7.2222	10.9063	3.6841	4.4289	49361.41	1.39	0.14	50.0000	1.5240	0.0635	740421.21	110.15	26.6667	11.11	-15.56	2.9391
cc9	1.5 serpentine, coil size 61	ref.1 2011	9020	0.7517	2.6435	0.09144	7.2222	10.9063	3.6841	0.1707	23312.60	0.11	0.01	50.0000	1.5240	0.0635	349688.95	7.36	26.6667	14.7778	-11.8889	0.1591
cc10	1.5 serpentine, coil size 62	ref.1 2011	16470	1.3725	4.8269	0.27432	7.2222	10.9063	3.6841	0.3117	25337.40	0.19	0.02	50.0000	1.5240	0.0635	380061.00	13.05	26.6667	13.8333	-12.8333	0.2557
cc11	1.5 serpentine, coil size 63	ref.1 2011	27260	2.2717	7.9891	0.67056	7.2222	10.9063	3.6841	0.5160	31466.53	0.25	0.02	50.0000	1.5240	0.0635	471998.00	17.93	26.6667	12.9444	-13.7222	0.3807
cc12	1.5 serpentine, coil size 64	ref.1 2011	39970	3.3308	11.7140	0.73152	7.2222	10.9063	3.6841	0.7565	35570.86	0.33	0.03	50.0000	1.5240	0.0635	533562.96	23.21	26.6667	12.8889	-13.7778	0.5540
cc13	1.5 serpentine, coil size 65	ref.1 2011	63440	5.2867	18.5924	0.73152	7.2222	10.9063	3.6841	1.2007	41262.20	0.45	0.04	50.0000	1.5240	0.0635	618933.03	33.65	26.6667	11.7222	-14.9444	0.8523
cc14	1.5 serpentine, coil size 66	ref.1 2011	112200	9.3500	32.8826	1.2192	7.2222	10.9063	3.6841	2.1236	48266.93	0.68	0.07	50.0000	1.5240	0.0635	724003.89	51.40	26.6667	11.5000	-15.1667	1.4206

Figure 31: HXCHGR DB tab

The figure above only displays the first type of heat exchanger type, the cooling coil. The database extends to the right containing similar data columns for the flat plate heat exchanger, the shell and tube heat exchanger, the cold plate heat exchanger and an 'other' category for more exotic types of heat exchangers. The heat exchangers currently modeled in the database include:

- 50 series cooling coil (double serpentine)
- 60 series cooling coil (1.5 serpentine)
- Unit cooler cooling coil (double serpentine)
- Notional flat plate heat exchanger (cross-flow)
- Notional cold plate heat exchanger
- Notional concentric tube heat exchanger (cross-flow)

Although the 50 series cooling coils and unit cooler cooling coils are no longer implemented on U.S. Navy vessels, they were included in the database in case an older ship's chilled water system were to be modeled with the use of this tool.

To accurately model the temperatures within the chilled water system, and to attempt to capture the temperature profile extending beyond the chilled water system into the heat exchanger and finally to the secondary fluid (be it air, demineralized water, or even oil), an extensive set of data is needed for the heat exchangers within the heat exchanger database. This exemplifies the difficulty that arises between creating a simple-to-use model, and a model that makes few assumptions to accurately portray the flow and temperature distribution within the chilled water system. As a compromise, the most essential parameters that describe the heat exchanger are kept, while the specific heat exchanger geometries are not. Essentially, the heat exchanger is treated as a box, using only average inlet and outlet values to simplify the calculations and to reduce the amount of information required by the user to add a heat exchanger to the database. Values calculated from assumptions made about the heat exchanger are highlighted in red. The rationale for each assumption is stated in the preceding paragraphs.

The first column of the 'HXCHGR DB' tab lists the name of the heat exchanger. The convention is as follows: the heat exchanger type and ascending number. The Matlab program uses this information to identify the individual heat exchangers.

The second column gives a brief description of the heat exchanger. This affords the user with some information about the heat exchanger if the user wished to select a particular heat exchanger for a particular heat load. This column does not have to be filled in, in that the program does not use any data contained in the description columns, but it is helpful to provide a description of heat exchangers added to the database for future users.

The third column provides references for the data contained for any specific heat exchanger. Again, the program does not require this information, but it is useful to provide source documentation if the need arises to further investigate a particular heat exchanger.

Columns 4-6 provide the heat capacity of the heat exchanger in BTU/hr, tons, and kW, respectively. The heat capacity is the heat transfer rate of the heat exchanger under certain conditions. The heat capacity should be greater or equal to the maximum heat load under any operating condition for a particular load for similar conditions.

Data for both sides of the heat exchanger is needed to accurately capture the performance of the heat exchanger. The chilled water side is referred to as the primary side or the primary loop. The air/demineralized water/oil/etc. side of the heat exchanger is referred to as the secondary side or the secondary loop.

To determine the head loss across the heat exchanger on the primary side, values from "21st Century HVAC System for Future Naval Surface Combatants-Concept Development Report" NSWCCD-98-TR-2007/06 were used. The head loss values for the 60 series cooling coils are listed in Table 8 below.

60 Series Cooling Coil Head Loss Values		
Coil Size	Water Pressure Drop (ft H_2O)	Water Pressure Drop (m H_2O)
61	0.3	0.09144
62	0.9	0.27432
63	2.2	0.67056
64	2.4	0.73152
65	2.4	0.73152
66	4	1.2192
67	4	1.2192
68	3.5	1.0668

Table 8: Pressure drop values for 60 series cooling coils (Frank & Helmick, 2007)

Similar data for the 50 series cooling coils or the unit cooler cooling coils was not available. Therefore, nominal values for head loss were used for those types of heat exchangers.

Columns 8-10 list the inlet, outlet and differential chilled water temperatures. The heat exchanger heat capacity was calculated based on an inlet chilled water temperature of 45°F or 7.22°C. The outlet chilled water temperature was calculated using the equation:

$$\dot{Q} = \dot{m}c_p(T_h - T_c)$$

Equation 1 (repeated)

The chilled water mass flow rate needs to be entered in column 11. For the cooling coils listed in the database, the chilled water mass flow rate was calculated using the equation below:

$$\dot{m} = 3.6\frac{gal}{min - ton} * tons\ capacity$$

Equation 57

The 3.6 gpm per ton capacity flow rate is the design flow rate of the cooling coils when determining the heat exchanger cooling capacity (Frank & Helmick, 2007).

To determine the heat transfer across the heat exchanger boundary, the surface area on the primary and secondary side is needed. This is difficult, since different manufacturers of the same coil size and type will have differing geometry. Therefore, the surface areas were calculated based on assumptions of the heat exchanger geometry. The cooling coil outer diameters were assumed to be 0.625 inches with a thickness of 0.025 inches[9]. The 50 series cooling coils and the unit coolers are double serpentine configurations with 8 rows and 12 tubers. The 60 series cooling coils are a 1.5 serpentine configuration with 6 rows and 12 tubers. The length of a row was assumed to span the width of the heat exchanger. Therefore, the inner surface area can be calculated using the equation:

$$SA_{inner} = (0.625in - 0.025in) * \pi * rows * tubers * width_{hxchgr}$$

Equation 58

Error is introduced in calculating the surface area since bends are not considered, the outer diameter and tube thickness may vary depending on the coil size, if a flatter coil is used instead of a cylindrical coil, and if turbospirals are used within the cooling coil. A turbospiral is a spiral piece of copper on the inside of the cooling coil which acts to trigger turbulent flow within the cooling coil. However, the above equation gives at least a rough approximation of surface area for a particular cooling coil type.

The outer heat exchanger surface area is even more error-prone due to complex fin geometry and variations in fin design and heat exchanger design. To get at least a rough approximation of outer surface area, the inner surface area was scaled up 15 times. This value was chosen based on the paper "The Design of Air Conditioning and Ventilation Systems for nuclear Submarines" which performed calculations in the analysis of a 46DW cooling coil. The paper initially used a factor of 15 and revised this number to 14.31 (Foltz, 1990). The value of 15 was chosen since there are many unknowns in the heat exchanger geometry and a precise value of 14.31 was unwarranted. The inner surface area of the coils and outer surface area of the coils are entered in columns 12 and 15, respectively.

The heat flux of the heat exchanger is calculated in column 13. This value is determined by dividing the cooling capacity by the inner surface area of the cooling coils. Values on the order of $1W/cm^2$ was found for the net flux of the cooling coils, which was to be expected due to the inefficiency of forced convection air on the secondary side. The low heat fluxes associated with cooling coils was the main driver in offering a section for 'other' types of heat exchangers. This category of heat exchangers could include two-phase flow heat exchangers, heat exchangers utilizing jets, some combination of the two, or

[9] These values were chosen based on the paper "21st Century HVAC System for Future Naval Surface Combatants-Concept Development Report" NSWCCD-98-TR-2007/06

some other exotic heat exchanger type that is capable of heat fluxes on the order of 300-500 W/cm^2 or higher.

The overall heat transfer coefficient, U, is calculated using the equation below:

$$U = \frac{\dot{Q}}{SA_{inner} * \left[\left(T_{in_{air}} - \frac{T_{in_{air}} - T_{out_{air}}}{2}\right) - \left(T_h - \frac{T_h - T_c}{2}\right)\right]}$$

Equation 2 (repeated, rearranged)

The overall heat transfer coefficient is with respect to the inner surface area of the cooling coils. The values for the overall heat transfer coefficient range from 0.02-0.15 W/cm^2-K, which are reasonable values for this type of heat exchanger.

Column 16 lists the convective heat transfer coefficient, $h_{c_{air}}$, on the secondary side of the heat exchanger. This is the most difficult parameter to be determined. The convective heat transfer coefficient is actually an average value. To determine this value analytically, a finite element approach would have to be taken, with the local convective heat transfer coefficient found at each location on the outer surface of the cooling coils and then integrated over the entire surface. This is not computationally feasible, especially since the outer cooling coil geometry and flow are not known. To get a notional value of the convective heat transfer coefficient, the average temperature on the outer surface (estimated) and the average temperature on the secondary side are taken in conjunction with the estimated outer surface area and the known heat transfer rate. The convective heat transfer coefficient is then computed using the equation:

$$h_{c_{air}} = \frac{\dot{Q}}{SA_{outer} * \left(\left[\left(T_{in_{air}} - \frac{T_{in_{air}} - T_{out_{air}}}{2}\right) - T_2\right]\right)}$$

Equation 59

where,

$$T_2 = \left(T_h - \frac{T_h - T_c}{2}\right) + \frac{\dot{Q}}{h_{c_{water}} SA_{inner}} + \frac{\dot{Q}_{per\ unit\ length}}{2\pi k_{copper}} ln\left(\frac{D_{outer}}{D_{inner}}\right)$$

Equation 60

and,

$$h_{c_{water}} = 0.023 \frac{V^{0.8} k^{0.6} (\rho c_p)^{0.4}}{D^{0.2} \nu^{0.4}}$$

Equation 12 (repeated)

and,

$$\dot{Q}_{per\ unit\ length} = \frac{Q * S\dot{A}_{inner}}{D_{inner}}$$

Equation 61

As can be seen from the above equations, several simplifying assumptions are made in determining the convective heat transfer coefficient on the secondary side.

1. The inner surface area assumes cylindrical tubing as opposed to flattened tubing. It also assumes the inner diameter of the tube is 0.6" for all cooling coils, and turbospirals are neglected.
2. The outer surface area is estimated to be 15 times that of the inner surface area, which would, in reality, vary from manufacturer to manufacturer.
3. In calculating T_2, the wall temperature on the outer surface of the cooling coils, the convective heat transfer coefficient on the water side is calculate using the equation described in Section 2. This equation is valid for flow through a cylindrical tube. The turbospirals within the cooling coils will have an effect on the convective heat transfer coefficient and the only way to determine this effect would be to generate parametric equations based on a specific manufacturer's heat exchanger. The turbospirals are ignored in order to easily compute a value for the convective heat transfer coefficient on the primary side.
4. The temperature rise across the copper material of the cooling coils is calculated by again neglecting the outer fins and treating the heat exchanger as a simple cylindrical tube. This assumption has little effect on the overall temperature rise since the resistance to heat flow caused by the fins would be very small in comparison to the film layer on the air side or even the film layer on the water side.
5. Pipe bends, entrance and exit effects, and friction resistance were also neglected with the thought that these are also all negligible in comparison to the temperature rise in the two film layers on either side of the heat exchanger boundary.

Because of these assumptions, the convective heat transfer coefficient on the secondary side is more of a notional value to be used in computations done by the Matlab program. To get the true convective heat transfer coefficient on the secondary side, the specific heat exchanger would have to be modeled in greater detail and the flow on the secondary side of the heat exchanger would also have to be modeled. The calculated values of the convective heat transfer coefficient on the secondary side are reasonable, however, as they do fall in the range expected for forced convection air. Forced convection air should result in values in the range of 5-200 W/m^2-K for the convective heat transfer coefficient. The values computed for the 60 series cooling coils falls within this range. Table 9 summarizes the calculated convective heat transfer coefficient on the air side for the 60 series cooling coils.

60 Series Cooling Coil	
Coil Size	**h_c [W/m^2-K]**
61	7.36
62	13.05
63	17.93
64	23.21
65	33.65
66	51.40
67	89.77
68	119.28

Table 9: Calculated values of convective heat transfer coefficient on air side of 60 series cooling coils

The inlet, outlet and differential temperatures on the secondary side are required in columns 17-19, respectively. For the 60 series cooling coil, these values were available in MIL-PRF-2939-G. For the 50 series cooling coils and the unit cooler cooling coils, the inlet temperatures were known. The outlet temperatures were assumed to be 58°F (14.44°C) for coil sizes 51 and 52, 56°F (13.33°C) for coil sizes 53 and 54, 54°F (12.22°C) for coil sizes 55 and 56, and 52°F (11.11°C) for coil sizes 57 and 58.

The mass flow rate of the air on the secondary side is required in column 20. This value was provided for the 60 series cooling coils. To determine the mass flow rate of the air on the secondary side for the 50 series cooling coils and the unit cooler cooling coils, the following equation was used:

$$\dot{m} = \frac{\dot{Q}}{c_p\left(T_{in_{air}} - T_{out_{air}}\right)}$$

Equation 1 (repeated, rearranged)

The specific heat capacity of the air was unknown, but was back-calculated using the known values of the 60 series cooling coil. The specific heat capacity was calculated to be roughly 1500 J/kg-K with a deviation of less than 2% for most of the cooling coils. This value also falls between that of the specific heat capacity of dry air at sea level (which has a value of 1003.5 J/kg-K) and water (which has a value of 4203 J/kg-K). A value of 1500 J/kg-K seems reasonable for the heat exchanger since the higher temperature and humidity would cause the specific heat capacity to fall within this range (but closer to the lower limit since air is being considered).

The dimensions of the heat exchangers are entered in columns 21-23. These values are used by the Matlab program to size the heat exchangers when constructing the three-dimensional plot of the chilled water system. The varying size of the heat exchangers within the 3-D plot allows quick visualization of where the larger heat loads are located. The heat exchanger dimensions for the cooling coils are listed in MIL-C-2939-E (outdated) and MIL-PRF-2939-G (current).

The dry and wet weights of the heat exchangers are entered in columns 24 and 25, respectively. These weights are used by the Matlab program when performing a weight analysis of the chilled water system.

With the heat load location, an accurate center of gravity of the chilled water system (with heat exchangers included) can also be determined.

Similar columns are included for data entry of the four other categories of heat exchangers: flat plate, shell and tube, cold plate and other types of heat exchangers.

3.1.1.3 Chiller DB tab

The last tab in the 'CSDT_input' Excel spreadsheet is the 'Chiller DB' tab. This tab includes data for four types of chillers (AC units) forming the chiller database. A screenshot of the 'Chiller DB' tab is shown in Figure 32 below.

Chiller Type	#
Centrifugal	4
Reciprocating	2
Screw	2
Other	0
Total	8

Note: The pressures and temperatures provided are notional values and are not based on the specified references. These pressures and temperatures and associated refrigerant tables are used to loosely model the refrigerant cycle within the AC unit to determine the power required by the compressor and to calculate the seawater mass flow rate. See Section 3.1.1 of Cooling System Early Stage Design Tool for Naval Applications (2013) for further explanation of the Chiller DB.

Centrifugal AC unit																	
							Dimensions			Refrigerant							
			Capacity			Weight	L	W	H	Type	P1	T1	P2	T2	P3	T3	Out temp
Centrifugal #	Description	Ref.	BTU/hr	Tons	kW	kg	m	m	m		MPa	C	MPa	C	MPa	C	C
c1	YK - SW cooled, centrifugal AC	ref.1	5118213	426.518	1500	20000	5.00	2.40	2.30	R134a	0.28	0.00	1.00	60.00	0.95	30.00	6.66667
c2	YK Special - SW cooled, centrifugal AC	ref.1	2047285	170.607	600	9500	3.90	1.80	1.78	R134a	0.28	0.00	1.00	60.00	0.95	30.00	6.66667
c3	Notional SW cooled, centrifugal AC	ref.1 Tak	2400000	200	703.371	13405.9	5.21	2.59	3.59	R134a	0.28	0.00	1.00	60.00	0.95	30.00	6.66667
c4	Notional SW cooled, centrifugal AC	ref.1 Tak	2400000	200	703.371	7927.35	4.97	1.70	2.18	R134a	0.28	0.00	1.00	60.00	0.95	30.00	5.55556
c5																	
c6																	
c7																	
c8																	
c9																	
c10																	
c11																	
c12																	
c13																	
c14																	
c15																	
c16																	
c17																	
c18																	
c19																	

Figure 32: Chiller DB tab

The four groups of chiller units include: centrifugal A/C unit types, reciprocating A/C unit types, screw A/C unit types, and other A/C unit types. The data required for each type of A/C unit is similar. If the user wishes to add to the database, all information needs to be documented within the spreadsheet.

Similar to the heat exchanger spreadsheet, the first three columns contain the name of the chiller using the same naming convention described for the heat exchangers. This name is what the Matlab program uses to identify the specific chillers. The second and third columns provide a description and/or name of the chiller and the source documents in which the chiller data was obtained.

Columns 4-6 include the capacity of the chiller in BTU/hr, tons, and kW, respectively.

The weight of the chiller (including coolant) is included in column 7. With such large weights associated with the chillers, these weights should be as accurate as possible since a large error in chiller weight would cause a large error in overall weight of the chilled water system. The Matlab program uses the weight of the chiller in performing the weight analysis of the chilled water system. Along with the chiller location, which is found through the use of the Matlab program, an accurate center of gravity of the chilled water system is possible.

Column 8-10 includes the dimensions of the chiller units. These values also need to be accurate in order to ensure the chiller units fit within the compartments in which they are placed. This is especially true for the larger chiller units. Chiller units c3 and c4 were taken CSDT v1.0, which sized the chillers parametrically (Fiedel, 2011).

Column 12 includes the refrigerant type used for the chiller.

Columns 13-18 include the pressures and temperatures of the refrigerant at various locations within the refrigerant cycle shown in Figure 33[10]. These pressures and temperatures are used in conjunction with the refrigerant tables to determine the corresponding enthalpies at these locations. The enthalpies are used to determine the heat transferred through the condenser into the seawater.

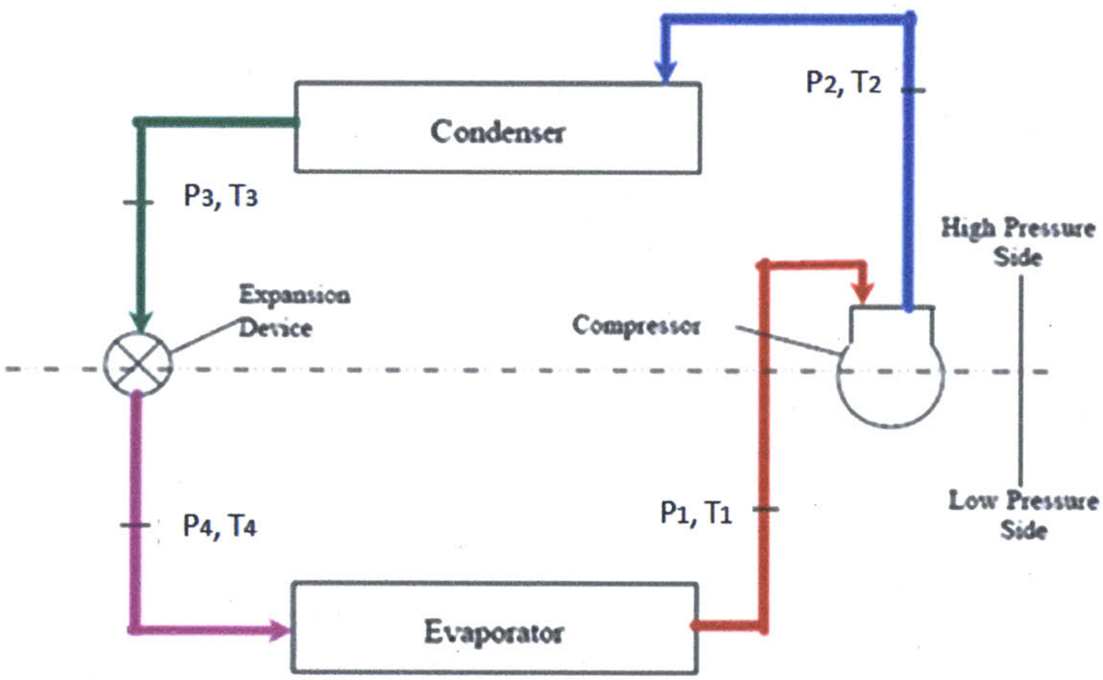


Figure 33: Refrigerant cycle with pressure and temperature variables shown (enggcyclopedia)

Lastly, Column 19 includes the chilled water outlet temperature of the chiller unit. This temperature is assumed to be met as long as the total heat load removed by the chiller is less than the capacity of the

[10] P_4 and T_4 are not needed in the chiller database since the enthalpy does not change across the expansion valve.

chiller[11]. Most of the chillers within the chiller database currently use the standard temperature of 44°F (6.67°C), but the ability is there to include chillers that output colder chilled water such as c4 which provides a chilled water temperature of 42°F (5.56°C).

3.1.2 Matlab Inputs

The main components of the chilled water design tool are the Matlab programs "geometry.m" and "analysis.m". The geometry module requires user input for principle ship dimensions and when a design decision needs to be made. The user interacts with the Matlab program through the use of command prompts in the command window. There are also some pop-up windows which appear throughout the program when a visual representation of the chilled water system would be beneficial in aiding design decisions.

The program starts out asking general questions about the ship's dimensions. These include:

- Length Overall (LOA)
- Beam
- Engine Room Deck Height Above the Keel
- Useable height in the engine room

The program provides default vales for these ship parameters if the user does not have a specific ship in mind. These default values are notional ship values taken from CSDT v1.0 (Fiedel, 2011). The user has the ability to overwrite the default values for one or more of the ship's dimensions. The dimensions must be inputted in metric, just as all subsequent parameters must also be inputted in metric. The default values provided by the program are:

- LOA = 143.561 m
- Beam = 20.390 m
- Engine Room Deck Height Above the Keel = 1.397 m
- Useable Height in the Engine Room = 3.098 m

After providing ship dimensions or accepting the default values, the program asks for the transverse bulkhead locations. The bulkhead locations must be entered as an array following the format:

$$[FP \quad BKHD\,1 \quad BKHD\,2 \quad BKHD\,3 \quad \ldots \quad BKHD\,N \quad AP]$$

The longitudinal axis is defined with midships at zero, the forward perpendicular (FP) at LOA/2 and the aft perpendicular (AP) at –LOA/2. The bulkhead location array also must include the FP in the first cell of the array and the AP in the last cell of the array. The default values are again notional values and are determined by the following array:

[11] If the chiller capacity is less than the total heat load serviced by the chiller, the outlet temperature will rise proportionately by the difference in heat transferred into the chilled water and heat transferred out by the chiller. The rate of temperature increase will depend on the thermal capacity of the chilled water.

$$\frac{LOA}{200} \times [100 \quad 90 \quad 82.5 \quad 67.5 \quad 52.5 \quad 37.5 \quad 20 \quad 5 \quad -10 \quad -27.5 \quad -43.5 \quad -50 \quad -80 \quad -100]$$

The user has the ability to overwrite the default bulkhead locations keeping in mind the array format, or can accept the default locations if the locations are not known. It would be ideal if these first inputs were generated from a separate module preceding the design of the chilled water system. This could potentially be an area of future work. However, due to time constraints, the program gathers the general ship dimensions and bulkhead locations from the user through the use of the command window.

After the ship dimensions and bulkhead locations have been identified, the Matlab program reads in the data provided by the Excel spreadsheet. The spreadsheet must be saved in the same folder as the program with the file name 'CSDT_input.xlsx' in order for the program to find it. If the Excel spreadsheet is not in the same folder as the program, an error message is displayed and the program ends. Any data entered previously is lost and would have to be re-entered after the excel spreadsheet is located in the correct folder.

At this point, no design decisions regarding the design of the chilled water system have been made. The first design decision encountered is the main piping configuration. The program offers three default main piping layouts. If the chilled water system is designed for an auxiliary ship, then a single main piping system should be selected, otherwise, a double main piping system should be chosen. The program offers two double main piping system layouts. The first layout is a simple rectangular loop. The second layout is a loop that can be modified to follow the shape of the hull. There are no cross-connections for either of the double main piping system layouts except at the bow and stern. An example of each piping layouts is shown in Figure 34 below.

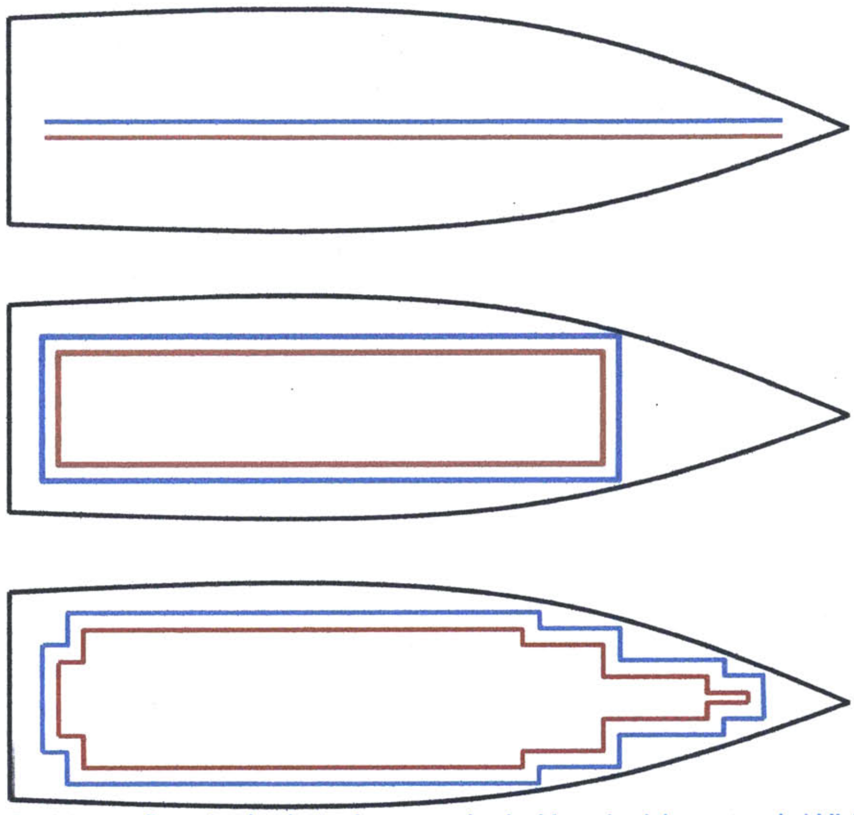


Figure 34: Single main piping configuration (top); simple rectangular double main piping system (middle); complex tapered double main piping system (bottom)

If the single main piping system is selected, the main piping height must be inputted. If the user does not have a main piping height, then the default value of 5.2 m is used. The single main piping system runs along the centerline of the ship 3 m from the bow to 3 m from the stern.

If the double main piping system is selected, then the main piping height port and starboard must be inputted. The default values are a height of 5.2 m on the port side and 10.2 m on the starboard side. The user can overwrite these default values, but should consider vertical separation of 1-2 decks for survivability consideration with one of the main piping heights corresponding to the damage control deck. The extents of the rectangular double main piping system is 3 m from the bow, 3 m from the stern, and half the beam minus 0.9 m from centerline. For the more complex double main piping system, there is a series of default locations corresponding to 90° bends in the piping. The bends results in a tapering of the double main piping system at the bow and at the stern. Figure 35 shows the default layout of the more complex double main piping system. If the user wishes to modify the layout of this main piping system, then the bend locations must be inputted in a matrix format. If the bends are symmetrical port and starboard, then the following format should be used:

$$[x_1 \quad y_1; x_2 \quad y_2; x_3 \quad y_3; x_4 \quad y_4; \ldots; x_n \quad y_n]$$

where the bend locations (in the x-y plane)are entered starting from the centerline forward and continuing counter-clockwise until centerline aft. In the figure, the locations of the first six points are all outer bends. The last point, point 7 occurs within the aft taper and is an inner corner. The bend locations are specified for the supply header only. The return header bend locations will be offset by the offset distance discussed shortly.

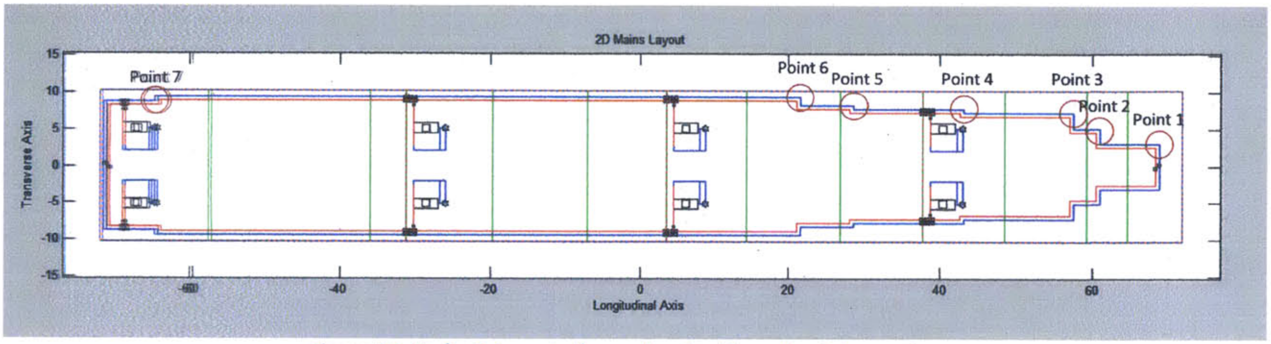


Figure 35: Default layout of complex double main piping system

If the bend locations are not symmetrical, then the following format should be used:

$$[x_1 \quad y_1; x_2 \quad y_2; x_3 \quad y_3; \ldots; x_m \quad y_m; x_{m+1} \quad y_{m+1}; x_{m+2} \quad y_{m+2}; x_{m+3} \quad y_{m+3}; \ldots; x_n \quad y_n]$$

The points entered should start from centerline forward and be entered counter-clockwise until centerline forward. Similar to the case above, the points in the taper near the bow should be the outer bends and the points in the taper near the stern should be the inner bends.

The main piping should be within 3 ft of the hull, except for curved sections of the hull which allows a maximum distance of 8 ft. Since the hullform is not defined within the program, this step cannot be done automatically. An area of future study could be to incorporate the hull structure as mentioned earlier. If the hullform is known, this process could be automated, eliminating the need of the simple rectangular layout and optimizing the layout of the tapered double main piping system.

Next, the program asks for the piping offset distance between the supply and return header. Figure 36 below gives a visual representation of the offset distance. The default offset distance for the header is 0.5 m. Similarly, the offset distance for the branch piping is also prompted for. The default branch piping distance is 0.1 m.

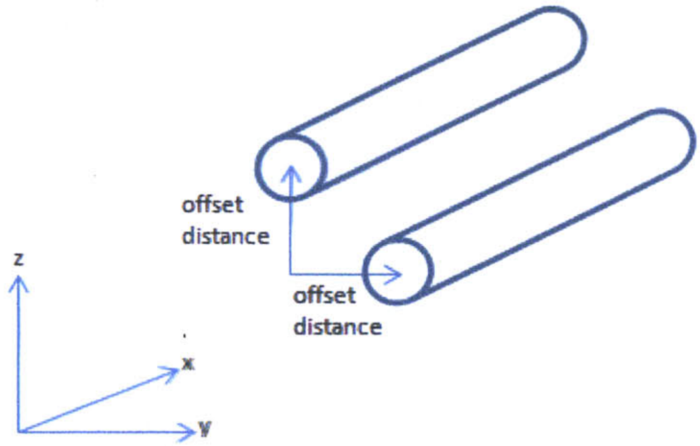


Figure 36: Offset distances

The next step in designing the chilled water system is to determine the number of chilled water zones. The heat loads are broken up into zones along the length of the ship. The zones can be isolated from one another during a casualty. The greater the number of zones, the more survivable he ship is; however, increasing the number of zones also increases the weight, space required, and ultimately, cost. All zones terminate at a transverse bulkhead. The fewest number of zones allowed by the program is two. While, it is possible for each compartment to be designated as a zone, the number of zones will generally be much less. The default number of zones is four. To aid in decision making, the program plots the heat load in each compartment and the heat load within each default zone. By default, the four zones are broken up into approximately equal lengths, with the zones terminating at the nearest transverse bulkhead. Figure 37 below shows the output provided by the program.

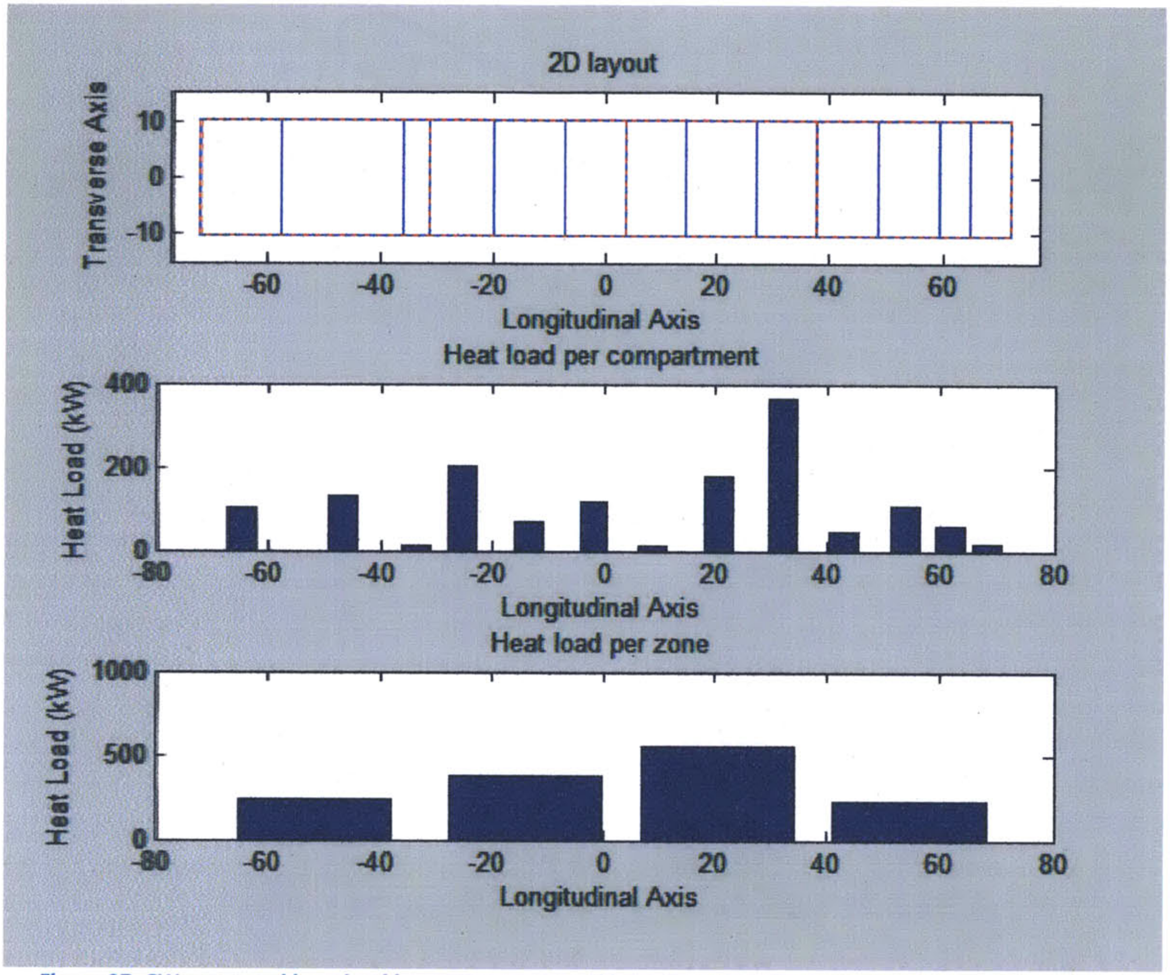


Figure 37: CW zones and heat load by compartment and by zone with 4 zones and default zonal boundaries

With the heat load plots, the user can make a better decision on how many zones may be needed and where to terminate each zone so that the heat loads in each zone are relatively close in magnitude. If new zonal boundaries are provided by the user, the zonal boundaries must be entered as an array starting from the FP and proceeding aft. After the user provides the number of zones (or accepts the default) and provides new zonal boundaries (or accepts the default) the program shows the final heat loads in each compartment and within each zone. An example is shown in Figure 38 below with the number of zones changed to five and the zonal boundaries redefined to produce a more even distribution of the total heat load within each zone.

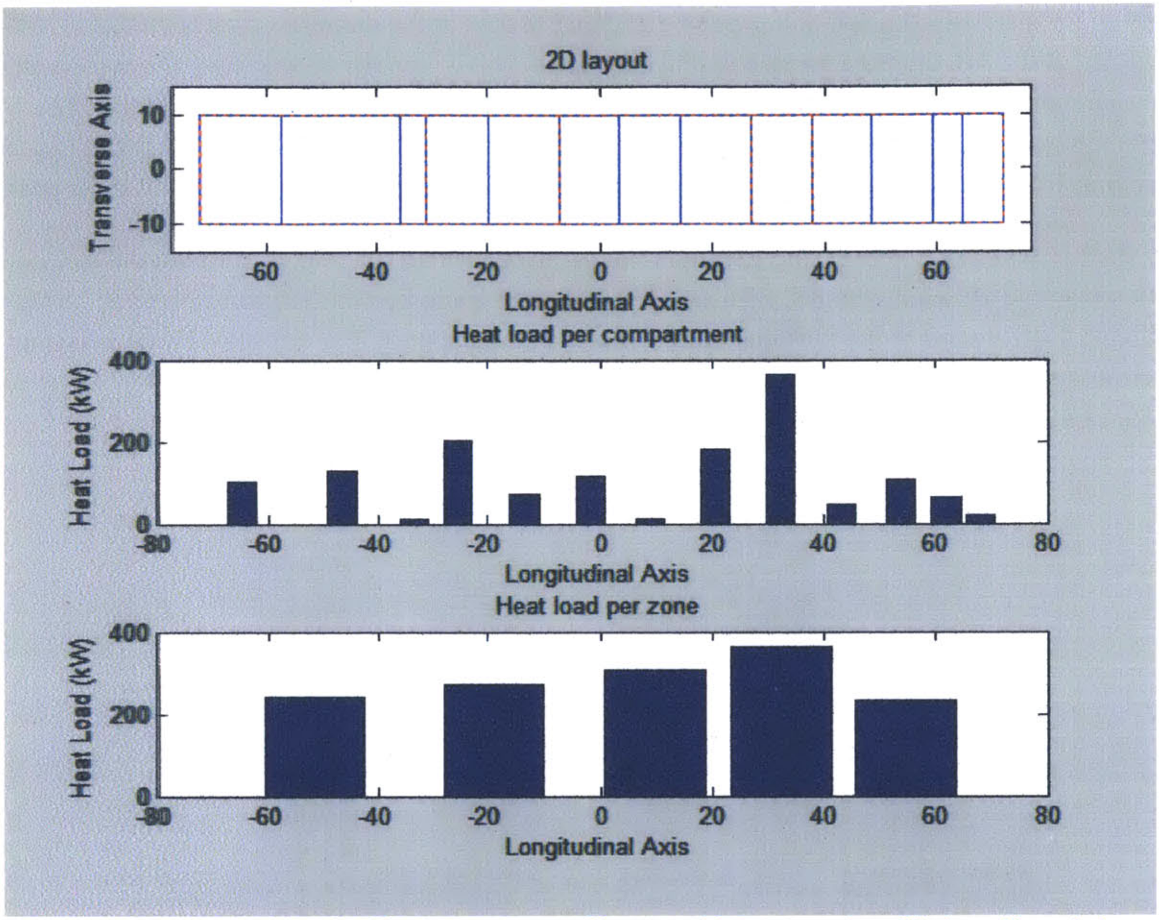


Figure 38: CW zones and heat load by compartment and by zone with 5 zones and modified zonal boundaries

The number of A/C units needs to be defined by the user next. The program provides default values which are dependent on the main piping configuration chosen. If a single main piping system is being designed, then the default number of A/C units is one per zone. If a double main piping system is being designed, then the default number of A/C units is two per zone (one port and one starboard). These default values are the minimum number of A/C units that can be installed for the number of zones chosen. If the user wishes to change the number of A/C units, then they must provide an array with the number of A/C units in each zone starting with the forward-most zone and proceeding aft.

To determine if the A/C units will fit within the compartments, the user is prompted to select the type of chiller to be used. First, the program provides a list of A/C units available from the A/C unit database within the Excel Spreadsheet. The categories include: centrifugal, reciprocating, screw, and other. By default, the program considers all types of A/C units and selects the A/C unit closest in capacity that satisfies the cooling needs within that zone. If the user wishes to select the type of A/C unit to be used, then a pop-up menu would appear which provides the user with the categories available. The program will then only consider the type of A/C unit selected by the user when designing the chilled water system. The user does not have the ability to select one type of A/C unit in one zone and another type of A/C unit in another zone.

The program then uses the A/C unit type selected by the user or the default setting to make an initial estimation of the A/C unit size needed within each zone. This is done by determining the total heat load within each zone and dividing that total evenly by the number of A/C units within that zone. The program then looks within the A/C unit database for the A/C unit that most closely meets the capacity calculated. The dimensions of that specific A/C unit are then used by the program for sizing purposes.

Once the A/C unit size is known, the A/C units can be placed within the ship. By default, the A/C unit is positioned in the aft-most compartment which is large enough to fit it in each zone. The A/C unit is positioned 1 m forward of the bulkhead and on the engineering deck. The transverse locations of the A/C units are dependent on the number of A/C units within the zone. Below are the default transverse locations of the A/C units for each case.

Number of A/C units per Zone	A/C Unit	Default Transverse Location
1	1	0
2	1	beam/4
	2	-beam/4
3	1	beam/4
	2	0
	3	-beam/4
4	1	beam/4
	2	-beam/4
	3	beam/4
	4	-beam/4
5	1	beam/4
	2	0
	3	-beam/4
	4	beam/4
	5	-beam/4
6	1	beam/4
	2	0
	3	-beam/4
	4	beam/4
	5	0
	6	-beam/4

Table 10: Default transverse A/C unit locations

For the case of 4 A/C units per zone, the first two A/C units are positioned in the aft-most compartment that can fit them and the remaining two are positioned in the adjacent compartment forward. For the cases of five or six A/C units per zone, the first three A/C units are positioned in the aft-most compartment that can fit them and the remaining A/C units are positioned in the adjacent compartment forward.

Although it is possible to have other configurations other than one pump to one A/C unit, this is the only option available by the program. The default location of the pump is 1 m forward of the A/C unit. This location cannot be modified by the user.

The program provides a plan view of the ship (treated as a rectangle with dimensions LOA x beam) with the transverse bulkheads, the chilled water zones, and the A/C units and pumps placed using the default locations mentioned above. If the user is satisfied, they can proceed through the design of the chilled water system; otherwise, the user has the ability to modify the A/C unit locations. The locations correspond to the center of the A/C unit. The locations have to be entered as a matrix starting from the forward most A/C unit portside working towards starboard, then aft. An example of the format is shown below.

$$[x_1 \quad y_1 \quad z_1; x_2 \quad y_2 \quad z_2; x_3 \quad y_3 \quad z_3; \ldots; x_n \quad y_n \quad z_n]$$

After the A/C unit locations have been identified, the program creates the structure of the main piping system. The structure includes connections from the A/C unit to the pump, then a riser section, the supply header, cross-connections (if a double main piping system), the return header, the return riser, and a connection to the A/C unit. Also, a recirculation line across the pump is modeled. This structure is created for each A/C unit. The program then outputs a plan view of the main piping structure including pumps and A/C units, and also a 3-dimensional representation of the main piping structure. The 3-dimensional representation can be zoomed in and out and can be rotated along all three axes. 2-D and 3-D examples of the main piping structure for each main piping layout using all default parameters are shown below in Figures 39-44.

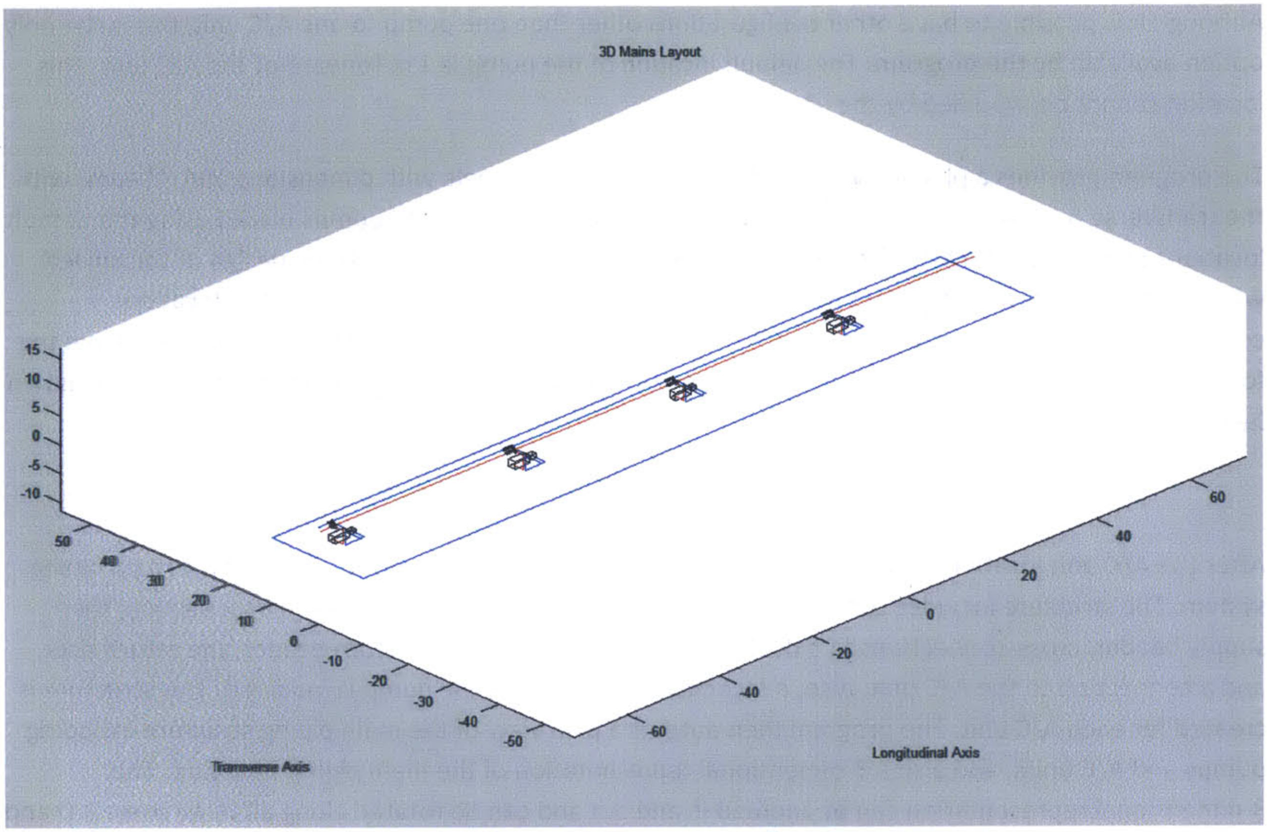


Figure 39: Default single main piping configuration 3-D

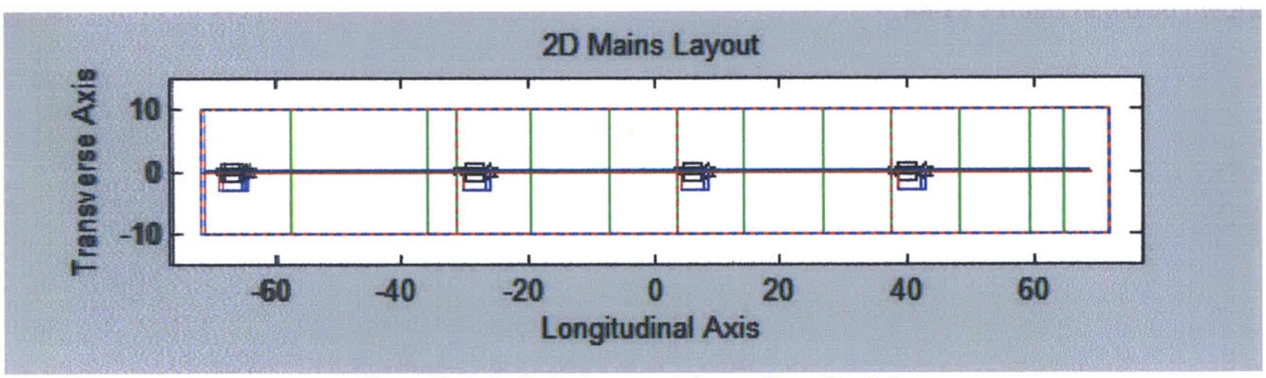


Figure 40: Default single main piping configuration 2-D

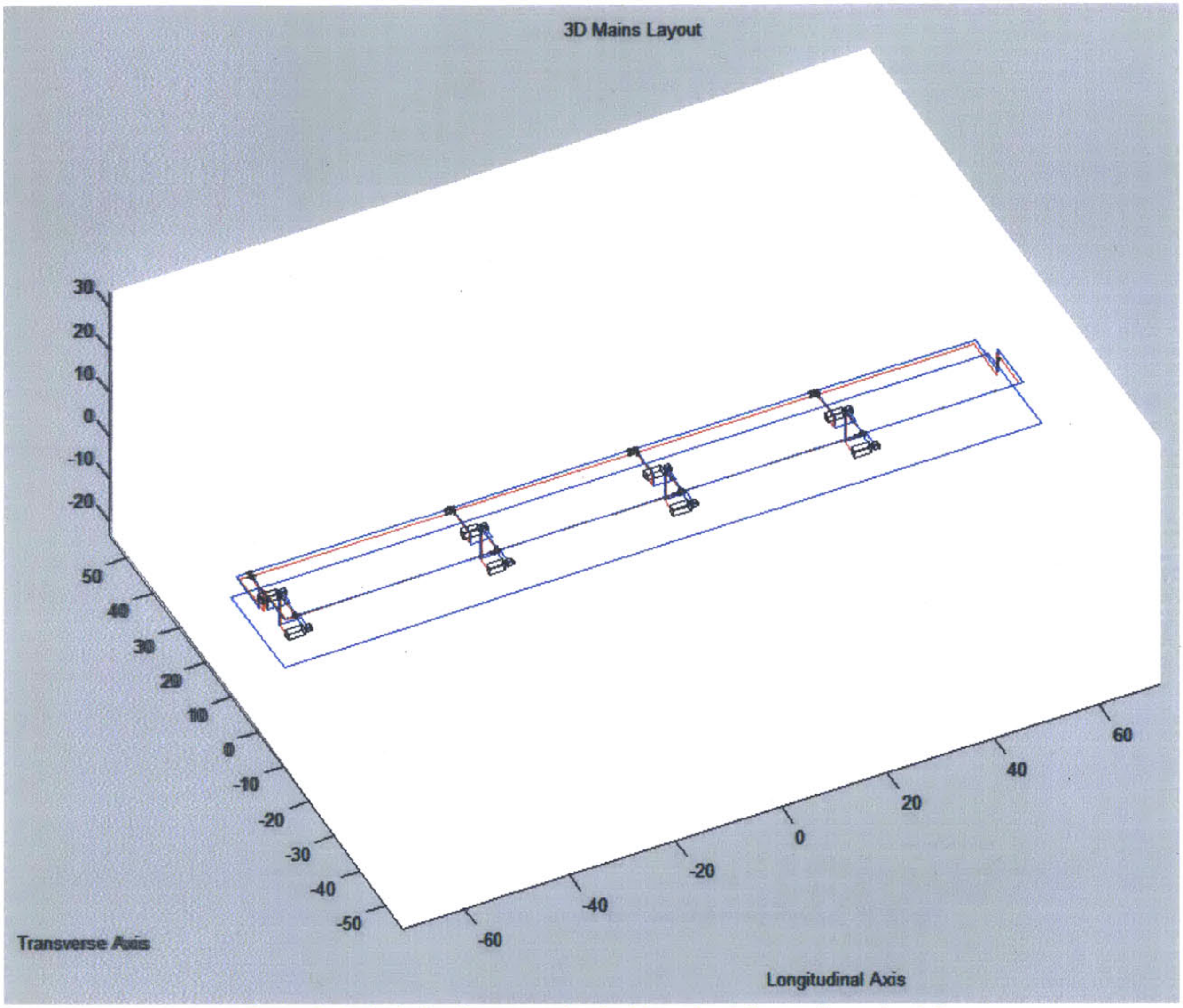


Figure 41: Default simple rectangular double main piping configuration 3-D

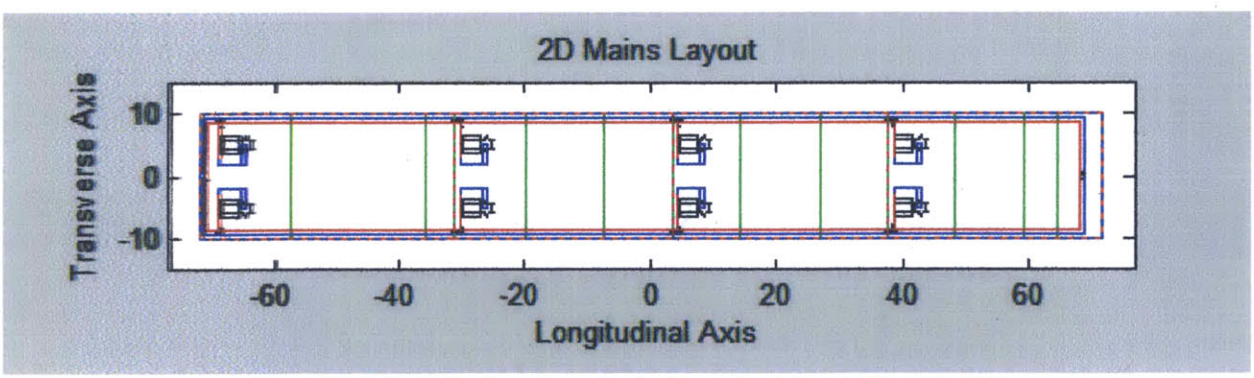


Figure 42: Default simple rectangular double main piping configuration 2-D

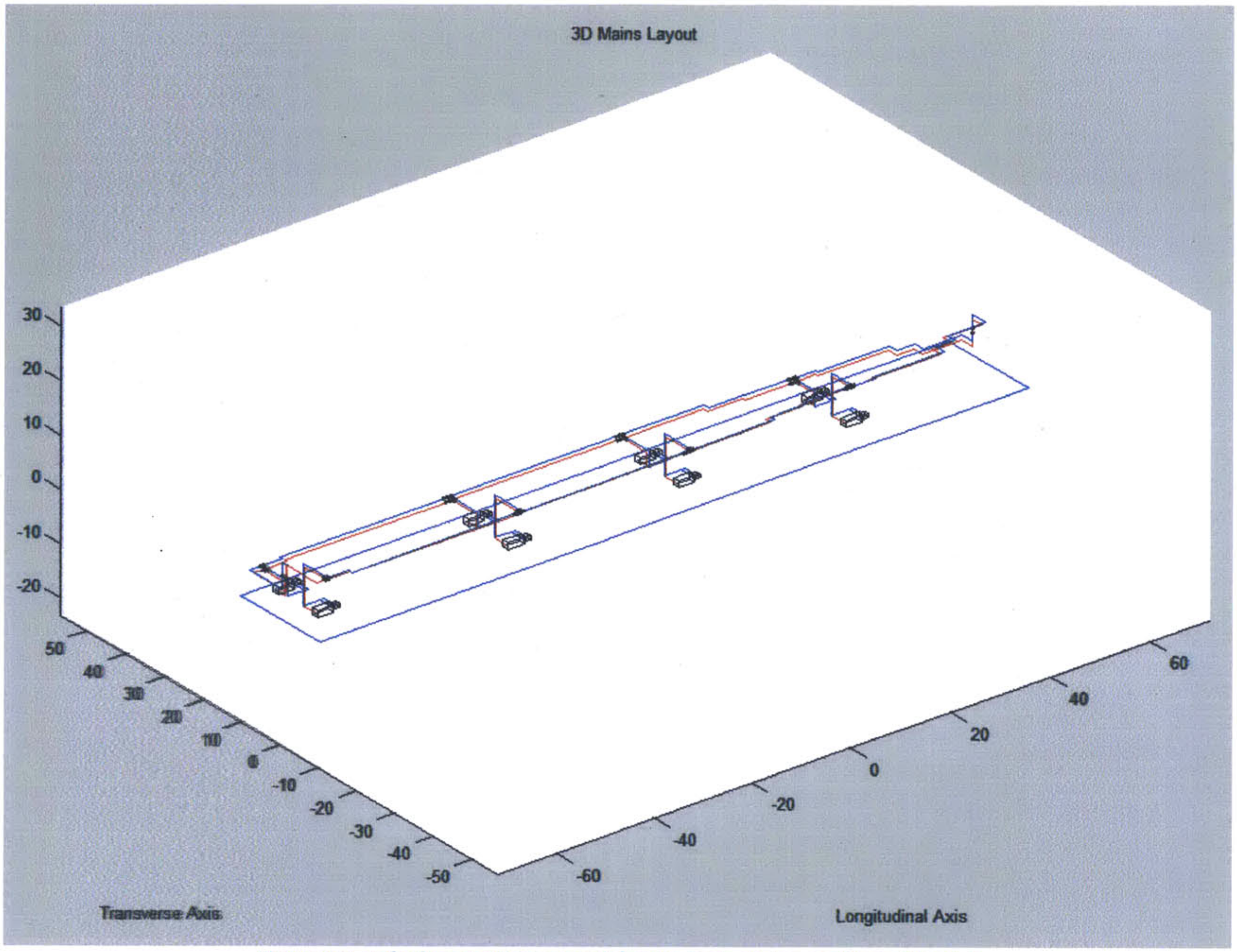


Figure 43: Default complex tapered double main piping configuration 3-D

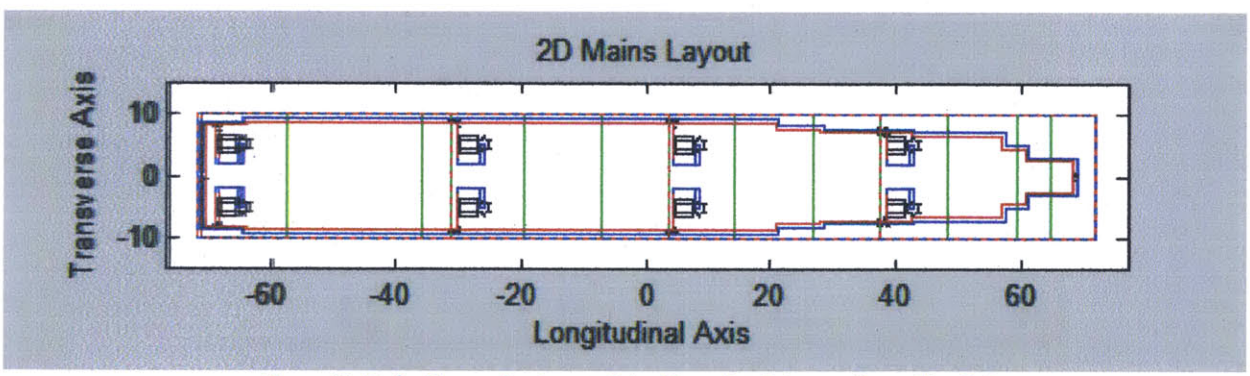


Figure 44: Default complex tapered double main piping configuration 2-D

Isolation valves are created by the program and added to specific locations within the main piping system. There are three isolation valves at the junction where the riser connects to the header. One isolation valve is located 1 ft forward of the junction. Another isolation valve is located 1 ft aft of the junction. A third isolation valve is located 2 ft from the end of the riser section. This configuration is repeated for each supply and return riser junction. In addition, isolation valves are located on either side of a bulkhead separating CW zones for both the supply and return headers. This may result in some redundancy occurring in some spots with isolation valves in close proximity to one another. Modifications may be made through the optional Modification module. This requires extensive knowledge of the CSDT program and the associated variables, but with careful programming, changes can be made as to the number of isolation valves and their locations. Below, Figure 45 shows a close-up of the isolation valves at the junction of the risers and supply and return headers as well as the isolation valves located on either side of a bulkhead separating two CW zones.

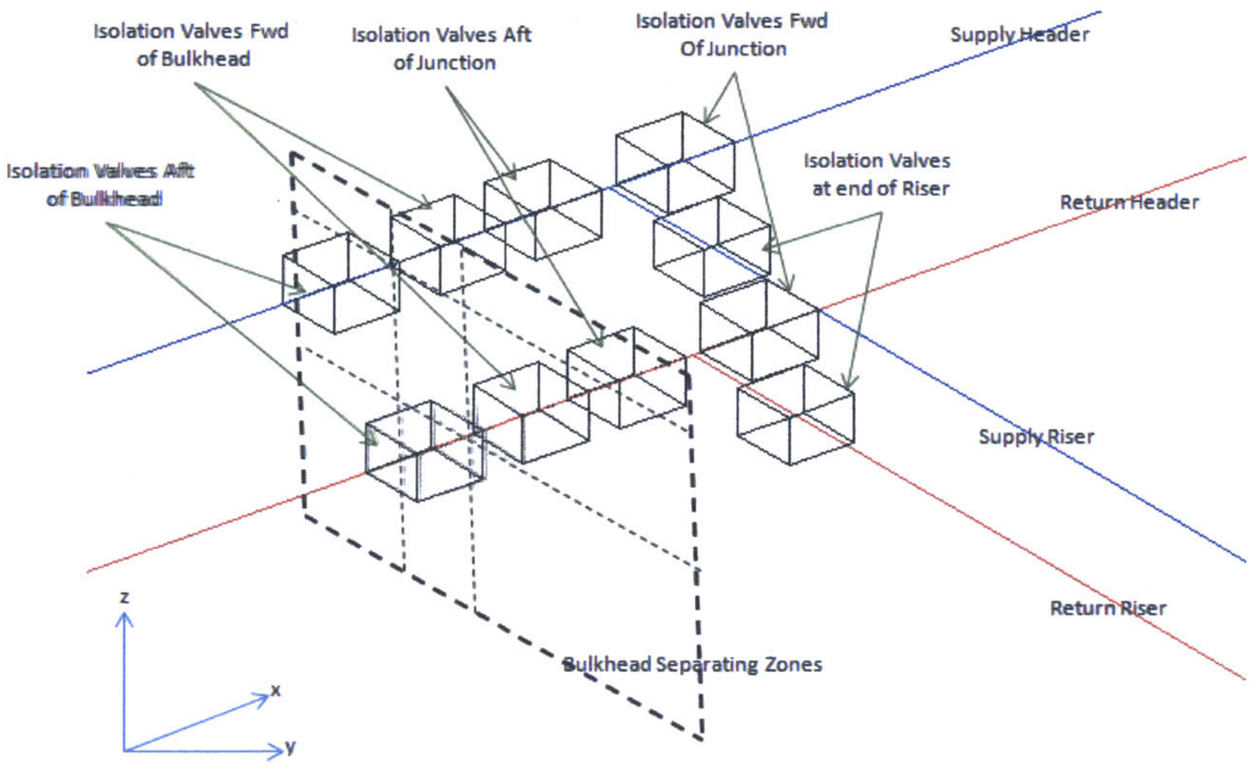


Figure 45: Isolation valve placement at main piping junctions and zonal boundaries

In addition to the isolation valve placement mentioned above, two isolation valves are placed at the athwartships cross-connection for the double main piping systems, one each for the supply and return headers.

A check valve is placed downstream of each chilled water pump to prevent flow going in the wrong direction and damaging the pump. The branch piping structure is then created. The vital/non-vital status of the branch piping determines if there is only a single path from the heat load to the main piping system or of there is a redundant path. The vital/non-vital status is determined by the program by reading in the priority of the heat load from the Excel Spreadsheet. I f the priority is less than three, the heat load is considered vital, else, it is considered non-vital. The set-point between vital and non-vital status can be changed by the user as well and the minimum priority could be greater than 8 if the user wishes to add more fidelity in the load priority.

Each branch is considered to be in parallel with all other branches. Although this is not true in all cases, it was too complex to create a structure that was generic enough to allow for series/parallel branch configurations. The development for a more generic approach that allows for series/parallel branch structures is a potential area for future work.

Since each heat load is in parallel with all other heat loads, each heat load has its own dedicated branch piping which connects it to the main piping supply and return headers. The branch structure is stored as a 10x2x3x180 matrix. The 10 in the matrix corresponds to 10 points describing the start, bends, and end of each branch. The 2 in the matrix corresponds to the primary and secondary branch for each heat load. For vital loads, there will be a branch in each of these indices; however, for a non-vital branch, there will not be a branch in the second index. The 3 in the matrix corresponds to the x,y,z coordinates of either the start, end or bend of a branch. Lastly, the 180 is variable depending on the number of heat loads listed in the excel spreadsheet. The case study utilized 180 loads taken from CSDT v1.0 (Fiedel, 2011).

The program creates the branch structure automatically. The structure is dependent on the location of the heat load with respect to the main piping structure. The simplest case is that of the single main piping structure. All heat loads within the longitudinal extent of the main piping system is connected at the same longitudinal location of the supply and return headers. The branch piping is created starting at the x-location of the heat load on the supply header. It continues along vertically up to the vertical location of the heat load. The pipe then continues transversely up to the heat load. The pipe is then offset vertically by the branch offset distance specified earlier and is connected to the return header in the reverse fashion, accounting for the branch offset distances as well as the header offset distances. An example of the branch connection is shown in Figure 46 below.

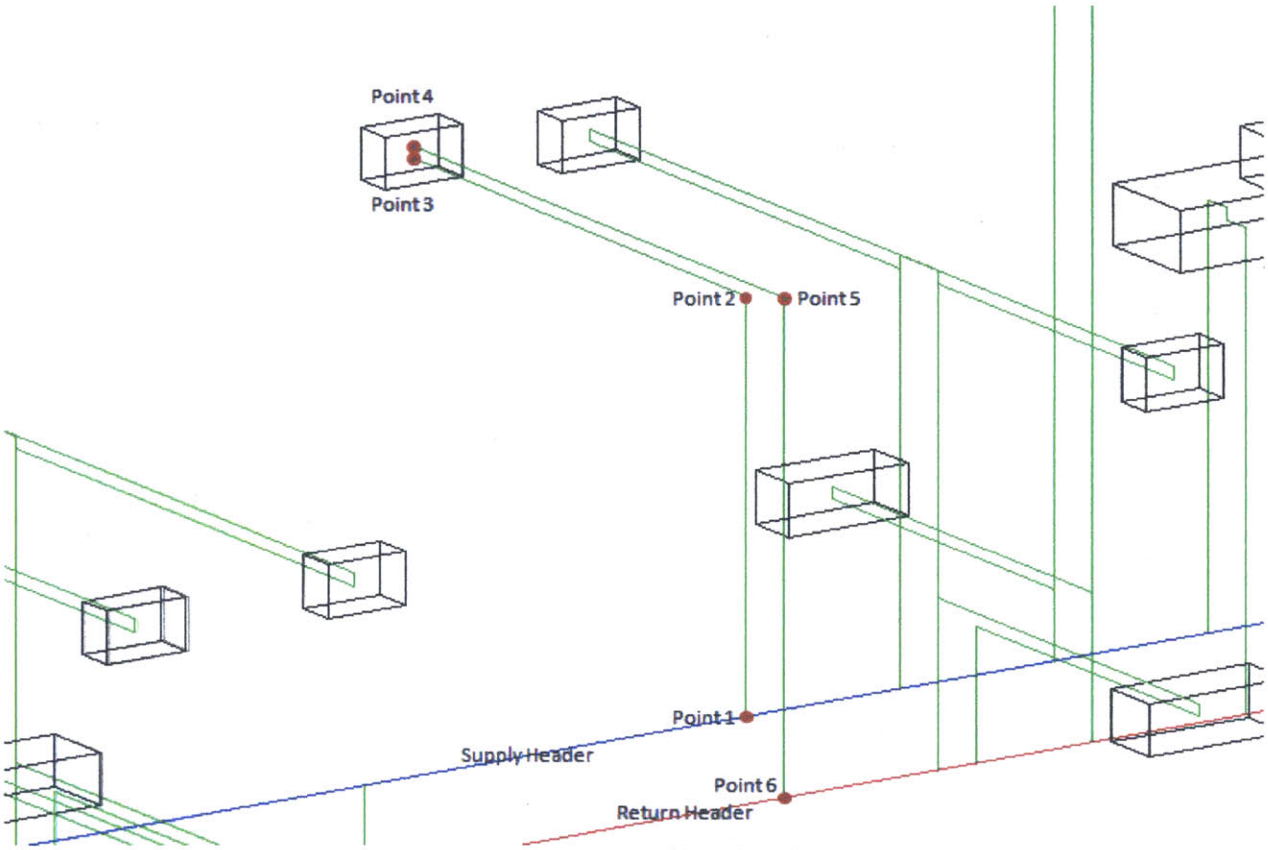


Figure 46: Formation of branch piping

For heat loads outside the longitudinal extent of the main piping, there would be additional bends in the branch piping, but the principle is the same.

For the double main configurations, there is a choice as to which supply and return header piping the heat load could be connected to. As mentioned before, if the heat load is vital, it would be connected to both, else it would be connected to the closest header. If the heat load is on the centerline of the ship, then it is connected to the starboard side by default.

Each branch also includes two gate valves and a globe valve. The gate valves are positioned one upstream of the heat load and the other downstream of the heat load. The globe valve is positioned downstream of the heat load. The gate valves allow for isolation of the branch in case of a casualty. The globe valve allows for throttling of flow through the branch. The locations of the valves are done automatically and does not allow for user manipulation within the Geometry module; however, the locations can potentially be modified through the use of the Modification module.

After the branch piping structure is defined, the program then sizes each heat load to a heat exchanger. The user input from the excel spreadsheet is used to identify what type of heat exchanger is to be used for each respective heat load. The heat exchanger of the proper type is then chosen based on having sufficient capacity to meet the demands of the heat load. The heat exchanger characteristics are then read in and stored in an array of arrays. One of the arrays includes the dimensions of the heat exchanger. This is used to properly size the heat exchangers in the subsequent plots.

At this point, the chilled water system is largely defined. The chilled water system interfaces with the seawater system through the A/C units. The next step of the program is then to model a generic seawater auxiliary system.

The program begins by locating four auxiliary seawater pumps. The four default locations are:

$$\left[\pm 0.3LOA \quad \pm 0.8\frac{Beam}{2} \quad EngDeckHtAboveKeel + \frac{SWPumpHt}{2}\right]$$

Seawater isolation valves are located in close proximity to the AUX SW pumps, two upstream and one downstream.

The piping of the AUX SW system is comprised of a connection from the sea chest to the pump then a riser section which forms a tee junction with the AUX SW supply header. There are two supply headers that run fore-aft. The supply headers are located port and starboard and are offset vertically to maintain vertical separation for survivability considerations. The two AUX SW supply headers are connected by two cross-connects. Figure 47 below shows the structure of the AUX SW piping created by the program.

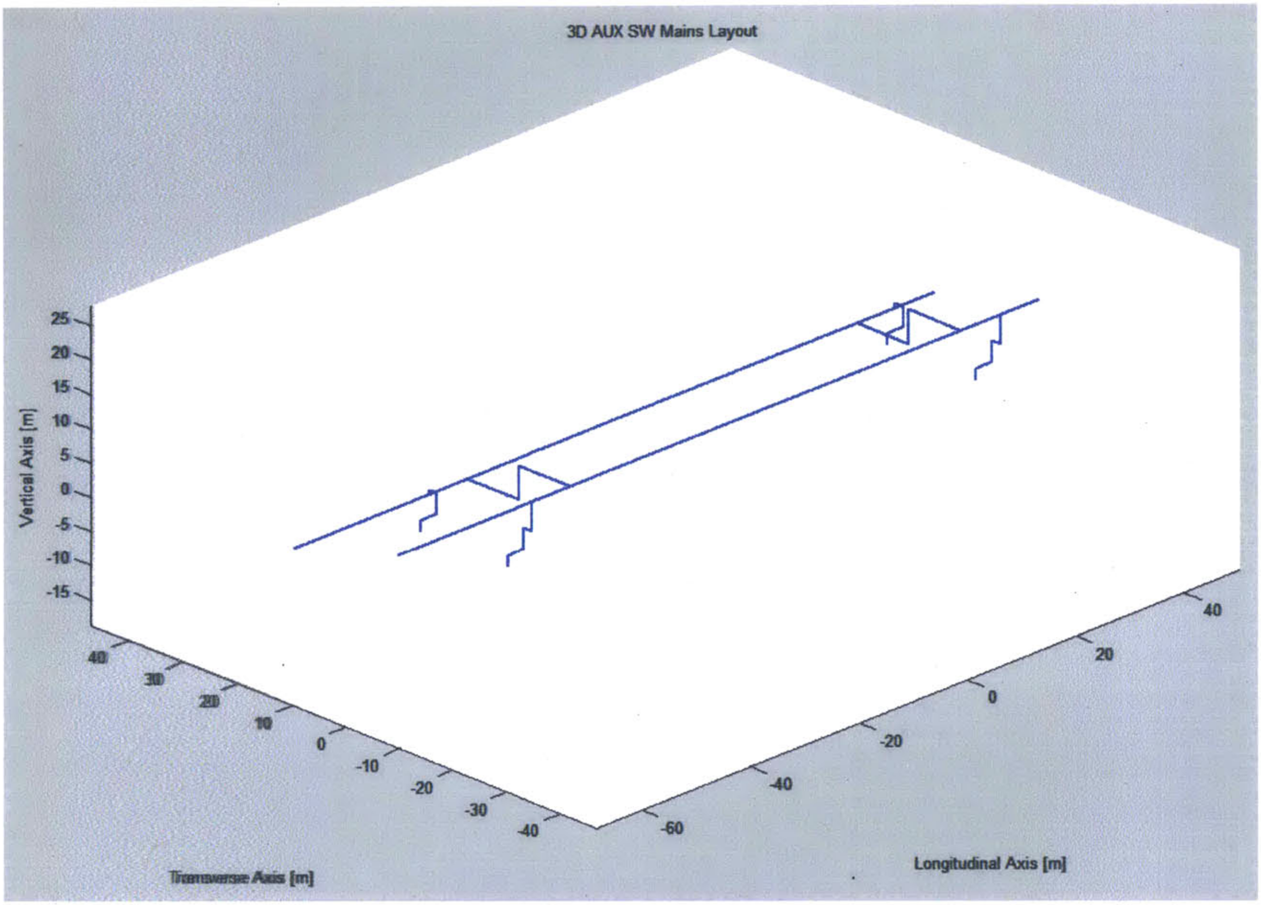


Figure 47: AUX SW piping structure

The AUX SW supply headers supply the seawater to the AUX SW branches. Each A/C unit has its own dedicated AUX SW branch. In addition, the user can specify the locations of other heat exchangers of the form SW/XX. The SW/XX heat exchangers also have their own dedicated AUX SW branch. Lastly, the user has the ability to specify if the shaft bearing is accounted for. If it is, the user specifies the location of the shaft bearing or uses the default value. The user also specifies the gpm flow rate to the shaft bearing and any SW/XX heat exchangers being accounted for in the design of the AUX SW system. Figure 48 below shows the AUX SW piping including the branch piping to the A/C units, a shaft bearing located at [-57.4 0 2.4]and two SW/XX heat exchangers located at [20 3 10] and [-30 -2 15].

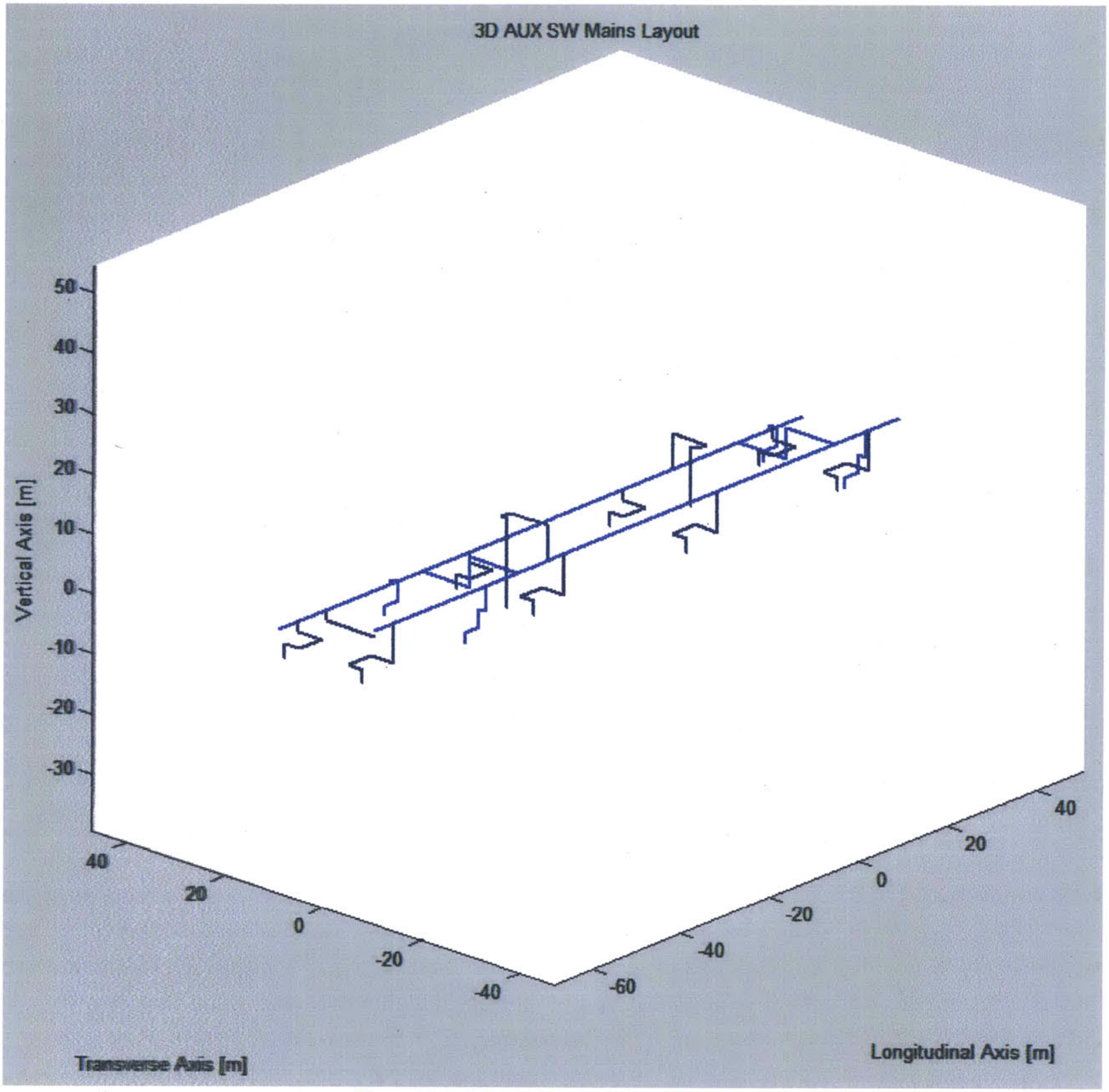


Figure 48: AUX SW piping with branch piping

The program outputs a 3-dimensional model of the chilled water system up to this point. The structure of the main piping system is included along with the A/C units, the pump, the structure of the branch piping system, the various check, gate, and globe valves, as well as the heat exchangers centered at the location of the heat load. The AUX SW system is also included in the plot. Examples of the plan and perspective views of the 3-dimensional chilled water models up to this point are shown in Figures 49-54 below.

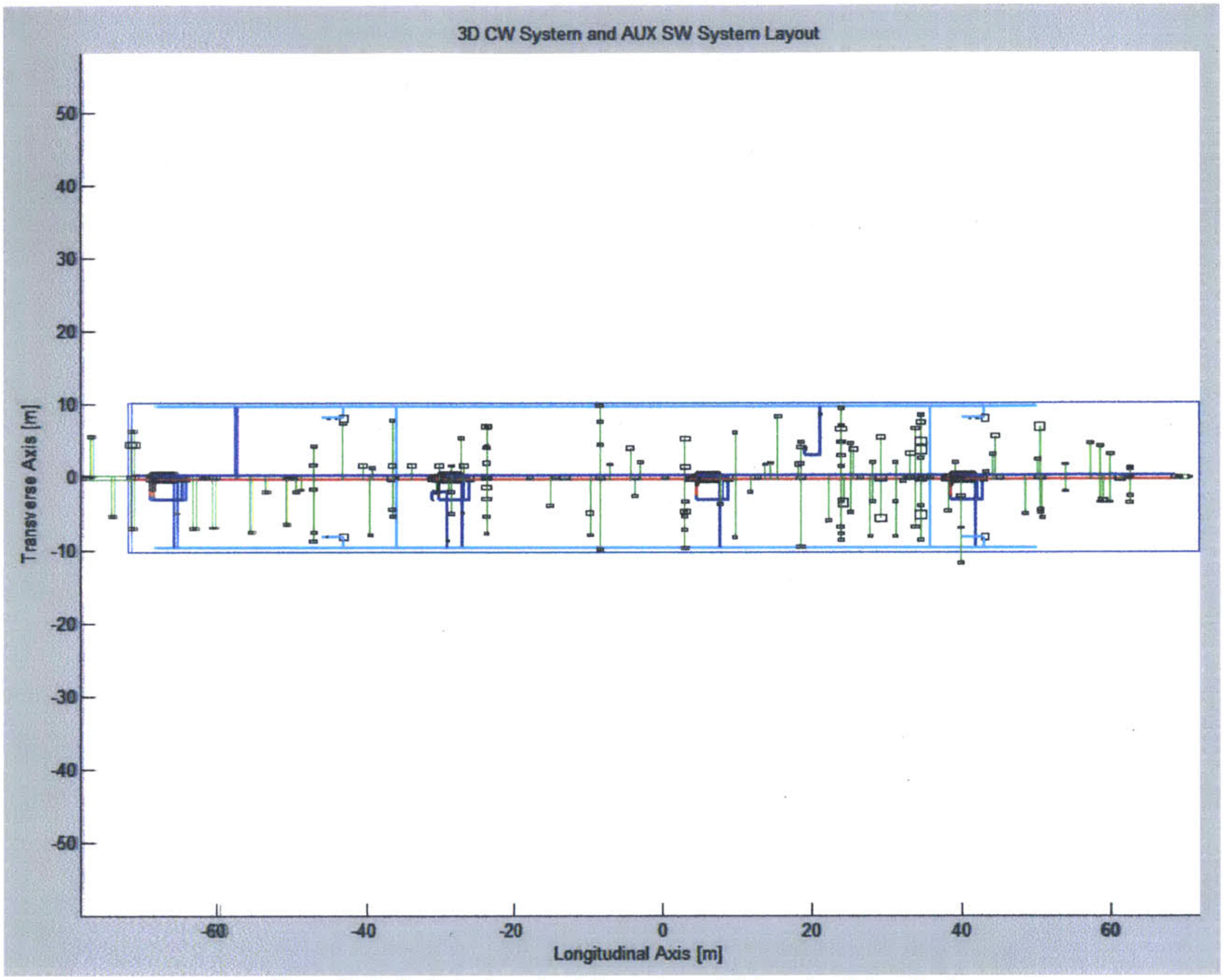


Figure 49: Default single main piping system with branches (plan view)

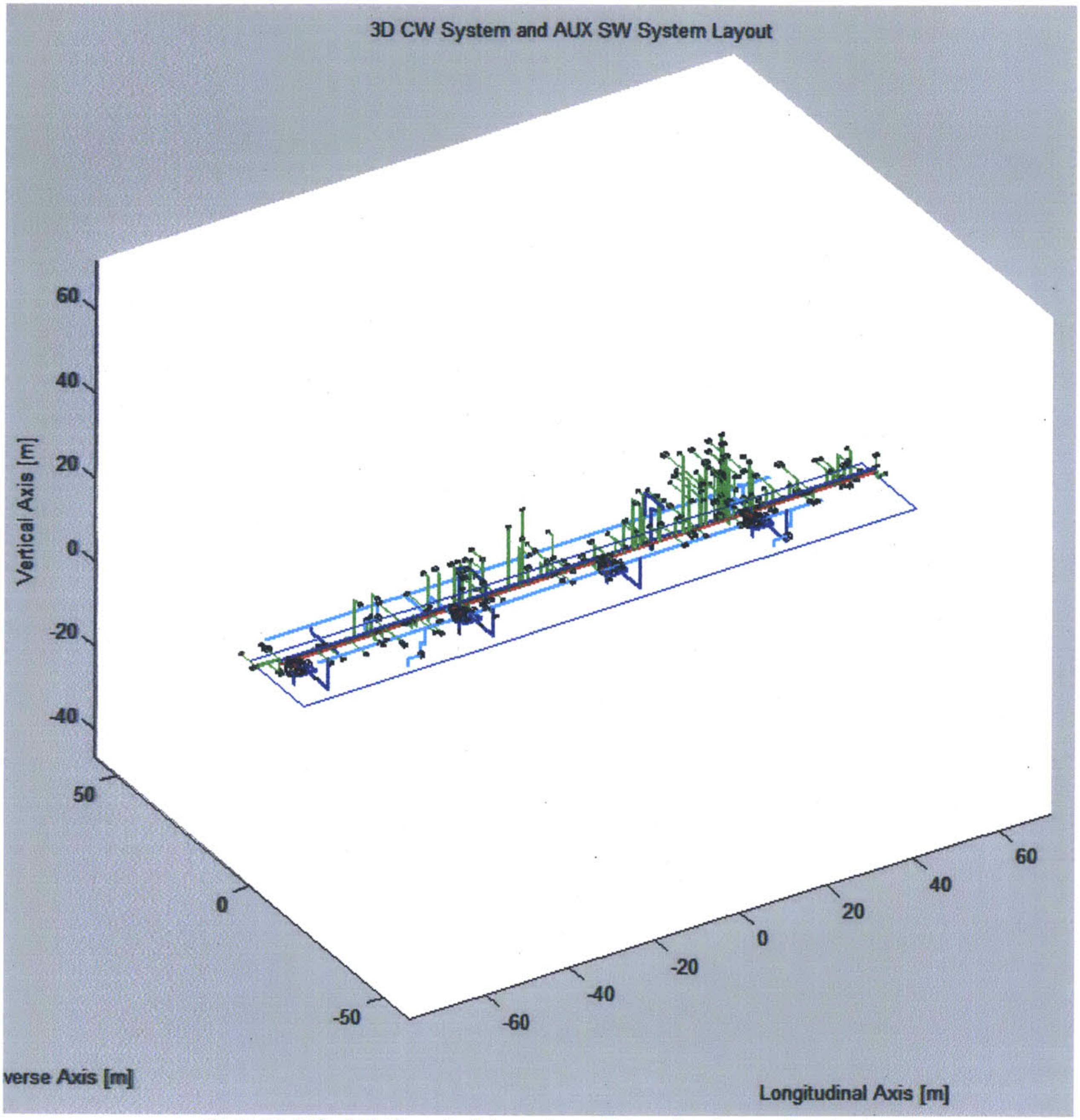


Figure 50: Default single main piping system with branches (perspective view)

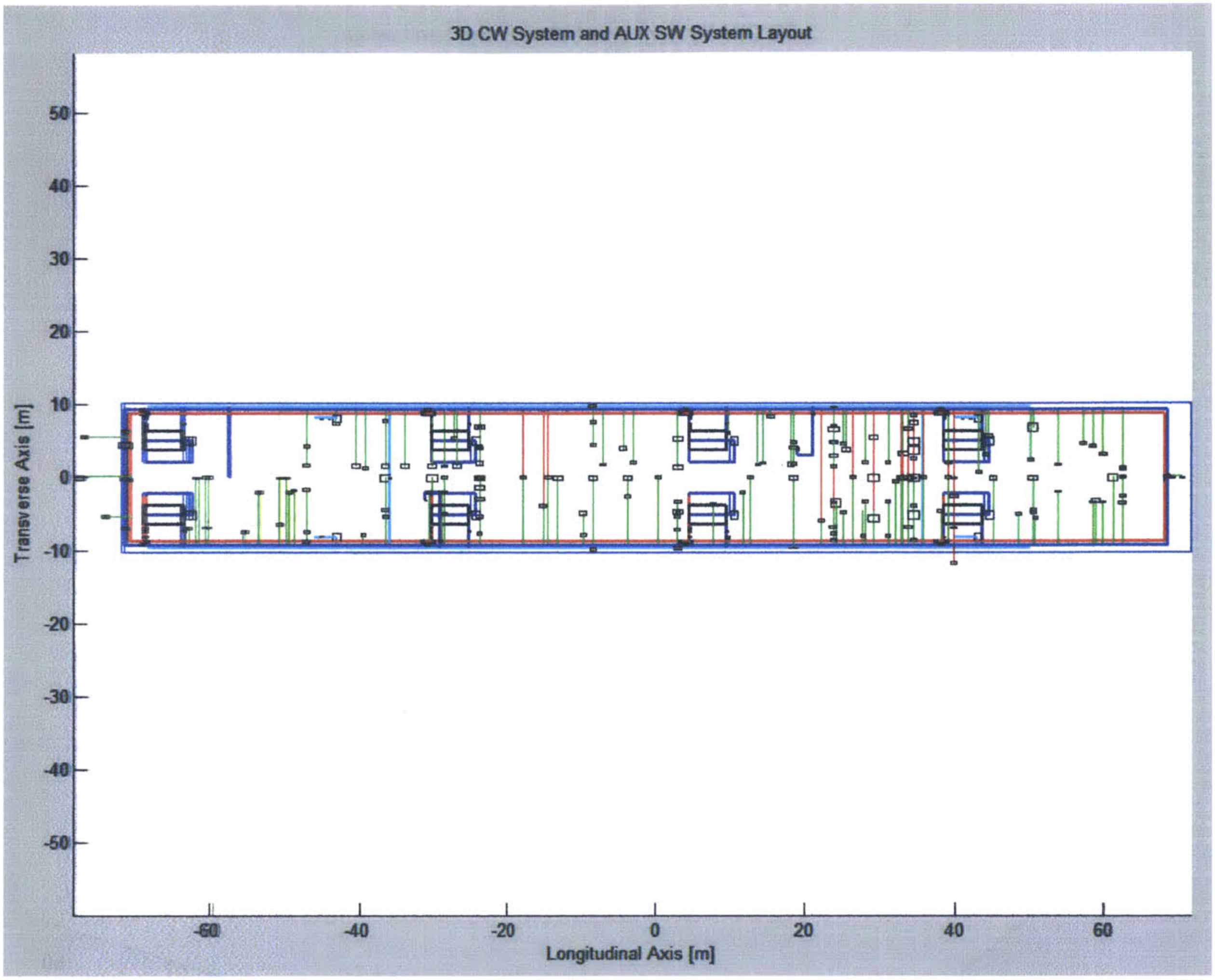


Figure 51: Default simple rectangular double main piping system with branches (plan view)

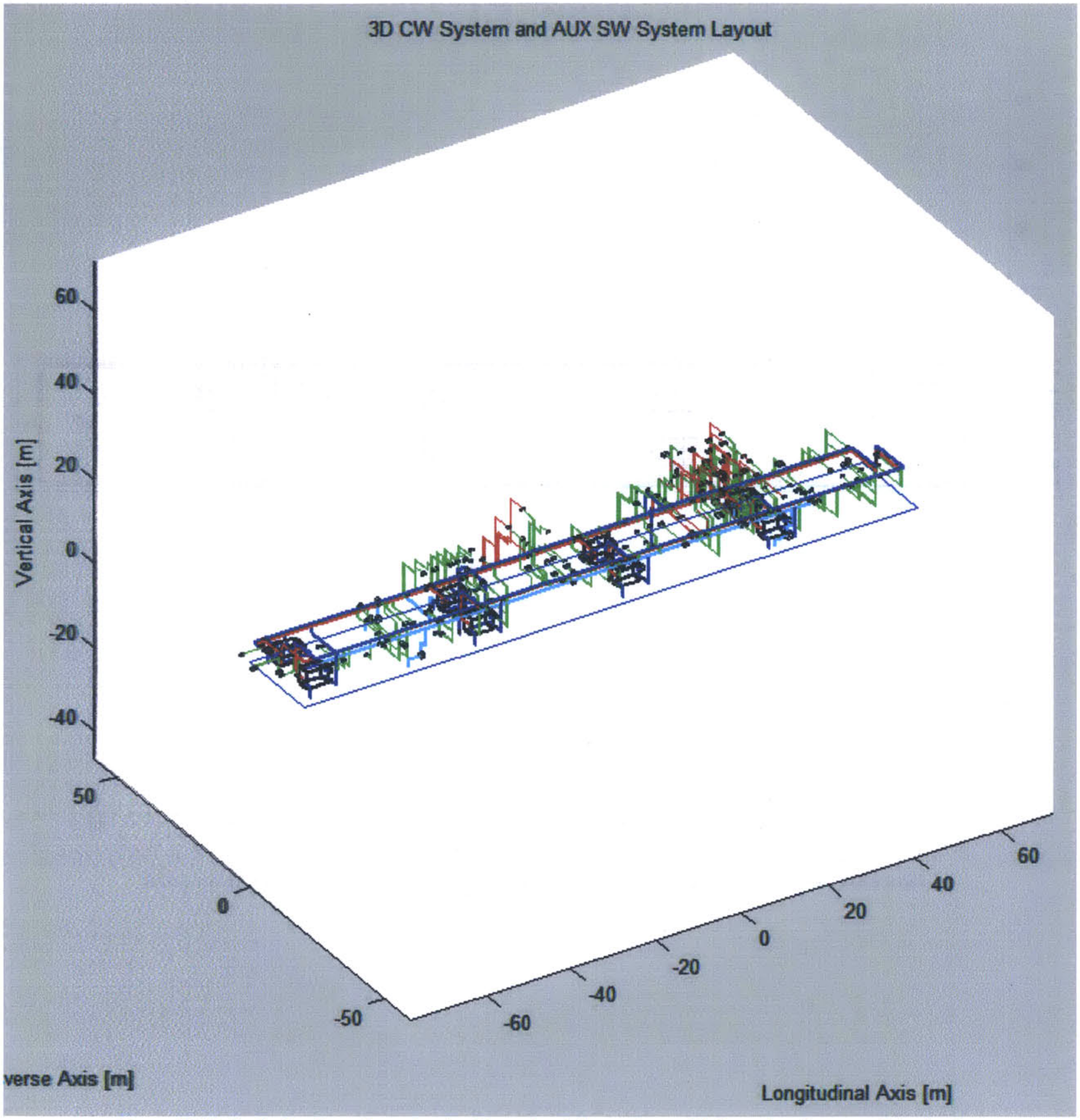


Figure 52: Default simple rectangular double main piping system with branches (perspective view)

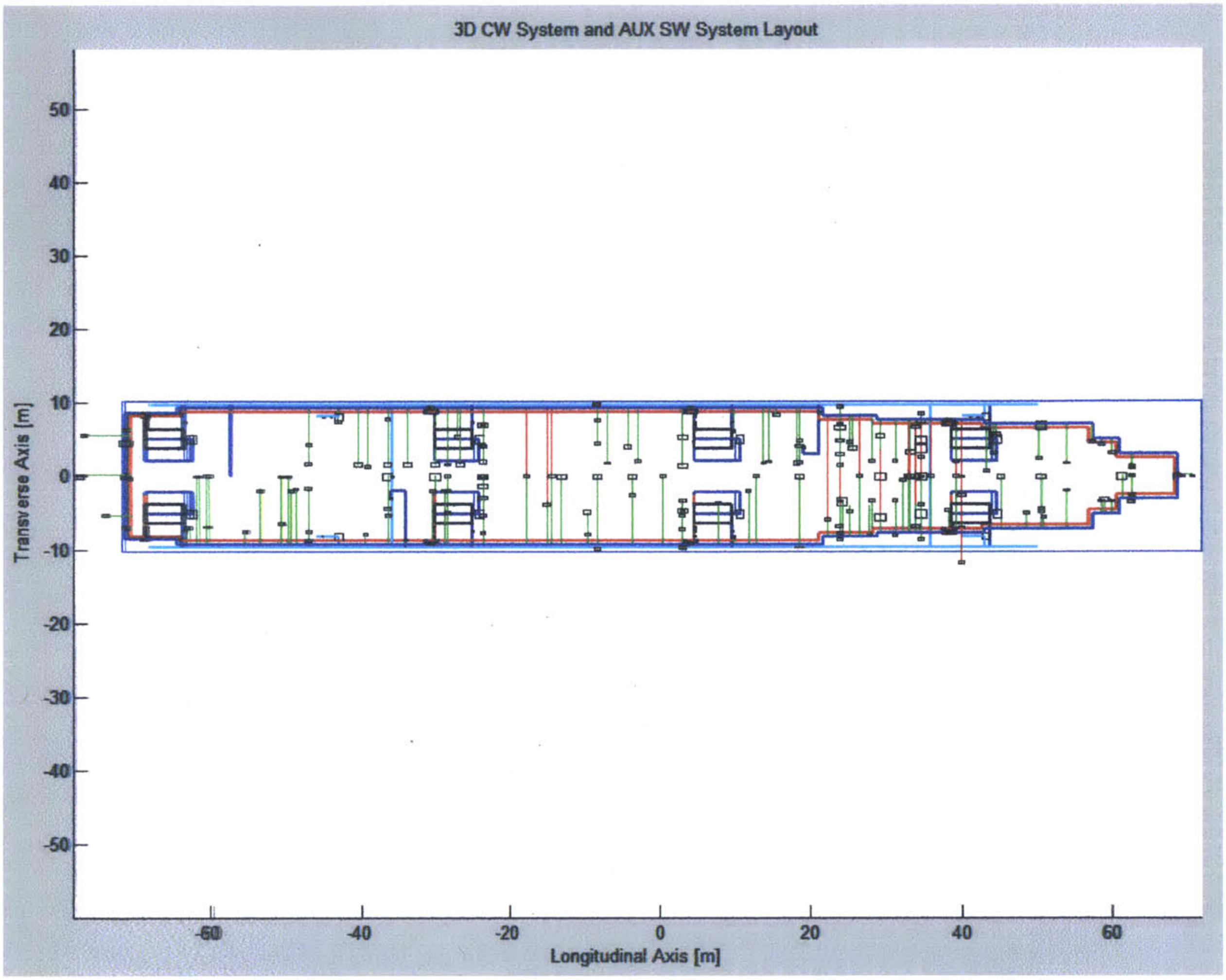


Figure 53: Default complex tapered double main piping system with branches (plan view)

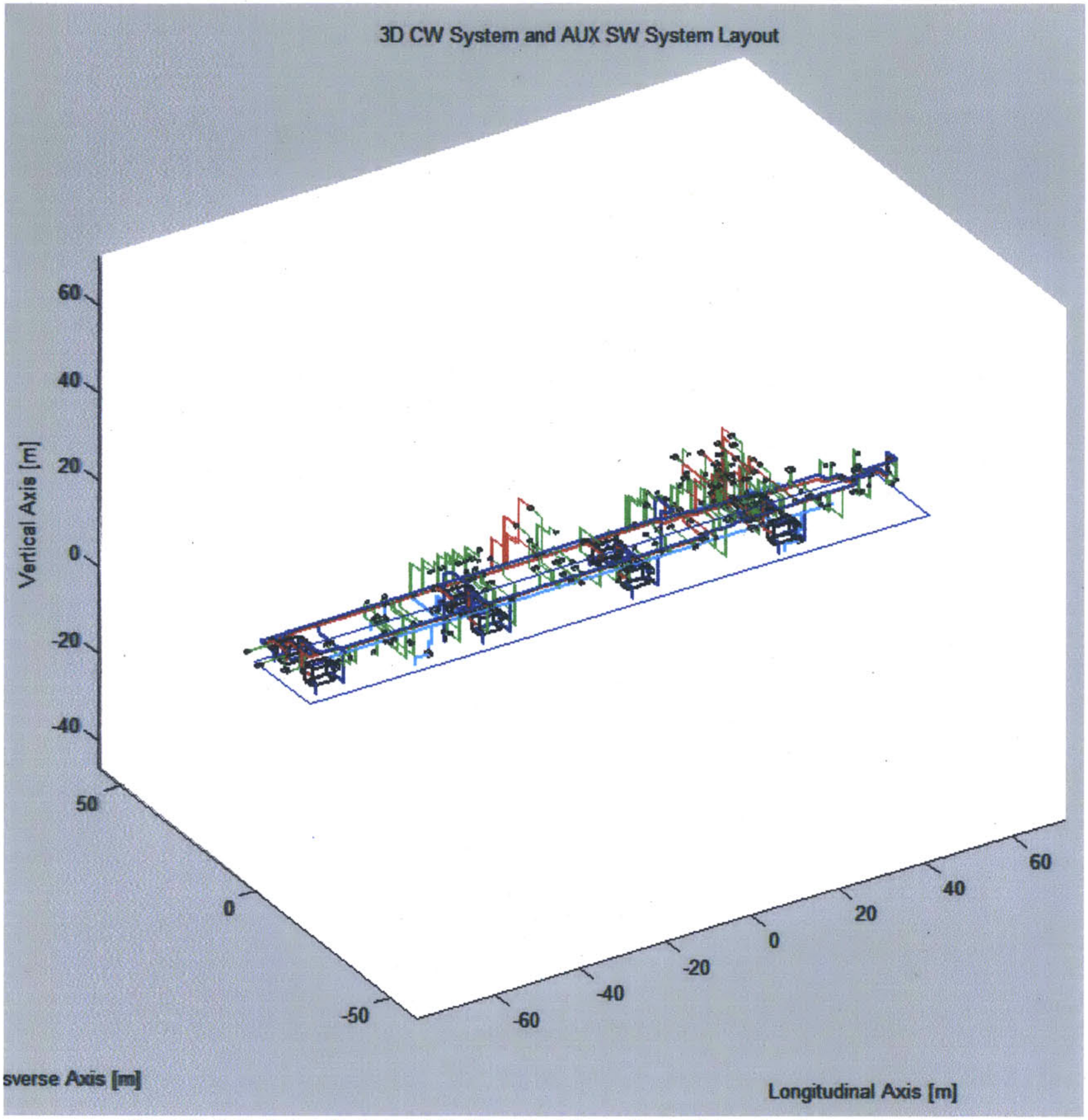


Figure 54: Default complex tapered double main piping system with branches (perspective view)

There is a lot represented in the above plots. To discern what is shown, the simple rectangular double main piping system with branches is shown in greater detail in the preceding figures. Figures 55-58 identifies each of the components in the 3-D plot of the CW/SW systems.

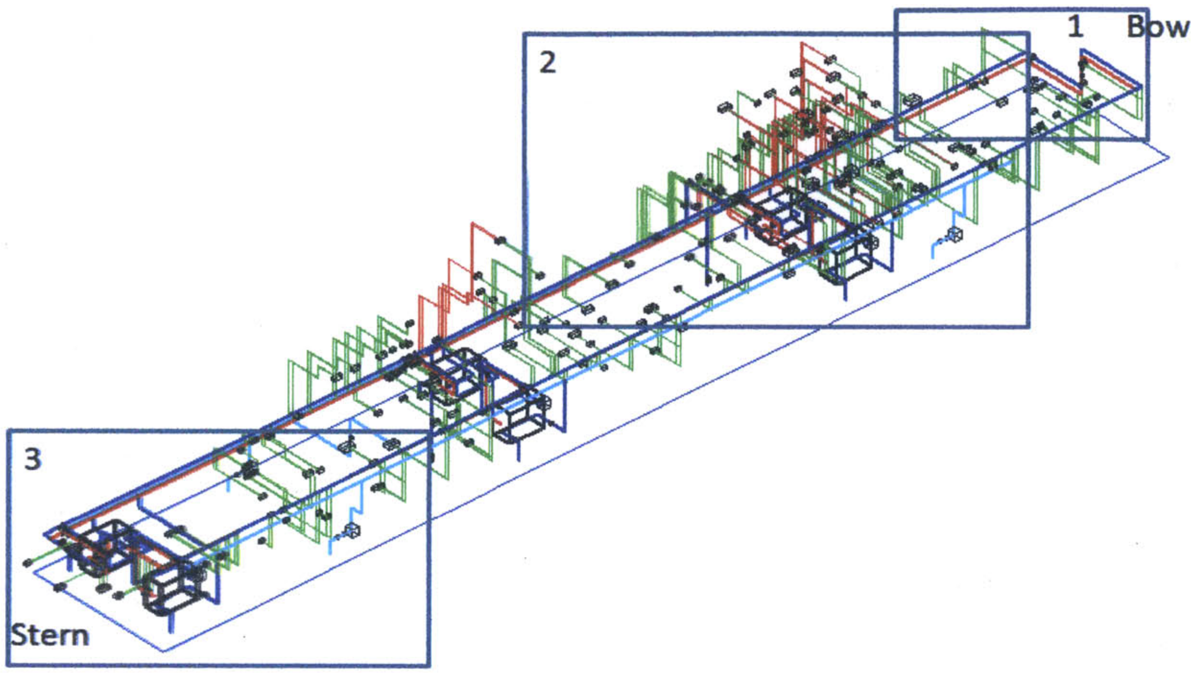


Figure 55: Chilled water system segmented into areas 1, 2 and 3

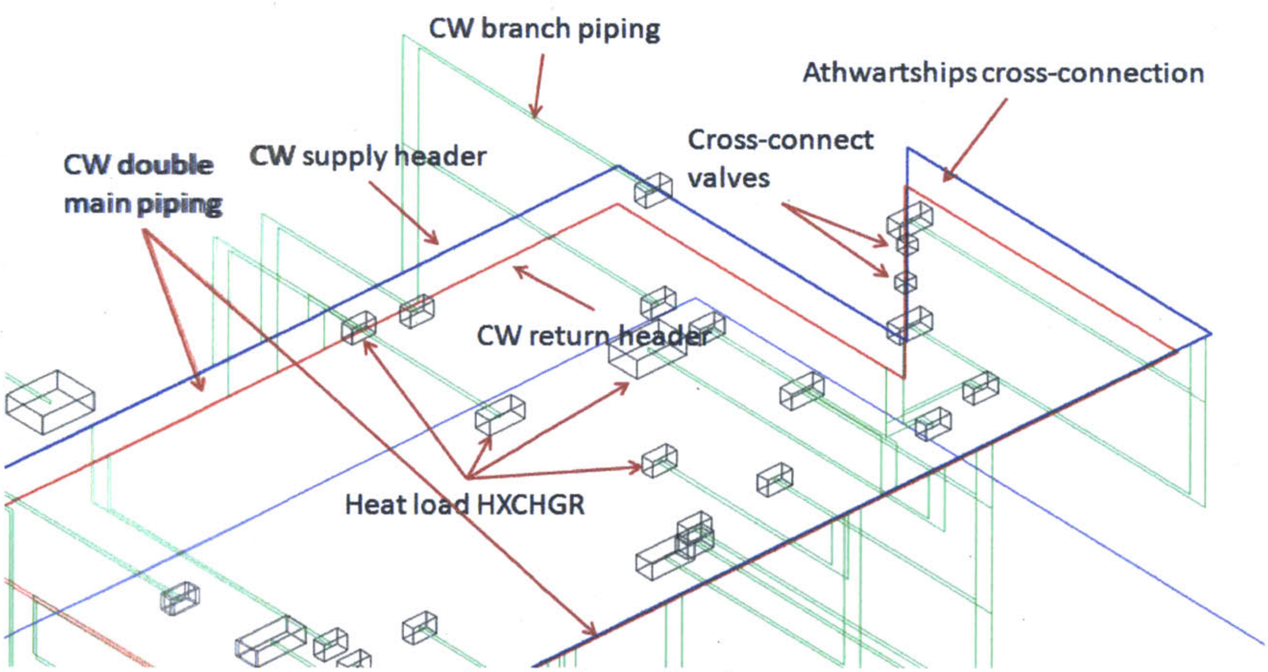


Figure 56: Close-up view of area 1

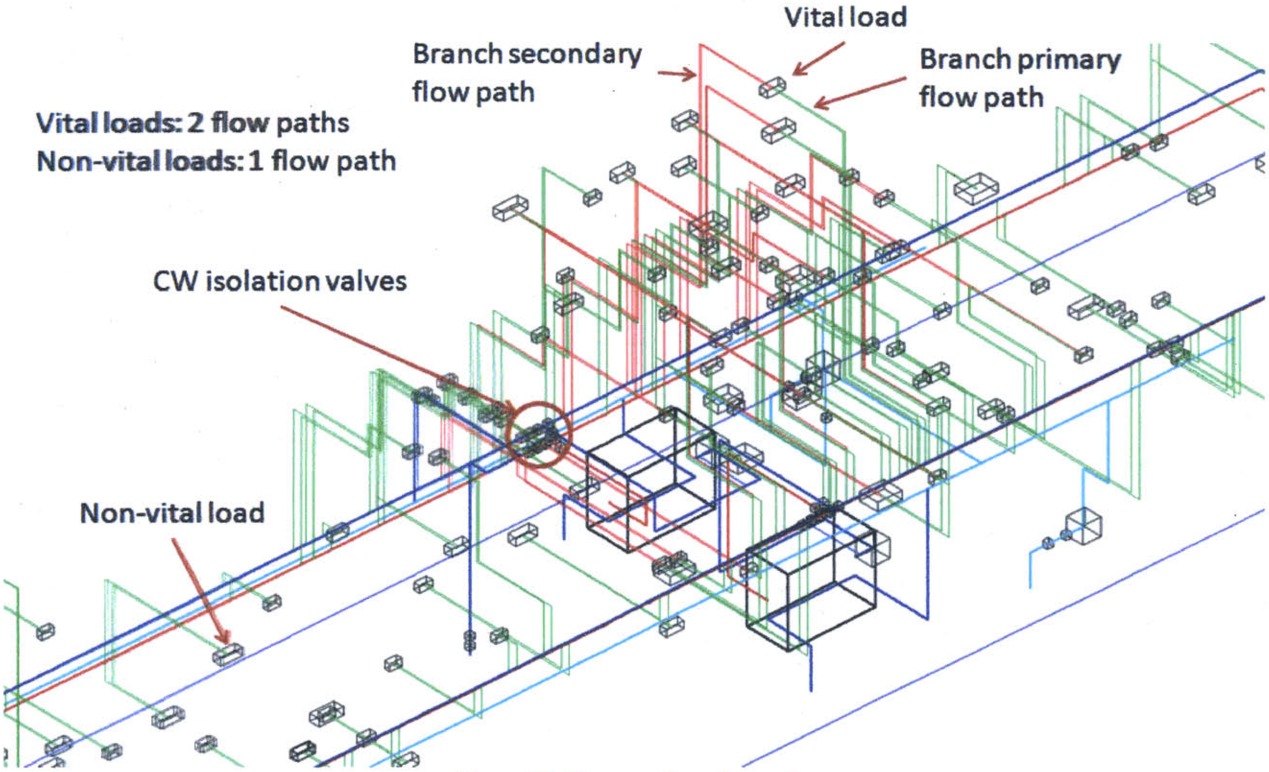


Figure 57: Close-up view of area 2

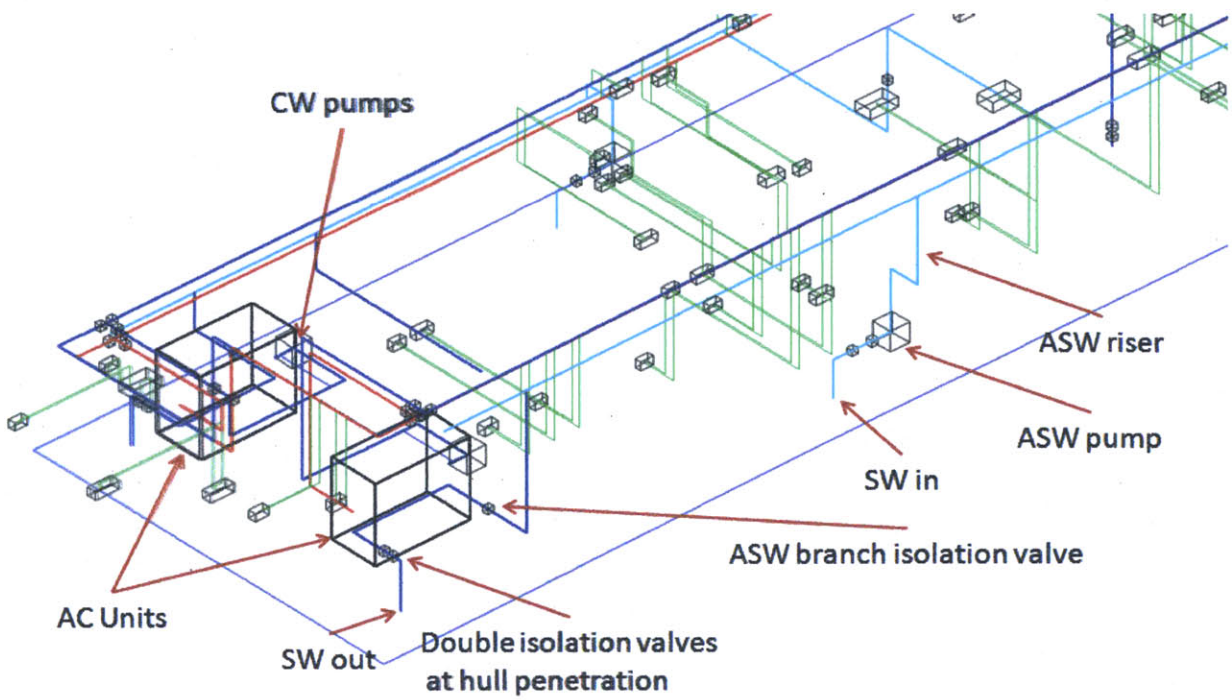


Figure 58: Close-up view of area 3

The program does not have the capability to modify the branch piping, such as the ability to route the branch piping around major pieces of machinery/equipment, etc. This level of refinement would have to be done in another program such as Paramarine (with an interface program between Matlab and Paramarine needed) or done through the use of the Modification module, but will require extensive knowledge of the program and programming expertise. Also, the number of hull penetrations are fixed based on the default AUX SW geometry created and the number of A/C units in the CW design (four for SW inlet and n number for SW outlet based on n number of A/C units included in the CW design). Further refinement could also be pursued in this area, allowing for grouping and placement of sea chests to minimize hull penetrations.

At this point, the program has gathered most of the user inputs required to design the initial layout of the chilled water system and auxiliary seawater system. The remaining portion of the program analyzes the system designed to determine the feasibility/performance of the system.

3.2 Analysis

As mentioned earlier, the Matlab program is broken up into two major modules. The first module utilized the user inputs to design the chilled water system and create the chilled water structure. The second module includes the analysis of the chilled water system modeled and is quite extensive. The analysis focuses on calculating the weight, the static temperature distribution, the temperature distribution and temperature response during transients of the chilled water system. This is accomplished through a structured process as summarized below:

- Step 1: Preliminary sizing of pipe diameters and preliminary calculation of branch velocities and branch mass flow rate based on heat load
- Step 2: Determination of network segments
- Step 3: Refining branch velocities and branch mass flow rates using network analysis accounting for head loss associated with bends, friction, and across valves
- Step 4: Account for entrance and exit effects utilizing refined branch velocities
- Step 5: Determination of pressure drop as a function of distance
- Step 6: Determination of stagnation points
- Step 7: Final calculation of velocities and mass flow rates using network analysis with network isolated at the stagnation points
- Step 8: Calculate branch inlet temperatures
- Step 9: Determination of A/C unit capacity required and selection of A/C units
- Step 10: Expansion tank sizing
- Step 11: Weight Analysis
- Step 12: Static Temperature Analysis
- Step 13: Transient Temperature Analysis

Each of the steps listed above is described in greater detail in the proceeding sections.

3.2.1 Step 1: Preliminary Sizing of Piping Diameters and Preliminary Calculation of Branch Velocities and Branch Mass Flow Rates Based on Heat Load

Using the same approach as within CSDT v1.0, the branch piping diameter was found parametrically using the equation:

$$D = \left(\frac{4KQ}{C\pi}\right)^{0.4}$$

Equation 62 (Fiedel, 2011)

where K is 4.5 gpm/ton, Q is the heat load [tons], C is 4 ft/(sec-in$^{0.5}$), and D is the inner pipe diameter. This gives a reasonable diameter to begin analyzing the chilled water system. The diameters are then rounded up to the nearest diameter listed in Table 2 and Table 3 along with the corresponding pipe thickness.

At this point, the inlet temperature (the temperature of the chilled water entering the heat exchanger) is assumed to be equal to 6.67°C. The branch mass flow rate is also assumed to equal 4.5 gpm/ton. These two initial conditions are not entirely accurate, but provide a starting point for the program and are later updated. With these initial conditions, all other conditions across the heat exchangers are found. Figure 59 shows the corresponding temperatures.

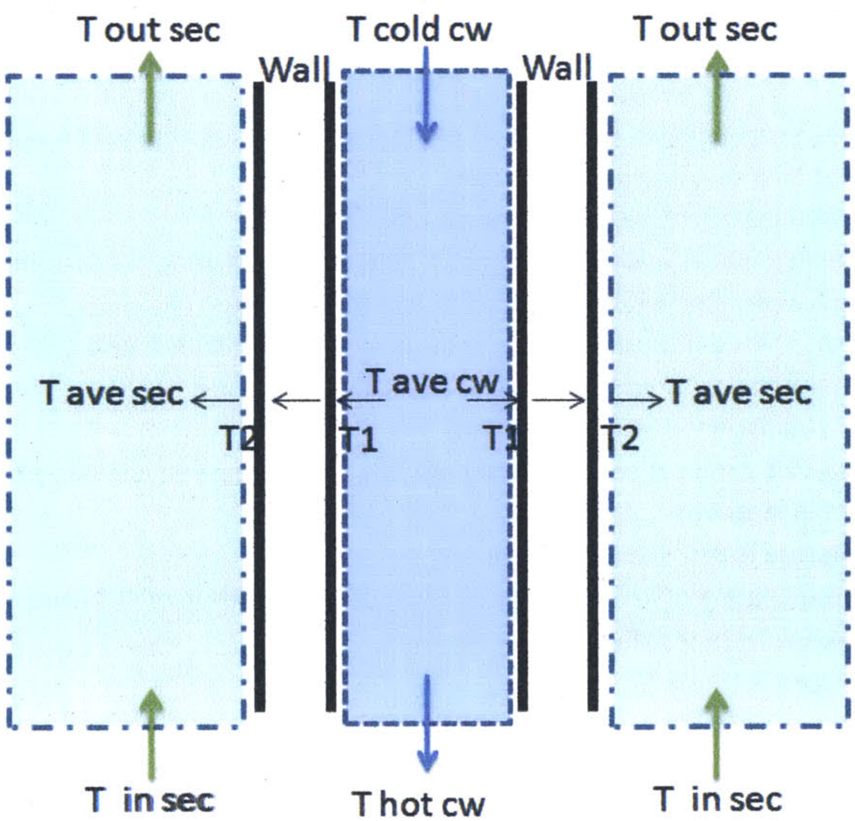


Figure 59: Temperatures within heat exchanger

A second simplifying assumption is the temperature distribution for cross flow. The temperature distribution from inlet to outlet on the primary and secondary side would resemble that shown in Figure 60below, but is simplified as a linear rise and fall.

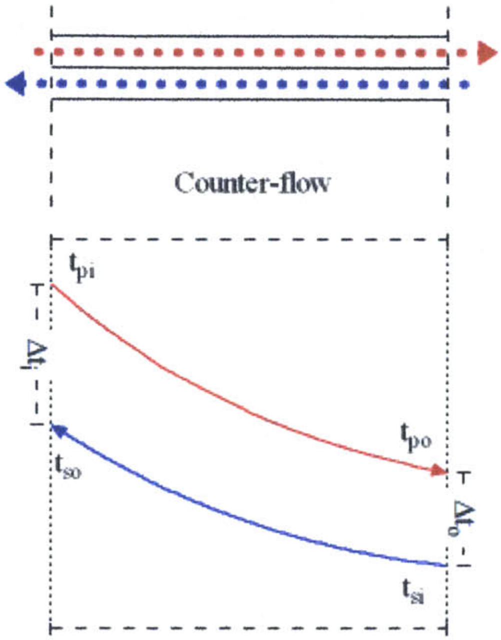


Figure 60: Temperature distribution for counter-flow (Engineering Toolbox)

The outlet temperature (the temperature of the chilled water exiting the heat exchanger) is found using the equation:

$$T_{out} = \frac{\dot{Q}}{\dot{m} \cdot c_p} + T_{in}$$

Equation 1 (repeated, rearranged)

where $\dot{Q}$ is the heat load [W], $\dot{m}$ is the mass flow rate of the chilled water in the branch [kg/s], c_p is the specific heat capacity of the chilled water, taken to be 4203 J/kg-K, and T_{in} is the inlet temperature of the chilled water [C].

The temperature at the inner wall of the heat exchanger is found using the equation:

$$T_1 = \frac{(T_{out} + T_{in})}{2} + \frac{\dot{Q}}{SA_{hxchgr_{inner}} \cdot h_{c_{cw}}}$$

Equation 63

where $SA_{hxchgr_{inner}}$ is the inner surface area of the heat exchanger [m^2], and $h_{c_{cw}}$ is the convective heat transfer coefficient of the chilled water [W/m^2-K]. The convective heat transfer coefficient is found as described in Chapter 2.

For heat exchangers with a tubular boundary (i.e., cooling coil), the temperature at the outer wall of the heat exchanger is found using the equation:

$$T_2 = T_1 + \frac{\dot{Q}\pi D_{hxchgr\ tube}}{SA_{hxchgr_{inner}}} \ln\left(\frac{\left(\frac{D_{hxchgr\ tube}}{2} + t_{hxchgr\ tube}\right)}{\frac{D_{hxchgr\ tube}}{2}}\right)\frac{1}{2\pi k_{hxchgr}}$$

Equation 64

where $D_{hxchgr\ tube}$ is the inner diameter of the tubes within the heat exchanger [m], $t_{hxchgr\ tube}$ is the thickness of the tubes within the heat exchanger [m], and k_{hxchgr} is the thermal conductivity of the tubes within the heat exchanger.

For heat exchangers with a slab boundary (i.e., flat plate heat exchanger), the temperature at the outer wall of the heat exchanger is found using the equation:

$$T_2 = T_1 + \frac{\dot{Q}k_{hxchgr}SA_{hxchgr}}{t_{hxchgr\ plate}}$$

where $t_{hxchgr\ plate}$ is the thickness of the plates within the heat exchanger [m].

The thermal conductivity of the heat exchanger tubing is based upon the percentages of copper and nickel in the composition of the pipe. The program assumes the composition of the piping is 90% copper and 10% nickel. Other possible compositions include: 80% copper and 20% nickel, 70% copper and 30% nickel, and 100% copper. The thermal conductivities for each of these compositions are shown in Table 11.

Piping Composition	Thermal Conductivity (W/m-K)
100% Copper	386
90% Copper – 10% Nickel	50
80% Copper – 20% Nickel	30
70% Copper – 30% Nickel	10

Table 11: Thermal conductivities of various copper-nickel compositions

The average temperature of the fluid on the secondary side of the heat exchanger is found using the equation:

$$T_{fluid_{avg}} = T_2 + \frac{\dot{Q}}{SA_{hxchgr_{outer}} \cdot h_{c_{fluid}}}$$

Equation 65

where $SA_{hxchgr_{outer}}$ is the outer surface area of the heat exchanger [m^2], and $h_{c_{fluid}}$ is the convective heat transfer of the fluid on the secondary side of the heat exchanger [W/m^2-K]. The convective heat transfer of the fluid on the secondary side is not computed as it was for the chilled water on the primary side, but instead had to be determined experimentally or by the manufacturer of the heat exchanger and provided to the program via the excel spreadsheet.

The differential temperature across the heat exchanger on the secondary side is found using the equation:

$$\Delta T_{fluid} = \frac{\dot{Q}}{\dot{m}_{fluid} \cdot c_{p_{fluid}}}$$

Equation 1 (repeated, rearranged)

where $\dot{m}_{fluid}$ is the mass flow rate of the fluid on the secondary side of the heat exchanger [kg/s], and $c_{p_{fluid}}$ is the specific heat capacity of the fluid on the secondary side [J/kg-K].

The inlet and outlet temperatures of the fluid on the secondary side are determined by the equations:

$$T_{fluid_{in}} = T_{fluid_{avg}} - \frac{\Delta T_{fluid}}{2}$$

Equation 66

and

$$T_{fluid_{out}} = T_{fluid_{avg}} + \frac{\Delta T_{fluid}}{2}$$

Equation 67

Of course, these temperatures are only valid once the system is in equilibrium and the temperatures reach steady-state.

An example of the various temperatures for each branch is shown in Figure 61 below. The example considers 180 heat loads with the first load equal to 1 MW cooled through a flat plate heat exchanger. All other heat loads are 60 kW or less and are cooled through a cooling coil.

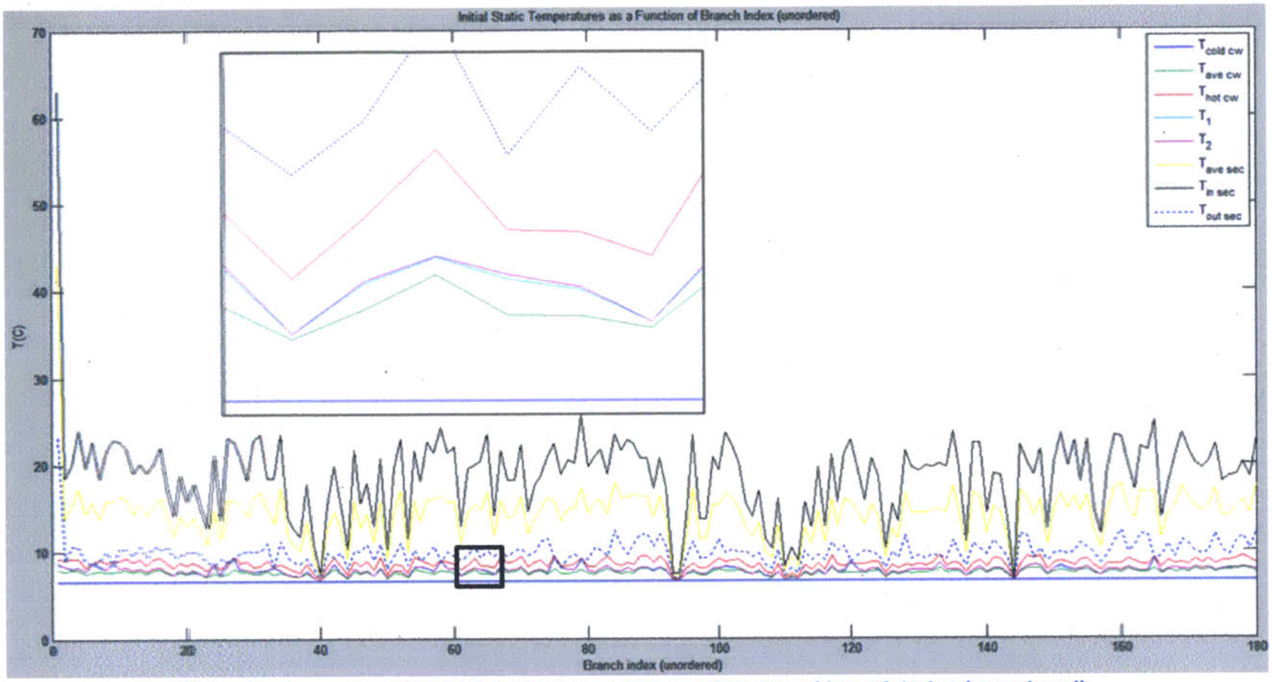


Figure 61: Example of initial static temperatures as a function of branch index (unordered)

3.2.2 Step 2: Determination of Network Segments

Up to this point, the program has data stored in vectors, such as the branch locations, but does not have the data ordered with respect to distance along the header. The determination of the network segments processes the data stored in the vectors and orders it with respect to the start of flow from a particular riser section and the direction of flow, either clockwise at the riser-header junction, or counterclockwise. Thus, a matrix is created which stores the index of various vectors, such as branch locations, with each row corresponding to a specific riser and direction.

Before proceeding, a quick description of variables is given:

- *curr_header_pt* – A point which keeps track of the current location in the supply header.
- *next_header_pt* – A point which keeps track of the next bend in the supply header.
- *branch_loc* – A matrix containing the x,y,z coordinates of each branch.
- *seg_valve_loc* – A matrix containing the x,y,z coordinates of each segregation valve.
- *branch_order* – A matrix which stores the riser number, the direction of flow (1 for clockwise, 2 for counterclockwise), and the branch index.
- *Location_x* – A matrix which stores the position of the points in which pressure is calculated with respect to the associated riser. The position is simply the distance travelled along the length of the pipe from the riser to the point of interest.
- *dPdX* – A matrix which stores the associated cause of the pressure drop at a specified point, i.e. 1=pressure drop due to friction along pipe walls, 3=pressure drop due to friction across segregation valves.

- *Pressure_height_h* – A matrix which stores the pressure drop associated with a change in height.

Initially, the program starts by defining *curr_header_pt* at the junction of the riser and supply header for the riser under consideration. Depending on the direction of flow (clockwise or counterclockwise), the program defines *next_header_pt* at the location of the next bend in the supply header. With these two points, the direction under consideration is found. Figure 62 gives a visualization of *curr_header_pt* and *next_header_pt*.

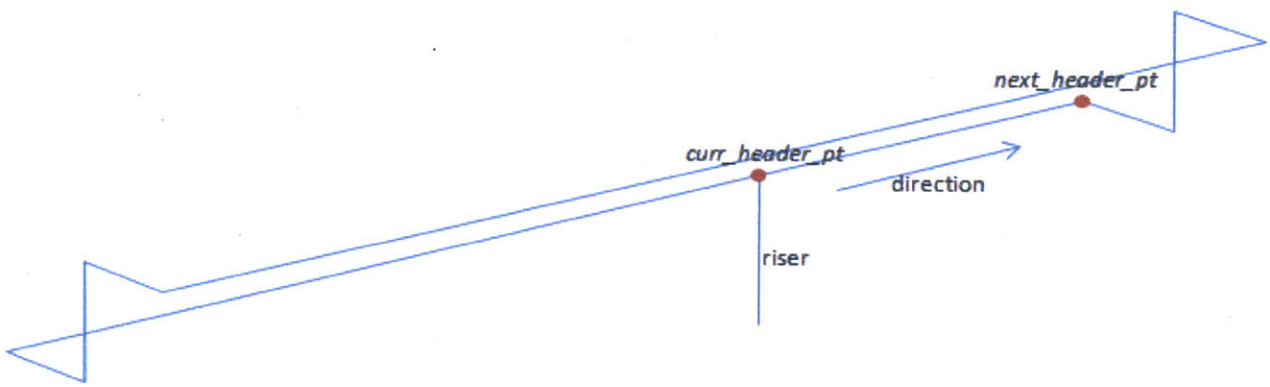


Figure 62: curr_header_pt and next_header_pt

The program searches through the vector containing the branches and determines the location of the next branch. Similarly, the program finds the location of the next valve. Figure 63 gives a visualization of *seg_valve_loc* and *branch_loc*.

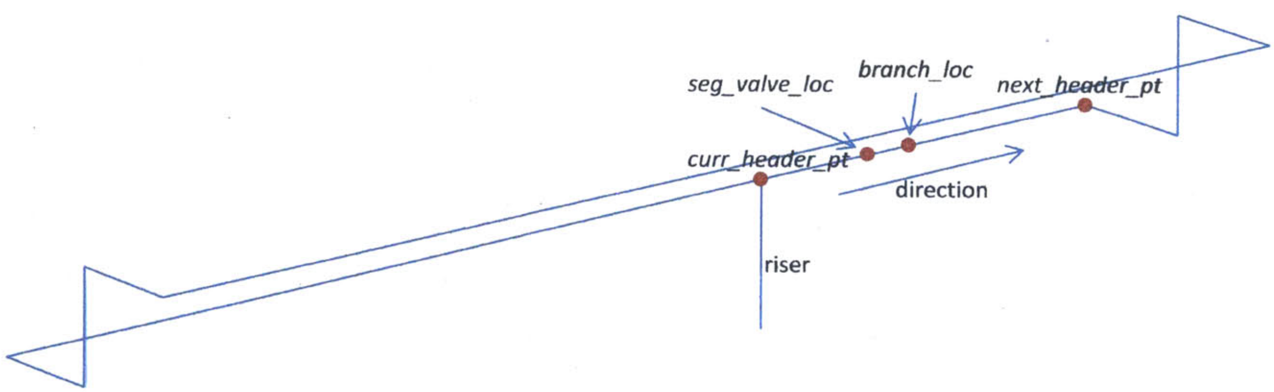


Figure 63: seg_valve_loc and next_header_pt

The program determines if the branch location is closer to *curr_header_pt* or if the valve location is closer. The *curr_header_pt* is updated to the closer of the two. This can be seen in Figure 64. If the next closest point was that of *branch_loc*, then the next element in *branch_order* is set equal to the index of *brancl_loc*. In any event, the distance between the next element and that of *curr_header_pt* is stored in

Location_x.

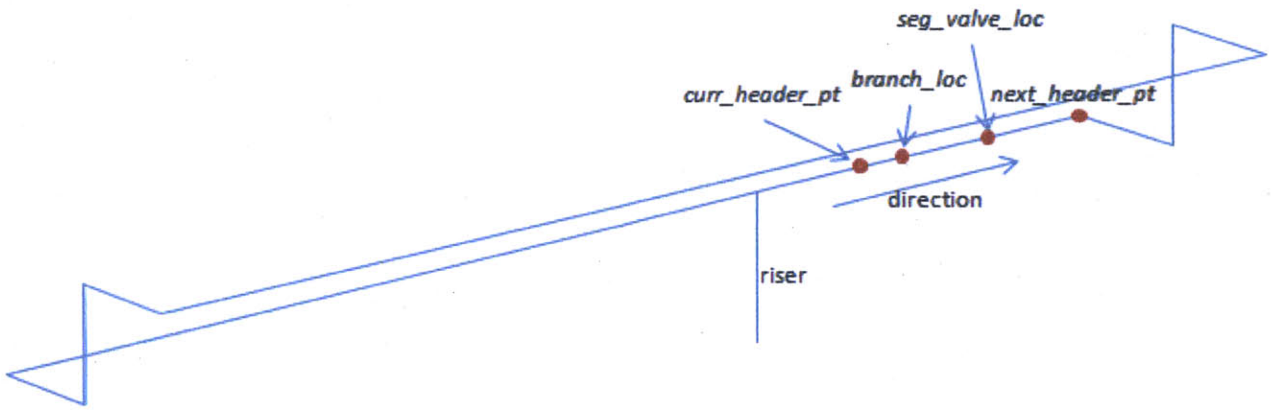


Figure 64: curr_header_pt updated to location of segregation valve

In addition, information pertaining to the pressure drop associated with the element and distance traveled is stored in the matrix, *dPdX*. The pressure drop associated with a change in height is stored in the matrix *Pressure_height_h*.

The process is repeated until there are no branches or valves between *curr_header_pt* and *next_header_pt* as seen in Figure 64. In that case, *curr_header_pt* is set to *next_header_pt*, *next_header_pt* is set to the next bend in the supply header piping, and the direction is updated. This can be seen in Figure 66. The whole process is repeated until a complete loop is performed.

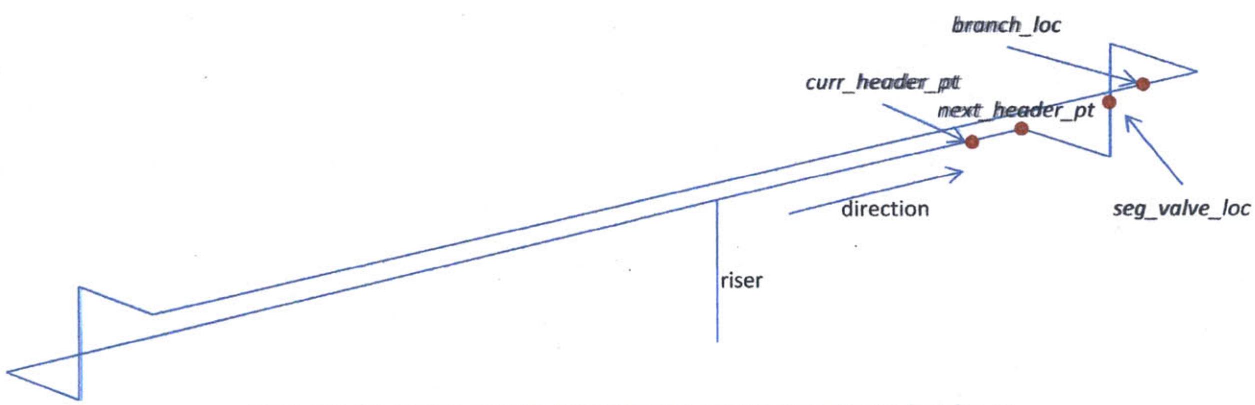


Figure 65: No branch or valve between curr_header_pt and next_header_pt

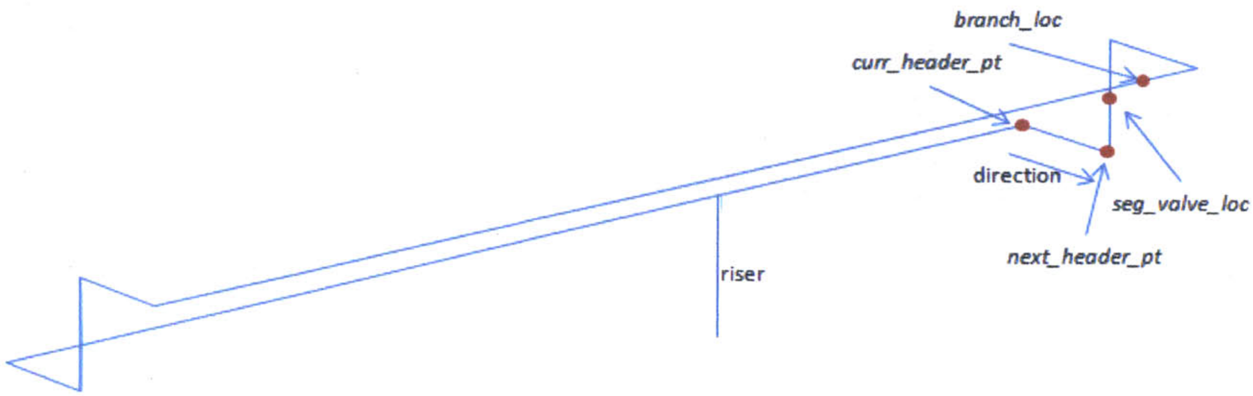


Figure 66: curr_header_pt updated to next_header_pt; next_header_pt set to next bend location

The process is then again repeated starting from the same riser with flow in the opposite direction. Afterwards, the program moves on to the next riser, where everything is repeated with clockwise flow and counterclockwise flow. This is continued until all risers are considered.

3.2.3 Step 3: Refining Branch Velocities and Branch Mass Flow Rates Using Network Analysis Accounting for Head loss Associated with Bends, Friction, and Across Valves

Previously, in step 1, it was assumed that the branch mass flow rates were equal to 4.5 gpm/ton. This is not necessarily true since the configuration of the piping network will have an effect on flow velocities and mass flow rates. In an attempt to get a more accurate value for branch mass flow rates, flow network analysis is used.

Initially, the total mass flow rate is assumed to be the sum of the branch mass flow rates found previously. The velocity for each branch is calculated from the contribution of each riser with flow going clockwise and counterclockwise. This is done taking into account the loss coefficient due to friction along the pipe walls, the loss coefficient due to bends in pipes, the loss coefficient due to friction across valves, and the loss coefficient due to friction across the heat exchangers. The sum of all loss coefficients within a branch yields the overall loss coefficient for that branch.

The overall loss coefficient is found by first calculating the Darcy friction factor. The Darcy friction factor is a function of the pipe diameter, the flow velocity within the branch, the thermal conductivity of the chilled water, the kinematic viscosity of the chilled water, the surface roughness factor, the density of the chilled water, and the specific heat capacity of the chilled water. Most of these values are assumed constant although there is temperature dependence, but for the range of temperatures considered, the error is negligible. The flow velocity, however, was assumed. The first iteration yields only an approximation of the Darcy friction factor.

The loss coefficient due to friction uses the calculated Darcy friction factor, and as a result is also just an approximation of the true loss coefficient due to friction. The other coefficients are also computed using the equations described in Ch. 2. Similarly, the overall loss coefficient is calculated for the header segments separating the branch piping junctions.

With the overall loss coefficients of the branches and the header segments, a resistive network can be set up in which the flow velocities can be solved. The velocities of each branch is solved for in this way, along with the velocities within the header segments utilizing the conservation of mass.

After the iteration is complete, a delta will exist between the initial velocity assumed within a branch and the computed velocity at the end of the iteration. The process is repeated until the velocities converge with a delta of less than 10^{-8} m/sec (usually within 4-5 iterations).

With a better approximation of the velocities with a branch and within the header segments, the mass flow rates through those segments can be determined. The more accurate velocities also yield more accurate outlet temperatures on the chilled water side, as well as temperatures on the secondary side.

3.2.4 Step 4: Account for Entrance and Exit Effects Utilizing Refined Branch Velocities

With more refined velocities, entrance and exit effects of the branches can be accounted for. Similar to the above step, the overall loss coefficients are calculated for each branch using the best estimate for flow velocity. The difference between the previous step is that in addition to accounting for the loss due to friction, bends, and valves, the loss coefficients due to flow entering a path and exiting from a path is also accounted for. These two loss coefficients are highly dependent on velocities, which is why time was spent getting a better approximation for velocity taking into account the other loss coefficients.

Also, similar to the method described above, the process is repeated until the differential velocities between iterations are negligible. Again, the mass flow rates, and various temperatures are re-calculated. Figure 67 shows the evolution of the branch velocities after each refinement made. Note that the branch index corresponds to the order of the branch junctions along the supply header. Therefore, branch index 1 is the first branch junction after the riser junction (assuming flow in the clockwise direction). The isolation valve between the last branch junction and the riser is considered shut so that flow is in one direction throughout the supply header piping. Additionally, all other A/C pumps are off so flow is in response to a single A/C unit and pump in operation.

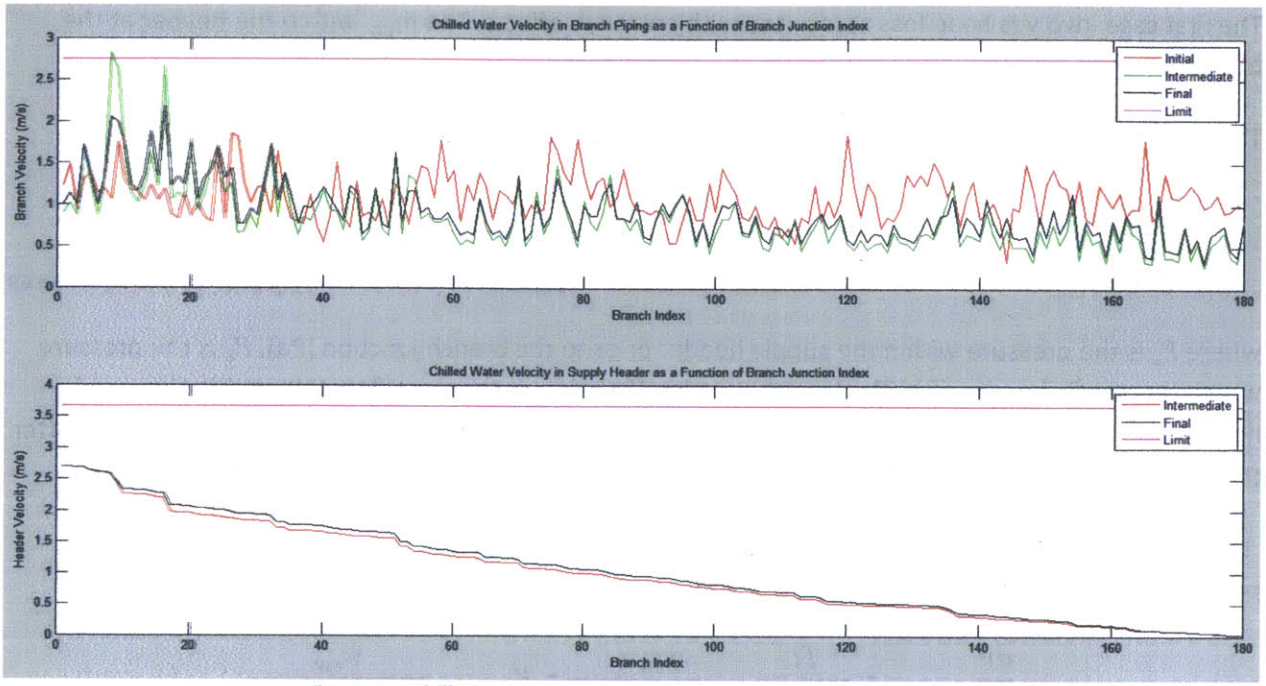


Figure 67: Chilled water velocities in branches and supply header

As can be seen in the above plot, the initial assumption of velocity based solely on the branch geometry is incorrect with the branch velocities fluctuating about a horizontal line. In contrast, once FNA is used, the trend in branch velocities shows a decrease in velocity as the distance from the branch junction to the riser increases. The velocity of the chilled water also shows a decrease along the length of the supply header.

3.2.5 Step 5: Determination of Pressure Drop as a Function of Distance

Using the information stored in the matrix *dPdX* along with the more accurate branch velocities and header segment velocities and the distance between pressure drop sources, the pressure as a function of distance along the supply header was determined.

Five sources of pressure drop were considered: the pressure drop associated with a branch junction, the pressure drop associated with friction along the pipe wall, the pressure drop across a valve, the pressure drop associated with a bend in the pipe, and the pressure drop across a heat exchanger. The pressure drop associated with changes in height was analyzed separately as discussed in Section 3.2.2.

The pressure drop along the header could be found using the equation:

$$\frac{dP}{dx} = K \frac{\dot{w}_{cw}}{2g\rho_{cw}A_{header}^2}$$

Equation 68 (Rennels & Hudson, 2012)

The first case involves head loss associated with entrance effects. The $\dot{m}_{cw}$ within the header at the branch junction was converted to $\dot{w}_{cw}$.

The differential pressure across the branch entrance was determined using the equation:

$$\frac{dP}{dx}_{entrance} = P_1 - P_2 = \frac{\dot{w}_{cw_2}}{2g\rho_{cw}A_{header}^2}\left(1.62 - 0.98\frac{\dot{w}_{cw_1}}{\dot{w}_{cw_2}} - 0.64\frac{\dot{w}_{cw_1}^2}{\dot{w}_{cw_2}^2} + 0.03\frac{\dot{w}_{cw_2}^6}{\dot{w}_{cw_1}^6}\right)$$

Equation 69 (derived from Equation 37 and Equation 68)

where P_1 is the pressure within the supply header prior to the branch junction [Pa], P_2 is the pressure within the supply header after the branch junction [Pa], $\dot{w}_{cw_1}$ is the mass flow rate within the supply header prior to the branch junction [lbm/sec], $\dot{w}_{cw_2}$ is the mass flow rate within the supply header after the branch junction [lbm/sec], and A_{header} is the cross-sectional area of the supply header.

The second case involves head loss associated with friction along the pipe walls. To determine the pressure drop along a length of pipe, the following equation was used:

$$\frac{dP}{dx}_{friction} = \frac{fL}{D_{header}}\frac{\dot{w}_{cw}}{2g\rho_{cw}A_{header}^2} = K_{friction}\frac{\dot{w}_{cw}}{2g\rho_{cw}A_{header}^2}$$

Equation 70 (derived from Equation 24 and Equation 68

where f is the Darcy friction factor associated with that segment of pipe, L is the length of pipe considered [m], and D_{header} is the diameter of the supply header [m].

The third case involves head loss across a segregation valve. To determine the pressure drop across a segregation valve, the following equation was used:

$$\frac{dP}{dx}_{valve} = K_{valve}\frac{\dot{w}_{cw}}{2g\rho_{cw}A_{header}^2}$$

Equation 70 (repeated)

where K_{valve} is the loss coefficient associated with the segregation valve. A value of 0.2 was used as a notional value for this type of valve (Rennels & Hudson, 2012).

The fourth case involves head loss associated with a bend in the supply header pipe. To determine the pressure drop associated with a pipe bend, the following equation was used:

$$\frac{dP}{dx}_{bend} = \left[f\frac{\pi}{2}\frac{r}{d} + \left(0.10 + 2.4f\sin\frac{\pi}{2} + \frac{6.6f\left(\sqrt{\sin\frac{\pi}{2}} + \sin\frac{\pi}{2}\right)}{\left(\frac{r}{d}\right)^2}\right)\right]\frac{\dot{w}_{cw}}{2g\rho_{cw}A_{header}^2}$$

Equation 71 (derived from Equation 35 and Equation 68)

where $\frac{r}{d}$ is the bend radius ratio (with a default value of 3.0).

The fifth case involves head loss associated with the heat exchanger. This value is specified in the heat exchanger database and read in by the Matlab program. The head loss across a heat exchanger is a set value and is not calculated as a function of mass flow rate. This exemplifies the tradeoff between having a model which accurately portrays flow under all circumstances and a model that is easy to use. If the flow across the heat exchanger is close to the design flow rate, then the actual head loss should also be close to the specified head loss.

The sum of the differential pressures yields the total differential pressure at each corresponding index within the matrix.

$$\frac{dP}{dx}_{total} = \frac{dP}{dx}_{entrance} + \frac{dP}{dx}_{friction} + \frac{dP}{dx}_{valve} + \frac{dP}{dx}_{bend} + \frac{dP}{dx}_{hxchgr}$$

Equation 72

The pressure is then computed along the pipe length, with each point representing a source of pressure drop. This can be shown with the following Matlab code snippet:

```
for i=1:max(size(Location_x))
        if i<max(size(Location_x))
                Pressure(m,n,i+1)=Pressure(m,n,i)-dPdX_total_h(m,n,i);
        end
end
```

Other sources of pressure drop such as sudden contraction or expansion of pipe could be accounted for in this section, but since the supply header is of constant diameter this was not considered. If greater generality of the program is desired, then some code would have to be added in this step of the program to account for the desired sources of changes in pressure.

3.2.6 Step 6: Determination of Stagnation Points

At the riser-header junction, a portion of the chilled water will flow clockwise and the remaining will flow counterclockwise. With several risers in parallel, there exist points between each pair of adjacent risers in which the clockwise flow exiting one riser junction will have the same pressure as the flow exiting the adjacent riser with counterclockwise flow exiting from it[12]. At this point, the flow stagnates. To break up the network into smaller independent networks, it is imperative to determine these stagnation points. These stagnation points represent the points in which the network can be isolated and analyzed independently.

[12] This assumes the pumps are well balanced, meaning that one pump will not overpower an adjacent pump causing flow to go in the wrong direction. Check valves are located downstream of each pump to ensure this does not happen.

Assuming the pressures at the base of each riser are all equal, the pressure differentials caused by changes in height were neglected. The magnitude of the pressure along the length of the header pipe for each riser were then compared with one another, and the intersection of the lines were considered the stagnation points within the header network. **Error! Reference source not found.** shows a representation of the pressures associated with each riser superimposed on one another. The reference point chosen (0 on the x-axis) corresponds to the riser junction of the forward-most portside riser junction. Positive proceeds clockwise along the supply header piping.

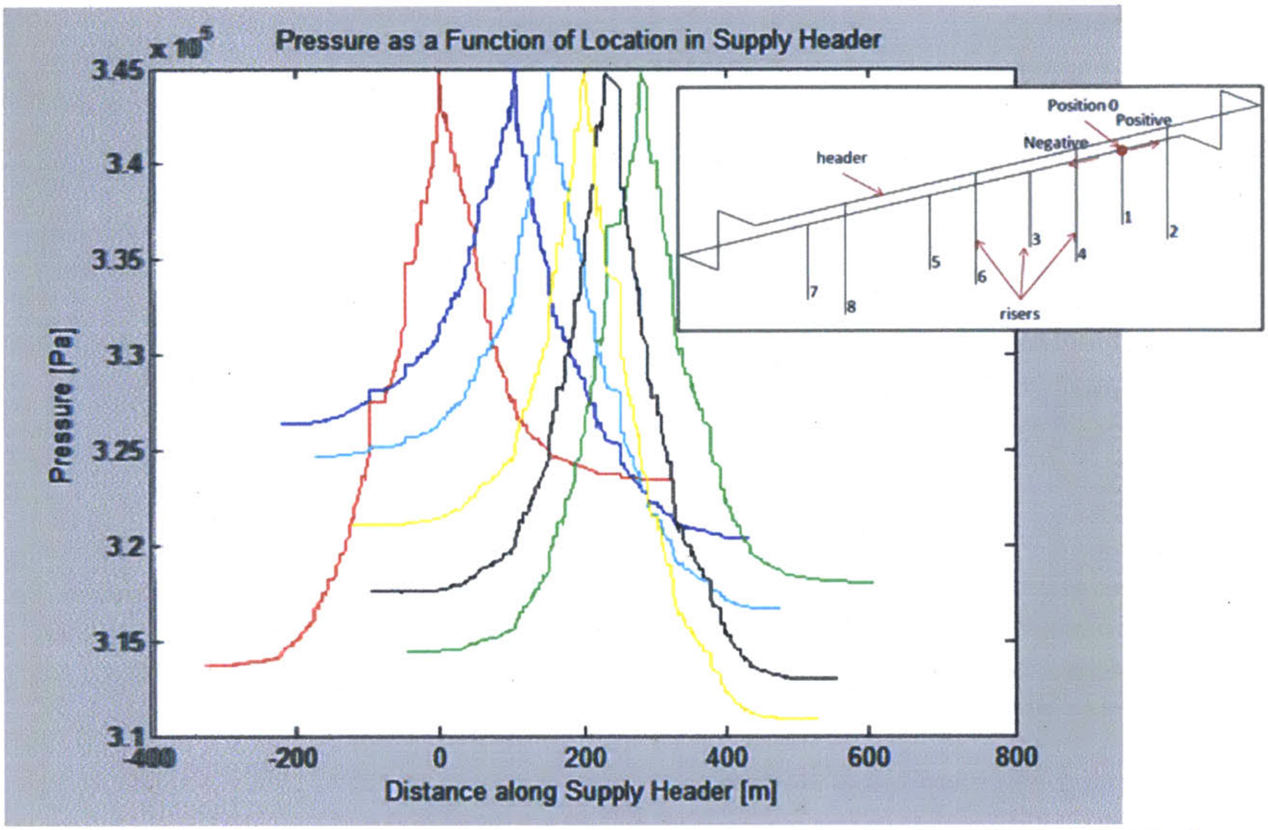


Figure 68: Pressure as a function of location in supply header for clockwise and counterclockwise flow for each chiller/pump superimposed

The pressure along the header pipe was plotted with respect to distance from the riser junction. It was interesting to see that the pressure drops were not symmetrical with clockwise flow and counterclockwise flow as one may assume. Since the pressure drops were a function of velocity, the difference in flow velocities at a point from clockwise flow or counterclockwise flow contributed to differences in head loss at the same point depending on the direction of flow. Figure 69 below shows the pressure drop from one junction with flow going clockwise (extending in the positive x-direction) and flow going counter-clockwise (extending in the negative x-direction). Figure 69 does not include the effects of changes in height along the pipe length.

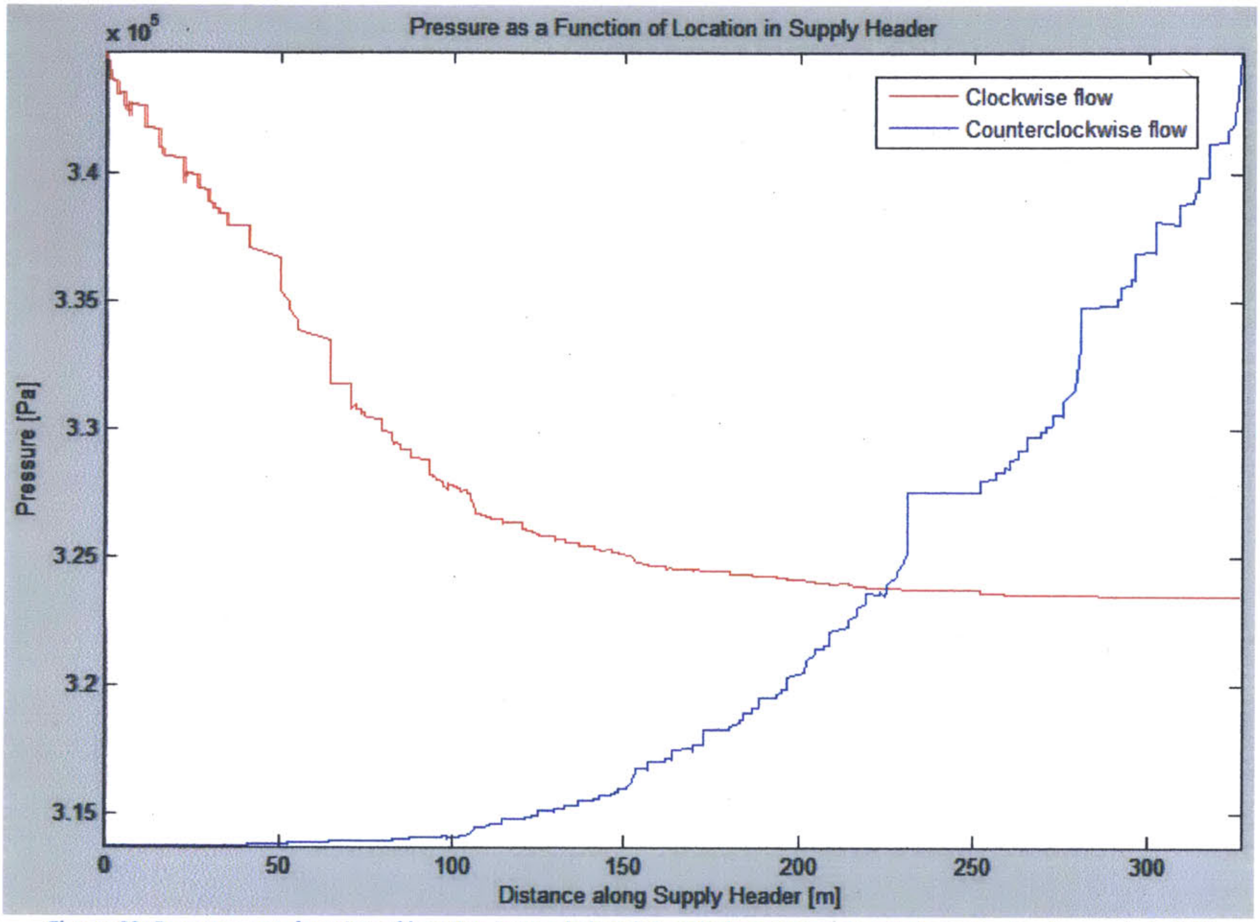


Figure 69: Pressure as a function of location in supply header excluding pressure variations due to changes in height

Accounting for changes in height along the length of the header piping, the following pressure distribution is found (Figure 69) for the same flow and junction as the figure above.

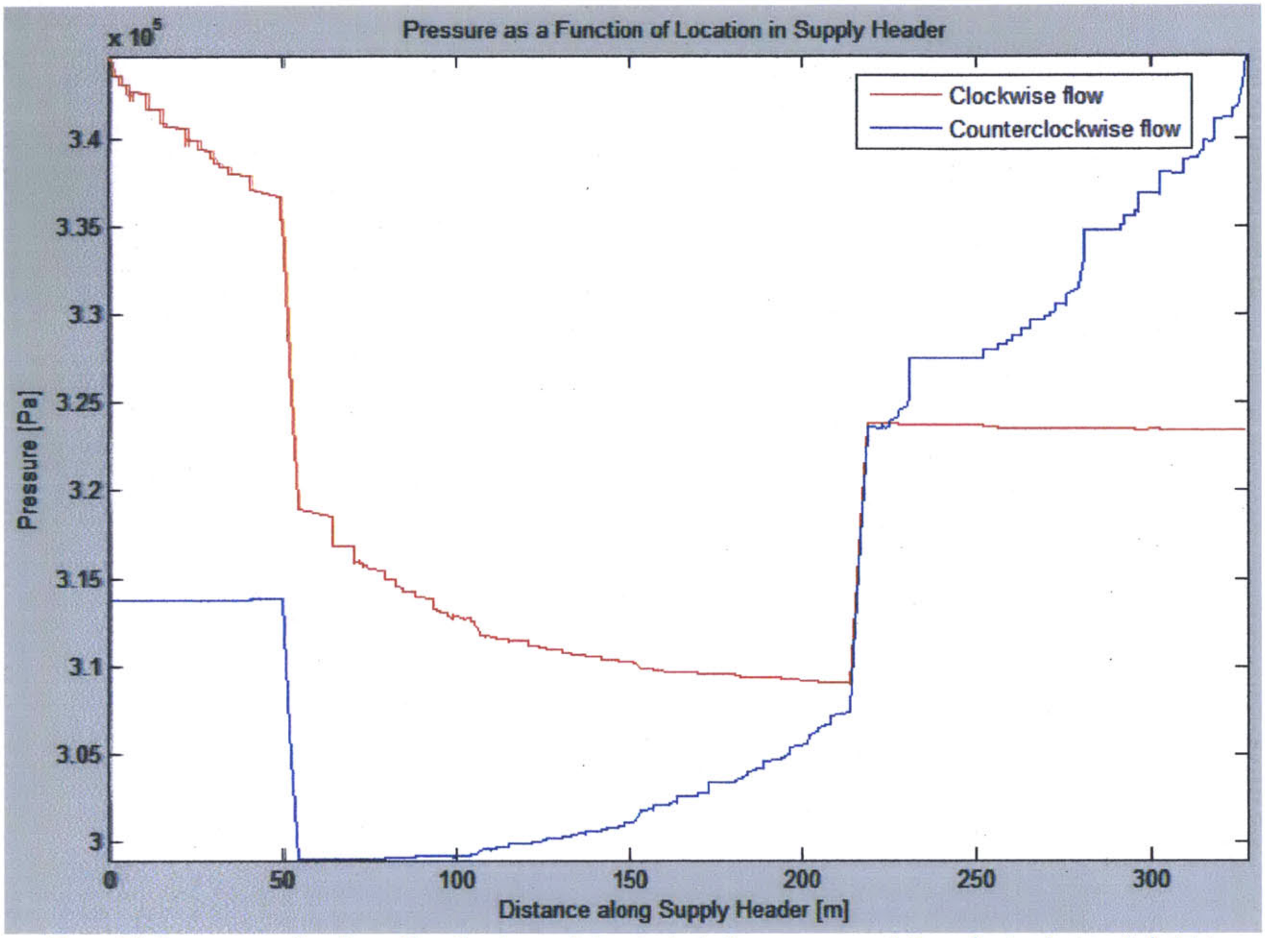


Figure 70: Pressure as a function of location in supply header including pressure variations due to changes in height

3.2.7 Step 7: Final Calculation of Velocities and Mass Flow Rates Using Network Analysis with Network Isolated at Stagnation Points

The chilled water network was first analyzed considering only one riser junction at a time, i.e., only accounting for flow from one A/C unit/pump in operation at a time. The network becomes much more complicated when there are several sources of flow in parallel. To circumvent the difficulties arising from parallel sources of flow, the piping network is isolated at the stagnation points discussed in Section 3.2.6. This can be seen in Figures 68-72 below.

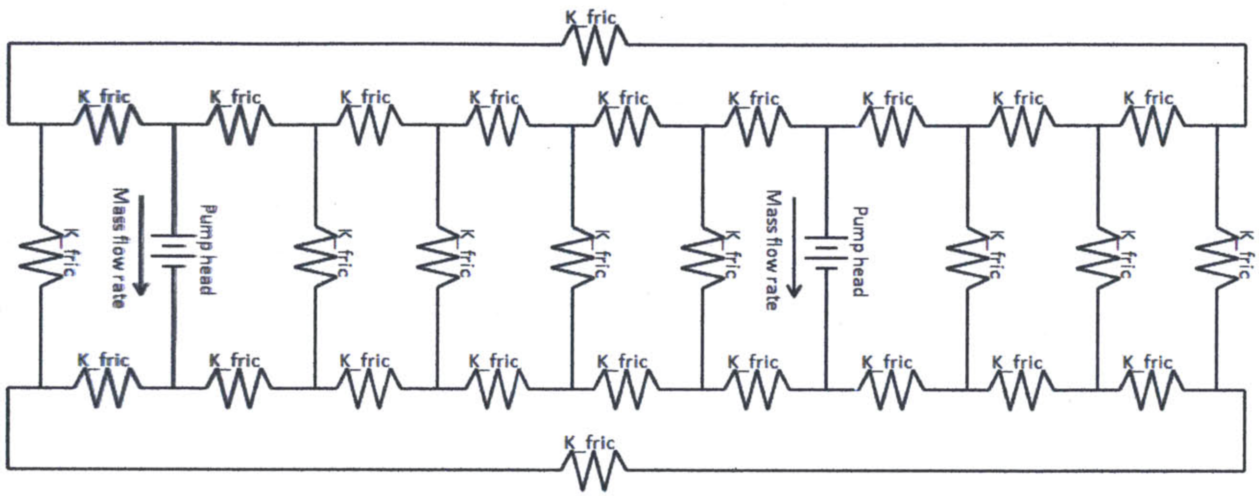


Figure 71: Electrical analogy of chilled water system including two pumps in parallel and several branches in parallel

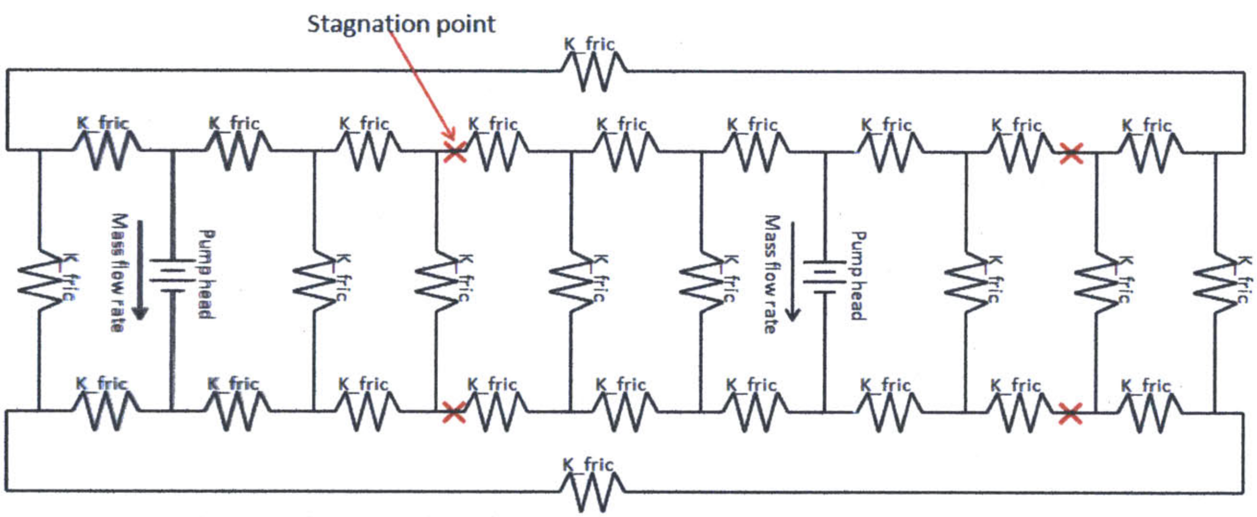


Figure 72: Electrical analogy of chilled water system with stagnation points shown in red

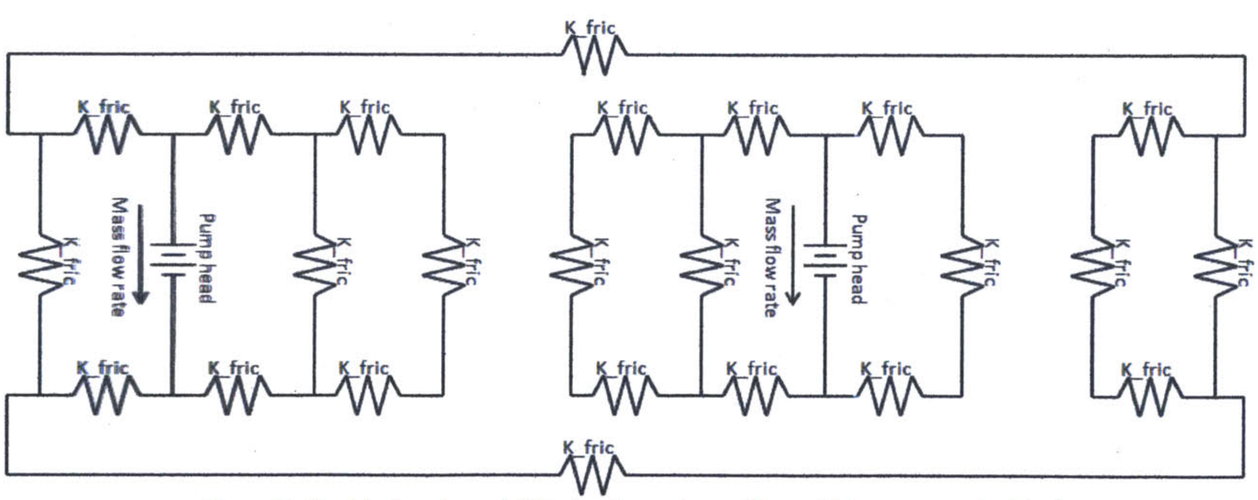


Figure 73: Electrical analogy of chilled water system with parallel pumps now isolated

Figure 74: Simplification of network

Figure 75: Network reduced to a single pump and a single equivalent resistance to flow per isolated loop

With the network segmented at the stagnation points, the velocity and mass flow rates are once again computed using network analysis. The network is isolated at the stagnation points, forming an isolated loop for each pump. The total head loss propagating clockwise from the riser junction and counterclockwise from the riser junction is found using flow network analysis. This is further reduced by considering the total head loss seen across each the pump. The pump head will equal the head loss.

An adequate pump would have to be selected such that the efficiency of the pump is satisfactory for the head calculated and mass flow rate calculated. This was not done by the program, but should be implemented in a future iteration of the CSDT. Using the pump curves shown in Section 2.4.3, a specific pump could be selected; however, equations would have to be developed possibly through the polyfit and polyval functions in Matlab in order to mathematically describe the pump curve plot for the 1510 series Bell & Gossett centrifugal pumps (or some other pump series). With pumps selected, the specific pump performance curve can be referenced to determine if the initial mass flow rate guessed at is correct based on the head loss of the system. If not, the mass flow rate would be adjusted and the process repeated (i.e., pressure distribution found using modified mass flow rates, stagnation points found, network simplified, total head loss across pump found, mass flow rate determined from pump performance curve for specified pump head).The CSDT currently assumes the mass flow rates were initially correct and the pressures at each riser junction are perfectly in balance.

Once the mass flow rate for each pump converges to a solution, the process is reversed. Starting from the simplified network with a single source and a single equivalent resistance to flow, the mass flow rate is found. To determine the mass flow rate propagating clockwise and counterclockwise from the riser junction, the network seen in Figure 74 is referred to. This is done by considering the conservation of mass at the junction (analogous to KCL) and FNA. The process is continued, solving for flow within each parallel branch applying FNA and mass conservation until the mass flow rates within each branch and header segment are known.

3.2.8 Step 8: Calculate Branch Inlet Temperatures

The assumption that the inlet temperature of the chilled water was equal to the outlet temperature of the A/C unit was revisited. The initial outlet temperature of the A/C unit was assumed to be 6.67°C (later on this is also revised to account for the actual outlet A/C unit temperature dependent on the actual A/C unit selected). It is known that the outlet temperature of the A/C unit will rise as the chilled water flows along the length of pipe. This rise in temperature is due to several factors, such as compression of the fluid across the pump, friction along the pipe walls, head loss across valves and bends, and head loss attributed to entrance and exit effects. These sources of head loss have already been determined by the program. The associated temperature rise is determined for the first branch junction by the equation:

$$\begin{aligned}\Delta T_{branch\ junction} &= T_{branch\ junction} - T_{AC\ out}\\ &= \left(T_{branch\ junction} - T_{pump\ out}\right) + \left(T_{pump\ out} - T_{pump\ in}\right) + \left(T_{pump\ in} - T_{AC\ out}\right)\\ &= K_2\frac{V^2}{2g}\frac{1}{C_2c_p} + \frac{H_p}{C_2c_p\eta} + K_1\frac{V^2}{2g}\frac{1}{C_2c_p}\end{aligned}$$

Equation 73

where $T_{branch\ junction}$ is the temperature of the chilled water at the junction of the branch piping and supply header [°C], $T_{AC\ out}$ is the outlet temperature of the chilled water from the A/C unit [°C], c_p is the specific heat capacity of the chilled water [Btu/lbf-°F], K_1 and K_2 are the overall loss coefficients for the segment of pipe from the A/C unit to the pump and for the segment of pipe from the pump to the branch junction, respectively, H_p is the pump head [ft], η is the pump efficiency, and C_2 is a conversion factor equal to 778.169262 [ft-lbf/Btu]. The heat loss through the pump was ignored since this is much smaller than the pump power.

The branch junction's downstream f the first branch junction includes the differential temperature discussed above plus the differential temperature arising from the distance between the two junctions. The equation is as follows:

$$\begin{aligned}\Delta T_{branch\ junction_{downstream}} &= T_{branch\ junction_{downstream}} - T_{branch\ junction}\\ &= K_3\frac{V^2}{2g}\frac{1}{C_2c_p} - T_{branch\ junction}\end{aligned}$$

Equation 74

where K_3 is the overall loss coefficient from the segment of piping extending from the first branch junction to the branch junction of interest.

In a similar manner, the chilled water inlet temperatures of the heat exchangers can be determined by the equation:

$$\Delta T_{in} = T_{in} - T_{branch\ junction_{downstream}} = K_4 \frac{V^2}{2g} \frac{1}{C_2 c_p} - T_{branch\ junction_{downstream}}$$

Equation 75

where K_4 is the overall loss coefficient from the segment of piping extending from the branch junction to the inlet of the heat exchanger.

With the refined chilled water inlet temperatures, the other temperatures of interest can be recalculated once again. The recalculated temperatures do not suffer from the initial assumptions of mass flow rate or branch inlet temperatures.

3.2.9 Step 9: Determination of A/C unit Capacity Required and Selection of A/C units

With the revised temperatures and revised piping network (isolated at the stagnation points), the A/C unit capacity required can be determined. The differential temperature across the A/C unit can be found, (assuming an outlet temperature of 6.67°C by default). In addition, the total mass flow rate is also known and is equal to the mass flow rate of the chilled water through the A/C unit. Therefore, the A/C unit capacity is found using the following equation:

$$AC_{capacity} = \dot{m} c_p \Delta T$$

Equation 76

The program selects all A/C units to have the same capacity. The capacity is chosen as the A/C unit within the chiller database that is closest (but greater) to the greater of the highest individual calculated A/C capacity or the average A/C capacity needed with 50% of the A/C units operational. For example, assuming there are four A/C units with capacities of 100 tons, 65 tons, 110 tons and 85 tons, the total capacity needed is 360 tons. Assuming 50% of the A/C units are operational at a given time, each A/C unit must at least supply 180 tons. The maximum individual A/C unit capacity is 110 tons, thus the greater of 110 tons and 180 tons is chosen. If the smallest available chiller available which is greater than or equal to 180 tons is a 200 ton chiller, the program will size each of the four chillers to 200 tons.

The user has the ability to override the program and select another A/C unit from the database. The outlet temperature of the A/C unit is then read in by the program to ensure the assumption of 45°F was valid. If it was, the program continues, if not, then the temperatures in the previous step are recalculated.

If a different A/C unit selection process is to be incorporated, this section of code would have to be modified. For example, an N-1 approach could be taken, where the A/C units are sized such that the cooling needs of the heat loads under the worst case operating condition could be met with a loss of one A/C unit. This approach would have a significant effect on reducing the total weight of the chilled water system as compared to the method employed by the CSDT.

3.2.10 Step 10: Expansion Tank Sizing

To size the expansion tank, the same method used within CSDT v1.0 was used (Fiedel, 2011). The expansion tank has to be large enough to supply chilled water to the pump for the time specified by the user (the default is 30 seconds). In addition, the tank acts as a surge volume accounting for the expansion of the fluid as it changes in temperature. The more limiting of the two criteria is what drives the size of the tank. This process was discussed in detail in Section 2.7.

3.2.11 Step 11: Weight Analysis

The CSDT also has the capability to perform a weight analysis of the chilled water system. The weight analysis includes the weight of the system as well as the LCG, VCG, and TCG of the system. The weight and center of gravity is broken down into the chilled water system and part of the auxiliary seawater system. Each system is then broken down further into the components which form each system. The weight breakdown structure is listed below:

1. Chilled Water System
    a. Piping
        i. Main Piping
        ii. Branch Piping
    b. Lagging
        i. Lagging – Main
        ii. Lagging – Branch
    c. Valves
        i. Globe Valves
            1. Main Globe Valves
            2. Branch Globe Valves
        ii. Gate Valves
            1. Main Gate Valves
            2. Branch Gate Valves
        iii. Check Valves
            1. Main Check Valves
            2. Branch Check Valves
    d. AC Units
    e. Expansion Tanks
    f. Pumps
    g. Brackets
    h. Instrumentation
    i. Chilled Water
    j. Heat Exchangers
2. Auxiliary Seawater System
    a. Piping
    b. Valves
    c. Pumps
    d. Brackets
    e. Salt Water

The weight breakdown is much more granular for the chilled water system, since this is the main focus of the CSDT program. The program also asks for a weight margin. The margin is added to the total weight of the systems.

Most of the components listed above are self explanatory as to how the weight and center of gravity were computed since the geometry, position and densities are known. However, some of the weights of the components were estimated.

To determine the weight of the valves within the chilled water system, typical valve weights were used based on valve size. The valves were sized according to the pipe diameter. Table 12 below lists the weights for various valve types.

Diameter [m]	Gate Valve Weight [kg]	Globe Valve Weight [kg]	Check Valve Weight [kg]
0.0127	3.2	3.1	
0.0190	4.2	4	
0.0254	5.8	5.7	
0.0381	11	10.6	
0.0508	15.4	15.4	13
0.0635			17
0.0762	35	35	24
0.1016	50	55	36
0.1270	70	80	57
0.1524	80	98	62
0.2032	135	165	96
0.2540	185	305	158
0.3048	280	425	238
0.3556	395	590	324
0.4064	530	830	483
0.4572	670	1040	548
0.5080	775	1260	782
0.6096	1150	1700	1150

Table 12: Valve weights

The pipe hangar weight was accounted for by using a hangar weight per unit distance of pipe length. This metric was dependent on the diameter of the pipe being supported by the pipe hangars and was determined based on the dimensions of the pipe hangar, the pipe hangar density, and the pipe hangar spacing along the length of the pipe (ASTM International, 2008). Table 13 lists the pipe hangar weight per meter of pipe for various pipe diameters.

Diameter [m]	Hangar Weight per Meter of Pipe [kg/m]
0.0063	0.0536
0.0095	0.0550
0.0127	0.0761
0.0190	0.0655
0.0254	0.0687
0.0317	0.0718
0.0381	0.0558
0.0508	0.1190
0.0635	0.1269
0.0762	0.1333
0.0889	0.1770
0.1016	0.1292
0.1270	0.2246
0.1524	0.2624
0.2032	0.3834
0.2534	0.3734
0.3048	0.4743
0.3556	0.4673
0.4064	0.5807
0.4572	0.5744
0.5080	0.6867
0.6096	0.6810

Table 13: Hangar weight per meter of pipe

3.2.12 Step 12: Static Temperature Analysis

At this point, the velocity distribution is known within the system. Using an energy balance approach, the temperatures are found along the supply header with the temperature rising as the fluid propagates towards the stagnation points. The rise in temperature is insignificant, however. The conversion of mechanical energy to thermal energy through head loss is on the order of 10^{-5}°C.The rise in temperature of the chilled water due to the temperature of the environment was neglected for the static temperature analysis since the fluid is flowing and the resistance to heat transfer is significant due to the pipe lagging[13]. The temperature across the heat exchanger is computed based on the mass flow rate and the heat load[14] and the chilled water exiting each heat exchanger within each branch is found. The temperature along the return header is found using an energy balance approach taking into account the outlet temperature and mass flow rate of each branch.

[13] The transient analysis does take this into account; however, and the rise in temperature is on the order of a few hundredths of a degree Celcius assuming a quiescent air temperature of 20°C and a lagging thickness of 0.75". Even with a higher environmental air temperature, the rise in temperature will still be insignificant (<0.1°C).

[14] Since the system is in steady-state, the heat load is equal to the heat transfer at each boundary between the heat soure and heat sink (neglecting internal heat generation which was found to be insignificant).

To determine the temperature distribution across the heat exchanger the same approach outlined in Section 3.2.1 was used.

3.2.13 Step 13: Transient Temperature Analysis

The transient analysis section of the program is very extensive making up half of the analysis module. The analysis is performed in several steps. The first step gathers information from the user regarding the initial conditions of the system and the changes which occur during an event. The second step determines the initial pressures within the chilled water system. The third step calculates the initial velocities and stagnation points. The fourth step calculates the pressures after the event occurred. The fifth step calculates the velocities after the event. The sixth step calculates the temperatures throughout the chilled water system with respect to location and time. Lastly, the seventh step plots the temperature responses.

3.2.13.1 Part A: User Input

The program begins the transient analysis by first gathering user input. First, the program prompts for the load condition to be considered during the transient: shore, design, cruise, or battle. The program takes this response and populates an excel spreadsheet 'Transient.xlsx' with the heat loads corresponding to the load condition. The user is then directed to the spreadsheet to fill in the remaining information needed. A screenshot of the spreadsheet is shown in Figure 76 below.

Notes:
The maximum heat load is taken from the maximum of the four load conditions: Shore, Design, Cruise and Battle.
The heat load at time t=0- is based off of the load condition specified in Analysis.m. Changes to the initial load value can be entered in the yellow column.
The user must enter the load value for each load after the transient under the column Heat Load at t=0+.
The coordinates for the chiller location is a sfollows: midships is at x=0 with the bow in the +x-direction, port is towards +y, and up is +z.
The chiller status must be specified for each chiller before and after the transient (specified as either on or off).

Heat Load				
Load Number	Load Name	Maximum (kW)	at t=0- (kW)	at t=0+ (kW)
1	RS10_5	0.56272	0.56272	0.56272
2	RS0405_2	3.79836	3.79836	0.00000
3	RS10_8	0.56272	0.56272	0.56272
4	RS75	16.88160	16.88160	16.88160
5	RS04	11.14924	11.14924	11.14924
6	RS10_11	0.91442	0.91442	0.91442
7	RS09	9.99356	9.99356	9.99356
8	RS60	34.39661	34.39661	34.39661
9	RS58	56.06098	56.06098	56.06098
10	RS28_2	1.68816	1.68816	1.68816
11	RS78	4.39625	4.39625	4.39625
12	RS27_2	1.30129	1.30129	1.30129
13	RS56_1	1.65299	1.65299	1.65299
14	RS25	8.12427	8.12427	8.12427
15	RS24	1.30129	1.30129	1.30129
16	RS16	59.47458	59.47458	59.47458
17	RS1G	0.14068	0.14068	0.14068
18	RS21_1	1.11383	1.11383	1.11383
19	RS21_6	0.14068	0.14068	0.14068
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Chiller					
Chiller Number	Chiller Location x (m)	Chiller Location y (m)	Chiller Location z (m)	Status (on/off) at t=0-	Status (on/off) at t=0+
1	30.430667	5.0975	3.362833	on	on
2	30.430667	-5.0975	3.362833	off	off
3	-16.22666	5.0975	3.362833	on	on
4	-16.22666	-5.0975	3.362833	off	off
5	-66.26752	5.0975	3.362833	on	off
6	-66.26752	-5.0975	3.362833	off	off

Transient

Figure 76: Transient Excel spreadsheet

The first column of the spreadsheet simply numbers the heat load according to location of the branch junction along the supply header. The first load corresponds to the first junction clockwise from the forward-most portside riser junction. The load numbers increase clockwise along the supply header. The second column gives the load name for the corresponding heat load. This is the same load name specified in the Excel spreadsheet 'CSDT_input' under the 'LoadData' tab. The maximum heat load is listed in the third column. The heat loads before and after the event need to be specified in the fourth and fifth column. The program populates the fourth column based on the load condition; however, if the initial heat loads deviate from this, the user needs to adjust the values. The chiller number column and chiller location columns are populated by the program. The chiller numbering is as follows: starting from the forward-most chiller portside proceeding starboard then aft. The last two columns need to be filled in by the user and correspond to the status of the chiller before and after the event.

3.2.13.2 Part B: Initial Pressures

The program takes the input from the spreadsheet and determines the pressure distribution along the length of the supply header. An example of the pressure distribution is shown in Figure 77 below.

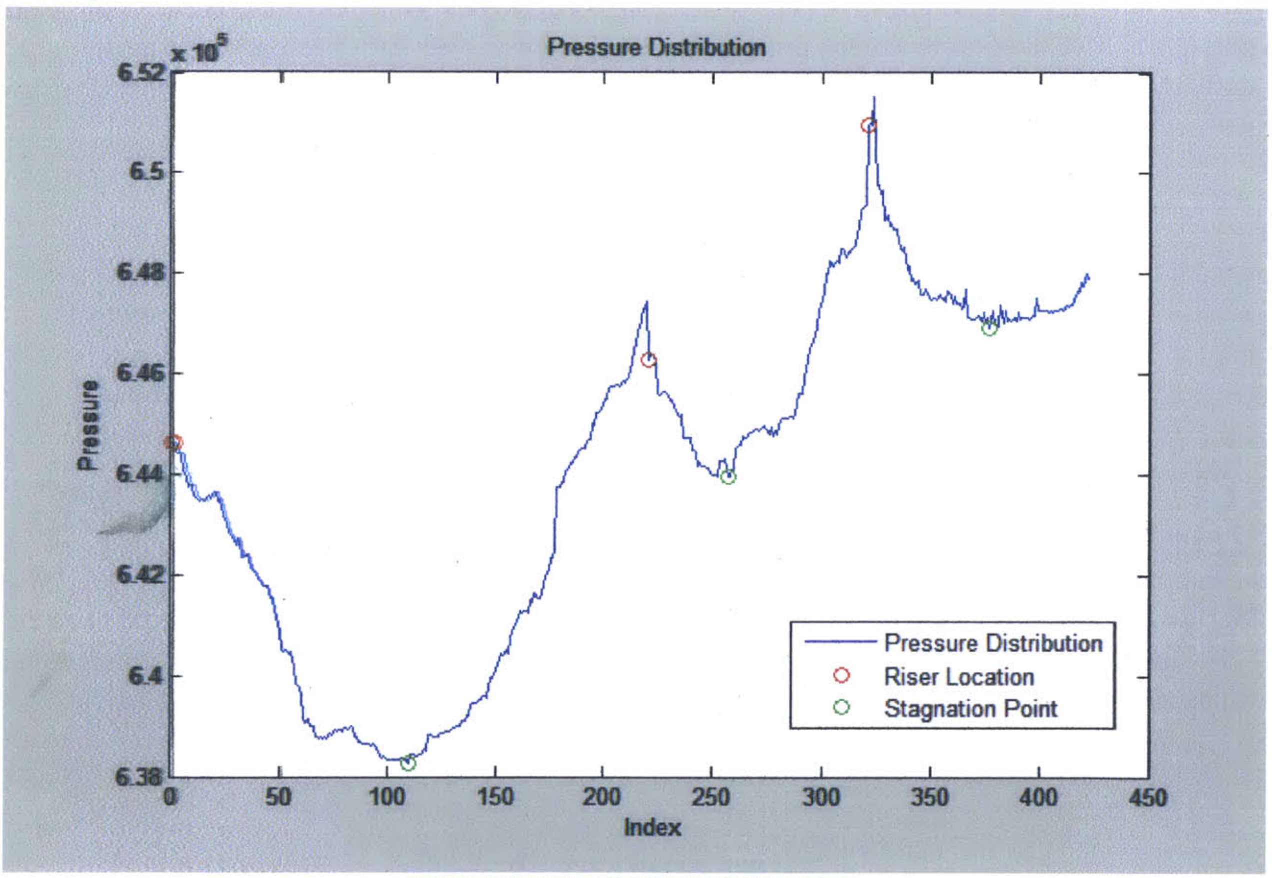


Figure 77: Pressure distribution before event

The above plot is formed by superimposing the pressures associated with flow going clockwise and counterclockwise with the source emanating from the chillers turned 'on' within the Excel spreadsheet. The peaks correspond to the riser locations of the chillers that are operational (red circles). The troughs correspond to the stagnation points (green circles). For the example above, there are six chillers with chillers one, three, and five are on and chillers two, four, and six are off.

There are a few areas of concern with the plot above which could be a potential candidate for future work. First, the peaks do not match up exactly with the risers. The index of the peak may be off by one or two. This is not a major concern, though because the peak is not used within the program, just the trough. More importantly, the beginning and end of the plot should line up with one another. It does not. This is because only a single iteration is done within the program. To achieve continuity at the boundaries of the plot, the process of determining the pressure distribution should be iterated. The stagnation points are found at the troughs of the pressure distribution plot. With this new information, the pressure distribution could then be recalculated. The plots again superimposed, and the stagnation points re-determined. This will result in a better approximation of the pressure distribution with the boundaries approaching one another. The process should then be repeated to the desired accuracy. This was not done because the absolute value of pressure is not needed. What is of importance is the pressure at a location relative to the pressure to other locations. Even with a second iteration of determining the pressure distribution is done, it was assumed the location of the minimum pressures will not change or will change very little.

3.2.13.3 Part C: Initial Velocities and Temperatures

The location of the stagnation points are used to determine the initial velocities as discussed in Section 3.2.6. It takes into account the loss coefficients due to friction, bends, valves, and entrance and exit effects. The temperatures are also calculated within each branch and return header. The supply header is assumed to be a constant temperature equal to the outlet temperature of the chiller. This may contribute to error on the order of a fraction of a degree, but will approach the true value during the transient temperature analysis.

3.2.13.4 Part D: Final Pressures

The same approach described in Section 3.2.13.2 is used to determine the pressure distribution and the subsequent stagnation points after the event. Figure 78 shows an example of the pressure distribution for an example in which chiller five is turned off, leaving only chillers one and three operational.

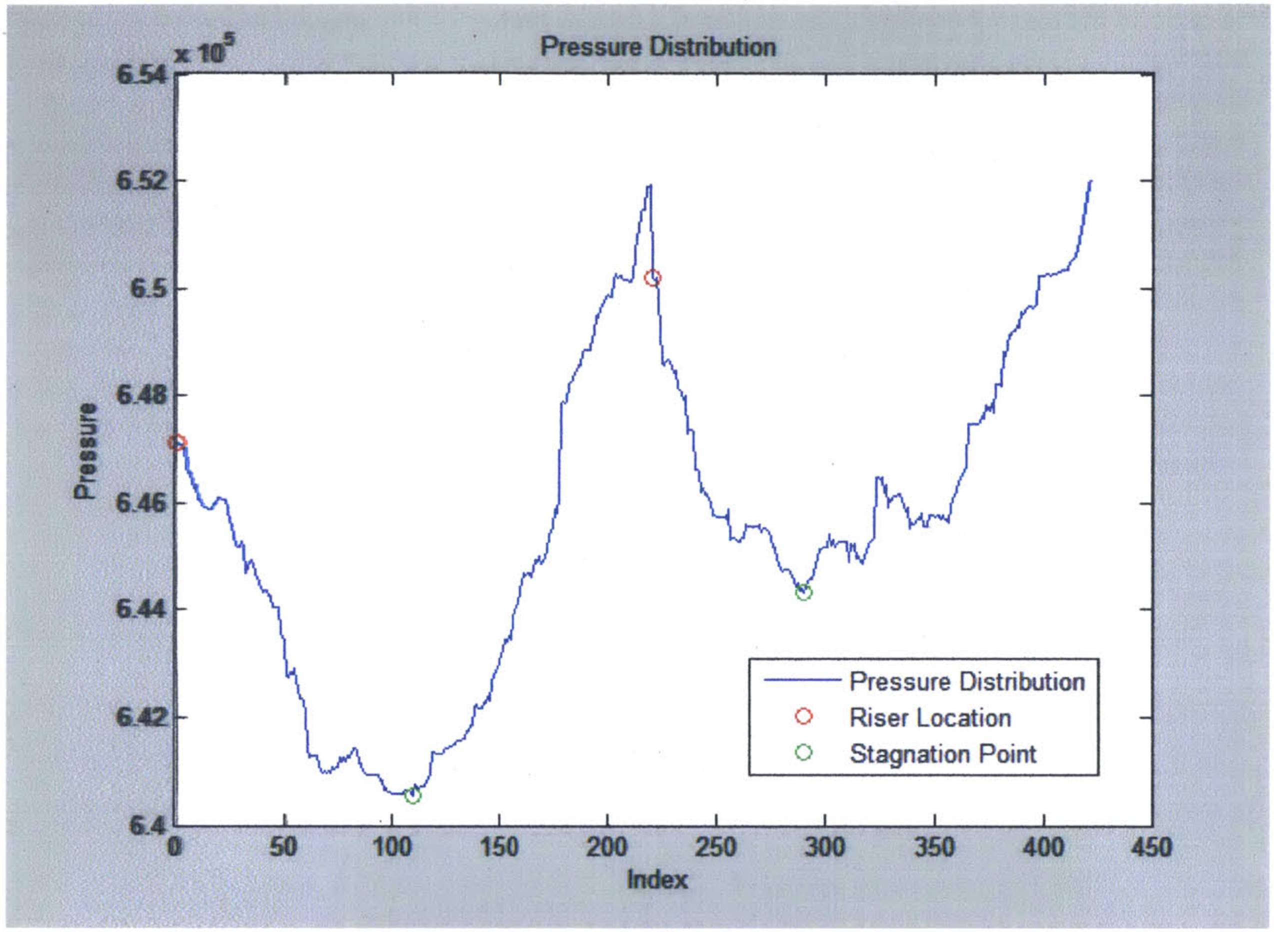


Figure 78: Pressure distribution after event

3.2.13.5 Part E: Final Velocities

The final velocities are computed in a similar manner as in Section 3.2.13.3. The difference comes in the calculation of the temperatures. A major assumption is the velocities change abruptly between the instant before and after the event. This assumption is made due to the difference in the timescale of the velocity transient and the temperature transient with the response of the temperature transient being much greater than the response of the fluid velocity transient. Further work could be done to eliminate this assumption and to incorporate the inertia of the fluid and the corresponding ramping up or down of the fluid velocity at each location within the piping structure.

3.2.13.6 Part F: Final Temperatures (Transient Response)

The purpose of the preceding steps was to determine the initial conditions prior to the event and the resulting change in velocity due to the reconfiguration of chiller operation. With this information, the transient temperature response can be determined using a finite element approach.

To perform the transient analysis, the cooling system was broken up into annular segments along the length of the pipe as shown in Figure 79. The length of the annular segment within the branches was determined by finding the minimum branch length and dividing it into five segments. If the segment length is greater than one meter, then the annular length for the branch piping is set to one meter, else the annular length calculated for the shortest branch is used for all branches. The supply header is then broken up into segments between distinct branch junction locations. The minimum distance between distinct branch junction locations is then found and the shortest length is segmented further into two segments. If this length is greater than one meter, then the annular length for the header piping is set to one meter, else the annular length calculated for the shortest header segment is used for all header segments. This approach is used to minimize the number of segments within the piping structure while maintaining some level of granularity. The user can not change the size of the annular lengths to prevent unstable responses.

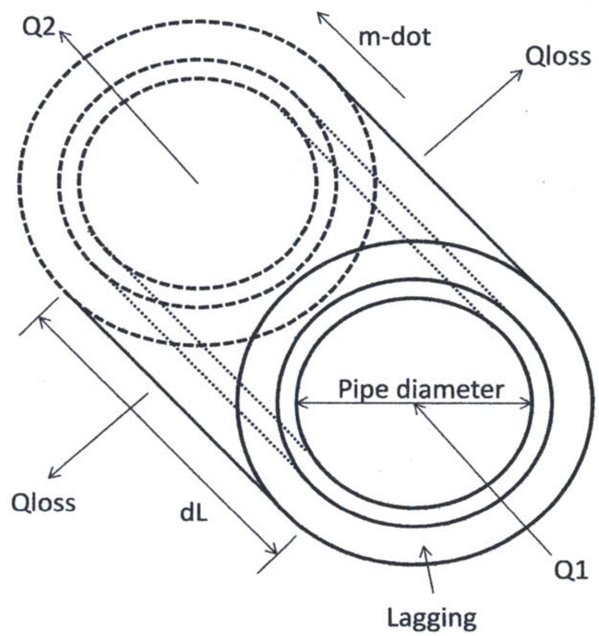


Figure 79: Annular element of cooling system piping

A time step is then determined. A very important criterion for the time step is it must be less than the length of the annular segment divided by the maximum velocity of the fluid. If this is violated, then an unsteady condition is possible, with temperatures dropping and increasing in greater amplitude after each time step. Therefore, to ensure a stable temperature response, values for the minimum time step is calculated for the branches and for the header (since they will have different maximum velocities and may have different annular lengths). The minimum of the two time steps calculated is then used rounded down to the nearest tenth of a second[15]. The user can change the time step but must be careful to not select a time step greater than the minimum recommended value. Decreasing the time

[15] The program takes the floor of the quotient. If the result is zero, then the program computes the time to the nearest hundredth of a second. If this is still too large, an error will be displayed.

step will have a profound effect on the computing time needed to iterate through the code as well as the amount of memory needed to store the large matrices.

The default length of time considered by the program is roughly 60 seconds. The user has the ability to change this value, again considering the impact of increasing time will increase the computing time and memory needed.

At this point, the main loop of the program determines the temperature at each node, incrementing time by the specified time step. The temperature of the annular element was taken to be the average temperature within the differential volume of fluid. To determine the change in temperature over a small time increment dt, the following equation was used:

$$\frac{dT}{dt} = \frac{1}{\rho c_p dV}\left(\dot{Q}_1 + \dot{Q}_2 + \dot{Q}_{loss} + \dot{Q}_{gen}\right)$$

Equation 77

where dV is the differential volume of the cylindrical element of fluid [m^3], $\dot{Q}_1$ is the rate of heat transfer into the volume from fluid entering the element [W], $\dot{Q}_2$ is the rate of heat transfer out of the volume from fluid exiting the element [W], $\dot{Q}_{loss}$ is the rate of heat transfer exiting the surface of the fluid in contact with the pipe wall [W], and $\dot{Q}_{gen}$ is the rate of heat transfer generated within the fluid due to friction [W].

For the heat flux across the surface of the pipe, heat transfer is by conduction across the pipe and lagging, but also by convection from the fluid to the pipe and from the lagging to the quiescent air external to the cooling system. The heat transfer equation for $\dot{Q}_{loss_{x,t}}$ for element x at time t follows:

$$\dot{Q}_{loss_{x,t}} = U_{x,t}A_x\left(T_\infty - T_{x,t}\right)$$

Equation 78

where T_∞ is the quiescent air temperature [°C], $T_{x,t}$ is the average fluid temperature for the x^{th} element at time t [°C], A_x is the surface area of the inner pipe wall for the x^{th} element, and $U_{x,t}$ is the overall heat transfer coefficient across the fluid to the quiescent air for the x^{th} element at time t [W/m^2-K]. The quiescent air temperature was taken to be 20°C at all locations. Segmenting the ship into blocks and determining the surrounding air temperature can also be an area of future work. The overall heat transfer coefficient can be computed as follows[16]:

$$U_{x,t} = \left(\frac{1}{h_{fluid_{x,t}}} + \frac{r_{1_x} ln\left(\frac{r_{2_x}}{r_{1_x}}\right)}{k_{Cu-Ni}} + \frac{r_1 ln\left(\frac{r_{3_x}}{r_{1_x}}\right)}{k_{lagging}} + \frac{r_{1_x}}{r_{3_x} h_{air_{x,t}}}\right)^{-1}$$

Equation 79

[16] The overall heat transfer coefficient is computed with respect to the inner pipe surface wall, and thus, the radius of the pipe is used as the reference radius. Accordingly, the surface area is that of the inner pipe surface wall.

where $r_{1_x}, r_{2_x}, r_{3_x}$ are the respective radii of the fluid, the copper-nickel alloy pipe, and the lagging for the x^{th} element [m], k_{Cu-Ni} is the thermal conductivity of the copper-nickel alloy pipe[17] [W/m-K], $k_{lagging}$ is the thermal conductivity of the lagging[18] [W/m-K], $h_{fluid_{x,t}}$ is the convective heat transfer coefficient of the fluid within the pipe [W/m^2-K] for the x^{th} element at time t , and $h_{air_{x,t}}$ is the convective heat transfer coefficient of the air external to the cooling system [W/m^2-K] for the x^{th} element at time t.

The convective heat transfer of the fluid within the pipe was determined using the equations below, depending on the flow regime. For laminar flow:

$$h_{fluid_{x,t}} = 3.66\frac{k_{fluid}}{2r_{1_x}}$$

Equation 80 (derived from Equation 11)

which is independent of time. For turbulent flow:

$$h_{fluid_{x,t}} = 0.023\frac{V_{x,t}^{0.8}k_{fluid}^{0.6}(\rho c_p)_{fluid}^{0.4}}{2r_{1_x}^{0.2}\nu_{fluid}^{0.4}}$$

Equation 81 (derived from Equation 12)

which is valid for $Pr > 0.5$ and $Re > 10{,}000$. If the Reynolds number falls within the transition range, then Gnielsinki's formula was used to determine the convective heat transfer coefficient.

The convective heat transfer of the quiescent air was computed using the equations for natural convection of horizontal cylinders:

$$\bar{h}_{air} = \frac{\overline{Nu}_D k_{air}}{2D_x}$$

Equation 82 (Incropera & DeWitt, 2002)

where, for a horizontal cylinder:

$$\overline{Nu}_D = \left\{0.60 + \frac{0.387Ra_{D_{x,t}}^{1/6}}{\left[1+\left(\frac{0.559}{Pr}\right)^{9/16}\right]^{8/27}}\right\}^2$$

Equation 83 (Incropera & DeWitt, 2002)

where $Ra_{D_{x,t}}$ is the Rayleigh number for the x^{th} element at time t. The Rayleigh number can be computed using the equation:

---

[17] The default value used is for Cu-Ni 70-30 alloy with a value of 50 W/m-K.
[18] The default value used for the insulation was 0.035 W/m-K.

$$Ra_{D_{x,t}} = \frac{g\beta(T_{s_{x,t}} - T_\infty)2r_{3_x}^3}{\alpha\nu}$$

Equation 84 (Incropera & DeWitt, 2002)

where g is gravity, β is the fluid coefficient of thermal volumetric expansion, $T_{s_{x,t}}$ is the surface temperature of the lagging for the x^{th} element at time t [°C], and α is the thermal diffusivity [m^2/s].

The fluid coefficient of thermal volumetric expansion for air can be found using the equation:

$$\beta = T_{air}^{-1}$$

Equation 85 (Incropera & DeWitt, 2002)

where T_{air} is the air temperature, taken to be the average between the surface temperature and the quiescent air temperature [°K].

The majority of the cooling system involves horizontal cylinders, so for preliminary analyses, this was the only equation used for the external environment.

The rate of heat generation within the fluid is based solely on friction of the fluid with the piping. Friction causes the conversion of mechanical energy to internal energy of the fluid. This conversion of energy can be accounted through the pressure drop that takes place along some length of pipe. The heat transfer equation for $\dot{Q}_{gen_{x,t}}$ for element x at time t follows:

$$\dot{Q}_{gen_{x,t}} = \left(\frac{K_{x,t}V_{x,t}^2}{2gc_p \cdot 778.169\frac{ft-lb}{BTU}}\right)\left(\frac{5°K}{9°F}\right)\left(\frac{\rho c_p dV_{x,t}}{dt}\right)$$

Equation 86 (Incropera & DeWitt, 2002)

where $K_{x,t}$ is the loss coefficient along the length of the annular segment for element x at time t (dimensionless), $V_{x,t}^2$ is the fluid velocity for element x at time t, $778.169\frac{ft-lb}{BTU}$ is a conversion factor, c_p is the specific heat capacity with units of $\frac{BTU}{lbf-°F}$ within the first set of brackets and units of $\frac{J}{kg-°K}$ within the third set of brackets[19], $dV_{x,t}$ is the differential volume of the fluid[20] within the annular element for element x at time t, and dt is the incremental time step set by the user. Of note, the loss coefficient is the sum of the loss coefficient due to friction of the fluid along the pipe, the loss coefficient due to bends within the pipe, the loss coefficient due to various valves, and the loss coefficient due to entrance and exit effects of piping. The loss coefficient due to friction along the length of the pipe is a continuous variable and is a function of the length of the pipe. However, the other loss coefficients are treated as discrete variables. Because of this, these loss coefficients are lumped into a single element. For example, a particular gate valve may extend into 3 elements (if the analysis is done with sufficient granularity). The loss coefficient associated with the gate valve would then be attributed to only one of these

[19] A consequence of working in both English units and metric units
[20] Not to be confused with the derivative of velocity

elements, say, the second of the three elements. This was initially considered; however, after careful consideration, the contribution of $\dot{Q}_{gen_{x,t}}$ due to friction is negligible for the speeds considered and only comes into play for fluid velocities approaching the speed of sound.

The remaining two variables, $\dot{Q}_1$ and $\dot{Q}_2$, is greatly dependent on the fluid velocity. The two variables can be thought of as accounting for the amount of heat transferred by the slug of water preceding the annular segment from the previous time step which is occupying the annular segment in the current time step and the amount of heat transferred by the slug of water which occupied the annular segment in the preceding time step which has since moved to the following segment in the current time step. This is the reason that the time step is so critical. The slug of water being transferred between time steps must be equal to or less than the actual volume of water occupying the annular segment, else instabilities may result.

The temperature at each node is then calculated by taking the temperature from the preceding time step at the same location and adding the corresponding differential temperature change over the time step in question. This is shown in the equation:

$$T_{x,t} = T_{x,t-1}\left(\frac{dT}{dt}\right)_{x,t} dt_{x,t}$$

Equation 87

where $T_{x,t}$ is the temperature at location x and time t, $T_{x,t-1}$ is the temperature at location x and time $t-1$, $\left(\frac{dT}{dt}\right)_{x,t}$ is the differential temperature at location x and time t over the time step, and $dt_{x,t}$ is the time step.

3.2.13.7 Part G: Plots

The last portion of the transient analysis plots the temperature response with respect to time and/or location. The first option provided by the program is the temperature response as a function of time. The program prompts the user to specify the general location under consideration: supply header, branch, or return header.

If the supply header (or return header) is selected, the program provides the user with pertinent indices including the indices corresponding to the riser locations, the indices corresponding to the stagnation points and the indices for all branch junctions. The program then prompts the user for the supply header (or return header) index which the user wishes to analyze. The output is a plot of temperature starting at the steady-state temperature at that location computed as described in Section 3.2.13.3 and the corresponding transient temperature response over the time interval specified.

If the branch is selected, the program provides the user with the number of branches in the chilled water system. The user must specify the branch which is to be analyzed. The program then displays the number of indices within the specified branch along with the index of the heat exchanger in that branch.

The user is then prompted for the branch index which is to be analyzed. The output is similar to that described above. An example of the temperature response is shown in Figure 80.

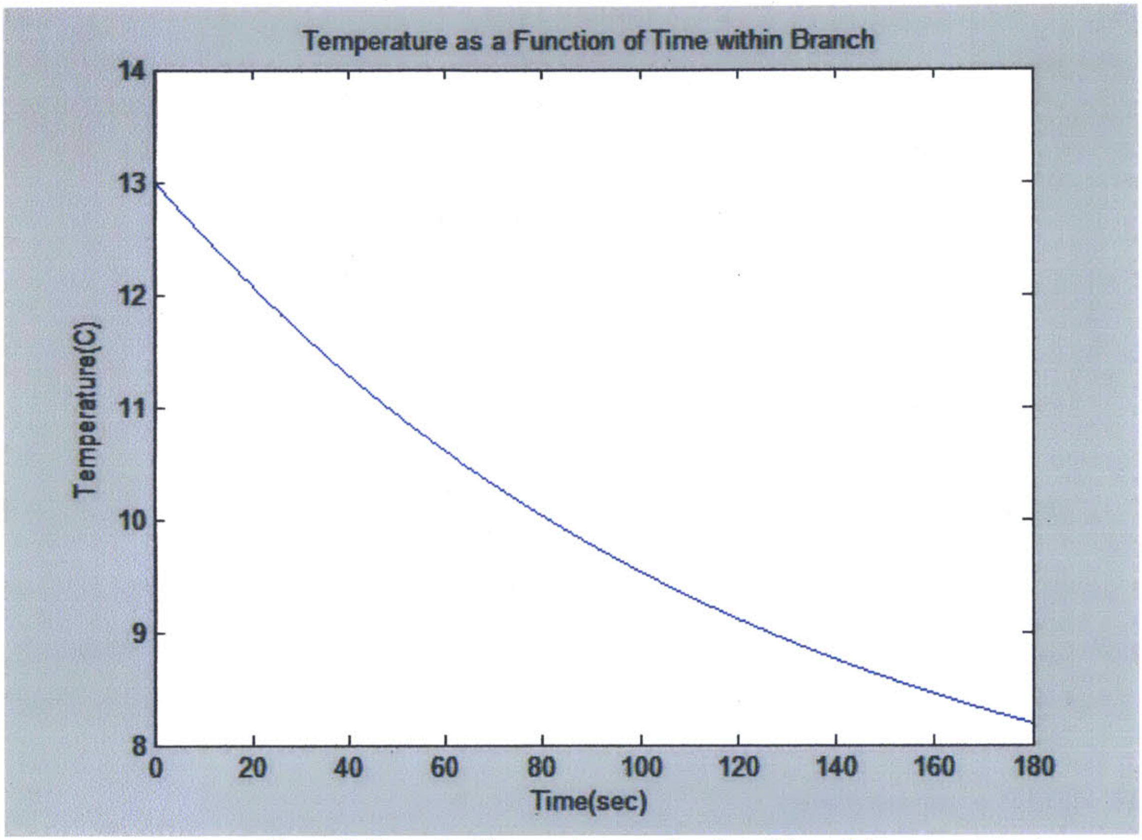


Figure 80: Example of temperature as a function of time plot

The user can look at other locations until they are satisfied and exits the loop. At this point, the program asks the user if they want a plot of the temperature distribution over a section of pipe at a specified time. The user selects the general location to be analyzed as before choosing between the supply header, the return header or a branch. If the supply header or return header is selected, the user is only prompted for the time at which the temperature distribution is to be plotted. If the user specifies a branch, the user must enter the branch number and the time. The program outputs the temperature distribution at the specified time. An example of the temperature distribution at a specific time is shown in Figure 81 below.

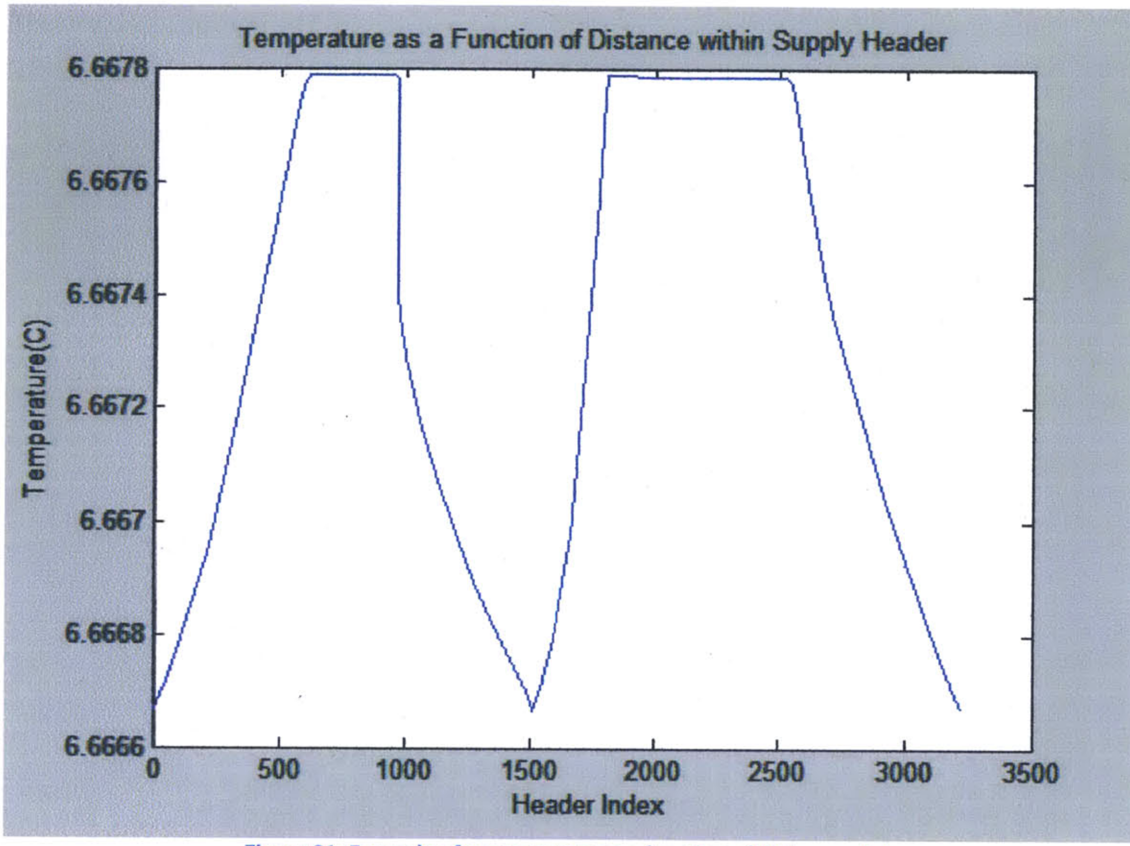


Figure 81: Example of temperature as a function of distance plot

3.2.14 Validation of the Model

To validate the model, the time dependent output of the model was compared with a simple example that could be solved analytically. The example focused on verifying how the CSDT models conductive heat transfer from the fluid through the pipe, through the lagging and to the surrounding quiescent air.

The example used to validate the model considered the outer surface temperature of the supply header pipe. The pipe considered was a nickel-copper 70-30 alloy with a density of 8950 kg/m^3, a thermal conductivity of 50 W/m^2-K and a specific heat capacity of 376.812 J/kg-K. The pipe had a diameter of 59.055 mm and a thickness of 2.1082 mm. The lagging had a thickness of 1 cm and a thermal conductivity of 0.035 W/m^2-K. The initial temperature of the pipe, fluid, lagging and quiescent air was 20°C. At time t=0- seconds, the fluid had a velocity of 1.5288 m/s, and the fluid temperature was 20°C. Friction was ignored along with the heat generated due to friction. At time t=0+ seconds, the fluid had the same velocity, but the fluid temperature was 6.6°C, representing the fluid exiting the chiller. The step response of the fluid temperature can be seen in Figure 82.

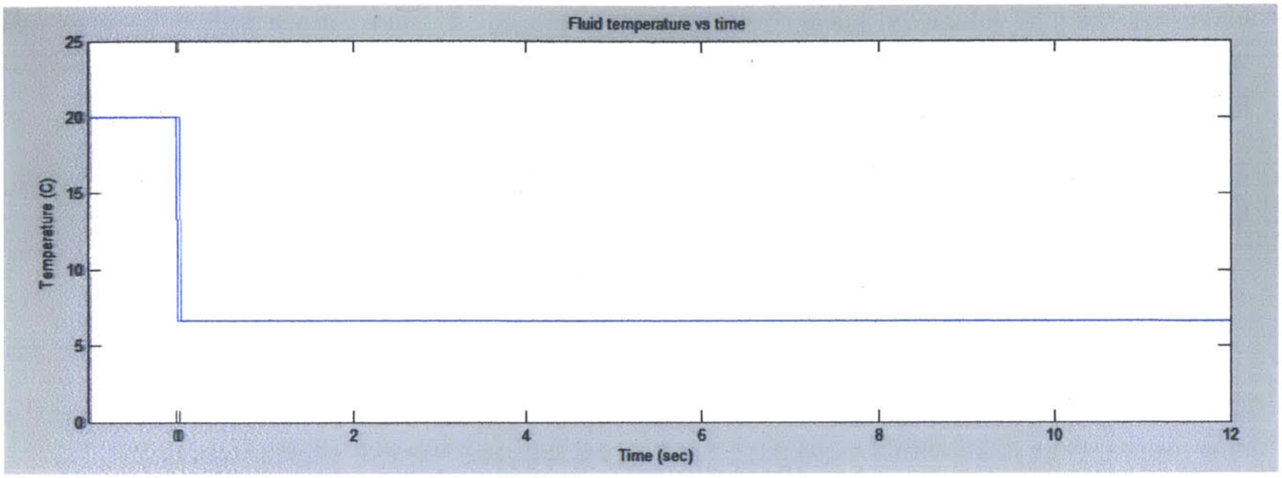


Figure 82: Fluid temperature versus time

The example was first modeled using the lumped capacitance method. The equation used to determine the outer pipe wall temperature was:

$$T = T_\infty + e^{-B_i F_o}(T_i - T_\infty)$$

Equation 88 (Incropera & DeWitt, 2002)

where T_∞ is the temperature of the bulk fluid [°C], in this case it is 6.6°C, T_i is the original temperature of the pipe wall [°C], in this case it is 20°C, F_o is the Fourier number, and B_i is the Biot number.

The Fourier number is dimensionless time that corresponds to the ratio of the heat conduction rate to the rate of thermal energy storage in a solid. The Fourier number can be found using the equation:

$$F_o = \frac{\alpha t}{L_c}$$

Equation 89 (Incropera & DeWitt, 2002)

where α is the thermal diffusivity [m^2/s], t is time [s], and L_c is the characteristic length.

The thermal diffusivity can be found using the equation:

$$\alpha = \frac{k}{\rho c_p}$$

Equation 90 (Incropera & DeWitt, 2002)

For the copper-nickel alloy pipe, the thermal diffusivity was found to be 1.4826x10^{-5} m^2/s.

The characteristic length can be found using the equation:

$$L_c = \frac{V}{A_s}$$

Equation 91 (Incropera & DeWitt, 2002)

where V is the pipe volume [m^3] over some arbitrary length, and A_s is the surface area of the inner wall [m^2] over the same arbitrary length. For the supply header pipe, the characteristic length was found to be 1.0729 mm, which is approximately half of the pipe thickness.

The Biot number corresponds to the ratio of the internal thermal resistance of a solid to the boundary layer thermal resistance. The Biot number can be found using the equation:

$$B_i = \frac{hL_c}{k}$$

Equation 92 (Incropera & DeWitt, 2002)

To calculate the Biot number, the convective heat transfer coefficient was needed. This depends on the flow regime of the fluid. With the diameter and fluid velocity, the Reynolds number was easily calculated to be 62,265. This corresponds to fully turbulent flow and the equation:

$$h = 0.023\frac{V^{0.8}k^{0.6}(\rho c_p)^{0.4}}{D^{0.2}\nu^{0.4}}$$

Equation 12 (repeated)

was valid in determining the convective heat transfer coefficient since $Re > 10{,}000$ and $Pr > 0.5$ (Pr for water at 6.6°C is about 10.7). The Biot number was determined to be 0.1615. This value is greater than what is recommended for the lumped capacitance model to be used ($B_i < 0.1$), but was computed due to its ease with the knowledge that the results of the lumped capacitance model would have some error associated with it.

To get a better estimate of the outer pipe wall surface temperature, the pipe wall was modeled as a semi-infinite wall. This is reasonable since the thickness of the wall is much less than the diameter of the pipe. With lagging on one side of the pipe, a wall of thickness L with an adiabatic condition on one surface and some surface condition on the other surface corresponds to a wall of thickness 2L with symmetric surface conditions on both walls due to the boundary condition at $x^* = 0$ is similarly $\frac{\partial \theta^*}{\partial x^*} = 0$. This is illustrated in Figure 83 below.

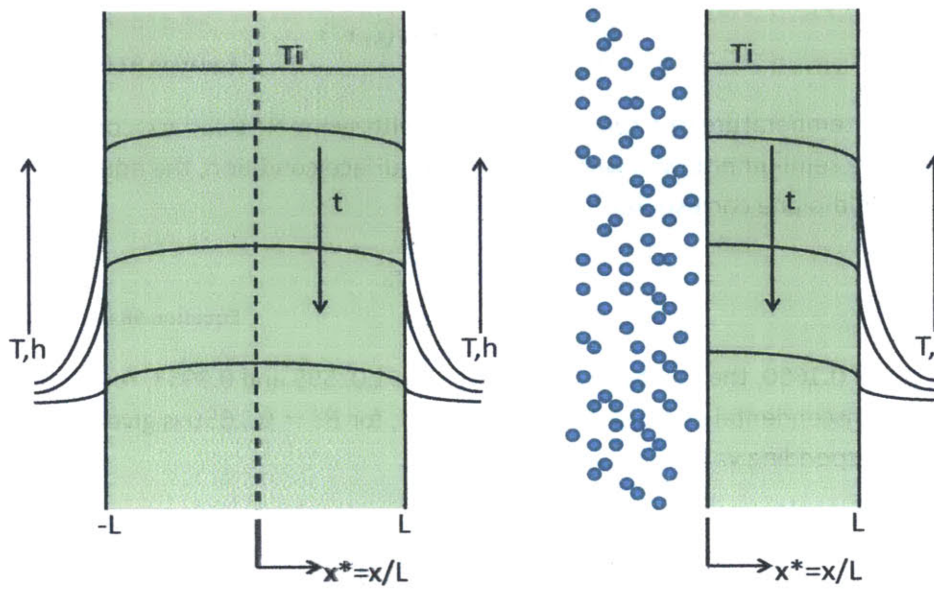


Figure 83: Equivalence of plane wall with symmetric convection (left) and adiabatic surface (right)

To simplify the analysis, radiation was considered negligible, and thus omitted from the analysis. With these assumptions, the temperature within the semi-infinite solid wall could be solved analytically. An exact analytical solution can be obtained through the infinite series:

$$\theta^* = \sum_{n=1}^{\infty} C_n e^{-\zeta_n^2 F_o} cos(\zeta_n x^*)$$

Equation 93 (Incropera & DeWitt, 2002)

where x^* is the dimensionless form of the cylinder radius with

$$x^* = \frac{x}{L}$$

Equation 94 (Incropera & DeWitt, 2002)

and the coefficient C_n is given by

$$C_n = \frac{4sin(\zeta_n)}{2\zeta_n + sin(2\zeta_n)}$$

Equation 95 (Incropera & DeWitt, 2002)

and the discrete values of ζ_n are positive roots of the transcendental equation

$$\zeta_n tan(\zeta_n) = Bi$$

Equation 96 (Incropera & DeWitt, 2002)

An approximate solution can be obtained by including only the first term of the infinite series. This reduces the above equation to:

$$\theta^* = C_1 e^{-\zeta_1^2 F_o} cos(\zeta_1 x^*)$$

Equation 97 (Incropera & DeWitt, 2002)

Since the mid-plane temperature of a semi- infinite wall with symmetric surface conditions corresponds to the outer wall of a semi-infinite wall with an adiabatic surface condition, the equation can be further reduced since the mid-plane corresponds to $x^* = 0$.

$$\theta^* = C_1 e^{-\zeta_1^2 F_o}$$

Equation 98 (Incropera & DeWitt, 2002)

For a Biot number of 0.1650, the coefficients C_1 and ζ_1 are 1.02595 and 0.3953, respectively. The first four roots of the transcendental equation $\zeta_n tan(\zeta_n) = Bi$, for $Bi = 0.1650$ is given in Table 14 below along with the corresponding values for C_n.

	$Bi = 0.1650$	
n	ζ_n	C_n
1	0.3953	1.02595
2	3.1933	1.00455
3	6.3093	1.00452
4	9.4423	1.00451

Table 14: First four roots and associated coefficients for Bi=0.1650

The CSDT model considered the rate of heat transferred from the cooler fluid and into the pipe. The model also considered the rate of heat transferred from the warmer quiescent air external to the lagging, through the lagging and into the pipe. The model actually computes the average temperature within the pipe and not the temperature external to the pipe; however, since the temperature gradient across the pipe wall is small, the average pipe temperature gives a good approximation to the external surface temperature of the pipe.

A plot of the pipe outer wall temperatures for the various analytical methods described above along with the predicted pipe outer wall temperature versus time is shown in Figure 84 below. As can be seen from the figure, the CSDT model is in close agreement with the analytical models. For small values of time, there is some disagreement with the series solution model. This is due to the error associated with the approximated series solution for values of $F_o < 0.2$ which corresponds to $t < 0.06\ sec$. To get a highly refined curve, the time step used within the model was 0.01 sec.

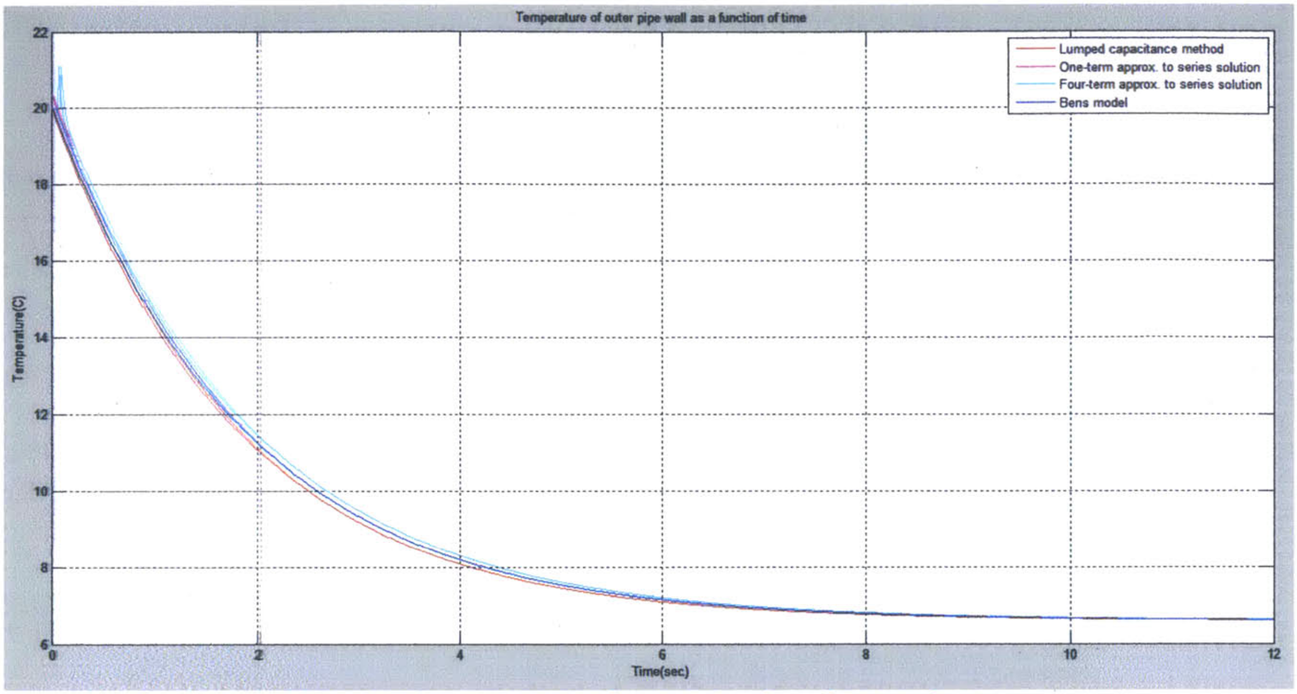


Figure 84: Surface temperature of outer pipe wall as a function of time

In addition to the example discussed above, the output of the transient analysis code was compared to the output of the steady-state code. After enough time, the temperature transient passes and a state of thermal equilibrium is reached. These temperature values were calculated for several elements of a simple cooling system network. The steady-state temperatures were calculated at the same locations of the cooling system network. The cooling system modeled comprised of four heat loads all of equal value (3 kW). The four heat loads were connected in parallel, with a single supply header and a single return header. The cooling system had a single chiller, pump and expansion tank.

There is much agreement between the two methods along the length of the piping system with differences less than 0.01°C. This gives greater confidence in the validity of the transient code.

3.3 Design Guidelines

In designing the cooling system, there are many criteria that must be satisfied. These criteria are in place to ensure adequate redundancy and survivability of the cooling system. These criteria focus on the main piping system separation, the isolation of the cooling systems vital and non-vital loads, and the additional capacity of the chillers to supply vital loads with cooling when the ship has sustained battle damage.

Depending on the level of redundancy required, the main piping system may consist of a single main or a double main. Single mains comprise of a single supply and return header which runs longitudinally, centerline of the ship. For the double main system, separation between the two mains is essential for survivability. Athwartship separation of the double main piping system is achieved by placing the mains close to the most outboard structure. The port and starboard mains are also separated vertically.

The risers are vertical sections of pipe that connect the chiller to the main piping. A segregation valve should be located on either side of the main where the riser connects to the main to allow restriction of flow either clockwise or counterclockwise from the junction. In addition, the riser should have a segregation valve right before the connection to the main to allow for total isolation of flow from that riser.

The design of the cooling system should also satisfy some damage loss criteria. The damage criteria may be damage along some length of the ship as a percent of the ship's length, or it may be a specified number of compartments (e.g., 2 compartment flooding). Considering a loss of all chillers located within the worst case damage scenario should not degrade the ability of the entire cooling system in supplying cooling to all vital loads.

4.0 Chapter 4: Simulation & Results

A simulation of a chilled water system was conducted utilizing the CSDT to model the chilled water system and the auxiliary seawater system. The simulation was conducted with all analyses performed.

The simulation included the same heat loads used within CSDT v1.0 (Fiedel, 2011). These heat loads are summarized in Appendix A. The simulation included all default values provided by the program with the exception of the number of zonal boundaries, which was set to three for more efficient sizing of the A/C units as well as the addition of auxiliary seawater piping to the shaft bearing and auxiliary seawater piping to three generic SW/XX heat exchangers. The breakdown of heat loads by compartment and by zone is shown in Figure 85. A 3-D representation is shown in Figure 86 below.

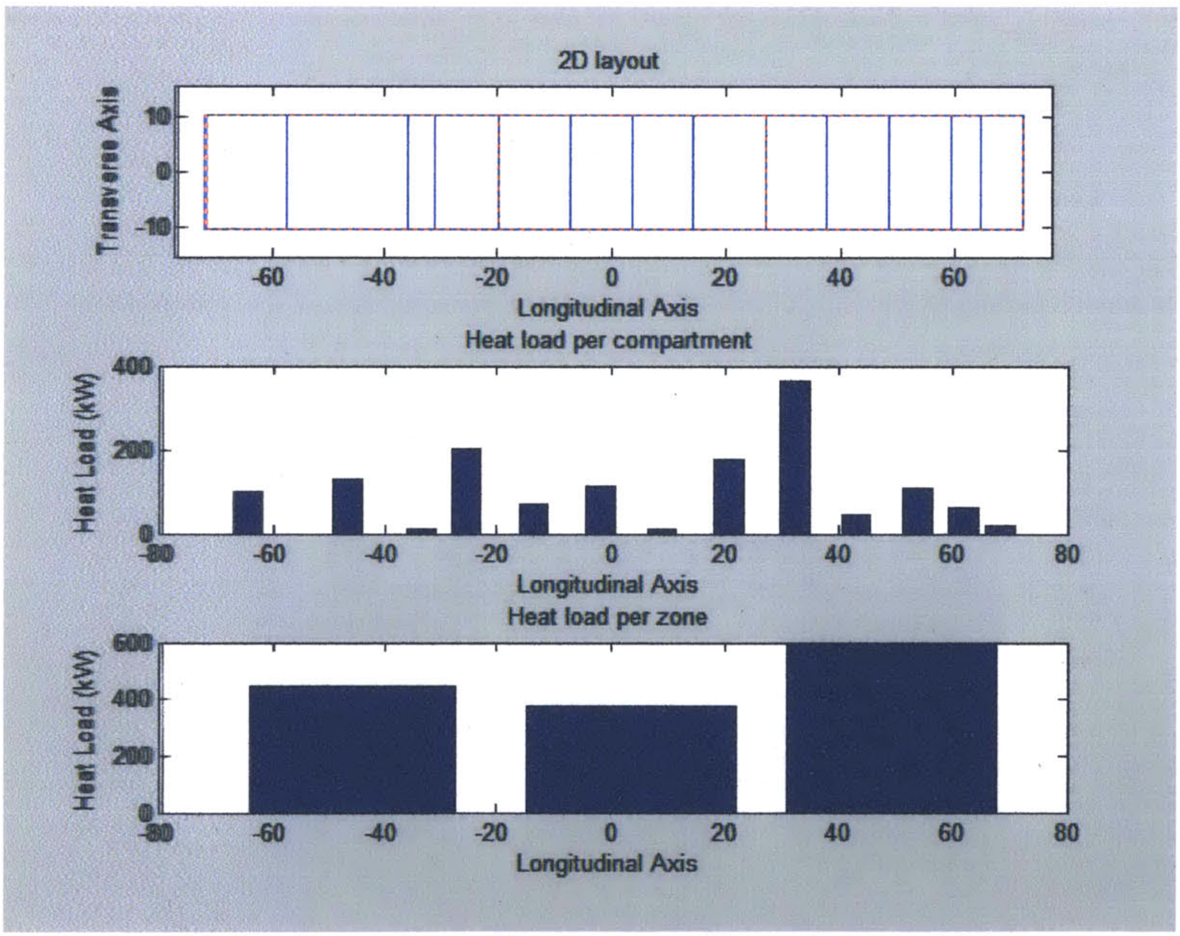


Figure 85: Breakdown of heat load by compartment and by zone for simulated design

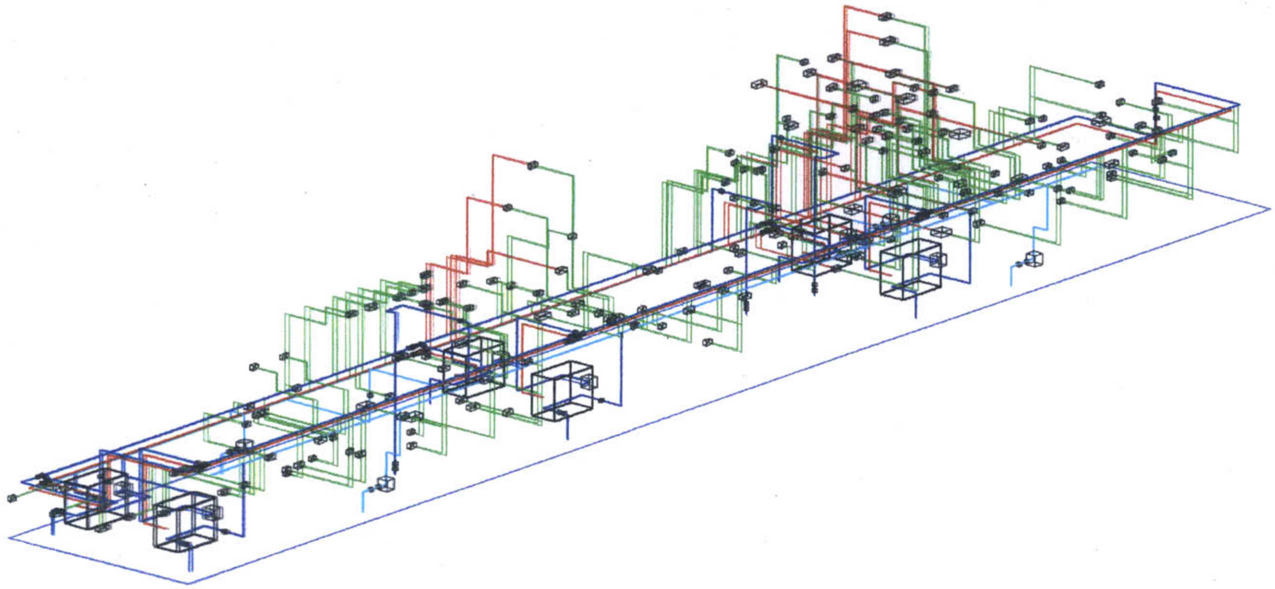


Figure 86: 3-D representation of chilled water system and auxiliary seawater system for simulated design

The program provides a few reports throughout the design of the chilled water system. The first two of these reports pertain to the sizing of the A/C units. For the simulated design, the reports are:

Report 1: Minimum Chiller Capacity

Chiller 1	Chiller Capacity(tons):	89.1544	Chiller Capacity(kW):	313.5430
Chiller 2	Chiller Capacity(tons):	134.8273	Chiller Capacity(kW):	474.1678
Chiller 3	Chiller Capacity(tons):	95.7981	Chiller Capacity(kW):	336.9077
Chiller 4	Chiller Capacity(tons):	18.4245	Chiller Capacity(kW):	64.7961
Chiller 5	Chiller Capacity(tons):	25.5806	Chiller Capacity(kW):	89.9633
Chiller 6	Chiller Capacity(tons):	39.2951	Chiller Capacity(kW):	138.1952
Total	Chiller Capacity(tons):	403.0800	Chiller Capacity(kW):	1417.5730

```
--------------------------------------------------------------------------------
Report 2: Default Chillers Selected
--------------------------------------------------------------------------------
Chiller 1  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
Chiller 2  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
Chiller 3  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
Chiller 4  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
Chiller 5  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
Chiller 6  Chiller Capacity(tons):  147.8595   Chiller Capacity(kW):  520.0000
--------------------------------------------------------------------------------
Total      Chiller Capacity(tons):  887.1568   Chiller Capacity(kW):  3120.0000
Capacity Installed/Minimum Capacity Required: 2.20
Minimum number of chillers needed to meet maximum heat load demands: 3
```

As can be seen by report 1, the largest capacity chiller is chiller 2. This makes sense when looking at the 3-D model of the chilled water system. Most of the heat loads are located in the forward-most zone. By default, the program allocates the starboard side chiller to support any loads which are centerline. Thus, with chiller 2 being the forward-most chiller on the starboard side, it is expected that this chiller will need to have the highest capacity.

Report 2 shows what the program sets each chiller's capacity to. They are all equal and are the smallest sized chillers within the chiller database which meets the requirements specified in Section 3.2.9. Report 2 also shows that the installed chiller capacity is 220% greater than what is needed, but this provides redundancy (at a cost and weight penalty). Only three of the six chillers are needed to meet the cooling needs of the ship at any given time.

Report 3 provides the sizing of the expansion tanks. For the simulated design, report 3 is:

```
--------------------------------------------------------------------------------
Report 3: Expansion Tank Sizing
--------------------------------------------------------------------------------
Expansion Tank Height(m):    1.980539
Expansion Tank Radius(m):    0.990270
Expansion Tank Thickness(mm): 4.000000
--------------------------------------------------------------------------------
```

4.1 Static Analysis

The first analysis performed was the static temperature analysis. All fluid flow and heat transfer is assumed to be in steady-state. When performing the static analysis, all four operating conditions should be considered along with all possible combinations of chillers in operation to ensure flow and cooling requirements are met under all conditions. An example of the static temperature output for the design condition and for a single chiller configuration is provided below.

Load: 1 Q(W): 562.7200 Diameter(m): 0.01532 Velocity(m/sec): 1.1977 Mass flow rate(kg/s): 0.2207 Thot(C): 7.2737 Telec(C): 7.9882
Load: 2 Q(W): 14490.0400 Diameter(m): 0.03975 Velocity(m/sec): 0.5093 Mass flow rate(kg/s): 0.1522 Thot(C): 12.6038 Telec(C): 10.4922
Load: 3 Q(W): 3798.3600 Diameter(m): 0.01951 Velocity(m/sec): 0.6579 Mass flow rate(kg/s): 0.1212 Thot(C): 7.7715 Telec(C): 8.2372
Load: 4 Q(W): 1336.4600 Diameter(m): 0.01532 Velocity(m/sec): 1.4005 Mass flow rate(kg/s): 1.7381 Thot(C): 8.9779 Telec(C): 9.7203
Load: 5 Q(W): 562.7200 Diameter(m): 0.01532 Velocity(m/sec): 0.6509 Mass flow rate(kg/s): 0.4909 Thot(C): 12.0705 Telec(C): 11.9295
Load: 6 Q(W): 1793.6700 Diameter(m): 0.01532 Velocity(m/sec): 1.3833 Mass flow rate(kg/s): 0.2549 Thot(C): 7.5206 Telec(C): 8.7480
Load: 7 Q(W): 16881.6000 Diameter(m): 0.03975 Velocity(m/sec): 1.2070 Mass flow rate(kg/s): 0.9103 Thot(C): 9.2790 Telec(C): 10.2692
Load: 8 Q(W): 17690.8617 Diameter(m): 0.03975 Velocity(m/sec): 1.2435 Mass flow rate(kg/s): 3.2040 Thot(C): 9.2213 Telec(C): 8.7854
Load: 9 Q(W): 11149.2417 Diameter(m): 0.03099 Velocity(m/sec): 1.0825 Mass flow rate(kg/s): 2.7893 Thot(C): 11.4490 Telec(C): 10.5077
Load: 10 Q(W): 1899.1800 Diameter(m): 0.01532 Velocity(m/sec): 0.6657 Mass flow rate(kg/s): 0.1227 Thot(C): 9.9416 Telec(C): 11.3539
Load: 11 Q(W): 914.4200 Diameter(m): 0.01532 Velocity(m/sec): 1.1023 Mass flow rate(kg/s): 0.5223 Thot(C): 8.6697 Telec(C): 10.8986
Load: 12 Q(W): 949.5900 Diameter(m): 0.01532 Velocity(m/sec): 1.4798 Mass flow rate(kg/s): 0.2726 Thot(C): 7.8026 Telec(C): 9.5877
Load: 13 Q(W): 9993.5555 Diameter(m): 0.03099 Velocity(m/sec): 0.5852 Mass flow rate(kg/s): 0.1078 Thot(C): 10.3148 Telec(C): 11.4785
Load: 14 Q(W): 1336.4600 Diameter(m): 0.01532 Velocity(m/sec): 0.9372 Mass flow rate(kg/s): 0.7068 Thot(C): 9.4018 Telec(C): 9.8892
Load: 15 Q(W): 34396.6117 Diameter(m): 0.05728 Velocity(m/sec): 0.5559 Mass flow rate(kg/s): 0.1024 Thot(C): 9.6898 Telec(C): 10.5305
Load: 16 Q(W): 9706.9200 Diameter(m): 0.03099 Velocity(m/sec): 1.4551 Mass flow rate(kg/s): 5.5393 Thot(C): 9.2215 Telec(C): 9.6364
Load: 17 Q(W): 56060.9800 Diameter(m): 0.05728 Velocity(m/sec): 1.4460 Mass flow rate(kg/s): 0.2664 Thot(C): 6.7926 Telec(C): 6.9841
Load: 18 Q(W): 6506.4500 Diameter(m): 0.02456 Velocity(m/sec): 1.2766 Mass flow rate(kg/s): 0.2352 Thot(C): 7.7937 Telec(C): 9.2436
Load: 19 Q(W): 1688.1600 Diameter(m): 0.01532 Velocity(m/sec): 0.5144 Mass flow rate(kg/s): 0.0948 Thot(C): 7.0202 Telec(C): 7.0979
Load: 20 Q(W): 3851.8184 Diameter(m): 0.01951 Velocity(m/sec): 0.9333 Mass flow rate(kg/s): 0.7039 Thot(C): 9.8766 Telec(C): 10.4425
Load: 21 Q(W): 4396.2500 Diameter(m): 0.02456 Velocity(m/sec): 0.9050 Mass flow rate(kg/s): 0.1667 Thot(C): 7.5704 Telec(C): 8.2636
Load: 22 Q(W): 932.3567 Diameter(m): 0.01532 Velocity(m/sec): 0.6661 Mass flow rate(kg/s): 0.3156 Thot(C): 10.2196 Telec(C): 11.9047
Load: 23 Q(W): 1301.2900 Diameter(m): 0.01532 Velocity(m/sec): 1.2206 Mass flow rate(kg/s): 0.3648 Thot(C): 8.1238 Telec(C): 11.4333
Load: 24 Q(W): 9144.5517 Diameter(m): 0.03099 Velocity(m/sec): 0.5808 Mass flow rate(kg/s): 0.1070 Thot(C): 10.5771 Telec(C): 11.7989
Load: 25 Q(W): 1652.9900 Diameter(m): 0.01532 Velocity(m/sec): 1.0160 Mass flow rate(kg/s): 0.7663 Thot(C): 9.0368 Telec(C): 10.3695
Load: 26 Q(W): 3165.3000 Diameter(m): 0.01951 Velocity(m/sec): 0.8428 Mass flow rate(kg/s): 0.6356 Thot(C): 9.4052 Telec(C): 10.4463
Load: 27 Q(W): 8124.2700 Diameter(m): 0.03099 Velocity(m/sec): 0.9389 Mass flow rate(kg/s): 0.1730 Thot(C): 8.1666 Telec(C): 9.3908
Load: 28 Q(W): 6524.7384 Diameter(m): 0.02456 Velocity(m/sec): 1.5714 Mass flow rate(kg/s): 0.2895 Thot(C): 7.9676 Telec(C): 10.1828
Load: 29 Q(W): 1301.2900 Diameter(m): 0.01532 Velocity(m/sec): 0.9491 Mass flow rate(kg/s): 0.4497 Thot(C): 8.7886 Telec(C): 10.6803
Load: 30 Q(W): 2110.2000 Diameter(m): 0.01532 Velocity(m/sec): 0.6374 Mass flow rate(kg/s): 0.1174 Thot(C): 8.5911 Telec(C): 9.3476
Load: 31 Q(W): 59474.5802 Diameter(m): 0.06962 Velocity(m/sec): 0.7490 Mass flow rate(kg/s): 0.3549 Thot(C): 9.9208 Telec(C): 9.8826
Load: 32 Q(W): 1336.4600 Diameter(m): 0.01532 Velocity(m/sec): 1.8033 Mass flow rate(kg/s): 4.6465 Thot(C): 8.3689 Telec(C): 8.3560
Load: 33 Q(W): 140.6800 Diameter(m): 0.01532 Velocity(m/sec): 0.6602 Mass flow rate(kg/s): 0.1216 Thot(C): 8.1805 Telec(C): 8.8232
Load: 34 Q(W): 879.2500 Diameter(m): 0.01532 Velocity(m/sec): 1.5795 Mass flow rate(kg/s): 1.9602 Thot(C): 8.6222 Telec(C): 9.4726
Load: 35 Q(W): 1113.8339 Diameter(m): 0.01532 Velocity(m/sec): 0.5395 Mass flow rate(kg/s): 0.0994 Thot(C): 8.7718 Telec(C): 9.3095
Load: 36 Q(W): 14771.4000 Diameter(m): 0.03975 Velocity(m/sec): 1.0280 Mass flow rate(kg/s): 0.1894 Thot(C): 6.8438 Telec(C): 7.0101
Load: 37 Q(W): 140.6800 Diameter(m): 0.01532 Velocity(m/sec): 0.6602 Mass flow rate(kg/s): 0.1216 Thot(C): 9.2125 Telec(C): 10.2939
Load: 38 Q(W): 13681.1300 Diameter(m): 0.03975 Velocity(m/sec): 1.9355 Mass flow rate(kg/s): 0.3566 Thot(C): 8.0287 Telec(C): 11.0402
Load: 39 Q(W): 9495.9000 Diameter(m): 0.03099 Velocity(m/sec): 1.6082 Mass flow rate(kg/s): 0.2963 Thot(C): 7.7684 Telec(C): 9.6984
Load: 40 Q(W): 1055.1000 Diameter(m): 0.01532 Velocity(m/sec): 0.9453 Mass flow rate(kg/s): 1.1732 Thot(C): 11.2034 Telec(C): 10.2587
Load: 41 Q(W): 633.0600 Diameter(m): 0.01532 Velocity(m/sec): 0.6619 Mass flow rate(kg/s): 0.1220 Thot(C): 7.7649 Telec(C): 8.2337
Load: 42 Q(W): 8370.4600 Diameter(m): 0.03099 Velocity(m/sec): 1.3501 Mass flow rate(kg/s): 1.0182 Thot(C): 8.9353 Telec(C): 10.0514
Load: 43 Q(W): 4712.7800 Diameter(m): 0.02456 Velocity(m/sec): 0.6619 Mass flow rate(kg/s): 0.1220 Thot(C): 8.6569 Telec(C): 9.5083
Load: 44 Q(W): 1582.6500 Diameter(m): 0.01532 Velocity(m/sec): 1.0436 Mass flow rate(kg/s): 0.7871 Thot(C): 9.3888 Telec(C): 10.1012
Load: 45 Q(W): 2233.6467 Diameter(m): 0.01951 Velocity(m/sec): 0.4437 Mass flow rate(kg/s): 0.0817 Thot(C): 8.3048 Telec(C): 8.5034
Load: 46 Q(W): 6506.4500 Diameter(m): 0.02456 Velocity(m/sec): 0.4303 Mass flow rate(kg/s): 0.0793 Thot(C): 10.4671 Telec(C): 10.8612
Load: 47 Q(W): 1758.5000 Diameter(m): 0.01532 Velocity(m/sec): 0.6043 Mass flow rate(kg/s): 0.1806 Thot(C): 11.4401 Telec(C): 11.7186
Load: 48 Q(W): 738.5700 Diameter(m): 0.01532 Velocity(m/sec): 1.6135 Mass flow rate(kg/s): 0.7645 Thot(C): 8.2651 Telec(C): 9.1491
Load: 49 Q(W): 7631.8900 Diameter(m): 0.03099 Velocity(m/sec): 0.6593 Mass flow rate(kg/s): 0.1970 Thot(C): 10.3194 Telec(C): 10.7171
Load: 50 Q(W): 4572.1000 Diameter(m): 0.02456 Velocity(m/sec): 0.7200 Mass flow rate(kg/s): 0.1327 Thot(C): 9.0640 Telec(C): 10.2859
Load: 51 Q(W): 7315.3600 Diameter(m): 0.03099 Velocity(m/sec): 1.1834 Mass flow rate(kg/s): 4.5051 Thot(C): 9.8934 Telec(C): 10.0986
Load: 52 Q(W): 2550.5284 Diameter(m): 0.01951 Velocity(m/sec): 0.5728 Mass flow rate(kg/s): 0.1055 Thot(C): 9.0456 Telec(C): 9.7664
Load: 53 Q(W): 1090.2700 Diameter(m): 0.01532 Velocity(m/sec): 0.9028 Mass flow rate(kg/s): 2.3262 Thot(C): 10.8290 Telec(C): 9.8605
Load: 54 Q(W): 7209.8500 Diameter(m): 0.03099 Velocity(m/sec): 0.5454 Mass flow rate(kg/s): 0.1005 Thot(C): 10.3312 Telec(C): 11.2972
Load: 55 Q(W): 1582.6500 Diameter(m): 0.01532 Velocity(m/sec): 0.7494 Mass flow rate(kg/s): 0.5652 Thot(C): 11.0052 Telec(C): 11.2278
Load: 56 Q(W): 8688.0451 Diameter(m): 0.03099 Velocity(m/sec): 0.9748 Mass flow rate(kg/s): 1.2097 Thot(C): 9.3302 Telec(C): 9.5643

Load: 57 Q(W): 4009.7317 Diameter(m): 0.02456 Velocity(m/sec): 0.7391 Mass flow rate(kg/s): 0.3502 Thot(C): 10.2752 Telec(C): 10.2236
Load: 58 Q(W): 20679.9600 Diameter(m): 0.03975 Velocity(m/sec): 0.6328 Mass flow rate(kg/s): 0.2998 Thot(C): 11.9975 Telec(C): 11.5491
Load: 59 Q(W): 949.5900 Diameter(m): 0.01532 Velocity(m/sec): 0.7829 Mass flow rate(kg/s): 0.9716 Thot(C): 11.6194 Telec(C): 10.3781
Load: 60 Q(W): 1055.1000 Diameter(m): 0.01532 Velocity(m/sec): 0.7428 Mass flow rate(kg/s): 0.5602 Thot(C): 10.0279 Telec(C): 10.1935
Load: 61 Q(W): 4853.4600 Diameter(m): 0.02456 Velocity(m/sec): 1.5035 Mass flow rate(kg/s): 0.2770 Thot(C): 6.7879 Telec(C): 6.9824
Load: 62 Q(W): 10309.3821 Diameter(m): 0.03099 Velocity(m/sec): 1.5035 Mass flow rate(kg/s): 0.2770 Thot(C): 6.8785 Telec(C): 7.2188
Load: 63 Q(W): 33235.6500 Diameter(m): 0.05728 Velocity(m/sec): 0.7198 Mass flow rate(kg/s): 0.1326 Thot(C): 9.1911 Telec(C): 10.4781
Load: 64 Q(W): 4150.0600 Diameter(m): 0.02456 Velocity(m/sec): 1.0580 Mass flow rate(kg/s): 2.7260 Thot(C): 10.6577 Telec(C): 9.9376
Load: 65 Q(W): 773.7400 Diameter(m): 0.01532 Velocity(m/sec): 0.9023 Mass flow rate(kg/s): 0.4275 Thot(C): 9.6225 Telec(C): 8.2558
Load: 66 Q(W): 1230.9500 Diameter(m): 0.01532 Velocity(m/sec): 0.9160 Mass flow rate(kg/s): 0.1688 Thot(C): 7.6588 Telec(C): 8.4374
Load: 67 Q(W): 16107.8600 Diameter(m): 0.03975 Velocity(m/sec): 1.3061 Mass flow rate(kg/s): 0.3904 Thot(C): 8.3391 Telec(C): 9.5223
Load: 68 Q(W): 8440.8000 Diameter(m): 0.03099 Velocity(m/sec): 0.5185 Mass flow rate(kg/s): 0.0955 Thot(C): 11.5725 Telec(C): 10.5705
Load: 69 Q(W): 879.2500 Diameter(m): 0.01532 Velocity(m/sec): 0.9211 Mass flow rate(kg/s): 0.4364 Thot(C): 9.8500 Telec(C): 10.1865
Load: 70 Q(W): 10093.7900 Diameter(m): 0.03099 Velocity(m/sec): 1.4415 Mass flow rate(kg/s): 3.7142 Thot(C): 10.3011 Telec(C): 10.0568
Load: 71 Q(W): 140.6800 Diameter(m): 0.01532 Velocity(m/sec): 0.8909 Mass flow rate(kg/s): 0.1641 Thot(C): 8.2475 Telec(C): 9.4336
Load: 72 Q(W): 10.0000 Diameter(m): 0.01532 Velocity(m/sec): 0.5045 Mass flow rate(kg/s): 0.0929 Thot(C): 9.0080 Telec(C): 9.4940
Load: 73 Q(W): 1301.2900 Diameter(m): 0.01532 Velocity(m/sec): 0.4883 Mass flow rate(kg/s): 0.0900 Thot(C): 9.6434 Telec(C): 10.1908
Load: 74 Q(W): 9179.3700 Diameter(m): 0.03099 Velocity(m/sec): 0.7965 Mass flow rate(kg/s): 0.3774 Thot(C): 10.8575 Telec(C): 10.9619
Load: 75 Q(W): 2040.9151 Diameter(m): 0.01532 Velocity(m/sec): 0.4323 Mass flow rate(kg/s): 0.0797 Thot(C): 10.7643 Telec(C): 11.1980
Load: 76 Q(W): 2426.7300 Diameter(m): 0.01951 Velocity(m/sec): 1.6158 Mass flow rate(kg/s): 1.2186 Thot(C): 8.2464 Telec(C): 9.3154
Load: 77 Q(W): 1371.6300 Diameter(m): 0.01532 Velocity(m/sec): 1.1415 Mass flow rate(kg/s): 1.4167 Thot(C): 8.9185 Telec(C): 9.3351
Load: 78 Q(W): 8194.6100 Diameter(m): 0.03099 Velocity(m/sec): 0.9386 Mass flow rate(kg/s): 0.4447 Thot(C): 8.8686 Telec(C): 10.8012
Load: 79 Q(W): 22368.1200 Diameter(m): 0.03975 Velocity(m/sec): 0.8118 Mass flow rate(kg/s): 0.1496 Thot(C): 10.0240 Telec(C): 12.1707
Load: 80 Q(W): 5873.3900 Diameter(m): 0.02456 Velocity(m/sec): 1.3458 Mass flow rate(kg/s): 0.2480 Thot(C): 7.6795 Telec(C): 9.0820
Load: 81 Q(W): 562.7200 Diameter(m): 0.01532 Velocity(m/sec): 1.6115 Mass flow rate(kg/s): 0.4816 Thot(C): 7.8138 Telec(C): 11.4428
Load: 82 Q(W): 3007.3867 Diameter(m): 0.01951 Velocity(m/sec): 1.1791 Mass flow rate(kg/s): 0.5587 Thot(C): 8.6740 Telec(C): 11.1431
Load: 83 Q(W): 9706.9200 Diameter(m): 0.03099 Velocity(m/sec): 0.9251 Mass flow rate(kg/s): 1.1481 Thot(C): 11.3757 Telec(C): 10.3744
Load: 84 Q(W): 5047.2467 Diameter(m): 0.02456 Velocity(m/sec): 1.4491 Mass flow rate(kg/s): 1.0929 Thot(C): 8.4894 Telec(C): 9.5042
Load: 85 Q(W): 1019.9300 Diameter(m): 0.01532 Velocity(m/sec): 0.8441 Mass flow rate(kg/s): 0.6366 Thot(C): 9.5194 Telec(C): 10.6319
Load: 86 Q(W): 1794.0217 Diameter(m): 0.01532 Velocity(m/sec): 1.2111 Mass flow rate(kg/s): 0.9134 Thot(C): 8.8200 Telec(C): 9.6659
Load: 87 Q(W): 9003.5200 Diameter(m): 0.03099 Velocity(m/sec): 0.5973 Mass flow rate(kg/s): 0.1100 Thot(C): 7.8836 Telec(C): 8.2940
Load: 88 Q(W): 1073.0367 Diameter(m): 0.01532 Velocity(m/sec): 0.7368 Mass flow rate(kg/s): 0.2202 Thot(C): 10.7329 Telec(C): 11.4595
Load: 89 Q(W): 562.7200 Diameter(m): 0.01532 Velocity(m/sec): 0.9675 Mass flow rate(kg/s): 0.2891 Thot(C): 9.6189 Telec(C): 10.7861
Load: 90 Q(W): 51243.7451 Diameter(m): 0.05728 Velocity(m/sec): 1.2069 Mass flow rate(kg/s): 0.2224 Thot(C): 7.8712 Telec(C): 9.3090
Load: 91 Q(W): 1266.1200 Diameter(m): 0.01532 Velocity(m/sec): 1.2119 Mass flow rate(kg/s): 1.5040 Thot(C): 8.9400 Telec(C): 9.4798
Load: 92 Q(W): 13575.6200 Diameter(m): 0.03975 Velocity(m/sec): 0.7970 Mass flow rate(kg/s): 0.1468 Thot(C): 7.7498 Telec(C): 8.4171
Load: 93 Q(W): 3622.8617 Diameter(m): 0.01951 Velocity(m/sec): 0.9765 Mass flow rate(kg/s): 0.7365 Thot(C): 9.3598 Telec(C): 9.9507
Load: 94 Q(W): 13716.6517 Diameter(m): 0.03975 Velocity(m/sec): 1.4488 Mass flow rate(kg/s): 1.0927 Thot(C): 8.4514 Telec(C): 9.4542
Load: 95 Q(W): 5134.8200 Diameter(m): 0.02456 Velocity(m/sec): 0.8736 Mass flow rate(kg/s): 0.4139 Thot(C): 9.4071 Telec(C): 11.5422
Load: 96 Q(W): 2954.2800 Diameter(m): 0.01951 Velocity(m/sec): 0.8703 Mass flow rate(kg/s): 1.0801 Thot(C): 10.9589 Telec(C): 9.9984
Load: 97 Q(W): 3024.6200 Diameter(m): 0.01951 Velocity(m/sec): 0.5339 Mass flow rate(kg/s): 0.2530 Thot(C): 11.7611 Telec(C): 11.0183
Load: 98 Q(W): 4185.2300 Diameter(m): 0.02456 Velocity(m/sec): 0.6228 Mass flow rate(kg/s): 0.4697 Thot(C): 10.7289 Telec(C): 10.5662
Load: 99 Q(W): 1336.4600 Diameter(m): 0.01532 Velocity(m/sec): 0.5236 Mass flow rate(kg/s): 0.1565 Thot(C): 12.1747 Telec(C): 12.0967
Load: 100 Q(W): 3094.9600 Diameter(m): 0.01951 Velocity(m/sec): 0.7614 Mass flow rate(kg/s): 0.3607 Thot(C): 10.4944 Telec(C): 10.4979
Load: 101 Q(W): 61090.2900 Diameter(m): 0.06962 Velocity(m/sec): 1.7350 Mass flow rate(kg/s): 0.3197 Thot(C): 7.9236 Telec(C): 10.3491
Load: 102 Q(W): 2321.2200 Diameter(m): 0.01951 Velocity(m/sec): 1.3408 Mass flow rate(kg/s): 1.6640 Thot(C): 9.1663 Telec(C): 9.9355
Load: 103 Q(W): 1055.1000 Diameter(m): 0.01532 Velocity(m/sec): 1.4468 Mass flow rate(kg/s): 0.6855 Thot(C): 8.1990 Telec(C): 10.6806
Load: 104 Q(W): 1195.7800 Diameter(m): 0.01532 Velocity(m/sec): 0.7905 Mass flow rate(kg/s): 0.5961 Thot(C): 11.4816 Telec(C): 10.4925
Load: 105 Q(W): 40691.3383 Diameter(m): 0.05728 Velocity(m/sec): 1.2108 Mass flow rate(kg/s): 0.9132 Thot(C): 8.9305 Telec(C): 9.8359
Load: 106 Q(W): 6811.3739 Diameter(m): 0.02456 Velocity(m/sec): 1.4163 Mass flow rate(kg/s): 1.7577 Thot(C): 8.9093 Telec(C): 9.7135
Load: 107 Q(W): 1547.4800 Diameter(m): 0.01532 Velocity(m/sec): 0.6405 Mass flow rate(kg/s): 0.1914 Thot(C): 10.5359 Telec(C): 10.8956
Load: 108 Q(W): 2567.4100 Diameter(m): 0.01951 Velocity(m/sec): 1.3771 Mass flow rate(kg/s): 0.2537 Thot(C): 7.7554 Telec(C): 9.3156
Load: 109 Q(W): 10304.8100 Diameter(m): 0.03099 Velocity(m/sec): 1.3579 Mass flow rate(kg/s): 0.6434 Thot(C): 8.3968 Telec(C): 10.9836
Load: 110 Q(W): 2110.2000 Diameter(m): 0.01532 Velocity(m/sec): 0.7578 Mass flow rate(kg/s): 0.3591 Thot(C): 9.4170 Telec(C): 11.1058
Load: 111 Q(W): 13540.4500 Diameter(m): 0.03975 Velocity(m/sec): 1.3829 Mass flow rate(kg/s): 0.4133 Thot(C): 8.5297 Telec(C): 9.9825
Load: 112 Q(W): 2426.7300 Diameter(m): 0.01951 Velocity(m/sec): 1.0349 Mass flow rate(kg/s): 2.6665 Thot(C): 11.4088 Telec(C): 10.6347

Load: 113 Q(W): 5310.6700 Diameter(m): 0.02456 Velocity(m/sec): 0.7838 Mass flow rate(kg/s): 0.1444 Thot(C): 9.6802 Telec(C): 11.4898
Load: 114 Q(W): 1547.8317 Diameter(m): 0.01532 Velocity(m/sec): 1.1669 Mass flow rate(kg/s): 0.5529 Thot(C): 8.6648 Telec(C): 11.0936
Load: 115 Q(W): 6717.4700 Diameter(m): 0.02456 Velocity(m/sec): 1.4154 Mass flow rate(kg/s): 1.0675 Thot(C): 8.5092 Telec(C): 9.5117
Load: 116 Q(W): 2004.6900 Diameter(m): 0.01532 Velocity(m/sec): 1.4180 Mass flow rate(kg/s): 3.6537 Thot(C): 9.8665 Telec(C): 9.6956
Load: 117 Q(W): 20222.7500 Diameter(m): 0.03975 Velocity(m/sec): 0.5400 Mass flow rate(kg/s): 0.0995 Thot(C): 8.6015 Telec(C): 9.0992
Load: 118 Q(W): 5169.9900 Diameter(m): 0.02456 Velocity(m/sec): 0.6175 Mass flow rate(kg/s): 0.1845 Thot(C): 9.7505 Telec(C): 12.5403
Load: 119 Q(W): 7913.2500 Diameter(m): 0.03099 Velocity(m/sec): 0.4919 Mass flow rate(kg/s): 0.2331 Thot(C): 12.4830 Telec(C): 11.4522
Load: 120 Q(W): 2638.1017 Diameter(m): 0.01951 Velocity(m/sec): 0.4944 Mass flow rate(kg/s): 0.2342 Thot(C): 13.2585 Telec(C): 12.1213
Load: 121 Q(W): 140.6800 Diameter(m): 0.01532 Velocity(m/sec): 0.8716 Mass flow rate(kg/s): 0.2605 Thot(C): 9.0765 Telec(C): 12.6581
Load: 122 Q(W): 2391.5600 Diameter(m): 0.01951 Velocity(m/sec): 1.1826 Mass flow rate(kg/s): 0.5603 Thot(C): 8.8622 Telec(C): 9.4765
Load: 123 Q(W): 246.1900 Diameter(m): 0.01532 Velocity(m/sec): 1.2500 Mass flow rate(kg/s): 0.2303 Thot(C): 8.7379 Telec(C): 11.3367
Load: 124 Q(W): 5697.8917 Diameter(m): 0.02456 Velocity(m/sec): 1.2537 Mass flow rate(kg/s): 0.3747 Thot(C): 8.2079 Telec(C): 11.8358
Load: 125 Q(W): 1406.8000 Diameter(m): 0.01532 Velocity(m/sec): 1.0156 Mass flow rate(kg/s): 0.1871 Thot(C): 8.6351 Telec(C): 10.4554
Load: 126 Q(W): 6489.2167 Diameter(m): 0.02456 Velocity(m/sec): 0.6337 Mass flow rate(kg/s): 0.1168 Thot(C): 10.9672 Telec(C): 12.6442
Load: 127 Q(W): 45721.0000 Diameter(m): 0.05728 Velocity(m/sec): 1.0176 Mass flow rate(kg/s): 0.3041 Thot(C): 8.6754 Telec(C): 12.3215
Load: 128 Q(W): 808.9100 Diameter(m): 0.01532 Velocity(m/sec): 0.5052 Mass flow rate(kg/s): 0.0931 Thot(C): 9.7232 Telec(C): 10.3620
Load: 129 Q(W): 5310.6700 Diameter(m): 0.02456 Velocity(m/sec): 0.8726 Mass flow rate(kg/s): 0.4135 Thot(C): 10.5864 Telec(C): 10.8923
Load: 130 Q(W): 4642.4400 Diameter(m): 0.02456 Velocity(m/sec): 1.0176 Mass flow rate(kg/s): 0.3041 Thot(C): 8.4828 Telec(C): 11.7853
Load: 131 Q(W): 703.4000 Diameter(m): 0.01532 Velocity(m/sec): 0.5437 Mass flow rate(kg/s): 0.1625 Thot(C): 11.1987 Telec(C): 11.2110
Load: 132 Q(W): 8264.9500 Diameter(m): 0.03099 Velocity(m/sec): 0.6532 Mass flow rate(kg/s): 0.3095 Thot(C): 9.8841 Telec(C): 11.3597
Load: 133 Q(W): 2743.2600 Diameter(m): 0.01951 Velocity(m/sec): 0.8716 Mass flow rate(kg/s): 0.2605 Thot(C): 9.3653 Telec(C): 10.1920
Load: 134 Q(W): 49132.4900 Diameter(m): 0.05728 Velocity(m/sec): 1.0251 Mass flow rate(kg/s): 1.2722 Thot(C): 9.2059 Telec(C): 9.5157
Load: 135 Q(W): 1969.5200 Diameter(m): 0.01532 Velocity(m/sec): 1.2662 Mass flow rate(kg/s): 1.5714 Thot(C): 8.7440 Telec(C): 9.2990
Load: 136 Q(W): 1828.8400 Diameter(m): 0.01532 Velocity(m/sec): 1.0899 Mass flow rate(kg/s): 2.8081 Thot(C): 11.0088 Telec(C): 10.2564
Load: 137 Q(W): 5838.2200 Diameter(m): 0.02456 Velocity(m/sec): 1.4927 Mass flow rate(kg/s): 0.2750 Thot(C): 7.5954 Telec(C): 9.0757
Load: 138 Q(W): 53141.8700 Diameter(m): 0.05728 Velocity(m/sec): 0.7651 Mass flow rate(kg/s): 0.1410 Thot(C): 9.6951 Telec(C): 11.4280
Load: 139 Q(W): 56729.2100 Diameter(m): 0.05728 Velocity(m/sec): 0.6348 Mass flow rate(kg/s): 0.1897 Thot(C): 10.4385 Telec(C): 10.7636
Load: 140 Q(W): 3235.6400 Diameter(m): 0.01951 Velocity(m/sec): 1.2818 Mass flow rate(kg/s): 0.6073 Thot(C): 8.6444 Telec(C): 9.3320
Load: 141 Q(W): 1090.2700 Diameter(m): 0.01532 Velocity(m/sec): 0.7335 Mass flow rate(kg/s): 0.3475 Thot(C): 10.6880 Telec(C): 10.6061
Load: 142 Q(W): 1160.6100 Diameter(m): 0.01532 Velocity(m/sec): 1.8508 Mass flow rate(kg/s): 1.3958 Thot(C): 8.0639 Telec(C): 9.2741
Load: 143 Q(W): 914.4200 Diameter(m): 0.01532 Velocity(m/sec): 1.1495 Mass flow rate(kg/s): 0.3435 Thot(C): 8.3476 Telec(C): 11.9064
Load: 144 Q(W): 4677.6100 Diameter(m): 0.02456 Velocity(m/sec): 1.1351 Mass flow rate(kg/s): 0.2091 Thot(C): 6.6783 Telec(C): 6.6907
Load: 145 Q(W): 1125.4400 Diameter(m): 0.01532 Velocity(m/sec): 0.5559 Mass flow rate(kg/s): 0.4193 Thot(C): 11.8761 Telec(C): 11.4286
Load: 146 Q(W): 4150.0600 Diameter(m): 0.02456 Velocity(m/sec): 0.6492 Mass flow rate(kg/s): 0.4896 Thot(C): 11.5723 Telec(C): 11.4963
Load: 147 Q(W): 6647.1300 Diameter(m): 0.02456 Velocity(m/sec): 1.2979 Mass flow rate(kg/s): 0.9788 Thot(C): 8.7186 Telec(C): 9.6766
Load: 148 Q(W): 5803.0500 Diameter(m): 0.02456 Velocity(m/sec): 1.2784 Mass flow rate(kg/s): 0.2355 Thot(C): 7.9104 Telec(C): 9.5242
Load: 149 Q(W): 1371.6300 Diameter(m): 0.01532 Velocity(m/sec): 1.3724 Mass flow rate(kg/s): 0.6503 Thot(C): 8.1854 Telec(C): 10.4845
Load: 150 Q(W): 1688.1600 Diameter(m): 0.01532 Velocity(m/sec): 0.6414 Mass flow rate(kg/s): 0.4838 Thot(C): 11.7375 Telec(C): 11.6253
Load: 151 Q(W): 8089.1000 Diameter(m): 0.03099 Velocity(m/sec): 1.3693 Mass flow rate(kg/s): 0.4092 Thot(C): 8.1498 Telec(C): 12.0246
Load: 152 Q(W): 17479.4900 Diameter(m): 0.03975 Velocity(m/sec): 0.8895 Mass flow rate(kg/s): 0.6709 Thot(C): 9.2240 Telec(C): 10.3571
Load: 153 Q(W): 13405.3972 Diameter(m): 0.03975 Velocity(m/sec): 1.3748 Mass flow rate(kg/s): 1.0368 Thot(C): 8.6606 Telec(C): 9.6849
Load: 154 Q(W): 4414.1867 Diameter(m): 0.02456 Velocity(m/sec): 1.0845 Mass flow rate(kg/s): 1.3460 Thot(C): 10.3225 Telec(C): 9.7252
Load: 155 Q(W): 4115.2417 Diameter(m): 0.02456 Velocity(m/sec): 0.6983 Mass flow rate(kg/s): 0.1287 Thot(C): 8.6181 Telec(C): 9.5572
Load: 156 Q(W): 12063.3100 Diameter(m): 0.03099 Velocity(m/sec): 1.1130 Mass flow rate(kg/s): 0.5274 Thot(C): 9.6023 Telec(C): 10.2825
Load: 157 Q(W): 2110.2000 Diameter(m): 0.01532 Velocity(m/sec): 0.5127 Mass flow rate(kg/s): 0.0945 Thot(C): 8.5273 Telec(C): 8.9352
Load: 158 Q(W): 8686.9900 Diameter(m): 0.03099 Velocity(m/sec): 0.5576 Mass flow rate(kg/s): 0.2642 Thot(C): 10.7842 Telec(C): 12.1026
Load: 159 Q(W): 1055.1000 Diameter(m): 0.01532 Velocity(m/sec): 1.0599 Mass flow rate(kg/s): 0.1953 Thot(C): 8.5952 Telec(C): 10.5059
Load: 160 Q(W): 16565.0700 Diameter(m): 0.03975 Velocity(m/sec): 0.6541 Mass flow rate(kg/s): 0.4933 Thot(C): 10.7039 Telec(C): 10.6342
Load: 161 Q(W): 2321.2200 Diameter(m): 0.01951 Velocity(m/sec): 0.5896 Mass flow rate(kg/s): 0.1086 Thot(C): 8.9778 Telec(C): 9.7352
Load: 162 Q(W): 3112.8967 Diameter(m): 0.01951 Velocity(m/sec): 1.1157 Mass flow rate(kg/s): 1.3846 Thot(C): 9.2052 Telec(C): 9.6523
Load: 163 Q(W): 4712.7800 Diameter(m): 0.02456 Velocity(m/sec): 0.7804 Mass flow rate(kg/s): 0.9685 Thot(C): 10.0280 Telec(C): 9.9535
Load: 164 Q(W): 3622.5100 Diameter(m): 0.01951 Velocity(m/sec): 0.6279 Mass flow rate(kg/s): 0.1157 Thot(C): 8.4752 Telec(C): 9.1629
Load: 165 Q(W): 22722.2819 Diameter(m): 0.03975 Velocity(m/sec): 1.3837 Mass flow rate(kg/s): 0.2549 Thot(C): 7.9144 Telec(C): 9.7183
Load: 166 Q(W): 8018.7600 Diameter(m): 0.03099 Velocity(m/sec): 1.3837 Mass flow rate(kg/s): 0.2549 Thot(C): 8.6365 Telec(C): 11.4865
Load: 167 Q(W): 8370.4600 Diameter(m): 0.03099 Velocity(m/sec): 1.7897 Mass flow rate(kg/s): 0.8480 Thot(C): 8.4977 Telec(C): 9.7227
Load: 168 Q(W): 5416.1800 Diameter(m): 0.02456 Velocity(m/sec): 0.6472 Mass flow rate(kg/s): 0.1934 Thot(C): 10.5603 Telec(C): 10.9541

Load: 169 Q(W): 7631.8900 Diameter(m): 0.03099 Velocity(m/sec): 0.8729 Mass flow rate(kg/s): 0.2609 Thot(C): 10.1801 Telec(C): 11.2730
Load: 170 Q(W): 19484.1800 Diameter(m): 0.03975 Velocity(m/sec): 1.3837 Mass flow rate(kg/s): 0.2549 Thot(C): 7.5372 Telec(C): 8.7949
Load: 171 Q(W): 8264.9500 Diameter(m): 0.03099 Velocity(m/sec): 0.9915 Mass flow rate(kg/s): 0.7478 Thot(C): 9.5766 Telec(C): 10.2574
Load: 172 Q(W): 8194.6100 Diameter(m): 0.03099 Velocity(m/sec): 0.6056 Mass flow rate(kg/s): 0.1116 Thot(C): 10.4918 Telec(C): 11.8355
Load: 173 Q(W): 562.7200 Diameter(m): 0.01532 Velocity(m/sec): 0.9219 Mass flow rate(kg/s): 1.1441 Thot(C): 10.3462 Telec(C): 10.6213
Load: 174 Q(W): 4766.9418 Diameter(m): 0.02456 Velocity(m/sec): 1.1980 Mass flow rate(kg/s): 0.2207 Thot(C): 8.7142 Telec(C): 11.1462
Load: 175 Q(W): 3763.1900 Diameter(m): 0.01951 Velocity(m/sec): 0.6144 Mass flow rate(kg/s): 0.1132 Thot(C): 8.6630 Telec(C): 9.3856
Load: 176 Q(W): 668.2300 Diameter(m): 0.01532 Velocity(m/sec): 0.4776 Mass flow rate(kg/s): 0.0880 Thot(C): 10.2805 Telec(C): 10.8944
Load: 177 Q(W): 3587.3400 Diameter(m): 0.01951 Velocity(m/sec): 0.9529 Mass flow rate(kg/s): 0.7186 Thot(C): 9.8808 Telec(C): 10.5393
Load: 178 Q(W): 8335.2900 Diameter(m): 0.03099 Velocity(m/sec): 0.6332 Mass flow rate(kg/s): 0.3000 Thot(C): 11.8267 Telec(C): 9.6590
Load: 179 Q(W): 1125.4400 Diameter(m): 0.01532 Velocity(m/sec): 1.1980 Mass flow rate(kg/s): 0.2207 Thot(C): 8.1077 Telec(C): 9.8154
Load: 180 Q(W): 1125.4400 Diameter(m): 0.01532 Velocity(m/sec): 0.9191 Mass flow rate(kg/s): 1.1406 Thot(C): 9.6896 Telec(C): 9.8763

The load number corresponds to the branch index. Q is the heat load [W]. The inner branch diameter [m] and chilled water velocity [m/sec] within the corresponding branch are shown. The mass flow rate [kg/sec] is also shown. Thot [C] corresponds to the temperature downstream of the heat exchanger. Telec [C] corresponds to the outlet (colder) temperature on the secondary side.

For the example above, the heat exchangers considered for all heat loads were the cooling coils since these were the most well-defined heat exchangers within the heat exchanger database. Because of this, the low Telec temperatures are to be expected since the hot inlet air temperatures are estimated to be 26.7°C. For other applications such as heat exchangers with high heat fluxes used for the removal of heat from high energy radars, the inlet temperature will play a critical role in determining an accurate outlet temperature on the secondary side. This outlet temperature is expected to be much higher as those shown above (on the order of 100°C).

4.2 Weight Analysis

The second analysis performed was the weight analysis. The weight of the chilled water system and the seawater system was determined along with a breakdown by components. The center of gravity for each component group and overall system was also included. A weight margin of 10% was included to account for miscellaneous items unaccounted for and for uncertainty in the design. For the simulated design, report 4 is:

Report 4: CW/SW Weight Summary

Item	Weight (MT)	LCG (m)	TCG (m)	VCG (m)
CW System:	159.1343	-6.8345	-0.1020	5.9911
Pipe:	29.1421	-2.5767	-0.0862	7.3868
Main:	24.9003	-5.4880	-0.0824	6.8900
Branch:	4.2418	14.5127	-0.1087	10.3037
Lagging:	0.9619	3.6922	-0.0945	8.4568
Main:	0.5204	-5.4880	-0.0824	6.8900
Branch:	0.4415	14.5127	-0.1087	10.3037
Valves:	20.7776	-5.0816	-0.4872	5.6592
Globe:	1.5560	7.0650	-2.1066	7.6973
Main:	0.0000	0.0000	0.0000	0.0000
Branch:	1.5560	7.0650	-2.1066	7.6973
Gate:	13.5336	-7.3983	-0.5057	7.5678
Main:	10.3600	-11.8358	0.0000	7.4500
Branch:	3.1736	7.0878	-2.1566	7.9525
Check:	5.6880	-2.8924	0.0000	0.5605
Main:	5.6880	-2.8924	0.0000	0.5605
Branch:	0.0000	0.0000	0.0000	0.0000
Chillers:	34.8000	-17.3545	0.0000	3.3628
Expansion tanks:	3.4869	-17.3545	0.0000	3.3628
Pumps:	7.2000	-13.8415	0.0000	3.3628
Brackets:	0.0000	5.3659	-0.0967	8.7425
Instrumentation:	0.3000	-17.3545	0.0000	3.3628
Chilled water:	46.5960	-5.8153	-0.0662	6.6338
Heat Exchangers:	15.8698	8.1791	-0.0265	9.4093
SW System:	17.2747	-4.7642	-0.1887	4.3308
Pipe:	2.9109	-7.7654	-0.3113	5.8375
Valves:	0.8000	-1.2000	0.0000	2.8576
Pumps:	6.0000	0.0000	0.0000	1.8970
Brackets:	0.3029	-7.7654	-0.3113	5.8375
Salt water:	7.2609	-7.7654	-0.3113	5.8375
Total:	176.4091	-6.6318	-0.1105	5.8285
Margin:	17.6409	-6.6318	-0.1105	5.8285
Total with margin:	194.0500	-6.6318	-0.1105	5.8285

As can be seen in the weight report, the chilled water system weighs approximately 159 MT, the auxiliary seawater system weighs approximately 17 MT and the combined systems with the added 10% weight margin weighs approximately 194 MT. The center of gravity is 6.6 m aft of midships, slightly starboard, and nearly 6 m from the baseline. This is also consistent with the 3-D model which is fairly symmetric forward-aft and port-starboard. The large, heavy A/C units will bring down the VCG so the 6.6 m VCG is reasonable.

4.3 Transient Analysis

Transient analysis is an important part of determining the feasibility and performance of a particular cooling system design. When in steady-state, the temperatures and flow velocities may be satisfactory, but without performing transient analysis on particular scenarios, it is not possible to guarantee the localized temperatures of certain regions or flow velocities are within acceptable limits.

Transient analyses were performed on the modeled chilled water system for two simultaneous events, a loss of a chiller with the chiller riser secured and no further action taken and a step load of a heat load with no action taken. The design heat loads were considered for the simulation. Figure 87 below shows the heat loads and the status of each chiller before and after the event. Only heat load RS58 (load number nine) differs before and after the event with a step response from 56 kW to 0 kW.

Notes:
The maximum heat load is taken from the maximum of the four load conditions: Shore, Design, Cruise and Battle.
The heat load at time t=0- is based off of the load condition specified in Analysis.m. Changes to the initial load value can be entered in the yellow column.
The user must enter the load value for each load after the transient under the column Heat Load at t=0+.
The coordinates for the chiller location is a sfollows: midships is at x=0 with the bow in the +x-direction, port is towards +y, and up is +z.
The chiller status must be specified for each chiller before and after the transient (specified as either on or off).

Heat Load				
Load Number	Load Name	Maximum (kW)	at t=0- (kW)	at t=0+ (kW)
1	RS10_5	0.56272	0.56272	0.56272
2	RS0405_2	3.79836	3.79836	3.79836
3	RS10_8	0.56272	0.56272	0.56272
4	RS75	16.88160	16.88160	16.88160
5	RS04	11.14924	11.14924	11.14924
6	RS10_11	0.91442	0.91442	0.91442
7	RS09	9.99356	9.99356	9.99356
8	RS60	34.39661	34.39661	34.39661
9	RS58	56.06098	56.06098	0.00000
10	RS28_2	1.68816	1.68816	1.68816
11	RS78	4.39625	4.39625	4.39625
12	RS27_2	1.30129	1.30129	1.30129
13	RS56_1	1.65299	1.65299	1.65299
14	RS25	8.12427	8.12427	8.12427
15	RS24	1.30129	1.30129	1.30129
16	RS16	59.47458	59.47458	59.47458
17	RS1G	0.14068	0.14068	0.14068
18	RS21_1	1.11383	1.11383	1.11383
19	RS21_6	0.14068	0.14068	0.14068

Chiller					
Chiller Number	Chiller Location x (m)	y (m)	z (m)	Status (on/off) at t=0-	at t=0+
1	30.430667	5.0975	3.362833	on	on
2	30.430667	-5.0975	3.362833	on	on
3	-16.22666	5.0975	3.362833	off	off
4	-16.22666	-5.0975	3.362833	on	on
5	-66.26752	5.0975	3.362833	off	off
6	-66.26752	-5.0975	3.362833	on	off

Transient

Figure 87: Input for simulated transient

4.3.1 Loss of Chiller

The loss of a chiller with the chiller riser secured will result in changes in the velocity within the supply header, a differing number and location of the stagnation points and resulting changes in the branch velocities. These differing velocities will then have an effect on the temperature response at each of the heat exchangers. Figure 88 shows the pressure distribution before and after the loss of chiller six along with the resulting effect on stagnation location. Figure 89 shows the temperature response at four locations in a single branch, immediately before, at and two different locations after the heat exchanger. The temperature variation as a function of time is due primarily to the change in the velocity within that branch, but some initial discrepancy may exist between the steady-state temperatures calculated and the transient temperatures calculated.

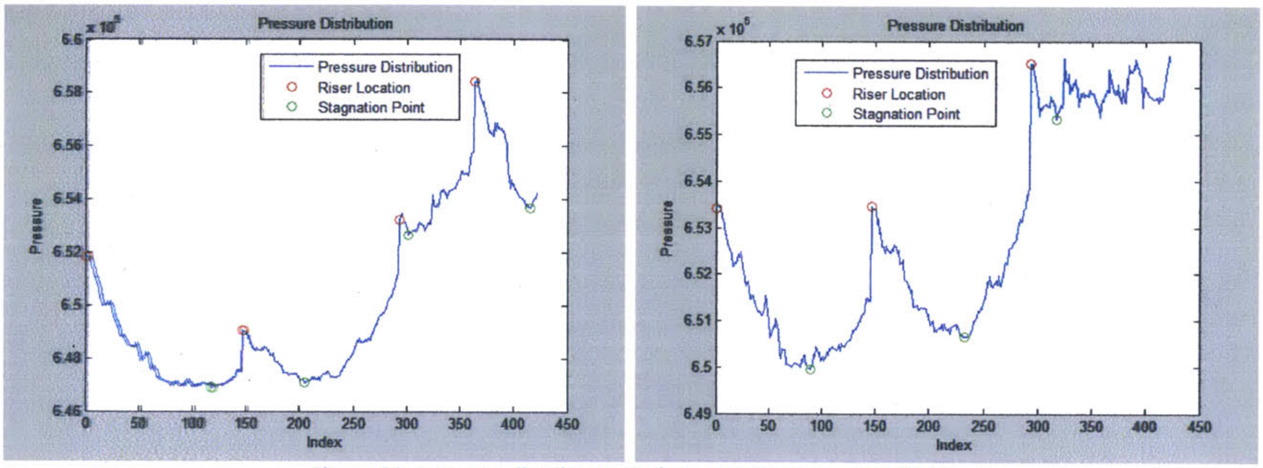


Figure 88: Pressure distribution before and after loss of chiller 6

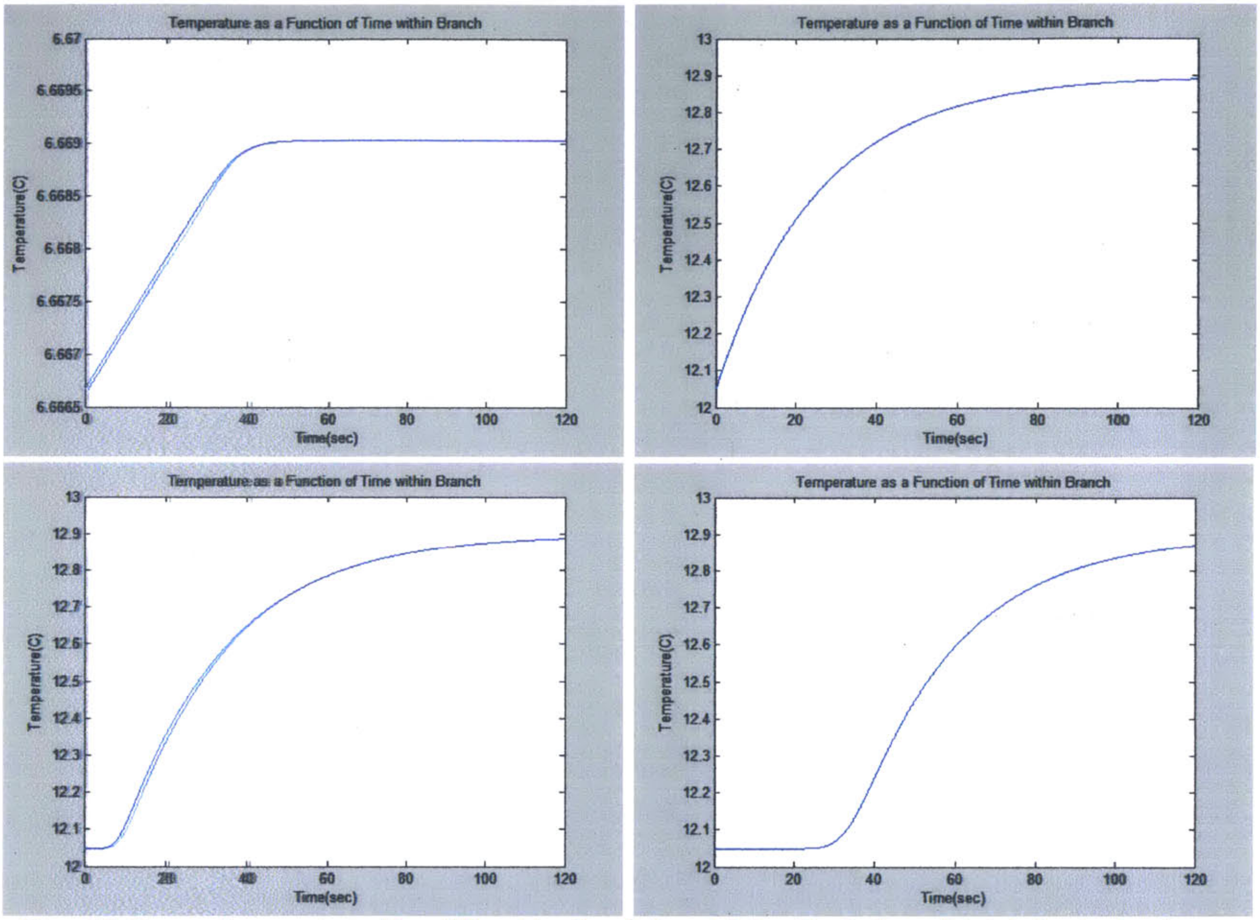


Figure 89: Temperature response at four different locations in branch 5. Location 1 - immediately before heat exchanger (upper left), location 2 - at heat exchanger (upper right), location 3 - a few meters downstream from the heat exchanger (lower left), location 4 - near the end of the branch (lower right)

The temperature response at location 1 may seem alarming, but after considering the temperature scale, it is reasonable to assume that the rise is due to error between the more simplified steady-state temperature analysis and the transient analysis. The difference in temperature is few hundredths of a degree Celsius and can be assumed constant. The temperature response at location 2 shows the correct behavior for a step-change in velocity. The beginning and ending values are also consistent with steady-state calculations using the initial and final velocities to determine the respective mass flow rates and resulting differential temperatures across the heat exchanger. The temperature response at locations 3 and 4 are in line with what is to be expected. The curve shifts to the right as the location analyzed moves downstream of the heat exchanger.

4.3.2 Step Load

Similar temperature responses to those described above are shown in the branch with a heat load step response.

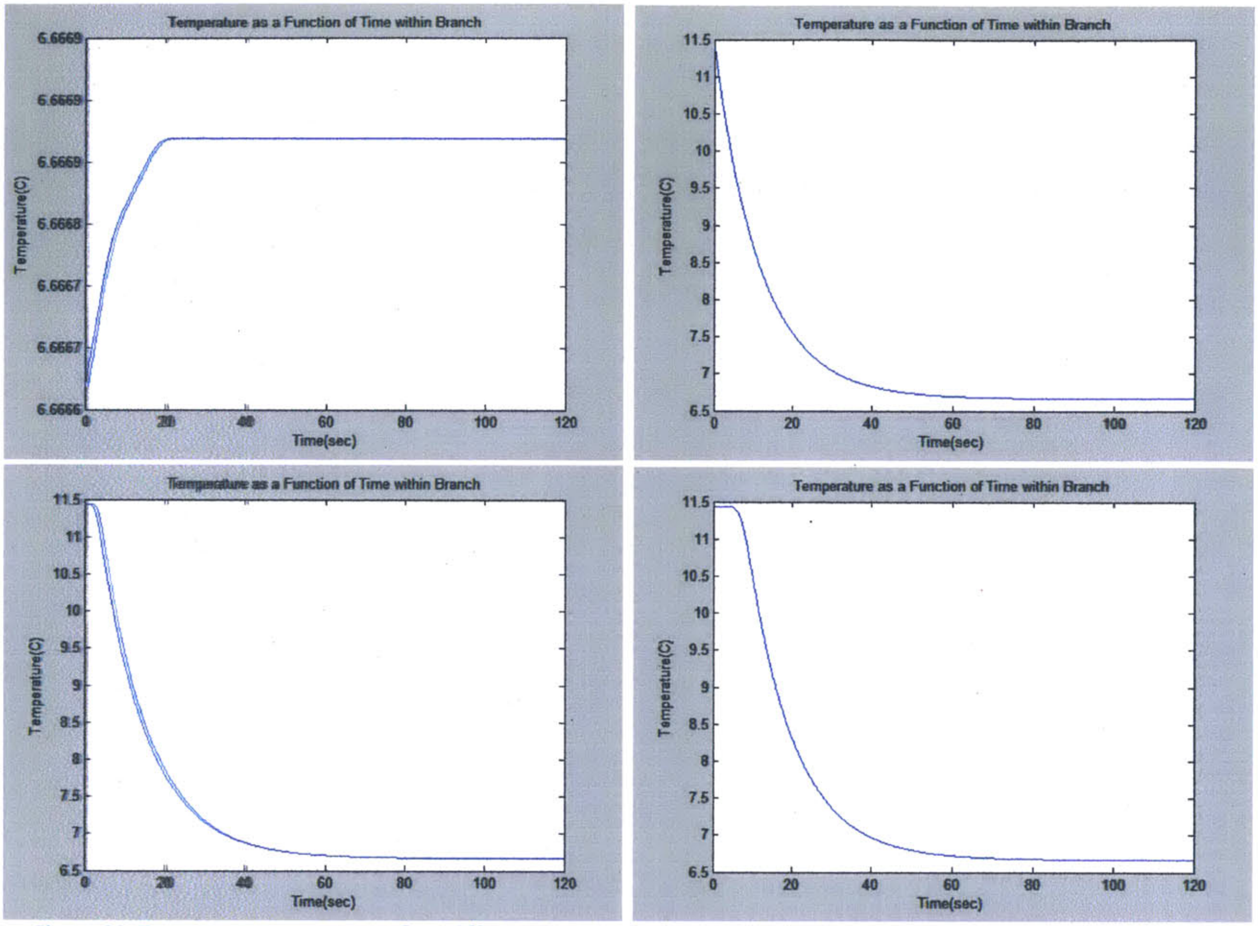


Figure 90: Temperature response at four different locations in branch 9 (heat load step response from 59 kW to 0 kW). Location 1 - immediately before heat exchanger (upper left), location 2 - at heat exchanger (upper right), location 3 - a few meters downstream from the heat exchanger (lower left), location 4 - near the end of the branch (lower right)

In the above figures, the heat load step response from 59 kW to 0 kW results in a decreasing temperature from approximately 11.25°C to 6.7°C as expected.

4.3.3 Temperature Distribution

The temperature distribution can also be found using the CSDT. Figures 74-75 show the temperature distribution along the supply header at 10 seconds and 120 seconds after the event, respectively. Figures 76-77 show the temperature distribution along the return header at 10 seconds and 120 seconds after the event, respectively.

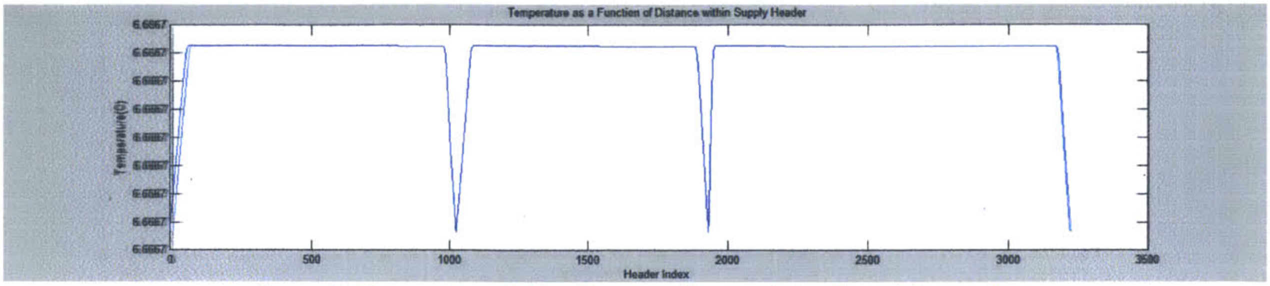


Figure 91: Temperature distribution along the supply header at 10 seconds

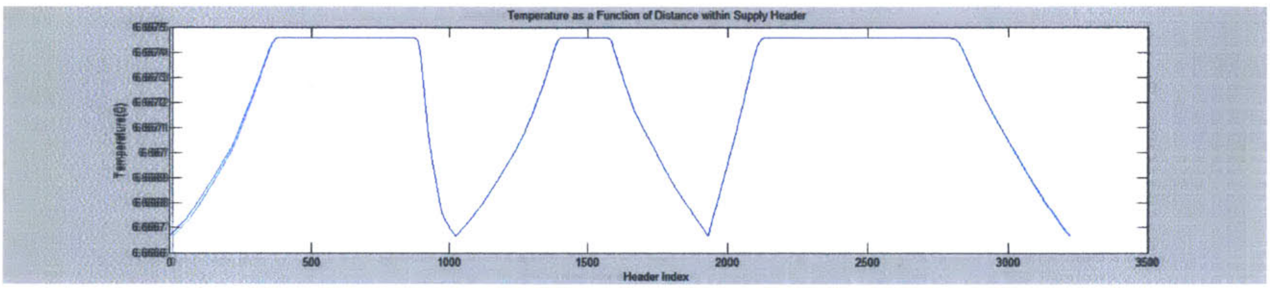


Figure 92: Temperature distribution along the supply header at 120 seconds

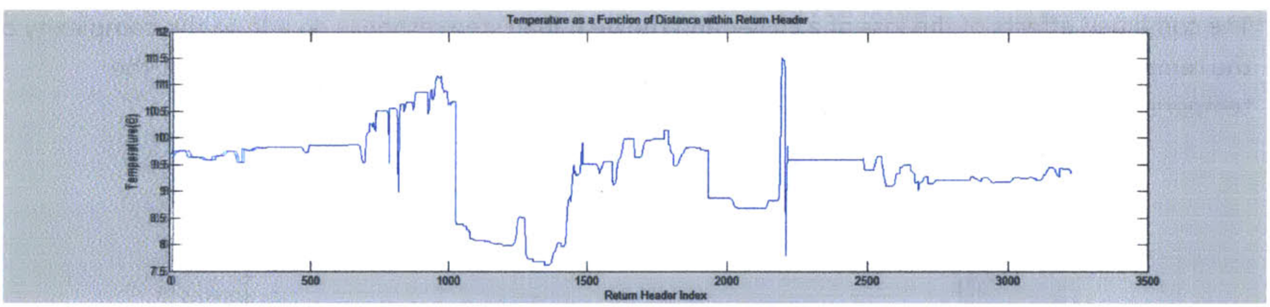


Figure 93: Temperature distribution along the supply header at 10 seconds

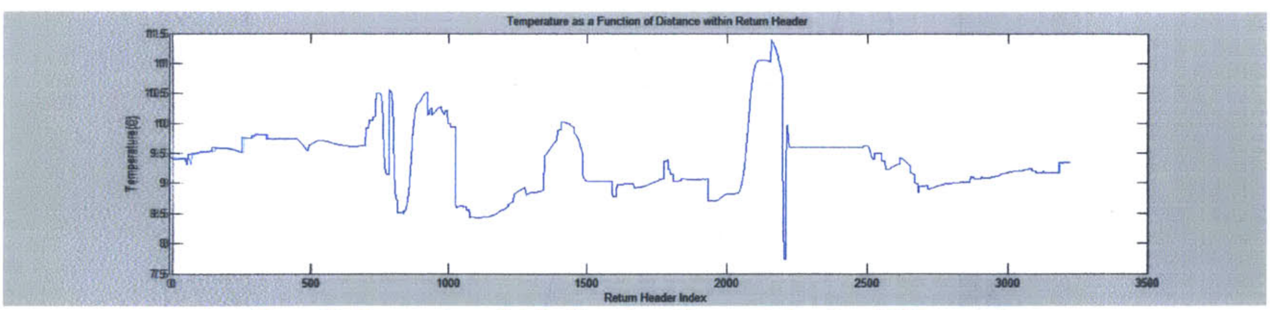


Figure 94: Temperature distribution along the supply header at 120 seconds

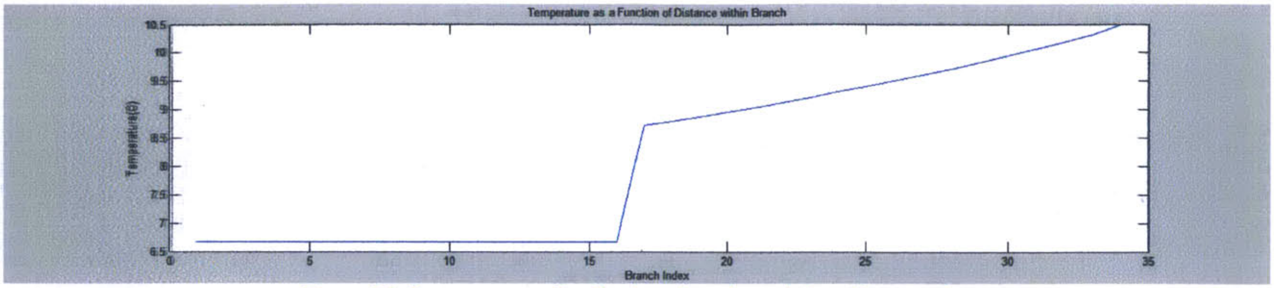


Figure 95: Temperature distribution along branch 9 at 10 seconds

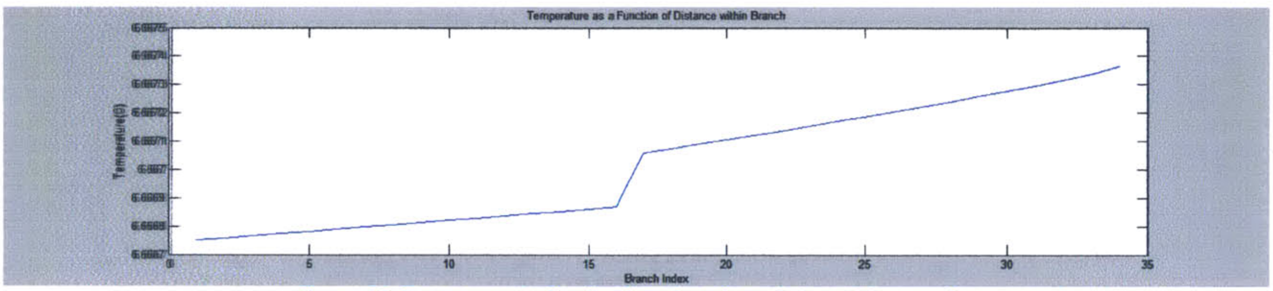


Figure 96: Temperature distribution along branch 9 at 120 seconds

The combined effects of the loss of a chiller and the heat load step response do add to the complexity of the temperature response. An example of this can be seen in Figure 97 below which shows the temperature response at the junction for riser 1 within the return header.

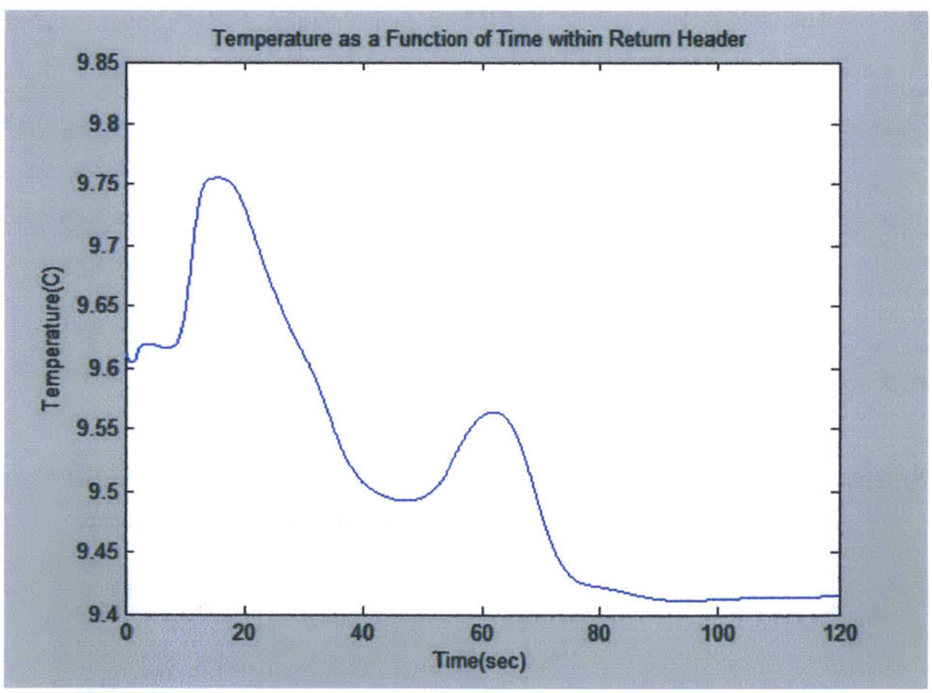


Figure 97: Temperature response within the return header at riser 1 junction

5.0 Chapter 5: Conclusions

5.1 General Conclusions

The intent of this thesis was to rapidly model and explore the design of the chilled water system using a mathematically rigorous approach. In this respect, the CSDT is a success. With relatively few inputs, the CSDT provides 2-D and 3-D visual representations of the chilled water and auxiliary seawater systems. In addition, the incorporation of FNA is essential in modeling the chilled water system. Without FNA, it is not possible to accurately determine the pressure and fluid velocity distribution within the system and without knowing these, it is impossible to determine the true temperature distribution within the system.

Other successes of the program include the analyses of the chilled water system. The three analyses available with the CSDT are the weight analysis, the static temperature analysis, and the transient temperature analysis.

The CSDT weight analysis not only captures the weight of the chilled water system, but also provides an accurate center of gravity of the chilled water system along with the weight and center of gravity of the auxiliary seawater system.

The static temperature analysis outputs the temperature at every junction and at the heat exchanger. With the properties of the secondary side known, the steady-state temperature of the exiting secondary fluid can also be known. This fluid may be the air blowing in a space, or the fluid surrounding a solid-state semiconductor chip. This brings the user one step closer in determining the average temperature within a space cooled by chilled water, or the surface temperature of electronic equipment.

The transient analysis is even more powerful. It provides the user with temperature fluctuations in time or space during transient states. The method employed within the CSDT to perform transient analyses has also been verified with two different analytical methods. The transient temperatures also reach steady-state values after sufficient time has elapsed. This gives greater confidence in the accuracy of the transient analysis.

The CSDT provides the naval architect with a tool to rapidly and accurately model the chilled water system under several operating conditions. Furthermore, the associated analyses also provide the naval architect with a means to easily determine the feasibility of their design.

5.2 Areas of Future Study

There are two broad categories for areas of future study. The first category involves improving the current version of the CSDT by offering more analyses, eliminating the few assumptions still remaining within the program, and improving the user interface. The second category involves incorporation of extensions beyond the chilled water system including the HVAC system, the SW system and the ship as a whole.

5.2.1 CSDT v3.0

There are a few areas the CSDT program could be improved, many involving the removal of the remaining assumptions and providing more capability to analyze the CW system, but the area that would have the most profound impact would be the user interface.

Currently, the user interacts through the program via the Excel spreadsheet which contains the library of heat exchangers and A/C units and through the command window. It would be beneficial if the interface between the user and the program was through the use of a Graphical User Interface (GUI). A well laid out GUI could provide all of the functionality of the program, but in a more user friendly way through the use of tabs, lists, graphs, charts and buttons which execute certain functions. More enhanced graphics could also provide a better means of displaying the 2-D and 3-D model of the CW system. Currently, valves, heat exchangers, and pumps are all displayed as a box. Better graphics could provide a means to display each of these components distinctly. Drag and drop ability would enhance the ability to place equipment accurately. Piping could be captured as a vector of nodes, with the ability to drag nodes to alter the shape of the piping. This ability would allow the user to easily route piping around equipment and to connect piping to each other while avoiding the need to manually describe each branch as a vector of points. For greater visualization, a program other than Matlab should be utilized. Some suggestions include: Python, Qt, GTK+, or C#.

Using one of the programming languages above will also allow for a better structuring of the program. The Matlab program uses some functions and is broken up into a few very large blocks of code, but the program was written in a brute-force fashion, with the focus more on correctness and less on efficiency or readability of the code.

Other than visual representation, the program could be improved by refining the FNA and removing assumptions made. FNA is used by the program, but could be structured to be more generic than it currently is. A CSDT v3.0 should include a generic node structure of the pipe network, with the ability to have any combination of branches in parallel and series. The FNA should then be able to solve the generic network, as opposed to assuming the branches are all in parallel. This can be done through the use of a flow solver. Flow solvers exist for solving current within electrical networks (the power flow problem), but could be modified to solve for fluid flow within a piping network. The incorporation of a flow solver would eliminate the need to solve for stagnation points and would remove all of the assumptions made in finding those stagnation points.

Other assumptions that still exist within the CSDT include: neglecting the inertia of the fluid, simplifying the temperature distribution to allow variation in only one dimension, simplifications in the modeling of the temperature distribution within the heat exchangers, simplifying the heat exchange process within the A/C units, and assuming constant properties of the CW and other fluids (density, specific heat capacity, etc.). Greater complexity of the CSDT program could overcome all of these assumptions.

The CSDT currently can perform weight analyses, static temperature analyses, and transient temperature analyses. An area of future study would be the inclusion of survivability analysis capability. This could be done by specifying a blast center and radius (assuming a spherical blast). All heat loads within the blast could be easily determined and would be considered damaged. The pipe branches are vectors. To see if the pipe is ruptured, the pipe would be discretized by where there is a bend. If any bend is located within the blast radius, then the pipe is damaged. If the line perpendicular to the pipe corners which passes through the blast center is between the pipe corners and has a length less than the blast radius, then that segment of pipe is damaged. The same procedure can be used to determine which segments of the header piping are damaged. Also, the chillers and pumps would be checked to see if they also fall within the blast radius (or if they are damaged due to flooding if the blast causes damage below the waterline). With the piping network redefined by damaged sections, the valves which would isolate those damaged sections would be assumed shut. This would further reduce the piping network. Once this is complete, undamaged heat loads would be checked for connectivity to a chiller/pump. Lastly, a priority queue would be used to determine which loads would get flow and which loads would have to be secured due to a lack of chilled water available.

Coupling of the survivability analysis and the transient analysis would be one step further. The transient analysis is performed on a select few scenarios, but is not general enough to be performed during casualties. The coupling of these two analyses would be beneficial in determining the transient temperatures, velocities, and pressures during a casualty.

Lastly, the CSDT provides three default layouts of the header mains. Providing more default layouts would be beneficial, especially when designing chilled water systems on other types of ships, such as an amphibious assault ship which has a large well-deck aft. The pipe bends are also created artificially. If the hullform were known beforehand, the header mains could be laid out according to the curvature of the hull.

5.2.2 System Extensions

The three areas in which extension of the chilled water system is most vital include the HVAC system, the SW system and the ship environment.

As stated earlier in the report, nearly ¾ of the heat load serviced by chilled water is related to the HVAC system. The CSDT currently needs to be provided the heat loads at the various spaces within the ship where the chilled water system interfaces with the HVAC system. It would be greatly beneficial to model the HVAC system by compartments with air flow modeled to determine the actual heat load produced based on environmental temperature, number of personnel in a room, heat dissipated by machinery, etc. With this information, the secondary side (air side) of the heat exchanger could be better modeled, and better yet, dynamically modeled, and then tied into the CSDT.

The SW system was modeled generically within the CSDT, but more time could be taken to more completely model the SW system. The most important aspect of modeling the SW system would be to accurately model the A/C units, including the closed loop of the refrigerant. This may prove to be difficult since there are several types of A/C units available, but it may be possible to model the most pertinent types, most notably a centrifugal A/C unit with R134a refrigerant. Several sizes of the A/C units could be modeled as well. With the interface between the CW system and the SW system defined, the interdependency between seawater temperature and the chilled water outlet temperature of the A/C unit could be determined. Other facets of the SW system could also be modeled, such as loads cooled directly with SW or loads that are cooled using a SW/FW heat exchanger (a cheaper alternative to a SW/CW heat exchanger). FNA would still have to be incorporated into this model since there is a strong relationship between parallel branch pressures and flow rates of the SW system.

A third area of future study could be modeling the ship as a whole. This would include a more macroscopic temperature profile of the ship, focusing on how air flows within the ship and how hot spots develop within the ship due to machinery, personnel, and other heat sources, and the effects of stagnant or forced air on those spaces. The macroscopic temperature distribution of the ship could be tied into the HVAC system, which could then be tied into the CW system. In addition, the ship temperature distribution and air flow could be directly tied into the CW system by accounting for the heat loss across the pipe walls and lagging. This loss should be negligible, but will eliminate an assumption of constant and quiescent air external to the CW piping within the CSDT.

Lastly, the reason the CSDT was first considered involved the increasing importance of the CW system based on projected heat loads of all electric ships, and the increasing heat fluxes associated with smaller, more dense equipment operating at increasing switching frequencies. The challenge then becomes removal of heat through less surface area. Focus should be spent on researching and developing methods and models of exotic heat removal techniques which can achieve these higher heat fluxes needed. With models developed, they can be experimentally verified and modeled using the CSDT. With more heat exchanger options available to the naval architect, greater flexibility is afforded in designing the CW system to meet the cooling demands of the future Navy.

References

(n.d.). Retrieved from enggcyclopedia: http://www.enggcyclopedia.com/wp-content/uploads/2012/01/refrigeration-cycle.png

(2007). Retrieved from Alpha Novatech: http://www.alphanovatech.com/almult.html

(2008). Retrieved from R134a - TetraFlouroEthane Properties: http://www.ohio.edu/mechanical/thermo/property_tables/R134a/R134a_TempSat1.html

(2013). Retrieved from ThomasNet: http://www.thomasnet.com/articles/image/centrifugal-pump.jpg

Adam, D. N. (2004). *Module 4: Heat Exchanger*. Retrieved from Dr. Nor Mariah Adam's Homepage: http://eng.upm.edu.my/~mariah/KMP3203/module4.htm

ASTM International. (2008). *Standard Practice for Design and installation of Rigid Pipe Hangars.*

Bell & Gossett. (1998). *Series 1510 Base Mounted Centrifugal Pump Performance cURVES - 60 Hz.* Morton Grove: Xylem Inc.

Bonney Forge. (2012). Retrieved from www.bonneyforge.com: http://www.bonneyforge.com/images/products/csvalves/gate/Gate%20Valve%20Diagram.gif

Cloutier, M. (n.d.). *Refrigeration Cycles.*

Conversion Factor of Weld Joint. (n.d.). Retrieved from AutoDesk Inventor: http://wikihelp.autodesk.com/Inventor/enu/2012/Help/0073-Autodesk73/0742-Engineer742/0743-Joints743/0744-Fixed_Jo744/0773-Weld_Joi773/0800-Conversi800

Copper Development Association, Inc. (2012). *Information about Copper Nickels and their Properties.* Retrieved from Copper.org: Copper Nickel: Alloys, Properties and Fabrication: http://www.copper.org/applications/cuni/txt_properties.html

Cross-Flooding area. (2004, November). Retrieved from Rules for Classification of Steel Ships: http://www1.veristar.com/Veristar/bvrules/E_11_a1_1_1.htm

DRS Technologies. (2011). Retrieved from 2011 Intgrated Marine Systems Catalog: www.drs.com/Products/PESG/PDF/IMScatalog.pdf

Energy-Film. (n.d.). Retrieved from http://www.energy-film.com/wordpress/wp-content/uploads/2010/05/xchanger.jpg

Engineering Toolbox. (n.d.). Retrieved from http://docs.engineeringtoolbox.com/documents/436/logarithmic_mean_temperature_difference.png

Fang, R., Jiang, W., Khan, J., & Dougal, R. (n.d.). *Thermal Modeling and Simulation of the Chilled Water System for Future All Electric Ship.* Columbia: University of South Carolina.

Fiedel, E. (2011). *Cooling System Early Stage Design Tool for Naval Applications.* Cambridge: MIT.

Foltz, D. F. (1990). *The Design of Air Conditioning and Ventilating Systems for nuclear Submarines Since the Nautilus.* New England: The Society of naval Architects and Marine Engineers.

Frank, M. V., & Helmick, D. (2007). *21st Century HVAC System for Future Naval Surface Combatants - Concept Development Report .* West Bethseda: Naval Surface Warfare Center Carderock Division.

Incropera, F. R., & DeWitt, D. P. (2002). *Fundamentals of Heat and Mass Transfer, 5th Edition.* New York: John Wiley & Sons.

Johnson, A., West, E., Miller, B., & Zouridakis, F. (2004). *DD(X) Rail-Gun Conversion Feasibilty Study.* Cambridge: MIT.

McGillan, J., Perotti, T., McCunney, E., & McGovern, M. (n.d.). *Shipboard Thermal Management Systems.* Naval Surface Warfeare Center, Carderock Division (NSWCCD).

MIL-C-2939E(SH). (1984). *Cooling Coils, Air, Duct Type and Gravity Type; Cooler Units, Air, Naval Shipboard Environmental Control Systems.*

MIL-C-2939F(SH). (1990). *Cooling Coils, Airr, Duct Type, and Gravity Type, Naval Shipboard Environmental Control Systems.*

Mills, A. F. (1999). *Basic Heat & Mass Transfer Second Edition.* Upper Saddle River: Prentice Hall.

MIL-PRF-2939G. (2001). *Cooling Coils, Air, Duct Type, and Gravity Type, naval Shipboard Environmental Control Systems.*

MIL-STD-1627B(SH). (1981). *Military Standard Bending of Pipe or Tube for Ship Piping System.* Washington, DC: NAVSEA.

MIL-T-16420K-1. (1978, April 24). *Military Specification, Tube Copper Alloy, Seamless and Welded.* Washington, DC: NAVSEA.

NAVSEA. (1987). *NAVSEA Design Practices and Criteria manual for Surface Ship Freshwater Systems, Chapter 532.* Washington, DC: NAVSEA.

Norsok Standard Fifth Edition. (2006, September).

Pruske, M. A., & Kiehne, T. M. (n.d.). *Thermal-Electric Co-Simulation of Power Conversion Systems aboard an All-Electric Ship.*

Rennels, D. C., & Hudson, H. M. (2012). *Pipe Flow.* Hoboken: John Wiley & Sons.

Sellens. (n.d.). Retrieved from http://me.queensu.ca/People/Sellens/images/Profiles.jpg

Solvay Fluor. (n.d.). Solkane 404A Thermodynamics. Hannover, Germany. Retrieved from www.n8fan.net/item/table-r404a-superheat-proerties-pressure/

Storage Tank Thickness Determination. (2013). Retrieved from Science and Engineers Guide: http://inclusive-science-engineering.com/storage-tank-thickness-determination/

Sunden, B. (2011). *Tubes, Crossflow over.* Retrieved from thermopedia: http://www.thermopedia.com/content/5637/TUBES_CROSSFLOW_OVER_FIG2.gif

System Curve and Pump Performance Curve. (n.d.). Retrieved from The Engineering Toolbox: http://www.engineeringtoolbox.com/pump-system-curves-d_635.html

Travkin, V. (2001). *Thermal Physics*. Retrieved from www.travkin-hspt.com: http://travkin-hspt.com/thermph/pic/fig7_01.gif

Appendix A: Simulated Heat Loads

Load Name	Cooling Load Shore Cond. (KW)	Cooling Load Design Cond. (KW)	Cooling Load Cruise Cond. (KW)	Cooling Load Battle Cond. (KW)	x-loc. (m)	y-loc. (m)	z-loc. (m)
RS01	9.56	7.91325	6.92849	7.91325	32.81	0.00	25.20
RS02	3.56	20.22275	20.22275	20.22275	33.14	0.00	22.83
RS02_1	3.56	1.96952	1.96952	1.96952	26.45	0.00	3.29
RS04	3.56	11.1492417	10.9030517	11.1492417	33.81	6.70	19.69
RS0405	0.24	13.54045	12.52052	13.54045	33.81	0.00	19.69
RS0405_1	45	5.31067	4.99414	5.31067	29.12	-5.52	19.69
RS04_1	20	6.50645	6.26026	6.50645	23.77	6.70	19.69
RS0405_2	9.7	3.79836	3.48183	3.79836	29.12	5.52	19.69
RS040505C	9.7	45.721	41.81713	45.721	29.12	0.00	19.69
RS05	0.9	10.30481	10.09379	10.30481	33.81	-6.70	19.69
RS05_1	1.74	6.64713	6.40094	6.64713	23.77	-6.70	19.69
RS07	-	5.41618	4.9238	5.41618	-14.39	0.00	22.83
RS07_1	-	3.62251	3.13013	3.62251	-17.74	0.00	19.69
RS07_2	-	8.01876	7.06917	8.01876	-15.06	-3.85	14.05
RS08	-	3.628617	3.628617	3.628617	39.17	0.00	16.59
RS12	-	6.71747	6.71747	6.71747	33.14	0.00	14.05
RS12_1	-	1.54748	1.54748	1.54748	34.48	-8.54	8.51
RS12_2	-	1.12544	1.12544	1.12544	23.77	-8.54	8.51
RS12_3	-	5.31067	5.31067	5.31067	33.14	0.00	14.05
RS12_4	-	1.37163	1.37163	1.37163	22.09	-5.86	8.51
RS12_5	-	3.02462	3.02462	3.02462	35.82	0.00	14.05
RS12_6	-	1.26612	1.26612	1.26612	39.84	-11.72	14.05
RS12_7	-	0.91442	0.91442	0.91442	23.77	-3.35	14.05
RS13	-	40.6913383	40.6913383	40.6913383	34.48	0.00	8.51
RS13_1	-	1.0551	1.0551	1.0551	34.48	0.00	8.51
RS14	-	61.09029	61.09029	61.09029	34.48	-5.02	5.80
RS14_1	-	56.72921	56.72921	56.72921	24.10	-3.35	5.80
RS17	-	9.17937	9.17937	9.17937	-36.49	1.67	14.05
RS17_1	-	3.1128967	3.1128967	3.1128967	-23.77	0.00	14.05
RS1819	-	7.31536	7.31536	7.31536	68.63	0.00	8.51
RS1819_1	-	7.63189	7.63189	7.63189	68.63	0.00	5.80
RS19	-	1.7585	1.7585	1.7585	69.29	0.00	2.95
RS20	-	33.23565	33.23565	33.23565	61.26	0.00	8.51
RS21	-	4.71278	4.71278	4.71278	62.60	1.34	11.33
RS21_1	-	1.1138339	1.1138339	1.1138339	57.24	4.69	8.51
RS21_2	-	0.87925	0.87925	0.87925	58.58	-3.35	8.51
RS21_3	-	0.77374	0.77374	0.77374	59.92	-3.35	11.33
RS21_4	-	4.85346	4.85346	4.85346	62.60	-3.35	8.51
RS21_5	-	0.63306	0.63306	0.63306	62.60	1.17	8.51

RS21_6	-	0.14068	0.14068	0.14068	58.58	4.35	8.51
RS23	-	1.310129	1.310129	1.310129	50.88	-5.52	5.80
RS2324	-	22.36812	22.36812	22.36812	50.55	0.00	5.80
RS24	-	1.30129	1.30129	1.30129	50.21	2.51	5.80
RS26	-	0.56272	0.56272	0.56272	39.84	-6.86	8.51
RS28	-	9.70692	9.70692	9.70692	23.77	4.85	8.51
RS28_1	-	1.33646	1.33646	1.33646	23.77	9.54	8.51
RS28_2	-	1.68816	1.68816	1.68816	34.48	8.54	8.51
RS28_3	-	0.94959	0.94959	0.94959	23.77	7.20	8.51
RS30	-	3.76319	3.76319	3.76319	3.01	-7.20	8.51
RS30_1	-	0.56272	0.56272	0.56272	3.01	-4.69	8.51
RS32	-	8.33529	8.33529	8.33529	-8.37	0.00	8.51
RS32_1	-	4.5721	4.5721	4.5721	-8.37	4.52	8.51
RS32_2	-	0.73857	0.73857	0.73857	-8.37	7.70	8.51
RS32_3	-	6.50645	6.50645	6.50645	-8.37	9.88	8.51
RS36	-	19.48418	19.48418	19.48418	-13.06	0.00	8.51
RS37	-	16.56507	16.56507	16.56507	-23.77	0.00	8.51
RS37_1	-	4.15006	4.15006	4.15006	-28.45	0.00	8.51
RS42	-	49.13249	49.13249	49.13249	-36.49	0.00	5.80
RS43	-	9.4959	9.4959	9.4959	59.92	3.18	5.80
RS43_1	-	16.10786	16.10786	16.10786	58.92	-3.18	5.80
RS43_2	-	0.94959	0.94959	0.94959	62.60	-2.51	5.80
RS46	-	8.26495	8.26495	8.26495	3.01	-3.19	5.80
RS46_1	-	3.58734	3.58734	3.58734	0.33	0.00	5.80
RS46_2	-	14.3676484	14.3676484	14.3676484	-3.68	0.00	5.80
RS46_3	-	7.63189	7.63189	7.63189	3.01	-3.18	5.80
RS47	-	1.12544	1.12544	1.12544	-3.68	-2.51	5.80
RS51	-	6.4892167	5.9968367	6.4892167	-47.20	-1.67	3.29
RS51_1	-	5.0472467	5.0472467	5.0472467	-47.20	1.67	3.29
RS53	-	3.09496	3.09496	3.09496	-63.27	-6.86	5.80
RS53_1	-	6.8113739	6.8113739	6.8113739	-60.59	0.00	5.80
RS53_2	-	1.19578	1.19578	1.19578	-60.59	-6.86	5.80
RS53_3	-	2.95428	2.95428	2.95428	-73.98	-5.35	5.80
RS55	-	1.7940217	1.7940217	1.7940217	-71.30	6.36	5.80
RS57	-	2.2336467	2.2336467	2.2336467	70.63	0.00	3.29
RS58	-	56.06098	53.84527	56.06098	34.48	3.85	3.29
RS60	-	34.3966117	34.3966117	33.4118517	34.48	5.02	5.80
RS61	-	14.49004	14.49004	14.49004	25.44	3.85	5.80
RS61_1	-	8.68699	8.61665	8.68699	-23.77	0.00	3.29
RS63	-	53.14187	52.33296	53.14187	-30.13	0.00	3.29
RS6674	-	13.7166517	13.7166517	13.7166517	-71.30	0.00	2.95
RS69	-	4.0097317	4.0097317	4.0097317	62.60	0.00	2.95
RS70	-	9.70692	9.63658	9.70692	45.19	0.00	2.95
RS71	-	12.06331	12.06331	12.06331	-23.77	-1.34	2.95
RS72	-	2.6381017	2.6381017	2.6381017	-48.87	-1.67	3.29

RS74	-	4.18523	4.18523	4.18523	-71.30	-6.86	5.80
RS75	-	16.8816	16.8816	16.8816	33.14	3.35	14.05
RS78	-	4.39625	4.39625	4.39625	39.17	2.18	16.59
RS8688	-	2.1102	2.1102	2.1102	14.39	2.01	16.59
RS102104	-	2.1102	2.1102	2.1102	11.72	-2.01	16.59
RS103105	-	1.58265	1.58265	1.58265	-7.30	1.84	14.05
RS5C	-	2.1102	2.1102	2.1102	-53.56	-2.01	14.05
RS8C	-	1.58265	1.58265	1.58265	62.60	0.00	3.29
RS6E	-	0.14068	0.14068	0.14068	53.90	-1.84	3.29
RS1G	-	0.14068	0.14068	0.14068	53.90	1.84	3.29
RS2G	-	1.09027	1.09027	1.09027	62.60	0.00	3.29
RS3G	-	4.67761	4.67761	4.67761	-28.45	-4.85	14.05
RS5G	-	1.0551	1.0551	1.0551	-23.77	4.35	11.33
RS6G	-	1.0551	1.0551	1.0551	9.71	-8.20	11.33
RS7G	-	9.1445517	9.1445517	9.1445517	18.41	4.18	5.80
RS03	-	1.89918	1.89918	0.17585	23.77	1.51	22.83
RS06	-	17.6908617	17.6908617	0.4575617	23.77	3.01	16.59
RS09	-	9.9935555	9.9935555	5.4917955	34.48	7.53	16.59
RS10	-	5.83822	5.83822	-	25.11	-4.69	16.59
RS10_1	-	1.33646	1.33646	-	25.11	4.69	16.59
RS10_2	-	1.09027	1.09027	0.3517	23.77	-7.70	16.59
RS10_3	-	2.74326	2.74326	1.58265	27.78	-8.03	16.59
RS10_4	-	0.7034	0.7034	0.3517	28.12	-3.18	16.59
RS10_5	-	0.56272	0.56272	0.3517	28.12	2.18	16.59
RS10_6	-	1.4068	1.4068	0.3517	31.13	-8.03	16.59
RS10_7	-	0.24619	0.24619	-	31.13	-3.18	16.59
RS10_8	-	0.56272	0.56272	0.3517	31.13	2.18	16.59
RS10_9	-	0.14068	0.14068	0.3517	32.14	-0.50	16.59
RS10_10	-	1.33646	1.33646	0.3517	34.48	-3.85	16.59
RS10_11	-	0.91442	0.91442	0.3517	34.48	2.68	16.59
RS10_12	-	8.37046	8.37046	-	-4.35	4.02	5.80
RS11	-	0.9323567	0.9323567	0.2289567	18.41	4.85	14.05
RS15	-	3.8518184	3.8518184	0.6865184	18.41	1.85	14.05
RS15_1	-	1.33646	1.33646	-	13.72	1.85	14.05
RS15_2	-	4.1152417	4.1152417	0.4575617	12.72	0.00	8.51
RS16	-	59.4745802	59.4745802	2.2881602	50.55	6.86	8.51
RS25	-	8.12427	8.12427	0.87925	44.52	5.69	8.51
RS25_1	-	1.37163	1.37163	-	50.55	-4.69	8.51
RS2527	-	9.00352	9.00352	-	39.84	-2.51	8.51
RS27	-	5.13482	5.13482	0.52755	38.16	-4.52	8.51
RS27_1	-	0.56272	0.56272	0.17585	48.54	-4.85	8.51
RS27_2	-	1.30129	1.30129	0.3517	43.18	0.84	8.51
RS27_3	-	1.01993	1.01993	0.3517	39.84	0.00	8.51
RS27_4	-	2.32122	2.32122	1.4068	7.70	-3.68	8.51
RS29	-	13.4073972	13.4073972	5.9493572	18.41	0.00	8.51

RS29_1	-	8.0891	8.0891	0.3517	18.41	-9.54	8.51
RS29_2	-	13.68113	13.68113	-	3.01	1.51	8.51
RS31	-	8.37046	8.37046	-	3.01	-9.71	8.51
RS31_1	-	22.7222819	22.7222819	20.8231019	3.01	-4.69	8.51
RS31_2	-	13.57562	13.57562	-	-77.33	0.00	5.80
RS34	-	4.7669418	4.7669418	1.6016418	-9.71	-7.87	8.51
RS35	-	8.19461	8.19461	1.23095	-9.71	-4.85	8.51
RS35_1	-	0.66823	0.66823	-	-8.37	-9.88	8.51
RS35_2	-	4.4141867	4.4141867	0.2289567	-23.77	-7.70	8.51
RS38	-	10.09379	10.09379	0.17585	-33.81	1.67	14.05
RS38_1	-	1.16061	1.16061	0.52755	-28.45	-2.01	14.05
RS38_2	-	1.82884	1.82884	-	-30.46	-2.01	14.05
RS38_3	-	3.23564	3.23564	-	-28.79	-2.01	14.05
RS38_4	-	1.0551	1.0551	-	-3.01	2.18	5.80
RS3973	-	0.01	0.01	0.01	-36.49	7.87	5.80
RS39	-	2.42673	2.42673	0.3517	-39.17	1.34	3.29
RS40	-	2.0409151	2.0409151	0.9154751	50.55	-4.35	5.80
RS44	-	20.67996	20.67996	10.1993	-23.77	7.03	5.80
RS45	-	6.5247384	6.5247384	0.6865184	15.40	8.37	5.80
RS45_1	-	0.87925	0.87925	-	9.71	6.19	5.80
RS45_2	-	8.6880451	8.6880451	0.9154751	-23.77	2.01	5.80
RS48	-	17.47949	17.47949	12.3095	-23.77	-2.85	5.80
RS48_1	-	8.4408	8.4408	0.87925	-30.13	1.67	14.05
RS49	-	7.20985	7.20985	0.3517	-23.77	4.18	14.05
RS49_1	-	1.68816	1.68816	0.17585	-23.77	-7.70	8.51
RS49_2	-	10.3093821	10.3093821	4.5766721	-26.78	1.67	14.05
RS49_3	-	1.23095	1.23095	-	-28.45	1.67	14.05
RS49_4	-	0.80891	0.80891	-	-39.50	-7.87	5.80
RS50	-	8.26495	8.26495	2.1102	-36.49	-4.35	5.80
RS50_1	-	4.64244	4.64244	1.0551	-36.49	-5.36	3.29
RS50_2	-	2.56741	2.56741	-	-55.57	-7.53	5.80
RS52	-	5.6978917	5.6978917	0.4575617	-47.20	-8.70	5.80
RS52_1	-	1.5478317	1.5478317	0.4575617	-50.88	-6.36	5.80
RS52_2	-	2.39156	2.39156	-	-47.20	-7.53	5.80
RS52_3	-	2.32122	2.32122	-	-61.93	0.00	5.80
RS54	-	51.2437451	51.2437451	0.9154751	-71.30	4.52	5.80
RS56	-	1.0730367	1.0730367	0.2289567	-76.66	5.69	5.80
RS56_1	-	1.65299	1.65299	-	44.19	3.18	3.29
RS59	-	14.7714	14.7714	5.6272	3.01	5.36	3.29
RS62	-	4.71278	4.71278	-	3.01	-5.36	3.29
RS62_1	-	4.15006	4.15006	1.23095	-27.11	5.36	3.29
RS64	-	5.80305	5.80305	2.4619	-23.77	-5.36	3.29
RS65	-	8.19461	8.19461	3.1653	-40.50	1.67	3.29
RS68	-	5.87339	5.87339	2.1102	-43.18	7.53	5.80
RS73	-	2.42673	2.42673	0.52755	-50.88	0.00	8.51

RS76	-	3.0073867	3.0073867	0.2289567	-47.20	4.35	5.80
RS77	-	5.16999	5.16999	-	-49.54	-2.01	14.05
RS79	-	1.79367	1.79367	0.3517	23.77	4.85	14.05
RS1D	-	2.00469	2.00469	0.3517	-49.88	0.00	8.51
RS8D	-	3.1653	3.1653	-	18.08	1.84	14.05
RS7E	-	2.5505284	2.5505284	0.6865184	-23.77	7.03	5.80

Appendix B: Refrigerant Characteristics

R134a Refrigerant - Saturated

T(°C)	P(kPa)	h_f(kJ/kg)	h_g(kJ/kg)
-40	51.2	0	225.86
-36	62.9	5.04	228.39
-32	76.7	10.1	230.92
-28	92.7	15.2	233.43
-26	101.7	17.76	234.68
-24	111.3	20.33	235.93
-2	121.7	22.91	237.17
-20	132.7	25.49	238.41
-18	144.6	28.09	239.64
-16	157.3	30.69	240.87
-14	170.8	33.3	242.09
-12	185.2	35.92	243.31
-10	200.6	38.55	244.52
-8	216.9	41.19	245.72
-6	234.3	43.84	246.92
-4	252.7	46.5	248.11
-2	272.2	49.17	249.29
0	292.8	51.86	250.46
2	314.6	54.55	251.62
4	337.7	57.25	252.78
6	362	59.97	253.92
8	387.6	62.69	255.05
12	443	68.19	257.29
16	504.3	73.73	259.47
20	571.7	79.32	261.6
24	645.8	84.98	263.68
26	685.4	87.83	264.7
28	726.9	90.7	265.69
30	770.2	93.58	266.67
32	815.4	96.48	267.64
34	862.6	99.4	268.58
36	911.9	102.33	269.5
38	963.2	105.29	270.41
40	1016.6	108.27	271.28
42	1072.2	111.26	272.14

44	1130.1	114.28	272.97
48	1252.9	120.39	274.55
52	1385.4	126.6	276.01
56	1528.2	132.92	277.32
60	1681.8	139.36	278.49
70	2116.8	156.14	280.51
80	2633.2	174.25	280.67
90	3244.2	194.78	277.27
100	3972.4	225.15	259.54
101.6	4059.1	241.49	241.49

Table 15: R134a Saturated table (R134a - TetraFlouroEthane Properties, 2008)

R134a Refrigerant – Superheated Vapor

T(°C)\P(MPa)	0.06	0.1	0.14	0.18	0.2	0.24	0.28	0.32	0.4	0.5
-20	240.8	239.5								
-10	248.6	247.5	246.4	245.2						
0	256.5	255.6	254.6	253.6	253.1	252				
10	264.7	263.8	262.9	262	261.6	260.7	259.7	258.7	256.6	
20	272.9	272.2	271.4	270.6	270.2	269.4	268.5	267.7	265.9	263.5
30	281.4	280.7	280	279.3	278.9	278.2	277.4	276.7	275.1	273
40	290	289.3	288.7	288.1	287.7	287.1	286.4	285.7	284.3	282.5
50	298.7	298.2	297.6	297	296.7	296.1	295.5	294.9	293.6	292
60	307.7	307.1	306.6	306.1	305.8	305.2	304.7	304.1	301	301.5
70	316.8	316.3	315.8	315.3	315	314.5	314	313.5	312.4	311.1
80	326	325.6	325.1	324.6	324.4	323.9	323.5	323	322	320.8
90	335.4	335	334.6	334.1	333.9	333.5	333.1	332.6	331.7	330.6
100	345	344.6	344.2	343.8	343.6	343.2	342.8	342.4	341.6	340.5
110							352.7	352.3	351.5	350.6
120							362.7	362.4	361.6	360.7
130										371
140										381.5
150										
160										
170										
180										

Table 16: R134a - Superheated vapor table (Ppressure 0.06MPa-0.5MPa) (R134a - TetraFlouroEthane Properties, 2008)

T(°C)\P(MPa)	0.6	0.7	0.8	0.9	1	1.2	1.4	1.6	1.8	2
-20										
-10										
0										
10										
20										
30	270.8	268.45								
40	280.6	278.58	276.5	274.2	271.7					
50	290.3	288.53	286.7	284.8	282.7	278.3				
60	300	298.43	296.8	295.1	293.4	289.6	285.5	280.7		
70	309.7	308.33	306.9	305.4	303.9	300.6	297.1	283.3	288.9	283.9
80	319.6	318.28	317	315.6	314.3	311.4	308.3	305.1	301.5	297.6
90	329.5	328.3	327.1	325.9	324.7	322.1	319.4	316.5	313.5	310.2
100	339.5	338.4	337.3	336.2	335.1	332.7	330.3	327.8	325.1	322.3
110	349.6	348.6	347.6	346.6	345.5	343.4	341.2	338.9	336.5	334.1
120	359.8	358.91	358	357	356.1	354.1	352.1	350	347.9	345.7
130	370.2	369.32	368.5	367.6	366.7	364.9	363	361.1	359.2	357.2
140	380.7	379.86	379.1	378.2	377.4	375.7	374	372.3	370.5	368.6
150			389.8	389	388.2	386.7	385.1	383.5	381.6	380.1
160			400.6	399.9	399.2	397.7	396.2	394.7	393.2	391.6
170						408.8	407.4	406	404.6	403.1
180						420.1	418.8	417.4	416.1	414.8

Table 17: R134a Superheated vapor table (pressure 0.6MPa-2.0MPa) (R134a - TetraFlouroEthane Properties, 2008)

R404a Refrigerant – Saturated

T(°C)	P_f(MPa)	P_g(MPa)	h_f(kJ/kg)	h_g(kJ/kg)
-60	0.0508	0.0484	123.1	330.92
-59	0.0537	0.0512	124.29	331.54
-58	0.0567	0.0541	125.48	332.15
-57	0.0598	0.0571	126.68	332.76
-56	0.0631	0.0603	127.88	333.37
-55	0.0665	0.0636	129.08	333.98
-54	0.07	0.067	130.28	334.59
-53	0.0737	0.0706	131.48	335.2
-52	0.0775	0.0743	132.69	335.81
-51	0.0815	0.0782	133.9	336.41
-50	0.0857	0.0823	135.11	337.02
-49	0.09	0.0865	136.33	337.62
-48	0.0945	0.0909	137.55	338.23
-47	0.0992	0.0955	138.77	338.83
-46	0.1041	0.1002	139.99	339.43
-45	0.1091	0.1051	141.22	340.03
-44	0.1143	0.1102	142.45	340.63
-43	0.1198	0.1155	143.69	341.23
-42	0.1254	0.1211	144.92	341.83
-41	0.1312	0.1268	146.16	342.43
-40	0.1373	0.1327	147.41	343.02
-39	0.1435	0.1388	148.66	343.62
-38	0.15	0.1451	149.91	344.21
-37	0.1567	0.1517	151.16	344.8
-36	0.1636	0.1585	152.42	345.39
-35	0.1708	0.1656	153.69	345.98
-34	0.1782	0.1728	154.95	346.56
-33	0.1858	0.1803	156.22	347.15
-32	0.1937	0.1881	157.5	347.73
-31	0.2019	0.1961	158.77	348.31
-30	0.2103	0.2044	160.06	348.89
-29	0.219	0.213	161.34	349.47
-28	0.228	0.2218	162.63	350.05
-27	0.2372	0.2309	163.92	350.62
-26	0.2468	0.2403	165.22	351.2
-25	0.2566	0.25	166.51	351.77
-24	0.2667	0.2599	167.82	352.33
-23	0.2771	0.2702	169.12	352.9

-22	0.2879	0.2808	170.43	353.47
-21	0.2989	0.2917	171.74	354.03
-20	0.3103	0.3029	173.06	354.59
-19	0.322	0.3145	174.38	355.15
-18	0.334	0.3263	175.7	355.7
-17	0.3464	0.3386	177.03	356.25
-16	0.3591	0.3511	178.36	356.8
-15	0.3721	0.364	179.69	357.35
-14	0.3856	0.3773	181.03	357.9
-13	0.3994	0.3909	182.37	358.44
-12	0.4135	0.4049	183.71	358.98
-11	0.428	0.4193	185.06	359.52
-10	0.443	0.4341	186.41	360.05
-9	0.4583	0.4492	187.76	360.58
-8	0.474	0.4647	189.12	361.11
-7	0.4901	0.4807	190.48	361.64
-6	0.5066	0.497	191.84	362.16
-5	0.5235	0.5138	193.21	362.68
-4	0.5409	0.531	194.58	363.19
-3	0.5586	0.5486	195.95	363.7
-2	0.5768	0.5667	197.32	364.21
-1	0.5955	0.5852	198.7	364.72
0	0.6146	0.6041	200	365.22
1	0.6342	0.6235	201.47	365.71
2	0.6542	0.6434	202.86	366.21
3	0.6747	0.6637	204.25	366.7
4	0.6957	0.6846	205.65	367.18
5	0.7171	0.7059	207.05	367.66
6	0.7391	0.7277	208.46	368.14
7	0.7615	0.75	209.86	368.61
8	0.7845	0.7728	211.28	369.08
9	0.808	0.7961	212.69	369.54
10	0.832	0.8199	214.11	370
11	0.8565	0.8443	215.54	370.45
12	0.8815	0.8692	216.97	370.9
13	0.9071	0.8946	218.4	371.34
14	0.9333	0.9206	219.84	371.78
15	0.96	0.9472	221.28	372.21
16	0.9873	0.9743	222.73	372.63
17	1.015	1.002	224.19	373.05
18	1.044	1.03	225.65	373.46

19	1.073	1.059	227.11	373.87
20	1.102	1.089	228.59	374.27
21	1.132	1.119	230.06	374.66
22	1.163	1.15	231.55	375.05
23	1.195	1.181	233.04	375.42
24	1.227	1.213	234.54	375.79
25	1.26	1.245	236.05	376.15
26	1.293	1.279	237.56	376.51
27	1.327	1.313	239.08	376.85
28	1.362	1.347	240.62	377.19
29	1.397	1.383	242.16	377.51
30	1.433	1.419	243.7	377.83
31	1.47	1.455	245.26	378.14
32	1.507	1.493	246.83	378.43
33	1.546	1.531	248.41	378.72
34	1.584	1.569	250	378.99
35	1.624	1.609	251.6	379.25
36	1.664	1.649	253.22	379.5
37	1.706	1.69	254.85	379.74
38	1.747	1.732	256.48	379.96
39	1.79	1.775	258.14	380.17
40	1.834	1.818	259.81	380.37
41	1.878	1.862	261.49	380.54
42	1.923	1.907	263.18	380.71
43	1.969	1.953	264.9	380.85
44	2.015	2	266.63	380.98
45	2.063	2.047	268.37	381.08
46	2.111	2.096	270.14	381.17
47	2.16	2.145	271.92	381.23
48	2.211	2.195	273.73	381.27
49	2.262	2.246	275.55	381.29
50	2.313	2.298	277.39	381.28
51	2.366	2.351	279.26	381.24
52	2.42	2.404	281.15	381.17
53	2.475	2.459	283.06	381.07
54	2.53	2.515	285	380.93
55	2.587	2.572	286.96	380.76
56	2.645	2.63	288.95	380.54
57	2.703	2.688	290.97	380.27
58	2.763	2.748	293.01	379.95
59	2.824	2.809	295.08	379.58

60	2.886	2.971	297.19	379.14
61	2.949	2.934	299.32	378.62
62	3.013	2.999	301.49	378.02
63	3.078	3.064	303.69	377.32
64	3.144	3.131	305.92	376.5
65	3.212	3.199	308.19	375.53
66	3.281	3.269	310.5	374.38
67	3.352	3.34	312.85	372.99
68	3.423	3.412	315.23	371.26

Table 18: R404a Saturated table (Solvay Fluor)

R404a Refrigerant – Superheated Vapor

T(°C)\P(MPa)	0.082	0.091	0.100	0.110	0.121	0.133	0.145	0.159	0.173	0.188	0.204
-50	337.02										
-45	340.82	340.52	340.20								
-40	344.65	344.38	344.08	343.76	343.41	343.02					
-35	348.52	348.27	347.99	347.69	347.36	347.00	346.62	346.20			
-30	352.43	352.19	351.93	351.65	351.34	351.01	350.65	350.26	349.84	349.39	348.89
-25	356.37	356.15	355.90	355.64	355.35	355.05	354.71	354.35	353.96	353.53	353.07
-20	360.35	360.14	359.92	359.67	359.40	359.11	358.80	358.46	358.09	357.70	357.27
-15	364.37	364.18	363.96	363.73	363.48	363.21	362.91	362.60	362.25	361.88	361.49
-10	368.44	368.25	368.05	367.83	367.59	367.34	367.06	366.76	366.44	366.10	365.72
-5	372.54	372.36	372.17	371.97	371.74	371.50	371.24	370.96	370.66	370.34	369.99
0	376.68	376.52	376.34	376.14	375.93	375.70	375.46	375.20	374.91	374.61	374.28
5	380.86	380.71	380.54	380.35	380.16	379.94	379.71	379.46	379.19	378.91	378.60
10	385.09	384.94	384.78	384.61	384.42	384.22	384.00	383.76	383.51	383.24	382.94
15	389.36	389.22	389.06	388.90	388.72	388.53	388.32	388.10	387.86	387.60	387.32
20	393.66	393.53	393.39	393.23	393.06	392.88	392.68	392.47	392.24	392.00	391.74
25	398.01	397.89	397.75	397.60	397.44	397.27	397.08	396.88	396.66	396.43	396.18
30	402.41	402.29	402.15	402.01	401.86	401.70	401.52	401.33	401.12	400.90	400.66
35	406.84	406.73	406.60	406.47	406.32	406.16	405.99	405.81	405.62	405.41	405.18
40				410.96	410.82	410.67	410.51	410.33	410.15	409.95	409.73
45						415.21	415.06	414.89	414.72	414.53	414.32
50									419.32	419.14	418.95
55											423.61
60											
65											
70											
75											

Table 19: R404 Superheated vapor table (pressure 0.082MPa-0.204MPa) (Solvay Fluor)

T(°C)\P(MPa)	0.222	0.240	0.260	0.281	0.303	0.326	0.351	0.377	0.405	0.434	0.465
-50											
-45											
-40											
-35											
-30	352.58	352.05									
-25	356.81	356.32	355.78	355.21	354.59						
-20	361.06	360.60	360.10	359.57	358.99	358.37	357.70				
-15	365.32	364.89	364.43	363.93	363.40	362.82	362.20	361.54	360.82	360.05	
-10	369.61	369.21	368.77	368.31	367.81	367.27	366.70	366.08	365.42	364.70	363.94
-5	373.92	373.54	373.14	372.70	372.23	371.73	371.20	370.62	370.00	369.34	368.63
0	378.26	377.91	377.52	377.11	376.68	376.21	375.70	375.16	374.59	373.97	373.31
5	382.63	382.29	381.93	381.55	381.14	380.39	380.22	379.72	379.18	378.60	377.99
10	387.03	386.71	386.37	386.01	385.62	385.20	384.76	384.28	383.78	383.24	382.66
15	391.46	391.16	390.83	390.49	390.12	389.73	389.31	388.87	388.39	387.88	387.34
20	395.92	395.63	395.33	395.00	394.65	394.28	393.89	393.47	393.02	392.54	392.03
25	400.41	400.14	399.85	399.54	399.21	398.86	398.49	398.09	397.66	397.21	396.73
30	404.94	404.68	404.41	404.11	403.80	403.47	403.11	402.73	402.33	401.90	401.45
35	409.50	409.26	409.00	408.72	408.42	408.10	407.76	407.41	407.02	406.62	406.19
40	414.10	413.87	413.62	413.35	413.07	412.77	412.45	412.10	411.74	411.35	410.94
45	418.74	418.51	418.27	418.02	417.75	417.46	417.16	416.83	416.48	416.12	415.73
50	423.41	423.19	422.97	422.72	422.46	422.19	421.90	421.58	421.25	420.90	420.53
55			427.69	427.46	427.21	426.95	426.67	426.37	426.05	425.72	425.36
60					431.99	431.74	431.47	431.19	430.88	430.56	430.22
65								436.03	435.74	435.44	435.11
70										440.34	440.03
75											

Table 20: R404 Superheated vapor table (pressure 0.222MPa-0.465MPa) (Solvay Fluor)

T(°C)\P(MPa)	0.497	0.531	0.567	0.604	0.643	0.685	0.728	0.773	0.820	0.869	0.921
-5	363.11										
0	367.87	367.05	366.17	365.22							
5	372.60	371.85	371.03	370.16	369.22	368.20					
10	377.33	376.62	375.87	375.06	374.19	373.26	372.25	371.17	370.00		
15	382.04	381.39	380.68	379.93	379.13	378.26	377.34	376.35	375.28	374.12	372.87
20	386.76	386.15	385.49	384.79	384.04	383.24	382.38	381.46	380.48	379.42	378.28
25	391.48	390.91	390.29	389.63	388.93	388.19	387.39	386.54	385.63	384.65	383.61
30	396.22	395.67	395.09	394.48	393.82	393.12	392.38	391.58	390.74	389.83	388.87
35	400.96	400.45	399.90	399.32	398.71	398.05	397.35	396.61	395.82	394.98	394.08
40	405.73	405.24	404.72	404.18	403.59	402.98	402.32	401.62	400.88	400.09	399.26
45	410.51	410.05	409.56	409.04	408.49	407.91	407.29	406.63	405.93	405.19	404.41
50	415.31	414.87	414.41	413.92	413.40	412.84	412.26	411.64	410.98	410.28	409.55
55	420.14	419.72	419.28	418.81	418.32	417.79	417.24	416.65	416.03	415.37	414.67
60	424.99	424.59	424.17	423.72	423.25	422.75	422.23	421.67	421.08	420.46	419.80
65	429.87	429.49	429.08	428.66	428.21	427.73	427.23	426.70	426.14	425.55	424.93
70	434.77	434.41	434.02	433.62	433.19	432.73	432.25	431.75	431.22	430.65	430.06
75	439.70	439.35	438.99	438.60	438.19	437.75	437.29	436.81	436.30	435.77	435.20
80		444.33	443.98	443.60	443.21	442.79	442.36	441.89	441.41	440.90	440.36
85				448.64	448.26	447.86	447.44	447.00	446.53	446.04	445.53
90							452.55	452.12	451.68	451.21	450.71
95									456.84	456.39	455.92
100											
105											
110											
115											
120											
125											
130											
135											
140											
145											

Table 21: R404a Superheated vapor table (pressure 0.497MPa-0.921MPa) (Solvay Fluor)

T(°C)\P(MPa)	0.974	1.030	1.089	1.150	1.213	1.279	1.347	1.419	1.493	1.569	1.649
-5											
0											
5											
10											
15											
20	377.05	375.72	374.27								
25	382.48	381.27	379.96	378.54	376.99						
30	387.83	386.72	385.53	384.24	382.84	381.32	379.66	377.83			
35	393.12	392.09	391.00	389.82	388.55	387.18	385.69	384.07	382.29	380.32	
40	398.36	397.41	396.40	395.31	394.14	392.89	391.54	390.08	388.50	386.77	384.86
45	403.57	402.69	401.74	400.73	399.66	398.51	397.27	395.94	394.51	392.96	391.27
50	408.76	407.93	407.05	406.11	405.11	404.04	402.90	401.68	400.38	398.97	397.45
55	413.94	413.15	412.33	411.45	410.51	409.52	408.46	407.33	406.13	404.84	403.46
60	419.10	418.37	417.58	416.76	415.88	414.95	413.97	412.92	411.80	410.61	409.34
65	424.27	423.57	422.83	422.05	421.23	420.36	419.43	418.45	417.41	416.30	415.12
70	429.43	428.77	428.07	427.34	426.56	425.73	424.86	423.94	422.49	421.93	420.84
75	434.61	433.98	433.31	432.61	431.88	431.10	430.28	429.41	428.49	427.52	426.49
80	439.79	439.19	438.56	437.89	437.19	436.45	435.68	434.86	433.99	433.08	432.11
85	444.98	444.41	443.81	443.18	442.51	441.81	441.07	440.29	439.47	438.61	437.69
90	450.19	449.65	449.07	448.47	447.83	447.16	446.46	445.72	444.94	444.12	443.26
95	455.42	454.90	454.35	453.77	453.16	452.52	451.85	451.14	450.40	449.62	448.80
100	460.67	460.16	459.63	459.08	458.50	457.89	457.24	456.57	455.86	455.12	454.34
105			464.94	464.41	463.85	463.26	462.65	462.00	461.33	460.62	459.87
110						468.66	468.07	467.45	466.80	466.12	465.41
115								472.90	472.28	471.63	470.94
120											476.49
125											
130											
135											
140											
145											

Table 22: R404a Superheated vapor table (pressure 0.974MPa-1.649MPa) (Solvay Fluor)

T(°C)\P(MPa)	1.732	1.818	1.907	2.000	2.096	2.195	2.298	2.404	2.515	2.630	2.748	2.871
-5												
0												
5												
10												
15												
20												
25												
30												
35												
40	382.75	380.37										
45	389.42	387.38	385.10	382.52								
50	395.80	394.00	392.03	389.84	387.38	384.57	381.28					
55	401.96	400.35	398.60	396.68	394.57	392.22	389.57	386.52	382.89			
60	407.97	406.51	404.93	403.22	401.36	399.32	397.06	394.54	391.68	388.35	384.33	379.14
65	413.87	412.52	411.08	409.53	407.86	406.05	404.07	401.90	399.49	396.78	393.69	390.06
70	419.67	418.43	417.10	415.68	414.16	412.52	410.75	408.83	406.73	404.42	401.85	398.95
75	425.40	424.25	423.02	421.71	420.31	418.82	417.21	415.48	413.61	411.57	409.35	406.89
80	431.09	430.01	248.86	427.64	426.35	424.97	423.50	421.92	420.23	418.40	416.42	414.27
85	436.73	435.72	434.64	433.50	432.30	431.02	429.66	428.20	426.65	424.99	423.21	421.28
90	442.35	441.39	440.38	439.31	438.18	436.98	435.71	434.37	432.94	431.41	429.78	428.03
95	447.94	447.03	446.08	445.07	444.01	442.88	441.70	440.44	439.11	437.70	436.19	434.59
100	453.52	452.66	451.75	450.80	449.79	448.74	447.62	446.44	445.20	443.88	442.48	441.00
105	459.09	458.27	457.41	456.51	455.55	454.55	453.50	452.39	451.22	449.98	448.68	447.29
110	464.66	463.88	463.06	462.19	461.29	460.34	459.34	458.29	457.19	456.02	454.80	453.50
115	470.23	469.48	468.70	467.87	467.01	466.11	465.16	464.16	463.12	462.02	460.86	459.64
120	475.80	475.09	474.33	473.55	472.72	471.86	470.96	470.01	469.02	467.97	466.87	465.72
125		480.70	479.97	479.22	478.43	477.61	476.74	475.84	474.89	473.90	472.85	471.76
130					484.14	483.35	482.52	481.66	480.75	479.80	478.81	477.77
135							488.30	487.47	486.60	485.69	484.74	483.75
140										491.57	490.66	489.71
145												495.66

Table 23: R404a Superheated vapor table (pressure 1.732MPa-2.871MPa) (Solvay Fluor)

Appendix C: Matlab Code

geometry.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cooling System Design Tool                          %
% Author: Ben Sanfiorenzo                              %
% Geometry module: Reads in excel data and user input %
% and creates the structure of the chilled water      %
% system. Provides 2D and 3D layout of CW structure.  %
% Last Modified: 5-8-13                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clc
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: Determine layout and geometry of the CW system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Conversions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ft_per_m = 3.2808399;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constants
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
g_mps2 = 9.807; %m/s^2
nu = 1.45*10^-6; %m^2/s - based on temp - assumed constant
rho = 1000; %kg/m^3 - based on temp - assumed constant
k_cw = 0.568; %W/m^2-K - based on temp - assumed constant
cp = 4203; %J/kg-K - based on temp - assumed constant

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ship's data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LOA = 143.561;                      %m (default)
beam = 20.39;                       %m (default)
eng_deck_ht_above_keel = 1.397; %m (default)
useable_ht_eng_rm = 3.098;          %m (default)
ship_data =
struct('LOA',LOA,'Beam',beam,'Engineering_deck_height_above_keel', ...

eng_deck_ht_above_keel,'Useable_height_in_engine_room',useable_ht_eng_rm);
fprintf('Note: ALL VALUES ARE IN METRIC\n\n')
fprintf('The default ship data is: \n')
ship_data
%reply = 'n';
reply = input('Would you like to modify it? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
```

```
    proceed = false;
    while ~proceed
        is_error = true;
        while is_error
            is_error = false;
            LOA = input('LOA [m]: ');
            if LOA <=0
                is_error = true;
                fprintf('Error!!! Please enter a positive number.\n')
            elseif LOA == 88888 %reset to default
                LOA = 143.561;
            end
        end
        is_error = true;
        while is_error
            is_error = false;
            beam = input('Beam [m]: ');
            if beam <=0 || ~isnumeric(beam)
                is_error = true;
                fprintf('Error!!! Please enter a positive number.\n')
            elseif beam == 88888 %reset to default
                beam = 20.39;
            end
        end
        is_error = true;
        while is_error
            is_error = false;
            eng_deck_ht_above_keel =
input('Engineering_deck_height_above_keel [m]: ');
            if eng_deck_ht_above_keel <=0 ||
~isnumeric(eng_deck_ht_above_keel)
                is_error = true;
                fprintf('Error!!! Please enter a positive number.\n')
            elseif eng_deck_ht_above_keel == 88888 %reset to default
                eng_deck_ht_above_keel = 1.397;
            end
        end
        is_error = true;
        while is_error
            is_error = false;
            useable_ht_eng_rm = input('Useable height in engine room [m]: ');
            if useable_ht_eng_rm <=0 || ~isnumeric(useable_ht_eng_rm)
                is_error = true;
                fprintf('Error!!! Please enter a positive number.\n')
            elseif useable_ht_eng_rm == 88888 %reset to default
                useable_ht_eng_rm = 3.098;
            end
        end
        fprintf('\nThe new ship data is: \n')
        ship_data =
struct('LOA',LOA,'Beam',beam,'Engine_deck_height_above_keel', ...

eng_deck_ht_above_keel,'Useable_height_in_engine_room',useable_ht_eng_rm)
        satisfactory = input('Satisfactory? [y/n]: ','s');
```

```
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Transverse bulkhead locations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The length, beam, transverse bulkhead locations would be known at this
%point. To create the CSDT independently, the code is written which asks
%for this information, but when working in conjunction with Damien's code,
%this will not be necessary
bulkhead_loc = [100 90 82.5 67.5 52.5 37.5 20 5 -10 -27.5 -43.5 -50 -80 -
100]*LOA/200; %(default)
fprintf('\nNote: Along the longitudinal axis, midships is defined as 0, the
forward\n')
fprintf('perpendicular is defined as LOA/2 and the aft perpendicular is
defined as -LOA/2.\n')
fprintf('The bulkhead_loc array also includes the FP in the first cell array
and the AP\n')
fprintf('in the last cell array.\n\n')
fprintf('The default transverse bulkhead locations are: \n')
bulkhead_loc
%reply = 'n';
reply = input('Would you like to change it? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    proceed = false;
    while ~proceed
        is_error = true;
        while is_error == true
            is_error = false;
            fprintf('Please enter the bulkhead locations from the bow to the
stern.\n')
            fprintf('Example: [75 60 40 20 5 -5 -15 -35 -50 -65 -75]\n')
            bulkhead_loc = input('Transverse bulkhead locations [m]: ');
            if length(bulkhead_loc)<2
                fprintf('Error!!! Not enough bulkhead_locations.\n')
            elseif bulkhead_loc == 88888 %reset to default
                bulkhead_loc = [100 90 82.5 67.5 52.5 37.5 20 5 -10 -27.5 -
43.5 -50 -80 -100]*LOA/200;
            else
                flag = false;
                for i=2:length(bulkhead_loc)
                    if bulkhead_loc(i)>bulkhead_loc(i-1)
                        flag = true;
                    end
                end
                if flag == true
```

```
                    is_error = true;
                    fprintf('Error!!! Bulkhead locations not entered from bow
to stern.\n')
                end
                if abs(bulkhead_loc(1) - LOA/2)>0.01*LOA ||
abs(bulkhead_loc(length(bulkhead_loc)) + LOA/2)>0.01*LOA
                    %is_error = true;
                    fprintf('Error!!! Bulkhead locations do not span the
length of the ship.\n')
                end
            end
            bulkhead_loc(1) = LOA/2;
            bulkhead_loc(length(bulkhead_loc)) = -LOA/2;
        end
        fprintf('The modified transverse bulkhead locations are: \n')
        bulkhead_loc
        satisfactory = input('Satisfactory? [y/n]: ','s');
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design CW system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('\nTo properly size and locate the piping and chiller units, the heat
load locations,\n')
fprintf('magnitude and priority (vital/non-vital) is necessary. The required
data can be \n')
fprintf('inputted into the excel spreadsheet CSDT_inputs.xlsx. If the
required data has not\n')
fprintf('been entered, please enter data now before proceeding through the
CSDT program.\n')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input File
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'CSDT_input.xlsx';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read Load Data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[num, txt] = xlsread(filename,'LoadData');
Num_Loads = num(1);
Condition_Labels = txt(11,4:7);
Load_Name = txt(13:12+Num_Loads,1);
Priority = num(6:5+Num_Loads,1); % vital loads priority 1-2; non-vital all
else
Load_Value_kW = num(6:5+Num_Loads,3:6);
Load_Value_kW(isnan(Load_Value_kW)) = 0;
Load_Loc_m = num(6:5+Num_Loads,7:9);
```

```
Hxchgr_Type = txt(13:12+Num_Loads,11);
size_num = size(num);
if size_num(2) > 9
    Hxchgr_Num = num(6:5+Num_Loads,11);
else
    Hxchgr_Num = nan*ones(1,Num_Loads);
end
clear num txt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read Hxchgr DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[num,txt] = xlsread(filename,'HXCHGR DB');
Num_CC_Types = num(1,1);
Num_50_Series_Types = num(2,1);
Num_60_Series_Types = num(3,1);
Num_Unit_Cooler_Types = num(4,1);
Num_Other_CC_Types = num(5,1);
Num_FP_Types = num(6,1);
Num_ST_Types = num(7,1);
Num_CP_Types = num(8,1);
Num_Other_Hxchgr_Types = num(9,1);
Num_Hxchgr_Types = num(10,1);

if Num_CC_Types > 0
    CC_Capacity_kW = num(16:15+Num_CC_Types,4);
    CC_hl_m = num(16:15+Num_CC_Types,5);
    CC_Area_Pri_cm2 = num(16:15+Num_CC_Types,10);
    CC_U = num(16:15+Num_CC_Types,12);
    CC_Tube_k = num(16:15+Num_CC_Types,13);
    CC_Tube_Diam_cm = num(16:15+Num_CC_Types,14);
    CC_Tube_Thick_cm = num(16:15+Num_CC_Types,15);
    CC_Area_Sec_cm2 = num(16:15+Num_CC_Types,16);
    CC_Fluid_hc = num(16:15+Num_CC_Types,17);
    CC_Fluid_Temp_In_C = num(16:15+Num_CC_Types,18);
    CC_Fluid_Mfr_kgps = num(16:15+Num_CC_Types,21);
    CC_Dim_m = [num(16:15+Num_CC_Types,22) num(16:15+Num_CC_Types,23)
num(16:15+Num_CC_Types,24)];
    CC_Weight_Dry_kg = num(16:15+Num_CC_Types,25);
    CC_Weight_Wet_kg = num(16:15+Num_CC_Types,26);
end

if Num_FP_Types > 0
    FP_Capacity_kW = num(16:15+Num_FP_Types,33);
    FP_hl_m = num(16:15+Num_FP_Types,34);
    FP_Area_cm2 = num(16:15+Num_FP_Types,39);
    FP_U = num(16:15+Num_FP_Types,41);
    FP_Plate_k = num(16:15+Num_FP_Types,42);
    FP_Plate_Thick_cm = num(16:15+Num_FP_Types,43);
    FP_Num_Gaps = num(16:15+Num_FP_Types,44);
    FP_Area_Sec_cm2 = num(16:15+Num_FP_Types,45);
    FP_Fluid_Type = txt(17:16+Num_FP_Types,48);
    FP_Fluid_cp = num(16:15+Num_FP_Types,47);
    FP_Fluid_hc = num(16:15+Num_CC_Types,48);
    FP_Fluid_Temp_In_C = num(16:15+Num_FP_Types,49);
```

```
    FP_Fluid_Mfr_kgps = num(16:15+Num_FP_Types,52);
    FP_Dim_m = [num(16:15+Num_FP_Types,53) num(16:15+Num_FP_Types,54)
num(16:15+Num_FP_Types,55)];
    FP_Weight_Dry_kg = num(16:15+Num_FP_Types,56);
    FP_Weight_Wet_kg = num(16:15+Num_FP_Types,57);
end

if Num_ST_Types > 0
    ST_Capacity_kW = num(16:15+Num_ST_Types,64);
    ST_hl_m = num(16:15+Num_ST_Types,65);
    ST_Area_cm2 = num(16:15+Num_ST_Types,70);
    ST_U = num(16:15+Num_ST_Types,72);
    ST_Tube_k = num(16:15+Num_ST_Types,73);
    ST_Tube_Diam_cm = num(16:15+Num_ST_Types,74);
    ST_Tube_Thick_cm = num(16:15+Num_ST_Types,75);
    ST_Area_Sec_cm2 = num(16:15+Num_ST_Types,76);
    ST_Fluid_Type = txt(17:16+Num_ST_Types,79);
    ST_Fluid_hc = num(16:15+Num_CC_Types,78);
    ST_Fluid_cp = num(16:15+Num_ST_Types,79);
    ST_Fluid_Temp_In_C = num(16:15+Num_ST_Types,80);
    ST_Fluid_Mfr_kgps = num(16:15+Num_ST_Types,83);
    ST_Dim_m = [num(16:15+Num_ST_Types,84) num(16:15+Num_ST_Types,85)
num(16:15+Num_ST_Types,86)];
    ST_Weight_Dry_kg = num(16:15+Num_ST_Types,87);
    ST_Weight_Wet_kg = num(16:15+Num_ST_Types,88);
end

if Num_CP_Types > 0
    CP_Capacity_kW = num(16:15+Num_CP_Types,95);
    CP_hl_m = num(16:15+Num_CP_Types,96);
    CP_Area_cm2 = num(16:15+Num_CP_Types,101);
    CP_U = num(16:15+Num_CP_Types,103);
    CP_Tube_k = num(16:15+Num_CP_Types,104);
    CP_Tube_Diam_cm = num(16:15+Num_CP_Types,105);
    CP_Tube_Thick_cm = num(16:15+Num_CP_Types,106);
    CP_Plate_k = num(16:15+Num_CP_Types,107);
    CP_Plate_Thick_cm = num(16:15+Num_CP_Types,108);
    CP_Dim_m = [num(16:15+Num_CP_Types,109) num(16:15+Num_CP_Types,110)
num(16:15+Num_CP_Types,111)];
    CP_Weight_Dry_kg = num(16:15+Num_CP_Types,112);
    CP_Weight_Wet_kg = num(16:15+Num_CP_Types,113);
end

if Num_Other_Hxchgr_Types > 0
    O_Capacity_kW = num(16:15+Num_Other_Hxchgr_Types,120);
    O_hl_m = num(16:15+Num_Other_Hxchgr_Types,121);
    O_Area_cm2 = num(16:15+Num_Other_Hxchgr_Types,126);
    O_U = num(16:15+Num_Other_Hxchgr_Types,128);
    O_Tube_k = num(16:15+Num_Other_Hxchgr_Types,129);
    O_Tube_Diam_cm = num(16:15+Num_Other_Hxchgr_Types,130);
    O_Tube_Thick_cm = num(16:15+Num_Other_Hxchgr_Types,131);
    O_Area_Sec_cm2 = num(16:15+Num_Other_Hxchgr_Types,132);
    O_Fluid_Type = txt(17:16+Num_Other_Hxchgr_Types,135);
    O_Fluid_hc = num(16:15+Num_CC_Types,134);
    O_Fluid_cp = num(16:15+Num_Other_Hxchgr_Types,135);
```

```
    O_Fluid_Temp_In_C = num(16:15+Num_Other_Hxchgr_Types,136);
    O_Fluid_Mfr_kgps = num(16:15+Num_Other_Hxchgr_Types,139);
    O_Dim_m = [num(16:15+Num_Other_Hxchgr_Types,140)
num(16:15+Num_Other_Hxchgr_Types,141) num(16:15+Num_Other_Hxchgr_Types,142)];
    O_Weight_Dry_kg = num(16:15+Num_Other_Hxchgr_Types,143);
    O_Weight_Wet_kg = num(16:15+Num_Other_Hxchgr_Types,144);
end
clear num txt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chiller DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[num,txt] = xlsread(filename,'Chiller DB');
Num_C_Chiller_Types = num(1,1); %centrifugal
Num_R_Chiller_Types = num(2,1); %reciprocating
Num_S_Chiller_Types = num(3,1); %screw
Num_O_Chiller_Types = num(4,1); %other
Num_Chiller_Types = num(5,1);

if Num_C_Chiller_Types > 0
    C_Chiller_Capacity_kW = num(11:10+Num_C_Chiller_Types,4);
    C_Chiller_Weight_kg = num(11:10+Num_C_Chiller_Types,5);
    C_Chiller_Dim_m = [num(11:10+Num_C_Chiller_Types,6)
num(11:10+Num_C_Chiller_Types,7) num(11:10+Num_C_Chiller_Types,8)];
    C_Chiller_Type = txt(13:12+Num_C_Chiller_Types,11);
    C_Chiller_P_MPa = [num(11:10+Num_C_Chiller_Types,10)
num(11:10+Num_C_Chiller_Types,12) num(11:10+Num_C_Chiller_Types,14)];
    C_Chiller_T_C = [num(11:10+Num_C_Chiller_Types,11)
num(11:10+Num_C_Chiller_Types,13) num(11:10+Num_C_Chiller_Types,15)];
    C_Chiller_Out_Temp_C = num(11:10+Num_C_Chiller_Types,16);
end

if Num_R_Chiller_Types > 0
    R_Chiller_Capacity_kW = num(11:10+Num_R_Chiller_Types,23);
    R_Chiller_Weight_kg = num(11:10+Num_R_Chiller_Types,24);
    R_Chiller_Dim_m = [num(11:10+Num_R_Chiller_Types,25)
num(11:10+Num_R_Chiller_Types,26) num(11:10+Num_R_Chiller_Types,27)];
    R_Chiller_Type = txt(13:12+Num_R_Chiller_Types,30);
    R_Chiller_P_MPa = [num(11:10+Num_R_Chiller_Types,29)
num(11:10+Num_R_Chiller_Types,31) num(11:10+Num_R_Chiller_Types,33)];
    R_Chiller_T_C = [num(11:10+Num_R_Chiller_Types,30)
num(11:10+Num_R_Chiller_Types,32) num(11:10+Num_R_Chiller_Types,34)];
    R_Chiller_Out_Temp_C = num(11:10+Num_R_Chiller_Types,35);
end

if Num_S_Chiller_Types > 0
    S_Chiller_Capacity_kW = num(11:10+Num_S_Chiller_Types,42);
    S_Chiller_Weight_kg = num(11:10+Num_S_Chiller_Types,43);
    S_Chiller_Dim_m = [num(11:10+Num_S_Chiller_Types,44)
num(11:10+Num_S_Chiller_Types,45) num(11:10+Num_S_Chiller_Types,46)];
    S_Chiller_Type = txt(13:12+Num_S_Chiller_Types,49);
    S_Chiller_P_MPa = [num(11:10+Num_S_Chiller_Types,48)
num(11:10+Num_S_Chiller_Types,50) num(11:10+Num_S_Chiller_Types,52)];
    S_Chiller_T_C = [num(11:10+Num_S_Chiller_Types,49)
num(11:10+Num_S_Chiller_Types,51) num(11:10+Num_S_Chiller_Types,53)];
```

```
    S_Chiller_Out_Temp_C = num(11:10+Num_S_Chiller_Types,54);
end

if Num_O_Chiller_Types > 0
    O_Chiller_Capacity_kW = num(11:10+Num_O_Chiller_Types,61);
    O_Chiller_Weight_kg = num(11:10+Num_O_Chiller_Types,62);
    O_Chiller_Dim_m = [num(11:10+Num_O_Chiller_Types,63)
num(11:10+Num_O_Chiller_Types,64) num(11:10+Num_O_Chiller_Types,65)];
    O_Chiller_Type = txt(13:12+Num_O_Chiller_Types,68);
    O_Chiller_P_MPa = [num(11:10+Num_O_Chiller_Types,67)
num(11:10+Num_O_Chiller_Types,69) num(11:10+Num_O_Chiller_Types,71)];
    O_Chiller_T_C = [num(11:10+Num_O_Chiller_Types,68)
num(11:10+Num_O_Chiller_Types,70) num(11:10+Num_O_Chiller_Types,72)];
    O_Chiller_Out_Temp_C = num(11:10+Num_O_Chiller_Types,73);
end
clear num txt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R134a Superheated Vapor DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = xlsread(filename,'R134a-superheated vapor');
R134a_SHV_T_C = num(2:22); %SHV temps
R134a_SHV_P_MPa = [num(1,2) num(1,3) num(1,4) num(1,5) num(1,6) num(1,7)
num(1,8) num(1,9) num(1,10)...
    num(1,11) num(1,12) num(1,13) num(1,14) num(1,15) num(1,16) num(1,17)
num(1,18) num(1,19) num(1,20) num(1,21)]; %SHV pressures
R134a_SHV_h = [num(2:22,2) num(2:22,3) num(2:22,4) num(2:22,5) num(2:22,6)
num(2:22,7) num(2:22,8) num(2:22,9) num(2:22,10)...
    num(2:22,11) num(2:22,12) num(2:22,13) num(2:22,14) num(2:22,15)
num(2:22,16) num(2:22,17) num(2:22,18) num(2:22,19) num(2:22,20)
num(2:22,21)]; %SHV enthalpies

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R134a Saturated DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = xlsread(filename,'R134a-saturated');
R134a_Sat_T_C = num(1:45); %Saturated temps
R134a_Sat_P_MPa = num(46:90); %Saturated pressures
R134a_Sat_hf = num(91:135); %Saturated enthalpies-fluid
R134a_Sat_hg = num(136:180); %Saturated enthalpies-gas

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R404a Superheated Vapor DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = xlsread(filename,'R404a-superheated vapor');
R404a_SHV_T_C = num(3:42); %SHV temps
R404a_SHV_P_MPa = [num(2,2) num(2,3) num(2,4) num(2,5) num(2,6) num(2,7)
num(2,8) num(2,9) num(2,10)...
    num(2,11) num(2,12) num(2,13) num(2,14) num(2,15) num(2,16) num(2,17)
num(2,18) num(2,19) num(2,20)...
    num(2,21) num(2,22) num(2,23) num(2,24) num(2,25) num(2,26) num(2,27)
num(2,28) num(2,29) num(2,30)...
    num(2,31) num(2,32) num(2,33) num(2,34) num(2,35) num(2,36) num(2,37)
num(2,38) num(2,39) num(2,40)...
```

```
    num(2,41) num(2,42) num(2,43) num(2,44) num(2,45) num(2,46) num(2,47)
num(2,48) num(2,49) num(2,50)...
    num(2,51) num(2,52) num(2,53) num(2,54) num(2,55) num(2,56) num(2,57)];
%SHV pressures
R404a_SHV_h = [num(3:42,2) num(3:42,3) num(3:42,4) num(3:42,5) num(3:42,6)
num(3:42,7) num(3:42,8) num(3:42,9) num(3:42,10)...
    num(3:42,11) num(3:42,12) num(3:42,13) num(3:42,14) num(3:42,15)
num(3:42,16) num(3:42,17) num(3:42,18) num(3:42,19) num(3:42,20)...
    num(3:42,21) num(3:42,22) num(3:42,23) num(3:42,24) num(3:42,25)
num(3:42,26) num(3:42,27) num(3:42,28) num(3:42,29) num(3:42,30)...
    num(3:42,31) num(3:42,32) num(3:42,33) num(3:42,34) num(3:42,35)
num(3:42,36) num(3:42,37) num(3:42,38) num(3:42,39) num(3:42,40)...
    num(3:42,41) num(3:42,42) num(3:42,43) num(3:42,44) num(3:42,45)
num(3:42,46) num(3:42,47) num(3:42,48) num(3:42,49) num(3:42,50)...
    num(3:42,51) num(3:42,52) num(3:42,53) num(3:42,54) num(3:42,55)
num(3:42,56) num(3:42,57)]; %SHV enthalpies

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R404a Saturated DB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = xlsread(filename,'R404a-saturated');
R404a_Sat_T_C = num(1:129); %Saturated temps
R404a_Sat_Pf_MPa = num(130:258); %Saturated pressures
R404a_Sat_Pg_MPa = num(259:387); %Saturated pressures
R404a_Sat_hf = num(388:516); %Saturated enthalpies-fluid
R404a_Sat_hg = num(517:645); %Saturated enthalpies-gas

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in pump curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = xlsread(filename,'PumpData');
Num_Pumps = num(1); %number of different pump curves in pump series 1510
Bell&Gosset
Pump_Mfr = zeros(Num_Pumps,4);
Pump_Head = zeros(Num_Pumps,4);
for i=1:Num_Pumps
    Pump_Mfr(i,:) = num(7+Num_Pumps*16+i*4:7+Num_Pumps*16+i*4+3);
    Pump_Head(i,:) = num(9+Num_Pumps*20+i*4:9+Num_Pumps*20+i*4+3);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main piping configuration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('\nThe chilled water system can be configured either using a single
main piping\n')
fprintf('system or a double main piping system. The single main piping system
will often\n')
fprintf('be cheaper, but offers less in terms of survivability. Single main
piping systems\n')
fprintf('are typically used for auxiliary ships or small combatants. Double
main piping\n')
fprintf('systems are generally used for large combatants. In addition, for
double main\n')
fprintf('piping systems, the loop could be simple, with few bends, or more
complex, with many\n')
```

```
fprintf('bends. A few generic examples are provided through the use of the
pop-up menu.\n')
%piping_config = 2;
piping_config = menu('Select the main piping configuration','Single
main','Double main');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define location of header piping for single main CW system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if piping_config == 1
    header_deck_ht = 5.2; %m (default)
    fprintf('\nThe default main piping height is: %4.2f m\n', header_deck_ht)
    reply = input('Would you like to change it? [y/n]: ','s');
    if isempty(reply)
        reply = 'y';
    end
    if reply == 'y' || reply == 'Y'
        proceed = false;
        while ~proceed
            header_deck_ht = input('Main piping height [m]: ');
            satisfactory = input('Satisfactory? [y/n]: ','s');
            if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
                proceed = true;
            else
                proceed = false;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define location of port and starboard header piping for double main CW
system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if piping_config == 2
    %piping_double_config = 1;
    piping_double_config = menu('Select a simple double main piping loop or a
double main with multiple bends',...
        'Simple loop','Multiple bends');
    port_header_deck_ht = 5.2; %m (default)
    stbd_header_deck_ht = 10.2; %m (default)
    fprintf('\nFor a double main system, proper separation of the main piping
is essential\n')
    fprintf('for survivability. Vertical separation of 1-2 decks is
recommended with one\n')
    fprintf('of the main piping systems on the damage control deck.\n')
    fprintf('The port and starboard main piping heights are %4.2f m and %4.2f
m, respectively\n', ...
        port_header_deck_ht,stbd_header_deck_ht)
    %reply = 'n';
    reply = input('Would you like to change them? [y/n]: ','s');
    if isempty(reply)
        reply = 'y';
    end
```

```
    if reply == 'y' || reply == 'Y'
        proceed = false;
        while ~proceed
            port_header_deck_ht = input('Port main height [m]: ');
            stbd_header_deck_ht = input('Starboard main height [m]: ');
            satisfactory = input('Satisfactory? [y/n]: ','s');
            if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
                proceed = true;
            else
                proceed = false;
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Define where bends occur for double main configuration with
    % multiple bends
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if piping_double_config == 2
        fprintf('\nThe main piping should be within 3 feet of the hull,
except for curved sections\n')
        fprintf('of the hull which allows a maximum distance of 8 feet (with
exemptions granted for\n')
        fprintf('situations in which freezing of the pipes could occur).\n')
        header_bends = [LOA/2-3 beam/2-0.7*beam/2;
            LOA/2-0.075*LOA beam/2-0.5*beam/2;
            LOA/2-0.1*LOA beam/2-0.3*beam/2;
            LOA/2-0.2*LOA beam/2-0.25*beam/2;
            LOA/2-0.3*LOA beam/2-0.2*beam/2;
            LOA/2-0.35*LOA beam/2-3/ft_per_m;
            -(LOA/2-0.05*LOA) beam/2-0.15*beam/2;
            LOA/2-3 -beam/2+0.7*beam/2;
            LOA/2-0.075*LOA -beam/2+0.5*beam/2;
            LOA/2-0.1*LOA -beam/2+0.3*beam/2;
            LOA/2-0.2*LOA -beam/2+0.25*beam/2;
            LOA/2-0.3*LOA -beam/2+0.2*beam/2;
            LOA/2-0.35*LOA -beam/2+3/ft_per_m;
            -(LOA/2-0.05*LOA) -beam/2+0.15*beam/2];
        fprintf('The default main piping bend locations are:\n')
        header_bends
        %reply = 'n';
        reply = input('\nWould you like to change them? [y/n]: ','s');
        if isempty(reply)
            reply = 'y';
        end
        if reply == 'y' || reply == 'Y'
            reply = input('Are the bend locations symmetric port and
starboard? [y/n]: ','s');
            if isempty(reply)
                reply = 'y';
            end
            if reply == 'y' || reply == 'Y'
                proceed = false;
                while ~proceed
```

```
                    is_error=true;
                    while is_error
                        is_error = false;
                        fprintf('Please enter the bend locations starting
from centerline forward and continuing counter-clockwise until centerline
aft.\n')
                        fprintf('Example: [20 5;19 6; 14 10; 8 12; -17 10; -
19 7; -20 4]\n')
                        test1 = input('Bend locations: ');
                        temp_var = max(abs(test1));
                        temp_var_2 = size(test1);
                        if temp_var(1)>LOA/2
                            fprintf('Error!!! Bend location exceeds ship
length\n')
                            is_error = true;
                        end
                        if temp_var(2)>beam/2
                            fprintf('Error!!! Bend location exceeds ship
beam\n')
                            is_error = true;
                        end
                        if temp_var_2(1)<2
                            fprintf('Error!!! Not enough bends\n')
                            is_error = true;
                        elseif temp_var_2(2)>2
                            fprintf('Error!!! Only include x and y bend
locations\n')
                            is_error = true;
                        end
                    end
                    header_bends = test1; %passes error check
                    for i=length(test1)+1:length(test1)*2
                        header_bends(i,1) = test1(i-length(test1),1);
                        header_bends(i,2) = -test1(i-length(test1),2);
                    end
                    fprintf('The new main piping bend locations are:\n')
                    header_bends
                    satisfactory = input('Satisfactory? [y/n]: ','s');
                    if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y')
|| strcmp(satisfactory,'yes')
                        proceed = true;
                    else
                        proceed = false;
                    end
                end
            else
                proceed = false;
                while ~proceed
                    is_error=true;
                    while is_error
                        is_error = false;
                        fprintf('Please enter the bend locations starting
from centerline forward and continuing counter-clockwise until centerline
forward.\n')
```

```
                        fprintf('Example: [20 5;15 6; 10 8; 8 10; -15 9;-20
5;-19 -5;-17 -6; -14 -7.2; 8 -7; 15 -4.8; 20 -4.2]\n')
                        test1 = input('Bend locations: ');
                        temp_var = max(abs(test1));
                        temp_var_2 = size(test1);
                        if temp_var(1)>LOA/2
                            fprintf('Error!!! Bend location exceeds ship
length\n')
                            is_error = true;
                        end
                        if temp_var(2)>beam/2
                            fprintf('Error!!! Bend location exceeds ship
beam\n')
                            is_error = true;
                        end
                        if temp_var_2(1)<2
                            fprintf('Error!!! Not enough bends\n')
                            is_error = true;
                        elseif temp_var_2(2)>2
                            fprintf('Error!!! Only include x and y bend
locations\n')
                            is_error = true;
                        end
                    end
                    header_bends = test1; %passes error check
                    fprintf('The new main piping bend locations are:\n')
                    header_bends
                    satisfactory = input('Satisfactory? [y/n]: ','s');
                    if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y')
|| strcmp(satisfactory,'yes')
                        proceed = true;
                    else
                        proceed = false;
                    end
                end
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define piping offsets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
offset_h = 0.5; %offset between supply and return header in the x,y,z
direction (default)
fprintf('\nThe default offset distance between the supply and return header
is: %4.2f m\n', offset_h)
%reply = 'n';
reply = input('Would you like to change it? [y/n]: ','s');
if reply == 'y' || reply == 'Y'
    proceed = false;
    while ~proceed
        offset_h = input('Supply and return header offset distance [m]: ');
        satisfactory = input('Satisfactory? [y/n]: ','s');
```

```
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
        end
    end
end
offset_b = 0.1; %offset between branch inlet and outlet in the x,y,z
direction (default)
fprintf('\nThe default offset distance between the branch inlet and outlet
is: %4.2f m\n', offset_b)
%reply = 'n';
reply = input('Would you like to change it? [y/n]: ','s');
if reply == 'y' || reply == 'Y'
    proceed = false;
    while ~proceed
        offset_b = input('Branch inlet and outlet offset distance [m]: ');
        satisfactory = input('Satisfactory? [y/n]: ','s');
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine zonal configuration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
zones = 3; %(default)
fprintf('\nThe cooling loads are broken up into zones along the length of the
ship,\n')
fprintf('with the ability of a zone to be isolated from the rest of the
cooling system.\n')
fprintf('The greater the number of zones, the more survivable the ship is,
but cost, \n')
fprintf('weight and space required go up. The minimum number of zones is 2.
The number of\n')
fprintf('zones also should not exceed the number of compartments (but
generally is much fewer).\n\n')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define total heat load within each compartment
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Load_Value_1 = zeros(1,length(bulkhead_loc)-1);
Load_Loc_1 = zeros(1,length(bulkhead_loc)-1);
for i=1:length(bulkhead_loc)-1
    for j=1:Num_Loads
        if Load_Loc_m(j,1) <= bulkhead_loc(i) && Load_Loc_m(j,1) >
bulkhead_loc(i+1)
            Load_Value_1(i) = Load_Value_1(i)+Load_Value_kW(j,2);
        elseif Load_Loc_m(j,1) > bulkhead_loc(1) && i==1
            Load_Value_1(i) = Load_Value_1(i)+Load_Value_kW(j,2);
```

```
        elseif Load_Loc_m(j,1) < bulkhead_loc(length(bulkhead_loc)) &&
i==length(bulkhead_loc)-1
            Load_Value_1(i) = Load_Value_1(i)+Load_Value_kW(j,2);
        end
    end
    Load_Loc_1(i) = (bulkhead_loc(i+1)-bulkhead_loc(i))/2+bulkhead_loc(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define zonal boundaries (default # of zones and default zonal
% boundaries)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
zonal_boundaries = zeros(length(zones));
zonal_length = LOA/zones; %m space zones equidistant (default)
for i=1:zones
    zonal_boundaries(i) = LOA/2 - i*zonal_length;
end
aft_bkhd_1 = zeros(1,length(zones));
for i=1:zones
    %find aft most bulkhead in zone
    for j=2:length(bulkhead_loc)
        if bulkhead_loc(j)>=zonal_boundaries(i)
            aft_bkhd_1(i) = bulkhead_loc(j);
        end
    end
end
zonal_boundaries = aft_bkhd_1;
for i=length(zonal_boundaries)+1:-1:2
    zonal_boundaries(i)=zonal_boundaries(i-1);
end
zonal_boundaries(1)=LOA/2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define total heat load within each zone (default # of zones
% and default zonal boundaries)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Load_Value_2 = zeros(1,length(zonal_boundaries)-1);
Load_Loc_2 = zeros(1,length(zonal_boundaries)-1);
for i=1:length(zonal_boundaries)-1
    for j=1:Num_Loads
        if Load_Loc_m(j,1) <= zonal_boundaries(i) && Load_Loc_m(j,1) >
zonal_boundaries(i+1)
            Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
        elseif Load_Loc_m(j,1) > zonal_boundaries(1) && i==1
            Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
        elseif Load_Loc_m(j,1) < zonal_boundaries(length(zonal_boundaries))
&& i==length(zonal_boundaries)-1
            Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
        end
    end
    Load_Loc_2(i) = (zonal_boundaries(i+1)-
zonal_boundaries(i))/2+zonal_boundaries(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Plot total heat load within each compartment and each zone
% (default # of zones and default zonal boundaries)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Figure 1 shows the transverse bulkheads (blue lines), zonal
boundaries (red dotted lines)\n')
fprintf('and the total heat load within each compartment and within each
zone.\n\n')
ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
figure(1)
subplot(3,1,1)
plot(ship_vec(1,:),ship_vec(2,:))
hold on
for i=2:(length(bulkhead_loc)-1)
    plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1])
end
zonal_boundaries = aft_bkhd_1;
plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -1 1
1],'r:')
for i=2:zones
    plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
end
axis equal
axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
xlabel('Longitudinal Axis')
ylabel('Transverse Axis')
title('2D layout')
subplot(3,1,2)
bar(Load_Loc_1,Load_Value_1)
xlabel('Longitudinal Axis')
ylabel('Heat Load (kW)')
title('Heat load per compartment')
subplot(3,1,3)
bar(Load_Loc_2,Load_Value_2)
xlabel('Longitudinal Axis')
ylabel('Heat Load (kW)')
title('Heat load per zone')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check to see if # of zones is sufficient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('There are currently %1.0f zones and %1.0f
compartments.\n',zones,length(bulkhead_loc)-1)
%reply = 'n';
reply = input('Would you like to change the number of zones? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% # of zones not sufficient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if reply == 'y' || reply == 'Y'
    proceed = false;
```

```
    while ~proceed
        is_error = true;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Get new number of zones
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        while is_error
            is_error = false;
            zones = input('Number of zones: ');
            if zones < 2
                is_error = true;
                fprintf('Error!!! The minimum number of zones is 2.\n')
            end
            if (zones-floor(zones))~=0
                is_error = true;
                fprintf('Error!!! Only integers are allowed for the number of
zones.\n')
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Modify zonal boundaries
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        zonal_boundaries = zeros(length(zones));
        zonal_length = LOA/zones; %m space zones equidistant (default)
        for i=1:zones
            zonal_boundaries(i) = LOA/2 - i*zonal_length;
        end
        aft_bkhd_1 = zeros(1,zones);
        for i=1:zones
            %find aft most bulkhead in zone
            for j=2:length(bulkhead_loc)
                if bulkhead_loc(j)>=zonal_boundaries(i)
                    aft_bkhd_1(i) = bulkhead_loc(j);
                end
            end
        end
        zonal_boundaries = aft_bkhd_1;
        for i=length(zonal_boundaries)+1:-1:2
            zonal_boundaries(i)=zonal_boundaries(i-1);
        end
        zonal_boundaries(1)=LOA/2;

        fprintf('\nIt is ideal to space the zones equally along the length of
the ship for\n')
        fprintf('survivability considerations or by heat load per zone for
comparably sized\n')
        fprintf('chillers in each zone. Also, zones should terminate at a
transverse\n')
        fprintf('bulkhead.\n')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Define total heat load within each zone (user defined # of zones
        % and default zonal boundaries)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Load_Value_2 = zeros(1,length(zonal_boundaries)-1);
```

```
        Load_Loc_2 = zeros(1,length(zonal_boundaries)-1);
        for i=1:length(zonal_boundaries)-1
            for j=1:Num_Loads
                if Load_Loc_m(j,1) <= zonal_boundaries(i) && Load_Loc_m(j,1)
> zonal_boundaries(i+1)
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                elseif Load_Loc_m(j,1) > zonal_boundaries(1) && i==1
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                elseif Load_Loc_m(j,1) <
zonal_boundaries(length(zonal_boundaries)) && i==length(zonal_boundaries)-1
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                end
            end
            Load_Loc_2(i) = (zonal_boundaries(i+1)-
zonal_boundaries(i))/2+zonal_boundaries(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % plot total heat load within each compartment and each zone (user
        % defined # of zones and default zonal boundaries)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        fprintf('\nFigure 1 shows the transverse bulkheads, zonal boundaries
and the\n')
        fprintf('total heat load within each compartment and within each
zone.\n')
        ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
        figure(1)
        subplot(3,1,1)
        plot(ship_vec(1,:),ship_vec(2,:))
        hold on
        for i=2:(length(bulkhead_loc)-1)
            plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1])
        end
        zonal_boundaries = aft_bkhd_1;
        plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -
1 1 1],'r:')
        for i=2:zones
            plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
        end
        axis equal
        axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
        xlabel('Longitudinal Axis')
        ylabel('Transverse Axis')
        title('2D layout')
        subplot(3,1,2)
        bar(Load_Loc_1,Load_Value_1)
        xlabel('Longitudinal Axis')
        ylabel('Heat Load (kW)')
        title('Heat load per compartment')
        subplot(3,1,3)
        bar(Load_Loc_2,Load_Value_2)
        xlabel('Longitudinal Axis')
        ylabel('Heat Load (kW)')
```

```
        title('Heat load per zone')

        satisfactory = input('Is the number of zones satisfactory? [y/n]:
','s')';
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check to see if the zonal boundaries are sufficient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('\nThe bulkhead locations are:\n')
bulkhead_loc
fprintf('\nThe default aft most locations for each zone are:\n')
zonal_boundaries
%reply = 'n';
reply = input('Would you like to change them? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Zonal boundaries are not sufficient. Redefine zonal boundaries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if reply == 'y' || reply == 'Y'
    proceed = false;
    while ~proceed
        is_error = true;
        while is_error
            is_error = false;
            fprintf('\nNote: Each zone must include at least one compartment
large enough to fit a chiller.\n')
            fprintf('Please enter the aft most location in each zone starting
from the bow to the stern.\n')
            fprintf('Example: [20 -25 -75]\n')
            zonal_boundaries = input('Zonal boundary locations: ');
            if zones>length(zonal_boundaries)
                is_error = true;
                fprintf('Error!!! Not enough zonal boundaries.\n')
            end
            if zones<length(zonal_boundaries)
                is_error = true;
                fprintf('Error!!! Too many zonal boundaries.\n')
            end
            flag = false;
            for i=2:length(zonal_boundaries)
                if zonal_boundaries(i)>zonal_boundaries(i-1)
                    flag = true;
                end
            end
```

```
            if flag == true
                is_error = true;
                fprintf('Error!!! Zonal bondaries not ordered from bow to
stern.\n')
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Define zonal boundaries (user defined # of zones and user
        % defined zonal boundaries)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        aft_bkhd_1 = zonal_boundaries;
        for i=length(zonal_boundaries)+1:-1:2
            zonal_boundaries(i)=zonal_boundaries(i-1);
        end
        zonal_boundaries(1)=LOA/2;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Define total heat load within each zone (user defined # of
        % zones and user defined zonal boundaries)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Load_Value_2 = zeros(1,length(zonal_boundaries)-1);
        Load_Loc_2 = zeros(1,length(zonal_boundaries)-1);
        for i=1:length(zonal_boundaries)-1
            for j=1:Num_Loads
                if Load_Loc_m(j,1) <= zonal_boundaries(i) && Load_Loc_m(j,1)
> zonal_boundaries(i+1)
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                elseif Load_Loc_m(j,1) > zonal_boundaries(1) && i==1
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                elseif Load_Loc_m(j,1) <
zonal_boundaries(length(zonal_boundaries)) && i==length(zonal_boundaries)-1
                    Load_Value_2(i) = Load_Value_2(i)+Load_Value_kW(j,2);
                end
            end
            Load_Loc_2(i) = (zonal_boundaries(i+1)-
zonal_boundaries(i))/2+zonal_boundaries(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Plot total heat load within each compartment and each zone
        % (user defined # of zones and user defined zonal boundaries)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        fprintf('Figure 1 shows the transverse bulkheads, zonal boundaries
and the\n')
        fprintf('total heat load within each compartment and within each
zone.\n')
        ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
        figure(1)
        subplot(3,1,1)
        plot(ship_vec(1,:),ship_vec(2,:))
        hold on
        for i=2:(length(bulkhead_loc)-1)
            plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1])
```

```
        end
        zonal_boundaries = aft_bkhd_1;
        plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -
1 1 1],'r:')
        for i=2:zones
            plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
        end
        axis equal
        axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
        xlabel('Longitudinal Axis')
        ylabel('Transverse Axis')
        title('2D layout')
        subplot(3,1,2)
        bar(Load_Loc_1,Load_Value_1)
        xlabel('Longitudinal Axis')
        ylabel('Heat Load (kW)')
        title('Heat load per compartment')
        subplot(3,1,3)
        bar(Load_Loc_2,Load_Value_2)
        xlabel('Longitudinal Axis')
        ylabel('Heat Load (kW)')
        title('Heat load per zone')

        satisfactory = input('Are the zonal boundaries satisfactory? [y/n]:
','s')';
        if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
            proceed = true;
        else
            proceed = false;
            fprintf('\nThe bulkhead locations are:\n')
            bulkhead_loc
            fprintf('\nThe current aft most locations for each zone are:\n')
            zonal_boundaries
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% chiller configuration inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('\nEach zone must have the capability of operating independently.
This\n')
fprintf('corresponds to having at least one chiller in each zone.\n')
if piping_config == 1 %single main
    chillers = ones(1,zones); %number of chillers per zone fwd->aft (default)
    fprintf('The default is one chiller per zone for the single main piping
system.\n')
    reply = 'n';
    %reply = input('Would you like to change this? [y/n]: ','s');
    if isempty(reply)
        reply = 'y';
    end
    if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
```

```
        proceed = false;
        while ~proceed
            is_error = true;
            while is_error
                is_error = false;
                fprintf('Please enter the number of chillers per zone
starting from the bow and progressing towards the stern.\n')
                fprintf('Example for 3 zones: [1 2 2]\n')
                chillers = input('Number of chillers per zone: ');
                if (min(chillers)<=0) || length(chillers)<zones
                    is_error = true;
                    fprintf('Error!!! Please enter at least one chiller per
zone.\n')
                end
                flag = false;
                for i=1:length(chillers)
                    if (chillers(i)-floor(chillers(i)))~=0
                        flag = true;
                    end
                end
                if flag == true;
                    is_error = true;
                    fprintf('Error!!! Please enter at least one chiller per
zone.\n')
                end
            end
            satisfactory = input('Satisfactory? [y/n]: ','s')';
            if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
                proceed = true;
            else
                proceed = false;
            end
        end
    end
elseif piping_config == 2 %double main
    chillers = 2*ones(1,zones); %number of chillers per zone fwd->aft
(default)
    fprintf('The default is two chillers per zone for the double main piping
system (one per main per zone).\n')
    reply = 'n';
    %reply = input('Would you like to change this? [y/n]: ','s');
    if isempty(reply)
        reply = 'y';
    end
    if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
        proceed = false;
        while ~proceed
            is_error = true;
            while is_error
                is_error = false;
                fprintf('Please enter the number of chillers per zone
starting from the bow and progressing towards the stern.\n')
                fprintf('Example for 3 zones: [1 2 2]\n')
                chillers = input('Number of chillers per zone: ');
```

```
                if (min(chillers)<=1) || length(chillers)<zones
                    is_error = true;
                    fprintf('Error!!! Please enter at least two chillers per
zone.\n')
                end
                flag = false;
                for i=1:length(chillers)
                    if (chillers(i)-floor(chillers(i)))~=0
                        flag = true;
                    end
                end
                if flag == true;
                    is_error = true;
                    fprintf('Error!!! Please enter at least two chillers per
zone.\n')
                end
            end
            satisfactory = input('Satisfactory? [y/n]: ','s')';
            if strcmp(satisfactory,'y') || strcmp(satisfactory,'Y') ||
strcmp(satisfactory,'yes')
                proceed = true;
            else
                proceed = false;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% define 2D and 3D vectors used for plotting squares and cubes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
two_D_x = [1 1 -1 -1 1];
two_D_y = [1 -1 -1 1 1];
three_D_x = [1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
three_D_y = [1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
three_D_z = [-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine what type of chiller is to be used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Currently, the chiller (AC unit) types available are:\n')
if Num_C_Chiller_Types > 0
    fprintf('     Centrifugal\n')
end
if Num_R_Chiller_Types > 0
    fprintf('     Reciprocating\n')
end
if Num_S_Chiller_Types > 0
    fprintf('     Screw\n')
end
if Num_O_Chiller_Types > 0
    fprintf('     Other\n')
end

chiller_type = 'd'; %default - considers all chillers, independent of type
```

```
%reply = 'n';
reply = input('Would you like to select which specific type of chiller is
used in the Chilled Water system? [y/n]: ','s');
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('Please select the chiller type from the pop-up menu\n')
    if Num_C_Chiller_Types > 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller
type','Centrifugal','Reiprocating','Screw','Other');
        if reply == 1
            chiller_type = 'c';
        elseif reply == 2
            chiler_type = 'r';
        elseif reply == 3
            chiller_type = 's';
        else
            chiller_type = 'o';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller
type','Centrifugal','Reiprocating','Screw');
        if reply == 1
            chiller_type = 'c';
        elseif reply == 2
            chiler_type = 'r';
        else
            chiller_type = 's';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller
type','Centrifugal','Reiprocating','Other');
        if reply == 1
            chiller_type = 'c';
        elseif reply == 2
            chiler_type = 'r';
        else
            chiller_type = 'o';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Centrifugal','Screw','Other');
        if reply == 1
            chiller_type = 'c';
        elseif reply == 2
            chiler_type = 's';
        else
            chiller_type = 'o';
        end
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Reiprocating','Screw','Other');
        if reply == 1
            chiller_type = 'r';
```

```
        elseif reply == 2
            chiler_type = 's';
        else
            chiller_type = 'o';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller type','Centrifugal','Reiprocating');
        if reply == 1
            chiller_type = 'c';
        else
            chiler_type = 'r';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller type','Centrifugal','Screw');
        if reply == 1
            chiller_type = 'c';
        else
            chiler_type = 's';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Centrifugal','Other');
        if reply == 1
            chiller_type = 'c';
        else
            chiler_type = 'o';
        end
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller type','Reiprocating','Screw');
        if reply == 1
            chiller_type = 'r';
        else
            chiler_type = 's';
        end
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Reiprocating','Other');
        if reply == 1
            chiller_type = 'r';
        else
            chiler_type = 'o';
        end
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Screw','Other');
        if reply == 1
            chiller_type = 's';
        else
            chiler_type = 'o';
        end
    elseif Num_C_Chiller_Types > 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types == 0
```

```
        reply = menu('Select a chiller type','Centrifugal');
        chiller_type = 'c';
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types > 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller type','Reciprocating');
        chiller_type = 'r';
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types > 0 && Num_O_Chiller_Types == 0
        reply = menu('Select a chiller type','Screw');
        chiller_type = 's';
    elseif Num_C_Chiller_Types == 0 && Num_R_Chiller_Types == 0 &&
Num_S_Chiller_Types == 0 && Num_O_Chiller_Types > 0
        reply = menu('Select a chiller type','Other');
        chiller_type = 'o';
    else
        fprintf('Error!!! No chillers in the database!!!\n')
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Guess at chiller dimensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Chiller_min_capacity_guess = max(sum(Load_Value_kW)/sum(chillers));
Chiller_capacity_guess = 1000000000;
if chiller_type == 'c'
    for i=1:Num_C_Chiller_Types
        if C_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
C_Chiller_Capacity_kW(i) < Chiller_capacity_guess
            Chiller_capacity_guess = C_Chiller_Capacity_kW(i);
            chiller_dim = C_Chiller_Dim_m(i,:);
        end
    end
elseif chiller_type == 'r'
    for i=1:Num_R_Chiller_Types
        if R_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
R_Chiller_Capacity_kW(i) < Chiller_capacity_guess
            Chiller_capacity_guess = R_Chiller_Capacity_kW(i);
            chiller_dim = R_Chiller_Dim_m(i,:);
        end
    end
elseif chiller_type == 's'
    for i=1:Num_S_Chiller_Types
        if S_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
S_Chiller_Capacity_kW(i) < Chiller_capacity_guess
            Chiller_capacity_guess = S_Chiller_Capacity_kW(i);
            chiller_dim = S_Chiller_Dim_m(i,:);
        end
    end
elseif chiller_type == 'o'
    for i=1:Num_O_Chiller_Types
        if O_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
O_Chiller_Capacity_kW(i) < Chiller_capacity_guess
            Chiller_capacity_guess = O_Chiller_Capacity_kW(i);
            chiller_dim = O_Chiller_Dim_m(i,:);
        end
```

```
        end
    elseif chiller_type == 'd'
        if Num_C_Chiller_Types > 0
            for i=1:Num_C_Chiller_Types
                if C_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
C_Chiller_Capacity_kW(i) < Chiller_capacity_guess
                    Chiller_capacity_guess = C_Chiller_Capacity_kW(i);
                    chiller_dim = C_Chiller_Dim_m(i,:);
                end
            end
        end
        if Num_R_Chiller_Types > 0
            for i=1:Num_R_Chiller_Types
                if R_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
R_Chiller_Capacity_kW(i) < Chiller_capacity_guess
                    Chiller_capacity_guess = R_Chiller_Capacity_kW(i);
                    chiller_dim = R_Chiller_Dim_m(i,:);
                end
            end
        end
        if Num_S_Chiller_Types > 0
            for i=1:Num_S_Chiller_Types
                if S_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
S_Chiller_Capacity_kW(i) < Chiller_capacity_guess
                    Chiller_capacity_guess = S_Chiller_Capacity_kW(i);
                    chiller_dim = S_Chiller_Dim_m(i,:);
                end
            end
        end
        if Num_O_Chiller_Types > 0
            for i=1:Num_O_Chiller_Types
                if O_Chiller_Capacity_kW(i) >= Chiller_min_capacity_guess &&
O_Chiller_Capacity_kW(i) < Chiller_capacity_guess
                    Chiller_capacity_guess = O_Chiller_Capacity_kW(i);
                    chiller_dim = O_Chiller_Dim_m(i,:);
                end
            end
        end
    end
    pump_dim = [1 1 1]; %m guess (default)
    min_dist = chiller_dim(1) + 3;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine chiller location longitudinally
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    compartments = zeros(1,zones);
    aft_bkhd = zeros(1,zones);
    aft_bkhd_sec = zeros(1,zones);
    for i=1:zones
        count = 0;
        %find aft most bulkhead in zone
        for j=2:length(bulkhead_loc)
            if (bulkhead_loc(j-1)-bulkhead_loc(j)) >= min_dist %minimum distance
between bulkheads that would fit chiller
                if bulkhead_loc(j)>=zonal_boundaries(i)-0.0001
```

```
                aft_bkhd(i) = bulkhead_loc(j);
                compartments(i) = compartments(i)+1;
                if count > 0
                    aft_bkhd_sec(i) = bulkhead_loc(count);
                    count = j;
                else
                    aft_bkhd_sec(i) = 12345;
                    count = j;
                end
            end
        end
    end
end
for i=zones:-1:2
    compartments(i)=compartments(i)-compartments(i-1);
end
chiller_loc = zeros(sum(chillers),3);
pump_loc = zeros(sum(chillers),3);
chiller_index = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine chiller and pump location for the case of 1-6
% chillers per zone. Need to change this section to incorporate
% chiller-pump combinations. Assume only 1-1 for now.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:zones
    if chillers(i) == 1
        if i==zones
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+3+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];
        else
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];
        end

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
        chiller_index=chiller_index+1;
    elseif chillers(i) == 2
        if i==zones
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+3+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];
            chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+3+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];
        else
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];
            chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];
        end

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
```

```
pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_index=chiller_index+2;
    elseif chillers(i) == 3
        chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
        chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_loc(chiller_index+2,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+2,:)=chiller_loc(chiller_index+2,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_index=chiller_index+3;
    elseif chillers(i) == 4
        chiller_loc(chiller_index+2,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+2,:)=chiller_loc(chiller_index+2,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_loc(chiller_index+3,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+3,:)=chiller_loc(chiller_index+3,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        if aft_bkhd_sec(i)~=12345;
            chiller_loc(chiller_index,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];

chiller_loc(chiller_index+1,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2 -beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        else
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
            chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
```

```
        end
        chiller_index=chiller_index+4;
    elseif chillers(i) == 5
        chiller_loc(chiller_index+2,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+2,:)=chiller_loc(chiller_index+2,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_loc(chiller_index+3,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+3,:)=chiller_loc(chiller_index+3,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_loc(chiller_index+4,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+4,:)=chiller_loc(chiller_index+4,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        if aft_bkhd_sec(i)~=12345;
            chiller_loc(chiller_index,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];

chiller_loc(chiller_index+1,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2 -beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        else
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
            chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        end
        chiller_index=chiller_index+5;
    elseif chillers(i) == 6
        chiller_loc(chiller_index+3,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+3,:)=chiller_loc(chiller_index+3,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        chiller_loc(chiller_index+4,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+4,:)=chiller_loc(chiller_index+4,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
```

```
        chiller_loc(chiller_index+5,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+5,:)=chiller_loc(chiller_index+5,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        if aft_bkhd_sec(i)~=12345;
            chiller_loc(chiller_index,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];

chiller_loc(chiller_index+1,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];

chiller_loc(chiller_index+2,:)=[aft_bkhd_sec(i)+1+chiller_dim(1)/2 -beam/4
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+2,:)=chiller_loc(chiller_index+2,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        else
            chiller_loc(chiller_index,:)=[aft_bkhd(i)+1+chiller_dim(1)/2
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index,:)=chiller_loc(chiller_index,:)+[chiller_dim(1)/2 0
0]+[1 0 0];
            chiller_loc(chiller_index+1,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 0
chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+1,:)=chiller_loc(chiller_index+1,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
            chiller_loc(chiller_index+2,:)=[aft_bkhd(i)+1+chiller_dim(1)/2 -
beam/4 chiller_dim(3)/2+eng_deck_ht_above_keel];

pump_loc(chiller_index+2,:)=chiller_loc(chiller_index+2,:)+[chiller_dim(1)/2
0 0]+[1 0 0];
        end
        chiller_index=chiller_index+6;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot 2D layout
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('The zones, transverse bulkheads and the default chiller\n')
fprintf('locations are shown in figure 2\n\n')
ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
chiller_vec = [chiller_dim(1)/2*[1 1 -1 -1 1]; chiller_dim(2)/2*[1 -1 -1 1
1]];
figure(2)
plot(ship_vec(1,:),ship_vec(2,:))
```

```
hold on
for i=1:sum(chillers)

plot(chiller_vec(1,:)+chiller_loc(i,1),chiller_vec(2,:)+chiller_loc(i,2),'g')
end
for i=2:(length(bulkhead_loc)-1)
    plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1])
end
plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -1 1
1],'r:')
for i=2:zones
    plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
end
scatter(pump_loc(:,1),pump_loc(:,2),'ch')
axis equal
axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
xlabel('Longitudinal Axis')
ylabel('Transverse Axis')
title('2D Chiller Layout')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine if chiller locations are to be modified
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%reply = 'n';
reply = input('Would you like to change the chiller locations? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('\nThe chiller locations are listed from forward to aft and from
port to starboard.\n')
    fprintf('The current chiller locations are: \n')
    chiller_loc
    fprintf('The transverse bulkhead locations are: \n')
    bulkhead_loc
    fprintf('Please enter the revised chiller locations from forward to aft
with the location\n')
    fprintf('corresponding to the center of the chiller.\n')
    fprintf('Example: [40 3 2;40 -3 2;5 0 2;-30 0 2;-68.5 0 2]\n')
    is_error = true;
    while is_error
        is_error = false;
        chiller_loc = input('Chiller locations [m]: ');
        if length(chiller_loc)~=sum(chillers)
            is_error = true;
            fprintf('Error!!! Please enter the locations for each
chiller.\n')
        end
        temp1 = max(chiller_loc);
        temp2 = min(chiller_loc);
        if temp1(1)>LOA/2 || temp2(1)<-LOA/2 || temp1(2)>beam/2 || temp2(2)<-
beam/2
            is_error = true;
```

```
            fprintf('Error!!! Please enter chiller locations within the
boundary of the hull.\n')
        end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Update pump locations
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:length(chiller_loc)
        pump_loc(i,:)=chiller_loc(i,:)+[chiller_dim(1)/2 0 0]+[1 0 0];
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %plot revised 2D layout
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    close
    fprintf('\nThe layout of the zones, transverse bulkheads and the revised
chiller\n')
    fprintf('locations is shown in figure 1\n\n')
    ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
    chiller_vec = [chiller_dim(1)/2*[1 1 -1 -1 1]; chiller_dim(2)/2*[1 -1 -1
1 1]];
    figure(2)
    plot(ship_vec(1,:),ship_vec(2,:))
    hold on
    for i=1:sum(chillers)

plot(chiller_vec(1,:)+chiller_loc(i,1),chiller_vec(2,:)+chiller_loc(i,2),'g')
    end
    for i=2:(length(bulkhead_loc)-1)
        plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1])
    end
    plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -1 1
1],'r:')
    for i=2:zones
        plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
    end
    scatter(pump_loc(:,1),pump_loc(:,2),'ch')
    axis equal
    axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
    xlabel('Longitudinal Axis')
    ylabel('Transverse Axis')
    title('2D Chiller Layout')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create supply and return piping structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if piping_config == 1
    %define header_loc_start and header_loc_end for single main with X
chillers in Y zones
    header_loc_start = zeros(sum(chillers),3);
 header_loc_end = zeros(sum(chillers),3);
    for i=1:sum(chillers)
```

```
        header_loc_start(i,:) = [chiller_loc(i)+chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
            eng_deck_ht_above_keel+chiller_dim(3)/2];
        header_loc_end(i,:) = [chiller_loc(i)-chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
            eng_deck_ht_above_keel+chiller_dim(3)/2];
    end
    seg_valve_index = 1;
    %create points for bends in header zone by zone
    for i=1:zones
        if chillers(i)==1
            %define supply header
            x_1_s = header_loc_start(i,1);
            x_2_s = pump_loc(i,1);
            x_3_s = header_loc_end(i,1);
            if i==1
               x_4a_s = LOA/2-3;
               x_1a_r = LOA/2-3;
               x_4b_s = zonal_boundaries(i);
               x_1b_r = zonal_boundaries(i);
            elseif i==zones
                x_4a_s = zonal_boundaries(i-1);
                x_1a_r = zonal_boundaries(i-1);
                x_4b_s = -LOA/2+0.5;
                x_1b_r = -LOA/2+0.5;
            else
                x_4a_s = zonal_boundaries(i-1);
                x_1a_r = zonal_boundaries(i-1);
                x_4b_s = zonal_boundaries(i);
                x_1b_r = zonal_boundaries(i);
            end
            y_1_s = header_loc_start(i,2);
            y_2_s = header_loc_start(i,2)-3;
            y_3_s = 0+offset_h/2;
            z_1_s = header_loc_start(i,3);
            z_2_s = header_deck_ht;
            header_loc_s(i,:,:) = [x_1_s y_1_s z_1_s;
                x_2_s y_1_s z_1_s;
                x_2_s y_2_s z_1_s;
                x_3_s y_2_s z_1_s;
                x_3_s y_2_s z_2_s;
                x_3_s y_3_s z_2_s;
                x_4a_s y_3_s z_2_s];
            header_loc_s_alt(i,:,:) = [x_3_s y_3_s z_2_s;
                x_4b_s y_3_s z_2_s];
            %define return header
            x_2_r = header_loc_end(i,1);
            y_1_r = 0-offset_h/2;
            y_2_r = header_loc_end(i,2)+offset_h-3;
            y_3_r = header_loc_end(i,2);
            z_1_r = header_deck_ht-offset_h;
            z_2_r = header_loc_end(i,3);
            header_loc_r(i,:,:) = [x_1a_r y_1_r z_1_r;
                x_2_r y_1_r z_1_r;
                x_2_r y_2_r z_1_r;
```

```
                    x_2_r y_2_r z_2_r;
                    x_2_r y_3_r z_2_r];
                header_loc_r_alt(i,:,:) = [x_1b_r y_1_r z_1_r;
                    x_2_r y_1_r z_1_r];
                %define recirc line
                x_1_rc = pump_loc(i,1)-0.75;
                y_1_rc = pump_loc(i,2);
                y_2_rc = header_loc_start(i,2)-3;
                z_1_rc = header_loc_start(i,3);
                recirc_line(i,:,:) = [x_1_rc y_1_rc z_1_rc;x_1_rc y_2_rc z_1_rc];
                %define isolation valves
                seg_valve_loc(seg_valve_index,:) = [x_3_s+1/ft_per_m y_3_s
z_2_s];
                seg_valve_loc(seg_valve_index+1,:) = [x_3_s-1/ft_per_m y_3_s
z_2_s];
                seg_valve_loc(seg_valve_index+2,:) = [x_3_s y_3_s-2/ft_per_m
z_2_s];
                seg_valve_loc(seg_valve_index+3,:) = [x_2_r+1/ft_per_m y_1_r
z_1_r];
                seg_valve_loc(seg_valve_index+4,:) = [x_2_r-1/ft_per_m y_1_r
z_1_r];
                seg_valve_loc(seg_valve_index+5,:) = [x_2_r y_1_r-2/ft_per_m
z_1_r];
                seg_valve_index = seg_valve_index+6;
            elseif chillers(i)==2
            elseif chillers(i)==3
            elseif chillers(i)==4
            elseif chillers(i)==5
            elseif chillers(i)==6
            end
        end
        %define cross-connect valve locations
        for j=1:zones-1
            for k=1:2
                if k==1
                    seg_valve_loc(4*j-4+seg_valve_index,:) =
[zonal_boundaries(j)+0.25 0+offset_h/2 header_deck_ht];
                    seg_valve_loc(4*j-3+seg_valve_index,:) =
[zonal_boundaries(j)+0.25 0-offset_h/2 header_deck_ht-offset_h];
                else
                    seg_valve_loc(4*j-2+seg_valve_index,:) =
[zonal_boundaries(j)-0.25 0+offset_h/2 header_deck_ht];
                    seg_valve_loc(4*j-1+seg_valve_index,:) =
[zonal_boundaries(j)-0.25 0-offset_h/2 header_deck_ht-offset_h];
                end

            end
        end

    end
    if piping_config == 2
        if piping_double_config == 1
            %define header_loc_start and header_loc_end for single main with X
chillers in Y zones
            header_loc_start = zeros(sum(chillers),3);
```

```
        header_loc_end = zeros(sum(chillers),3);
        for i=1:sum(chillers)
            header_loc_start(i,:) = [chiller_loc(i)+chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
                eng_deck_ht_above_keel+chiller_dim(3)/2];
            header_loc_end(i,:) = [chiller_loc(i)-chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
                eng_deck_ht_above_keel+chiller_dim(3)/2];
        end
        seg_valve_index = 1;
        %create points for bends in header zone by zone
        index = 0;
        for i=1:zones
            if chillers(i)==2
                for j=1:2
                    index = index+1;
                    %define supply and return headers
                    x_1_s = header_loc_start(index,1);
                    x_2_s = pump_loc(index,1);
                    x_3_s = header_loc_end(index,1);
                    x_2_r = header_loc_end(index,1);
                    if i==1
                        x_4a_s = LOA/2-3;
                        x_1a_r = LOA/2-3-offset_h;
                        x_4b_s = zonal_boundaries(i);
                        x_1b_r = zonal_boundaries(i);
                    elseif i==zones
                        x_4a_s = zonal_boundaries(i-1);
                        x_1a_r = zonal_boundaries(i-1);
                        x_4b_s = -LOA/2+0.5;
                        x_1b_r = -LOA/2+0.5+offset_h;
                    else
                        x_4a_s = zonal_boundaries(i-1);
                        x_1a_r = zonal_boundaries(i-1);
                        x_4b_s = zonal_boundaries(i);
                        x_1b_r = zonal_boundaries(i);
                    end
                    if j==1%port side
                        y_1_s = header_loc_start(index,2);
                        y_2_s = header_loc_start(index,2)-3;
                        y_3_s = beam/2-3/ft_per_m;
                        y_4_s = 0+offset_h/2;
                        y_1_r = 0-offset_h/2;
                        y_2_r = beam/2-3/ft_per_m-offset_h;
                        y_3_r = header_loc_end(index,2)+offset_h-3;
                        y_4_r = header_loc_end(index,2);
                        z_2_s = port_header_deck_ht;
                        z_1_r = port_header_deck_ht-offset_h;
                    elseif j==2 %starboard side
                        y_1_s = header_loc_start(index,2);
                        y_2_s = header_loc_start(index,2)+3;
                        y_3_s = -beam/2+3/ft_per_m;
                        y_4_s = 0+offset_h/2;
                        y_1_r = 0-offset_h/2;
                        y_2_r = -beam/2+3/ft_per_m+offset_h;
```

```
        y_3_r = header_loc_end(index,2)-offset_h+3;
        y_4_r = header_loc_end(index,2);
        z_2_s = stbd_header_deck_ht;
        z_1_r = stbd_header_deck_ht-offset_h;
    end
    z_1_s = header_loc_start(index,3);
    z_2_r = header_loc_end(index,3);
    if i == 1
        header_loc_s(index,:,:) = [x_1_s y_1_s z_1_s;
            x_2_s y_1_s z_1_s;
            x_2_s y_2_s z_1_s;
            x_3_s y_2_s z_1_s;
            x_3_s y_2_s z_2_s;
            x_3_s y_3_s z_2_s;
            x_4a_s y_3_s z_2_s;
            x_4a_s y_4_s z_2_s];
        header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
            x_4b_s y_3_s z_2_s;
            x_4b_s y_3_s z_2_s];
        header_loc_r(index,:,:) = [x_1a_r y_1_r z_1_r;
            x_1a_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r;
            x_2_r y_3_r z_1_r;
            x_2_r y_3_r z_2_r;
            x_2_r y_4_r z_2_r];
        header_loc_r_alt(index,:,:) = [x_1b_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r];
    elseif i == zones
        header_loc_s(index,:,:) = [x_1_s y_1_s z_1_s;
            x_2_s y_1_s z_1_s;
            x_2_s y_2_s z_1_s;
            x_3_s y_2_s z_1_s;
            x_3_s y_2_s z_2_s;
            x_3_s y_3_s z_2_s;
            x_4a_s y_3_s z_2_s;
            x_4a_s y_3_s z_2_s];
        header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
            x_4b_s y_3_s z_2_s;
            x_4b_s y_4_s z_2_s];
        header_loc_r(index,:,:) = [x_1a_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r;
            x_2_r y_3_r z_1_r;
            x_2_r y_3_r z_2_r;
            x_2_r y_4_r z_2_r;
            x_2_r y_4_r z_2_r];
        header_loc_r_alt(index,:,:) = [x_1b_r y_1_r z_1_r
            x_1b_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r];
    else
        header_loc_s(index,:,:) = [x_1_s y_1_s z_1_s;
            x_2_s y_1_s z_1_s;
            x_2_s y_2_s z_1_s;
            x_3_s y_2_s z_1_s;
            x_3_s y_2_s z_2_s;
```

```
            x_3_s y_3_s z_2_s;
            x_4a_s y_3_s z_2_s;
            x_4a_s y_3_s z_2_s];
        header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
            x_4b_s y_3_s z_2_s;
            x_4b_s y_3_s z_2_s];
        header_loc_r(index,:,:) = [x_1a_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r;
            x_2_r y_3_r z_1_r;
            x_2_r y_3_r z_2_r;
            x_2_r y_4_r z_2_r;
            x_2_r y_4_r z_2_r];
        header_loc_r_alt(index,:,:) = [x_1b_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r;
            x_2_r y_2_r z_1_r];
    end
    %define recirc line
    if j==1
        x_1_rc = pump_loc(index,1)-0.75;
        y_1_rc = pump_loc(index,2);
        y_2_rc = header_loc_start(index,2)-3;
        z_1_rc = header_loc_start(index,3);
    elseif j==2
        x_1_rc = pump_loc(index,1)-0.75;
        y_1_rc = pump_loc(index,2);
        y_2_rc = header_loc_start(index,2)+3;
        z_1_rc = header_loc_start(index,3);
    end
    recirc_line(index,:,:) = [x_1_rc y_1_rc z_1_rc;x_1_rc
y_2_rc z_1_rc];
    %define athwartship cross-connect points and
    %athwartship cross-connect valve locations
    if i==1 && j==1
        x_11_cc_s = x_4a_s;
        y_11_cc_s = y_4_s;
        z_11_cc_s = z_2_s;
        x_11_cc_r = x_1a_r;
        y_11_cc_r = y_1_r;
        z_11_cc_r = z_1_r;
    elseif i==1 && j==2
        x_12_cc_s = x_4a_s;
        y_12_cc_s = y_4_s;
        z_12_cc_s = z_2_s;
        x_12_cc_r = x_1a_r;
        y_12_cc_r = y_1_r;
        z_12_cc_r = z_1_r;
    elseif i==zones && j==1
        x_21_cc_s = x_4b_s;
        y_21_cc_s = y_4_s;
        z_21_cc_s = z_2_s;
        x_21_cc_r = x_1b_r;
        y_21_cc_r = y_1_r;
        z_21_cc_r = z_1_r;
    elseif i==zones && j==2
        x_22_cc_s = x_4b_s;
```

```
                    y_22_cc_s = y_4_s;
                    z_22_cc_s = z_2_s;
                    x_22_cc_r = x_1b_r;
                    y_22_cc_r = y_1_r;
                    z_22_cc_r = z_1_r;
                end
                %define cross-connects
                %define isolation valves
                if j==1
                    sign = 1;
                else
                    sign = -1;
                end
                seg_valve_loc(seg_valve_index,:) = [x_3_s+1/ft_per_m
y_3_s z_2_s];
                seg_valve_loc(seg_valve_index+1,:) = [x_3_s-1/ft_per_m
y_3_s z_2_s];
                seg_valve_loc(seg_valve_index+2,:) = [x_3_s y_3_s-
sign*2/ft_per_m z_2_s];
                seg_valve_loc(seg_valve_index+3,:) = [x_2_r+1/ft_per_m
y_2_r z_1_r];
                seg_valve_loc(seg_valve_index+4,:) = [x_2_r-1/ft_per_m
y_2_r z_1_r];
                seg_valve_loc(seg_valve_index+5,:) = [x_2_r y_2_r-
sign*2/ft_per_m z_1_r];
                seg_valve_index = seg_valve_index+6;
            end
        end
    end
    %define athwartship cross-connect
    cc1_loc_s = [x_11_cc_s y_11_cc_s z_11_cc_s; x_12_cc_s y_12_cc_s
z_12_cc_s];
    cc2_loc_s = [x_21_cc_s y_21_cc_s z_21_cc_s; x_22_cc_s y_22_cc_s
z_22_cc_s];
    cc1_loc_r = [x_11_cc_r y_11_cc_r z_11_cc_r; x_12_cc_r y_12_cc_r
z_12_cc_r];
    cc2_loc_r = [x_21_cc_r y_21_cc_r z_21_cc_r; x_22_cc_r y_22_cc_r
z_22_cc_r];
    seg_valve_loc(seg_valve_index,:) = [(x_11_cc_s+x_12_cc_s)/2
(y_11_cc_s+y_12_cc_s)/2 (z_11_cc_s+z_12_cc_s)/2];
    seg_valve_loc(seg_valve_index+1,:) = [(x_21_cc_s+x_22_cc_s)/2
(y_21_cc_s+y_22_cc_s)/2 (z_21_cc_s+z_22_cc_s)/2];
    seg_valve_loc(seg_valve_index+2,:) = [(x_11_cc_r+x_12_cc_r)/2
(y_11_cc_r+y_12_cc_r)/2 (z_11_cc_r+z_12_cc_r)/2];
    seg_valve_loc(seg_valve_index+3,:) = [(x_21_cc_r+x_22_cc_r)/2
(y_21_cc_r+y_22_cc_r)/2 (z_21_cc_r+z_22_cc_r)/2];
    %define cross-connect valves across zones
    for j=1:zones-1
        for k=1:2
            if k==1
                seg_valve_loc(8*j-8+seg_valve_index+4,:) =
[zonal_boundaries(j)+0.25 beam/2-3/ft_per_m port_header_deck_ht];%supply fwd
port
```

```
                    seg_valve_loc(8*j-7+seg_valve_index+4,:) =
[zonal_boundaries(j)+0.25 beam/2-3/ft_per_m-offset_h port_header_deck_ht-
offset_h]; %return fwd port
                    seg_valve_loc(8*j-6+seg_valve_index+4,:) =
[zonal_boundaries(j)+0.25 -beam/2+3/ft_per_m stbd_header_deck_ht];%supply fwd
stbd
                    seg_valve_loc(8*j-5+seg_valve_index+4,:) =
[zonal_boundaries(j)+0.25 -beam/2+3/ft_per_m+offset_h stbd_header_deck_ht-
offset_h]; %return fwd stbd
                else
                    seg_valve_loc(8*j-4+seg_valve_index+4,:) =
[zonal_boundaries(j)-0.25 beam/2-3/ft_per_m port_header_deck_ht]; %supply aft
port
                    seg_valve_loc(8*j-3+seg_valve_index+4,:) =
[zonal_boundaries(j)-0.25 beam/2-3/ft_per_m-offset_h port_header_deck_ht-
offset_h]; %return aft port
                    seg_valve_loc(8*j-2+seg_valve_index+4,:) =
[zonal_boundaries(j)-0.25 -beam/2+3/ft_per_m stbd_header_deck_ht]; %supply
aft stbd
                    seg_valve_loc(8*j-1+seg_valve_index+4,:) =
[zonal_boundaries(j)-0.25 -beam/2+3/ft_per_m+offset_h stbd_header_deck_ht-
offset_h]; %return aft stbd
                end
            end
        end

    elseif piping_double_config == 2
        %define header_loc for double main loop w/ ext with 2 zones
        %define header_loc_start and header_loc_end for single main with X
chillers in Y zones
        header_loc_start = zeros(sum(chillers),3);
        header_loc_end = zeros(sum(chillers),3);
        for i=1:sum(chillers)
            header_loc_start(i,:) = [chiller_loc(i)+chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
                eng_deck_ht_above_keel+chiller_dim(3)/2];
            header_loc_end(i,:) = [chiller_loc(i)-chiller_dim(1)/2
chiller_loc(i+length(chiller_loc)) ...
                eng_deck_ht_above_keel+chiller_dim(3)/2];
        end
        %create points for bends in header zone by zone
        temp_zonal_boundaries=zeros(1,length(zonal_boundaries)+1);
        for m=1:length(zonal_boundaries)
            temp_zonal_boundaries(m+1)=zonal_boundaries(m);
        end
        temp_zonal_boundaries(1)=LOA/2;
        %find number of bends in each zone and maximum number
        %of bends in any zone
        index_hb = zeros(1,zones*2);
        for m=1:zones
            for k=1:length(header_bends)
                if header_bends(k)>temp_zonal_boundaries(m+1) &&
header_bends(k)<temp_zonal_boundaries(m)
                    if header_bends(k+length(header_bends))>=0
                        index_hb(m)=index_hb(m)+1;
```

```
                    else
                        index_hb(m+zones)=index_hb(m+zones)+1;
                    end
                end
            end
        end
        index_hb_max=max(index_hb);
        if index_hb_max < 1
            index_hb_max = 1;
        end
        header_loc_s = zeros((zones)*2,index_hb_max*2+7,3);
        header_loc_r = zeros((zones)*2,index_hb_max*2+5,3);
        index = 0;
        y_3_s_port = 0;
        y_3_s_stbd = 0;
        seg_valve_index = 1;
        for i=1:zones
            if chillers(i)==2
                for j=1:2
                    index=index+1;
                    %define supply and return headers
                    x_1_s = header_loc_start(index,1);
                    x_2_s = pump_loc(index,1);
                    x_3_s = header_loc_end(index,1);
                    x_3_r = header_loc_end(index,1);
                    x_4a_s=ones(1,index_hb_max);
                    x_4b_s=ones(1,index_hb_max);
                    x_2a_r=ones(1,index_hb_max);
                    x_2b_r=ones(1,index_hb_max);
                    y_4a_s=ones(1,index_hb_max);
                    y_4b_s=ones(1,index_hb_max);
                    y_2a_r=ones(1,index_hb_max);
                    y_2b_r=ones(1,index_hb_max);
                    %define bend locations in supply header
                    if i == 1 %first zone
                        if j == 1 %port side
                            count = 0;
                            for k=1:length(header_bends)
                                if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                                    if header_bends(k+length(header_bends))>0
                                        x_4a_s(index_hb_max-count) =
header_bends(k);
                                        x_1a_r(count+1) = header_bends(k)-
offset_h;
                                        y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
                                        y_2a_r(count+1) =
header_bends(k+length(header_bends))-offset_h;
                                        y_3_s_port =
header_bends(k+length(header_bends));
                                        count = count+1;
                                    end
                                end
                            end
```

```
                    if (index_hb_max-count)>0
                        for k=1:(index_hb_max-count)
                            x_4a_s(k) = x_3_s;
                            x_1a_r(index_hb_max-k+1) = x_3_s;
                            y_4a_s(k) = y_3_s_port;
                            y_2a_r(index_hb_max-k+1) = y_3_s_port-
offset_h;
                        end
                    end
                else %starboard side
                    count = 0;
                    for k=1:length(header_bends)
                        if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                            if header_bends(k+length(header_bends))<0
                                x_4a_s(index_hb_max-count) =
header_bends(k);
                                x_1a_r(count+1) = header_bends(k)-
offset_h;
                                y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
                                y_2a_r(count+1) =
header_bends(k+length(header_bends))+offset_h;
                                y_3_s_stbd =
header_bends(k+length(header_bends));
                                count = count+1;
                            end
                        end
                    end
                    if (index_hb_max-count)>0
                        for k=1:(index_hb_max-count)
                            x_4a_s(k) = x_3_s;
                            x_1a_r(index_hb_max-k+1) = x_3_s;
                            y_4a_s(k) = y_3_s_stbd;
                            y_2a_r(index_hb_max-k+1) =
y_3_s_stbd+offset_h;
                        end
                    end
                end
            elseif i == zones %last zone
                if j == 1 %port side
                    count = 0;
                    y_5_s_temp = y_3_s_port;
                    y_1_r_temp = y_3_s_port - offset_h;
                    for k=1:length(header_bends)
                        if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                            if header_bends(k+length(header_bends))>0
                                x_4a_s(index_hb_max-count) =
header_bends(k);
                                x_1a_r(count+1) =
header_bends(k)+offset_h;
                                y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
```

```
                                        y_2a_r(count+1) =
header_bends(k+length(header_bends))-offset_h;
                                        y_3_s_port =
header_bends(k+length(header_bends));
                                        count = count+1;
                                    end
                                end
                            end
                            if (index_hb_max-count)>0
                                for k=1:(index_hb_max-count)
                                    x_4a_s(k) = x_3_s;
                                    x_1a_r(index_hb_max-k+1) = x_3_s;
                                    y_4a_s(k) = y_3_s_port;
                                    y_2a_r(index_hb_max-k+1) = y_3_s_port-
offset_h;
                                end
                            end
                        else %starboard side
                            count = 0;
                            y_5_s_temp = y_3_s_stbd;
                            y_1_r_temp = y_3_s_stbd + offset_h;
                            for k=1:length(header_bends)
                                if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                                    if header_bends(k+length(header_bends))<0
                                        x_4a_s(index_hb_max-count) =
header_bends(k);
                                        x_1a_r(count+1) =
header_bends(k)+offset_h;
                                        y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
                                        y_2a_r(count+1) =
header_bends(k+length(header_bends))+offset_h;
                                        y_3_s_stbd =
header_bends(k+length(header_bends));
                                        count = count+1;
                                    end
                                end
                            end
                            if (index_hb_max-count)>0
                                for k=1:(index_hb_max-count)
                                    x_4a_s(k) = x_3_s;
                                    x_1a_r(index_hb_max-k+1) = x_3_s;
                                    y_4a_s(k) = y_3_s_stbd;
                                    y_2a_r(index_hb_max-k+1) =
y_3_s_stbd+offset_h;
                                end
                            end
                        end
                    else %middle zones
                        if j == 1 %port side
                            count = 0;
                            y_5_s_temp = y_3_s_port;
                            y_1_r_temp = y_3_s_port - offset_h;
                            for k=1:length(header_bends)
```

```
                                if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                                    if header_bends(k+length(header_bends))>0
                                        x_4a_s(index_hb_max-count) =
header_bends(k);
                                        x_1a_r(count+1) = header_bends(k)-
offset_h;
                                        y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
                                        y_2a_r(count+1) =
header_bends(k+length(header_bends))-offset_h;
                                        y_3_s_port =
header_bends(k+length(header_bends));
                                        count = count+1;
                                    end
                                end
                            end
                            if (index_hb_max-count)>0
                                for k=1:(index_hb_max-count)
                                    x_4a_s(k) = x_3_s;
                                    x_1a_r(index_hb_max-k+1) = x_3_s;
                                    y_4a_s(k) = y_3_s_port;
                                    y_2a_r(index_hb_max-k+1) = y_3_s_port-
offset_h;
                                end
                            end
                        else %starboard side
                            count = 0;
                            y_5_s_temp = y_3_s_stbd;
                            y_1_r_temp = y_3_s_stbd + offset_h;
                            for k=1:length(header_bends)
                                if header_bends(k)<=temp_zonal_boundaries(i)
&& header_bends(k)>temp_zonal_boundaries(i+1)
                                    if header_bends(k+length(header_bends))<0
                                        x_4a_s(index_hb_max-count) =
header_bends(k);
                                        x_1a_r(count+1) = header_bends(k)-
offset_h;
                                        y_4a_s(index_hb_max-count) =
header_bends(k+length(header_bends));
                                        y_2a_r(count+1) =
header_bends(k+length(header_bends))+offset_h;
                                        y_3_s_stbd =
header_bends(k+length(header_bends));
                                        count = count+1;
                                    end
                                end
                            end
                            if (index_hb_max-count)>0
                                for k=1:(index_hb_max-count)
                                    x_4a_s(k) = x_3_s;
                                    x_1a_r(index_hb_max-k+1) = x_3_s;
                                    y_4a_s(k) = y_3_s_stbd;
                                    y_2a_r(index_hb_max-k+1) =
y_3_s_stbd+offset_h;
```

```
                        end
                    end
                end
            end

            if i==1
                x_4b_s = zonal_boundaries(i);
                x_1b_r = zonal_boundaries(i);
            elseif i==zones
                x_5a_s = zonal_boundaries(i-1);
                x_4b_s = -LOA/2+0.5;
                x_1b_r = -LOA/2+0.5+offset_h;
            else
                x_5a_s = zonal_boundaries(i-1);
                x_4b_s = zonal_boundaries(i);
                x_1b_r = zonal_boundaries(i);
            end
            if j==1%port side
                y_1_s = header_loc_start(index,2);
                y_2_s = header_loc_start(index,2)-3;
                y_3_s = y_3_s_port;
                y_5_s = 0+offset_h/2;
                y_1_r = 0-offset_h/2;
                y_2_r = beam/2-3/ft_per_m-offset_h;
                y_3_r = header_loc_end(index,2)+offset_h-3;
                y_4_r = header_loc_end(index,2);
                z_2_s = port_header_deck_ht;
                z_1_r = port_header_deck_ht-offset_h;
            elseif j==2 %starboard side
                y_1_s = header_loc_start(index,2);
                y_2_s = header_loc_start(index,2)+3;
                y_3_s = y_3_s_stbd;
                y_5_s = 0+offset_h/2;
                y_1_r = 0-offset_h/2;
                y_2_r = -beam/2+3/ft_per_m+offset_h;
                y_3_r = header_loc_end(index,2)-offset_h+3;
                y_4_r = header_loc_end(index,2);
                z_2_s = stbd_header_deck_ht;
                z_1_r = stbd_header_deck_ht-offset_h;
            end
            z_1_s = header_loc_start(index,3);
            z_2_r = header_loc_end(index,3);
            if i == 1
                header_loc_s(index,1,:) = [x_1_s y_1_s z_1_s];
                header_loc_s(index,2,:) = [x_2_s y_1_s z_1_s];
                header_loc_s(index,3,:) = [x_2_s y_2_s z_1_s];
                header_loc_s(index,4,:) = [x_3_s y_2_s z_1_s];
                header_loc_s(index,5,:) = [x_3_s y_2_s z_2_s];
                header_loc_s(index,6,:) = [x_3_s y_3_s z_2_s];
                for k=1:index_hb_max-1
                    header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                    header_loc_s(index,6+2*k,:) = [x_4a_s(k)
y_4a_s(k+1) z_2_s];
                end
```

```
                    for k=index_hb_max
                        header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                    end
                    header_loc_s(index,index_hb_max*2+6,:) =
[x_4a_s(index_hb_max) y_5_s z_2_s];
                    header_loc_s(index,index_hb_max*2+7,:) =
[x_4a_s(index_hb_max) y_5_s z_2_s];
                    header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
                        x_4b_s y_3_s z_2_s;
                        x_4b_s y_3_s z_2_s];
                    header_loc_r(index,1,:) = [x_1a_r(1) y_1_r z_1_r];
                    header_loc_r(index,2,:) = [x_1a_r(1) y_1_r z_1_r];
                    for k=1:index_hb_max-1
                        header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                        header_loc_r(index,2*k+2,:) = [x_1a_r(k+1)
y_2a_r(k) z_1_r];
                    end
                    for k=index_hb_max
                        header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                    end
                    header_loc_r(index,index_hb_max*2+2,:) = [x_3_r
y_2a_r(index_hb_max) z_1_r];
                    header_loc_r(index,index_hb_max*2+3,:) = [x_3_r y_3_r
z_1_r];
                    header_loc_r(index,index_hb_max*2+4,:) = [x_3_r y_3_r
z_2_r];
                    header_loc_r(index,index_hb_max*2+5,:) = [x_3_r y_4_r
z_2_r];
                    header_loc_r_alt(index,:,:) = [x_1b_r
y_2a_r(index_hb_max) z_1_r;
                        x_3_r y_2a_r(index_hb_max) z_1_r;
                        x_3_r y_2a_r(index_hb_max) z_1_r];
                elseif i == zones
                    header_loc_s(index,1,:) = [x_1_s y_1_s z_1_s];
                    header_loc_s(index,2,:) = [x_2_s y_1_s z_1_s];
                    header_loc_s(index,3,:) = [x_2_s y_2_s z_1_s];
                    header_loc_s(index,4,:) = [x_3_s y_2_s z_1_s];
                    header_loc_s(index,5,:) = [x_3_s y_2_s z_2_s];
                    header_loc_s(index,6,:) = [x_3_s y_3_s z_2_s];
                    for k=1:index_hb_max-1
                        header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                        header_loc_s(index,6+2*k,:) = [x_4a_s(k)
y_4a_s(k+1) z_2_s];
                    end
                    for k=index_hb_max
                        header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                    end
                    header_loc_s(index,index_hb_max*2+6,:) =
[x_4a_s(index_hb_max) y_5_s_temp z_2_s];
```

```
                    header_loc_s(index,index_hb_max*2+7,:) = [x_5a_s
y_5_s_temp z_2_s];
                    header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
                        x_4b_s y_3_s z_2_s;
                        x_4b_s y_5_s z_2_s];
                    header_loc_r(index,1,:) = [zonal_boundaries(i-1)
y_1_r_temp z_1_r];
                    header_loc_r(index,2,:) = [x_1a_r(1) y_1_r_temp
z_1_r];
                    for k=1:index_hb_max-1
                        header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                        header_loc_r(index,2*k+2,:) = [x_1a_r(k+1)
y_2a_r(k) z_1_r];
                    end
                    for k=index_hb_max
                        header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                    end
                    header_loc_r(index,index_hb_max*2+2,:) = [x_3_r
y_2a_r(index_hb_max) z_1_r];
                    header_loc_r(index,index_hb_max*2+3,:) = [x_3_r y_3_r
z_1_r];
                    header_loc_r(index,index_hb_max*2+4,:) = [x_3_r y_3_r
z_2_r];
                    header_loc_r(index,index_hb_max*2+5,:) = [x_3_r y_4_r
z_2_r];
                    header_loc_r_alt(index,:,:) = [x_1b_r y_1_r z_1_r;
                        x_1b_r y_2a_r(index_hb_max) z_1_r;
                        x_3_r y_2a_r(index_hb_max) z_1_r];
                else
                    header_loc_s(index,1,:) = [x_1_s y_1_s z_1_s];
                    header_loc_s(index,2,:) = [x_2_s y_1_s z_1_s];
                    header_loc_s(index,3,:) = [x_2_s y_2_s z_1_s];
                    header_loc_s(index,4,:) = [x_3_s y_2_s z_1_s];
                    header_loc_s(index,5,:) = [x_3_s y_2_s z_2_s];
                    header_loc_s(index,6,:) = [x_3_s y_3_s z_2_s];
                    for k=1:index_hb_max-1
                        header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                        header_loc_s(index,6+2*k,:) = [x_4a_s(k)
y_4a_s(k+1) z_2_s];
                    end
                    for k=index_hb_max
                        header_loc_s(index,5+2*k,:) = [x_4a_s(k)
y_4a_s(k) z_2_s];
                    end
                    header_loc_s(index,index_hb_max*2+6,:) =
[x_4a_s(index_hb_max) y_5_s_temp z_2_s];
                    header_loc_s(index,index_hb_max*2+7,:) = [x_5a_s
y_5_s_temp z_2_s];
                    header_loc_s_alt(index,:,:) = [x_3_s y_3_s z_2_s;
                        x_4b_s y_3_s z_2_s;
                        x_4b_s y_3_s z_2_s];
```

```
                          header_loc_r(index,1,:) = [zonal_boundaries(i-1)
y_1_r_temp z_1_r];
                          header_loc_r(index,2,:) = [x_1a_r(1) y_1_r_temp
z_1_r];
                          for k=1:index_hb_max-1
                              header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                              header_loc_r(index,2*k+2,:) = [x_1a_r(k+1)
y_2a_r(k) z_1_r];
                          end
                          for k=index_hb_max
                              header_loc_r(index,2*k+1,:) = [x_1a_r(k)
y_2a_r(k) z_1_r];
                          end
                          header_loc_r(index,index_hb_max*2+2,:) = [x_3_r
y_2a_r(index_hb_max) z_1_r];
                          header_loc_r(index,index_hb_max*2+3,:) = [x_3_r y_3_r
z_1_r];
                          header_loc_r(index,index_hb_max*2+4,:) = [x_3_r y_3_r
z_2_r];
                          header_loc_r(index,index_hb_max*2+5,:) = [x_3_r y_4_r
z_2_r];
                          header_loc_r_alt(index,:,:) = [x_1b_r
y_2a_r(index_hb_max) z_1_r;
                              x_3_r y_2a_r(index_hb_max) z_1_r;
                              x_3_r y_2a_r(index_hb_max) z_1_r];
                      end
                      %define recirc line
                      if j==1
                          x_1_rc = pump_loc(index,1)-0.75;
                          y_1_rc = pump_loc(index,2);
                          y_2_rc = header_loc_start(index,2)-3;
                          z_1_rc = header_loc_start(index,3);
                      elseif j==2
                          x_1_rc = pump_loc(index,1)-0.75;
                          y_1_rc = pump_loc(index,2);
                          y_2_rc = header_loc_start(index,2)+3;
                          z_1_rc = header_loc_start(index,3);
                      end
                      recirc_line(index,:,:) = [x_1_rc y_1_rc z_1_rc;x_1_rc
y_2_rc z_1_rc];
                      %define athwartship cross-connect points
                      if i==1 && j==1
                          x_11_cc_s = x_4a_s(index_hb_max);
                          y_11_cc_s = y_5_s;
                          z_11_cc_s = z_2_s;
                          x_11_cc_r = x_1a_r(1);
                          y_11_cc_r = y_1_r;
                          z_11_cc_r = z_1_r;
                      elseif i==1 && j==2
                          x_12_cc_s = x_4a_s(index_hb_max);
                          y_12_cc_s = y_5_s;
                          z_12_cc_s = z_2_s;
                          x_12_cc_r = x_1a_r(1);
                          y_12_cc_r = y_1_r;
```

```
                    z_12_cc_r = z_1_r;
                elseif i==zones && j==1
                    x_21_cc_s = x_4b_s;
                    y_21_cc_s = y_5_s;
                    z_21_cc_s = z_2_s;
                    x_21_cc_r = x_1b_r;
                    y_21_cc_r = y_1_r;
                    z_21_cc_r = z_1_r;
                elseif i==zones && j==2
                    x_22_cc_s = x_4b_s;
                    y_22_cc_s = y_5_s;
                    z_22_cc_s = z_2_s;
                    x_22_cc_r = x_1b_r;
                    y_22_cc_r = y_1_r;
                    z_22_cc_r = z_1_r;
                end
                %define cross-connects
                %define isolation valves
                if j==1
                    sign = 1;
                else
                    sign = -1;
                end
                seg_valve_loc(seg_valve_index+0,:) = [x_3_s y_3_s
z_2_s]+[1/ft_per_m 0 0];
                seg_valve_loc(seg_valve_index+1,:) = [x_3_s y_3_s
z_2_s]+[-1/ft_per_m 0 0];
                seg_valve_loc(seg_valve_index+2,:) = [x_3_s y_3_s
z_2_s]+[0 -sign*2/ft_per_m 0];
                seg_valve_loc(seg_valve_index+3,:) = [x_3_s y_3_s
z_2_s]+[1/ft_per_m 0 0]+offset_h*[0 -sign -1];
                seg_valve_loc(seg_valve_index+4,:) = [x_3_s y_3_s
z_2_s]+[-1/ft_per_m 0 0]+offset_h*[0 -sign -1];
                seg_valve_loc(seg_valve_index+5,:) = [x_3_s y_3_s
z_2_s]+[0 -sign*2/ft_per_m 0]+offset_h*[0 -sign -1];
                if i<zones
                    %define cross-connect valves across zones
                    seg_valve_loc(seg_valve_index+6,:) =
[zonal_boundaries(i) y_3_s z_2_s]+[0.25 0 0];
                    seg_valve_loc(seg_valve_index+7,:) =
[zonal_boundaries(i) y_3_s z_2_s]+[-0.25 0 0];
                    seg_valve_loc(seg_valve_index+8,:) =
[zonal_boundaries(i) y_3_s z_2_s]+[0.25 0 0]+offset_h*[0 -sign -1];
                    seg_valve_loc(seg_valve_index+9,:) =
[zonal_boundaries(i) y_3_s z_2_s]+[-0.25 0 0]+offset_h*[0 -sign -1];
                    seg_valve_index = seg_valve_index+10;
                else
                    seg_valve_index = seg_valve_index+6;
                end
            end
        end
    end

    %define athwartship cross-connect
```

```
        cc1_loc_s = [x_11_cc_s y_11_cc_s z_11_cc_s; x_12_cc_s y_12_cc_s
z_12_cc_s];
        cc2_loc_s = [x_21_cc_s y_21_cc_s z_21_cc_s; x_22_cc_s y_22_cc_s
z_22_cc_s];
        cc1_loc_r = [x_11_cc_r y_11_cc_r z_11_cc_r; x_12_cc_r y_12_cc_r
z_12_cc_r];
        cc2_loc_r = [x_21_cc_r y_21_cc_r z_21_cc_r; x_22_cc_r y_22_cc_r
z_22_cc_r];
        seg_valve_loc(seg_valve_index,:) = [(x_11_cc_s+x_12_cc_s)/2
(y_11_cc_s+y_12_cc_s)/2 (z_11_cc_s+z_12_cc_s)/2];
        seg_valve_loc(seg_valve_index+1,:) = [(x_21_cc_s+x_22_cc_s)/2
(y_21_cc_s+y_22_cc_s)/2 (z_21_cc_s+z_22_cc_s)/2];
        seg_valve_loc(seg_valve_index+2,:) = [(x_11_cc_r+x_12_cc_r)/2
(y_11_cc_r+y_12_cc_r)/2 (z_11_cc_r+z_12_cc_r)/2];
        seg_valve_loc(seg_valve_index+3,:) = [(x_21_cc_r+x_22_cc_r)/2
(y_21_cc_r+y_22_cc_r)/2 (z_21_cc_r+z_22_cc_r)/2];
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define segregation valve dimensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
seg_valve_dim = [1/ft_per_m 1/ft_per_m 1/ft_per_m];
seg_valve_vec = [seg_valve_dim(1)/2*[1 1 -1 -1 1];seg_valve_dim(2)/2*[1 -1 -1
1 1];seg_valve_dim(3)/2*[1 1 1 1 1]];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot 2D layout w/ mains
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ship_vec = [LOA/2*[1 1 -1 -1 1];beam/2*[1 -1 -1 1
1];eng_deck_ht_above_keel*[1 1 1 1 1]];
figure(3)
plot(ship_vec(1,:),ship_vec(2,:))
hold on
for i=1:length(seg_valve_loc)

plot(seg_valve_vec(1,:)+seg_valve_loc(i,1),seg_valve_vec(2,:)+seg_valve_loc(i
,2),'k')
end
for i=1:sum(chillers)

plot(chiller_vec(1,:)+chiller_loc(i,1),chiller_vec(2,:)+chiller_loc(i,2),'k')
end
for i=2:(length(bulkhead_loc)-1)
    plot(bulkhead_loc(i)*[1 1],beam/2*[1 -1],'g')
end
plot(zonal_boundaries(1)*[1 1 0 0 1]+LOA/2*[0 0 1 1 0],beam/2*[1 -1 -1 1
1],'r:')
for i=2:zones
    plot(zonal_boundaries(i)*[0 1 1 0]+zonal_boundaries(i-1)*[1 0 0
1],beam/2*[1 1 -1 -1],'r:')
end
scatter(chiller_loc(:,1),chiller_loc(:,2),'ks')
scatter(pump_loc(:,1),pump_loc(:,2),'kh')
axis equal
```

```
axis ([-LOA/2-5 LOA/2+5 -beam/2-5 beam/2+5])
for i=1:sum(chillers)
    plot(header_loc_s(i,:,1),header_loc_s(i,:,2),'b')
    plot(header_loc_r(i,:,1),header_loc_r(i,:,2),'r')
    plot(header_loc_s_alt(i,:,1),header_loc_s_alt(i,:,2),'b')
    plot(header_loc_r_alt(i,:,1),header_loc_r_alt(i,:,2),'r')
end
for i=1:sum(chillers)
    plot(recirc_line(i,:,1),recirc_line(i,:,2),'b')
end
xlabel('Longitudinal Axis')
ylabel('Transverse Axis')
title('2D Mains Layout')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot 3D layout w/ mains
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4)
hold on
for i=1:sum(chillers) %plot mains
    plot3(header_loc_s(i,:,1),header_loc_s(i,:,2),header_loc_s(i,:,3),'b')
    plot3(header_loc_r(i,:,1),header_loc_r(i,:,2),header_loc_r(i,:,3),'r')

plot3(header_loc_s_alt(i,:,1),header_loc_s_alt(i,:,2),header_loc_s_alt(i,:,3)
,'b')

plot3(header_loc_r_alt(i,:,1),header_loc_r_alt(i,:,2),header_loc_r_alt(i,:,3)
,'r')
end
for i=1:sum(chillers) %plot recirc line
    plot3(recirc_line(i,:,1),recirc_line(i,:,2),recirc_line(i,:,3),'b')
end
if piping_config == 2 %plot athwartship cc piping for double mains
    plot3(cc1_loc_s(:,1),cc1_loc_s(:,2),cc1_loc_s(:,3),'b')
    plot3(cc2_loc_s(:,1),cc2_loc_s(:,2),cc2_loc_s(:,3),'b')
    plot3(cc1_loc_r(:,1),cc1_loc_r(:,2),cc1_loc_r(:,3),'r')
    plot3(cc2_loc_r(:,1),cc2_loc_r(:,2),cc2_loc_r(:,3),'r')
end
plot3([LOA/2 LOA/2 -LOA/2 -LOA/2 LOA/2],[beam/2 -beam/2 -beam/2 beam/2
beam/2],[0 0 0 0 0]) %plot ship boundaries
chiller_vec_3D = [chiller_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    chiller_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    chiller_dim(3)/3*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
pump_vec_3D = [pump_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    pump_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    pump_dim(3)/3*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
seg_valve_vec_3D = [seg_valve_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1
-1];
    seg_valve_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    seg_valve_dim(3)/3*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
for i=1:sum(chillers)

plot3(chiller_vec_3D(1,:)+chiller_loc(i,1),chiller_vec_3D(2,:)+chiller_loc(i,
2),chiller_vec_3D(3,:)+chiller_loc(i,3),'k')
```

```
plot3(pump_vec_3D(1,:)+pump_loc(i,1),pump_vec_3D(2,:)+pump_loc(i,2),pump_vec_
3D(3,:)+pump_loc(i,3),'k')
end
for i=1:length(seg_valve_loc)

plot3(seg_valve_vec_3D(1,:)+seg_valve_loc(i,1),seg_valve_vec_3D(2,:)+seg_valv
e_loc(i,2),seg_valve_vec_3D(3,:)+seg_valve_loc(i,3),'k')
end
axis equal
xlabel('Longitudinal Axis')
ylabel('Transverse Axis')
title('3D Mains Layout')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine if a branch is vital or non-vital
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
inputs = length(Load_Loc_m);
vital = false(1,inputs);
for i=1:inputs
    if Priority(i) < 3
        vital(i) = true;
    else
        vital(i) = false;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define branch location by defining an array of 10 points for the
% start, bends and end of the branch piping starting at the supply
% header junction and ending at the return header junction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
branch_loc = zeros(10,2,3,inputs); %10 points describing branch location - 2
branches per vital load - 1 branch per non-vital load
branch_loc_vital = zeros(10,3,inputs); %delete
branch_gate_loc = zeros(2,2,3,inputs); %2 points describing gate valve
locations - 2 sets per vital load - 1 set per non-vital load
branch_globe_loc = zeros(1,2,3,inputs); %1 point describing globe valve
locations - 2 sets per vital load - 1 set per non-vital load
gate_valve_b = zeros(2,inputs);
globe_valve_b = zeros(2,inputs);
if piping_config == 1 %single main with n zones
    length_b = zeros(2,inputs);
    for i=1:inputs
        if Load_Loc_m(i,1)> header_loc_s(1,7,1)
            %fwd of header
            x_1 = header_loc_s(1,7,1)-0.1;
            y_1 = offset_h/2;
            z_1 = header_loc_s(1,6,3);
            if Load_Loc_m(i,3)>z_1
                sign = -1;
            else
                sign = 1;
            end
            branch_loc(:,1,:,i) = [x_1 y_1 z_1;
```

```
                x_1 y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-sign*offset_b/2;
                x_1 y_1-offset_h Load_Loc_m(i,3)-sign*offset_b/2;
                x_1 y_1-offset_h z_1-offset_h];
            if Load_Loc_m(i,3) > z_1
                sign = 1;
            else
                sign = -1;
            end
            branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
            branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
            gate_valve_b(1,i) = gate_valve_b(1,i)+2;
            branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
            globe_valve_b(1,i) = globe_valve_b(1,i)+1;
        elseif Load_Loc_m(i,1)<header_loc_s_alt(zones,2,1)
            %aft of header
            x_1 = header_loc_s_alt(zones,2,1)+0.1;
            y_1 = offset_h/2;
            z_1 = header_loc_s(1,6,3);
            if Load_Loc_m(i,3)>z_1
                sign = 1;
            else
                sign = -1;
            end
            branch_loc(:,1,:,i) = [x_1 y_1 z_1;
                x_1 y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-sign*offset_b/2;
                x_1 y_1-offset_h Load_Loc_m(i,3)-sign*offset_b/2;
                x_1 y_1-offset_h z_1-offset_h];
            if Load_Loc_m(i,3) > z_1
                sign = 1;
            else
                sign = -1;
            end
```

```
            branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
            branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
            gate_valve_b(1,i) = gate_valve_b(1,i)+2;
            branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
            globe_valve_b(1,i) = globe_valve_b(1,i)+1;
        else
            %within header boundaries
            y_1 = offset_h/2;
            z_1 = header_loc_s(1,6,3);
            if (y_1-Load_Loc_m(i,2))*(Load_Loc_m(i,3)-z_1)>0
                sign = 1;
            else
                sign = -1;
            end
            branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-sign*offset_b/2;
                Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
            if Load_Loc_m(i,3) > z_1
                sign = 1;
            else
                sign = -1;
            end
            branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
            branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
            gate_valve_b(1,i) = gate_valve_b(1,i)+2;
            branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
            globe_valve_b(1,i) = globe_valve_b(1,i)+1;
        end
    end
elseif piping_config == 2 %double main with n zones
    if piping_double_config == 1 %double main simple loop with n zones
        length_b = zeros(2,inputs);
        for i=1:inputs
            %connect each load to the supply and return header
            %regardless of vital or non-vital
            if Load_Loc_m(i,1) > header_loc_s(1,8,1) && ...
```

```
                    ((Load_Loc_m(i,2) >= offset_h/2 && Load_Loc_m(i,2) <=
header_loc_s(1,7,2))||...
                    (Load_Loc_m(i,2) <= -offset_h/2 && Load_Loc_m(i,2) >=
header_loc_s(2,7,2)))
                %port & stbd fwd of header loop
                x_1 = header_loc_s(1,8,1);
                %set z_1 depending on port or stbd side
                if Load_Loc_m(i,2) > 0
                    z_1 = header_loc_s(1,8,3);
                else
                    z_1 = header_loc_s(2,8,3);
                end
                %set sign depending on z-location
                if Load_Loc_m(i,3) >= z_1
                    sign = 1;
                else
                    sign = -1;
                end
                branch_loc(:,1,:,i) = [x_1 Load_Loc_m(i,2) z_1;
                    x_1 Load_Loc_m(i,2) Load_Loc_m(i,3)-sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1-offset_h Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1-offset_h Load_Loc_m(i,2) z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,1) > header_loc_s(1,8,1) && Load_Loc_m(i,2) >
-offset_h/2 && Load_Loc_m(i,2) < offset_h/2
                %port fwd of header loop within x-conn section
                x_1 = header_loc_s(1,8,1);
                z_1 = header_loc_s(1,8,3);
                %set sign depending on z-location
                if Load_Loc_m(i,3) >= z_1
                    sign = 1;
                else
                    sign = -1;
                end
                branch_loc(:,1,:,i) = [x_1 offset_h/2+0.1 z_1;
```

```
                         x_1 offset_h/2+0.1 Load_Loc_m(i,3)-sign*offset_b/2;
                         Load_Loc_m(i,1) offset_h/2+0.1 Load_Loc_m(i,3)-
sign*offset_b/2;
                         Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                         Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                         Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                         Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                         Load_Loc_m(i,1) offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                         x_1-offset_h offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                         x_1-offset_h offset_h/2+0.1 z_1-offset_h];
                    branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                    branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                    gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                    branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                    globe_valve_b(1,i) = globe_valve_b(1,i)+1;
                elseif Load_Loc_m(i,1) > header_loc_s_alt(zones*2,3,1) &&
Load_Loc_m(i,1) < header_loc_s(1,8,1) && Load_Loc_m(i,2) > 0
                    %port mid-zones
                    y_1 = header_loc_s(1,6,2);
                    z_1 = header_loc_s(1,8,3);
                    if (y_1-Load_Loc_m(i,2))*(Load_Loc_m(i,3)-z_1)>0
                        sign = 1;
                    else
                        sign = -1;
                    end
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                    else
                        branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
```

```
                        Load_Loc_m(i,1) y_1 deck;
                        Load_Loc_m(i,1) y_1/2 deck;
                        Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2-offset_b deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1-offset_h deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                end
                %set sign depending on z-location
                if Load_Loc_m(i,3) >= z_1
                    sign_z = 1;
                else
                    sign_z = -1;
                end
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign_z*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
                if vital(i) %vital load
                    y_1 = header_loc_s(zones*2,6,2);
                    z_1 = header_loc_s(zones*2,8,3);
                    sign = sign*-1;
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1+offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                    else
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1 deck;
```

```
                                    Load_Loc_m(i,1) y_1/2 deck;
                                    Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                                    Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                                    Load_Loc_m(i,1) y_1/2-offset_b deck-
sign*offset_b;
                                    Load_Loc_m(i,1) y_1+offset_h deck-sign*offset_b;
                                    Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                        end
                        %set sign depending on z-location
                        if Load_Loc_m(i,3) >= z_1
                            sign_z = 1;
                        else
                            sign_z = -1;
                        end
                        branch_gate_loc(1,2,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign_z*0.15];
                        branch_gate_loc(2,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.15];
                        gate_valve_b(2,i) = gate_valve_b(1,i)+2;
                        branch_globe_loc(1,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.3];
                        globe_valve_b(2,i) = globe_valve_b(1,i)+1;
                    end
                elseif Load_Loc_m(i,1) > header_loc_s_alt(zones*2,3,1) &&
Load_Loc_m(i,1) < header_loc_s(1,8,1) && Load_Loc_m(i,2) <= 0
                    %stbd mid-zones
                    y_1 = header_loc_s(zones*2,6,2);
                    z_1 = header_loc_s(zones*2,8,3);
                    if (y_1-Load_Loc_m(i,2))*(Load_Loc_m(i,3)-z_1)>0
                        sign = -1;
                    else
                        sign = 1;
                    end
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
```

```
                        Load_Loc_m(i,1) y_1+offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                else
                   branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                        Load_Loc_m(i,1) y_1 deck;
                        Load_Loc_m(i,1) y_1/2 deck;
                        Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2+offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2+offset_b deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1+offset_h deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                end
                %set sign depending on z-location
                if Load_Loc_m(i,3) >= z_1
                    sign_z = 1;
                else
                    sign_z = -1;
                end
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign_z*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
                if vital(i) %vital load
                    y_1 = header_loc_s(1,6,2);
                    z_1 = header_loc_s(1,8,3);
                    sign = sign*-1;
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
```

```
                            Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                    else
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1 deck;
                            Load_Loc_m(i,1) y_1/2 deck;
                            Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1/2-offset_b deck-
sign*offset_b;
                            Load_Loc_m(i,1) y_1-offset_h deck-sign*offset_b;
                            Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                    end
                    %set sign depending on z-location
                    if Load_Loc_m(i,3) >= z_1
                        sign_z = 1;
                    else
                        sign_z = -1;
                    end
                    branch_gate_loc(1,2,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign_z*0.15];
                    branch_gate_loc(2,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.15];
                    gate_valve_b(2,i) = gate_valve_b(1,i)+2;
                    branch_globe_loc(1,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign_z*0.3];
                    globe_valve_b(2,i) = globe_valve_b(1,i)+1;
                end
            elseif Load_Loc_m(i,1) < header_loc_s_alt(zones*2,3,1) &&
Load_Loc_m(i,2) > -offset_h/2 && Load_Loc_m(i,2) < offset_h/2
                %port aft of header loop within x-conn section
                x_1 = header_loc_s_alt(zones*2-1,3,1);
                z_1 = header_loc_s_alt(zones*2-1,3,3);
                if (Load_Loc_m(i,3)-z_1)<0
                    sign = -1;
                else
                    sign = 1;
                end
                branch_loc(:,1,:,i) = [x_1 offset_h/2+0.1 z_1;
                    x_1 offset_h/2+0.1 Load_Loc_m(i,3)-sign*offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1 Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
```

```
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h offset_h/2+0.1 z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,1) < header_loc_s_alt(zones*2,3,1) &&
(Load_Loc_m(i,2) >= offset_h/2||Load_Loc_m(i,2) <= -offset_h/2)
                %port & stbd aft of header loop
                x_1 = header_loc_s_alt(zones*2-1,3,1);
                %set z_1 depending on port or stbd side
                if Load_Loc_m(i,2) > 0
                    z_1 = header_loc_s_alt(zones*2-1,3,3);
                else
                    z_1 = header_loc_s_alt(zones*2,3,3);
                end
                if (Load_Loc_m(i,3)-z_1)<0
                    sign = -1;
                else
                    sign = 1;
                end
                branch_loc(:,1,:,i) = [x_1 Load_Loc_m(i,2) z_1;
                    x_1 Load_Loc_m(i,2) Load_Loc_m(i,3)-sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h Load_Loc_m(i,2) z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
```

```
            end
        end
    elseif piping_double_config == 2 %double main complex loop with n zones
        length_b = zeros(2,inputs);
        for i=1:inputs
            flag = 1;
            for j=1:length(header_bends)-1 %find y location of junction
                if header_bends(j,2)<0 %stbd side
                    if flag == 1; %first time through loop catches transition
from port to stbd
                        flag = 0;
                        y_temp = header_bends(j,2);
                    end
                end
            end
            %connect each load to the supply and return header
            %regardless of vital or non-vital
            if Load_Loc_m(i,1)>header_bends(1,1) && Load_Loc_m(i,2)>-
offset_h/2 && Load_Loc_m(i,2)<offset_h/2
                %fwd of header loop between x-conn
                x_1 = header_bends(1,1);
                z_1 = header_loc_s(1,8,3);
                branch_loc(:,1,:,i) = [x_1 offset_h/2+0.1 z_1;
                    x_1 offset_h/2+0.1 Load_Loc_m(i,3)-offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1 Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1
Load_Loc_m(i,3)+offset_b/2;
                    x_1-offset_h offset_h/2+0.1 Load_Loc_m(i,3)+offset_b/2;
                    x_1-offset_h offset_h/2+0.1 z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,1)>header_bends(1,1) && ...
                    ((Load_Loc_m(i,2)>=offset_h/2 &&
Load_Loc_m(i,2)<=header_bends(1,2))||...
                    (Load_Loc_m(i,2) <= -offset_h/2 && Load_Loc_m(i,2) >=
y_temp))
                %port & stbd fwd of header loop
                x_1 = header_bends(1,1);
                %set z_1 depending on port or stbd side
                if Load_Loc_m(i,2) > 0
```

```
                    z_1 = header_loc_s(1,8,3);
                else
                    z_1 = header_loc_s(2,8,3);
                end
                branch_loc(:,1,:,i) = [x_1 Load_Loc_m(i,2) z_1;
                    x_1 Load_Loc_m(i,2) Load_Loc_m(i,3)-offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+offset_b/2;
                    x_1-offset_h Load_Loc_m(i,2) Load_Loc_m(i,3)+offset_b/2;
                    x_1-offset_h Load_Loc_m(i,2) z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,1)<header_bends(length(header_bends),1) &&
Load_Loc_m(i,2)>-offset_h/2 && Load_Loc_m(i,2)<offset_h/2
                %aft of header loop between x-conn
                x_1 = -LOA/2+0.5;
                z_1 = header_loc_s_alt(zones*2-1,3,3);
                if (Load_Loc_m(i,3)-z_1)<0
                    sign = -1;
                else
                    sign = 1;
                end
                branch_loc(:,1,:,i) = [x_1 offset_h/2+0.1 z_1;
                    x_1 offset_h/2+0.1 Load_Loc_m(i,3)-sign*offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1 Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h offset_h/2+0.1
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h offset_h/2+0.1 z_1-offset_h];
```

```
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,1)<header_bends(length(header_bends),1) &&
(Load_Loc_m(i,2)>=offset_h/2 || Load_Loc_m(i,2)<offset_h/2)
                %aft of header loop
                x_1 = -LOA/2+0.5;
                %set z_1 depending on port or stbd side
                if Load_Loc_m(i,2) > 0
                    z_1 = header_loc_s_alt(zones*2-1,3,3);
                else
                    z_1 = header_loc_s_alt(zones*2,3,3);
                end
                if (Load_Loc_m(i,3)-z_1)<0
                    sign = -1;
                else
                    sign = 1;
                end
                branch_loc(:,1,:,i) = [x_1 Load_Loc_m(i,2) z_1;
                    x_1 Load_Loc_m(i,2) Load_Loc_m(i,3)-sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                    x_1+offset_h Load_Loc_m(i,2) z_1-offset_h];
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
            elseif Load_Loc_m(i,2)>0
                %port
                flag = 1;
                for j=1:length(header_bends)-1 %find y location of junction
                    if header_bends(j,2)>0 %port side
                        if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>header_bends(j+1,1)
```

```
                          y_1 = header_bends(j,2);
                      end
                  elseif flag == 1; %first time through loop catches
transition from port to stbd
                      flag = 0;
                      if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>-LOA/2+0.5
                          % y_1 = header_bends(j,2)
                      end
                  end
              end
              %port mid-zones
              z_1 = header_loc_s(1,8,3);
              if (y_1-Load_Loc_m(i,2))*(Load_Loc_m(i,3)-z_1)>0
                  sign = 1;
              else
                  sign = -1;
              end
              deck = 12.5;
              if Load_Loc_m(i,3) < deck
                  branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                      Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
              else
                 branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                      Load_Loc_m(i,1) y_1 deck;
                      Load_Loc_m(i,1) y_1/2 deck;
                      Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                      Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                      Load_Loc_m(i,1) y_1/2-offset_b deck-sign*offset_b;
                      Load_Loc_m(i,1) y_1-offset_h deck-sign*offset_b;
                      Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
              end
              branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
```

```
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
                if vital(i) %vital load
                    flag = 1;
                    for j=1:length(header_bends)-1 %find y location of
junction
                        if header_bends(j,2)<=0 %stbd side
                            if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>header_bends(j+1,1)
                                y_1 = header_bends(j,2);
                            end
                        elseif flag == 1; %first time through loop catches
transition from port to stbd
                            flag = 0;
                            if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>-LOA/2+0.5
                                % y_1 = header_bends(j,2)
                            end
                        end
                    end
                    z_1 = header_loc_s(zones*2,8,3);
                    sign = sign*-1;
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1+offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                    else
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                            Load_Loc_m(i,1) y_1 deck;
                            Load_Loc_m(i,1) y_1/2 deck;
                            Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                            Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
```

```
                            Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                            Load_Loc_m(i,1) y_1/2-offset_b deck-
sign*offset_b;
                            Load_Loc_m(i,1) y_1+offset_h deck-sign*offset_b;
                            Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                    end
                    branch_gate_loc(1,2,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                    branch_gate_loc(2,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                    gate_valve_b(2,i) = gate_valve_b(1,i)+2;
                    branch_globe_loc(1,2,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                    globe_valve_b(2,i) = globe_valve_b(1,i)+1;
                end
            elseif Load_Loc_m(i,2)<=0
                %stbd
                flag = 1;
                for j=1:length(header_bends)-1 %find y location of junction
                    if header_bends(j,2)<=0 %stbd side
                        if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>header_bends(j+1,1)
                            y_1 = header_bends(j,2);
                        end
                    elseif flag == 1; %first time through loop catches
transition from port to stbd
                        flag = 0;
                        if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>-LOA/2+0.5
                            % y_1 = header_bends(j,2)
                        end
                    end
                end
                %stbd mid-zones
                z_1 = header_loc_s(zones*2,8,3);
                if (y_1-Load_Loc_m(i,2))*(Load_Loc_m(i,3)-z_1)>0
                    sign = -1;
                else
                    sign = 1;
                end
                deck = 12.5;
                if Load_Loc_m(i,3) < deck
                    branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                        Load_Loc_m(i,1) y_1 Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
```

```
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1+offset_h Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                else
                   branch_loc(:,1,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                        Load_Loc_m(i,1) y_1 deck;
                        Load_Loc_m(i,1) y_1/2 deck;
                        Load_Loc_m(i,1) y_1/2
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2)
Load_Loc_m(i,3)+sign*offset_b/2;
                        Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2+offset_b Load_Loc_m(i,3)-
sign*offset_b/2;
                        Load_Loc_m(i,1) y_1/2+offset_b deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1+offset_h deck-sign*offset_b;
                        Load_Loc_m(i,1) y_1+offset_h z_1-offset_h];
                end
                branch_gate_loc(1,1,:,i) = [branch_loc(1,1,1,i)
branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                branch_gate_loc(2,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                gate_valve_b(1,i) = gate_valve_b(1,i)+2;
                branch_globe_loc(1,1,:,i) = [branch_loc(10,1,1,i)
branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                globe_valve_b(1,i) = globe_valve_b(1,i)+1;
                if vital(i) %vital load
                    flag = 1;
                    for j=1:length(header_bends)-1 %find y location of
junction
                        if header_bends(j,2)>0 %port side
                            if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>header_bends(j+1,1)
                                y_1 = header_bends(j,2);
                            end
                        elseif flag == 1; %first time through loop catches
transition from port to stbd
                            flag = 0;
                            if Load_Loc_m(i,1)<header_bends(j,1) &&
Load_Loc_m(i,1)>-LOA/2+0.5
                                % y_1 = header_bends(j,2)
                            end
                        end
                    end
                    z_1 = header_loc_s(1,8,3);
                    sign = sign*-1;
                    deck = 12.5;
                    if Load_Loc_m(i,3) < deck
                        branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
```

```
                                Load_Loc_m(i,1) y_1
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2)
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2)
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2)
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) y_1-offset_h Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                        else
                            branch_loc(:,2,:,i) = [Load_Loc_m(i,1) y_1 z_1;
                                Load_Loc_m(i,1) y_1 deck;
                                Load_Loc_m(i,1) y_1/2 deck;
                                Load_Loc_m(i,1) y_1/2
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2)
   Load_Loc_m(i,3)+sign*offset_b/2;
                                Load_Loc_m(i,1) Load_Loc_m(i,2) Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) y_1/2-offset_b Load_Loc_m(i,3)-
   sign*offset_b/2;
                                Load_Loc_m(i,1) y_1/2-offset_b deck-
   sign*offset_b;
                                Load_Loc_m(i,1) y_1-offset_h deck-sign*offset_b;
                                Load_Loc_m(i,1) y_1-offset_h z_1-offset_h];
                        end
                        branch_gate_loc(1,2,:,i) = [branch_loc(1,1,1,i)
   branch_loc(1,1,2,i) branch_loc(1,1,3,i)]+[0 0 sign*0.15];
                        branch_gate_loc(2,2,:,i) = [branch_loc(10,1,1,i)
   branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.15];
                        gate_valve_b(2,i) = gate_valve_b(1,i)+2;
                        branch_globe_loc(1,2,:,i) = [branch_loc(10,1,1,i)
   branch_loc(10,1,2,i) branch_loc(10,1,3,i)]+[0 0 sign*0.3];
                        globe_valve_b(2,i) = globe_valve_b(1,i)+1;
                    end
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine length of branch piping
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        for j=1:9
            for k=1:2
                distance = sqrt((branch_loc(j,k,1,i)-branch_loc(j+1,k,1,i))^2+...
```

```
                (branch_loc(j,k,2,i)-branch_loc(j+1,k,2,i))^2+...
                (branch_loc(j,k,3,i)-branch_loc(j+1,k,3,i))^2);
            length_b(k,i) = length_b(k,i)+distance;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define hxchgr associated with each heat load
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hxchgr_U = zeros(1,inputs);
hxchgr_fluid_mfr = zeros(1,inputs);
hxchgr_fluid_temp_in = zeros(1,inputs);
hxchgr_weight_wet = zeros(1,inputs);
hxchgr_weight_dry = zeros(1,inputs);
hxchgr_hl = zeros(1,inputs);
hxchgr_cp = zeros(1,inputs);
hxchgr_dim = zeros(inputs,3);
hxchgr_area_pri = zeros(1,inputs);
hxchgr_area_sec = zeros(1,inputs);
hxchgr_hc = zeros(1,inputs);
hxchgr_tube_k = zeros(1,inputs);
hxchgr_tube_diam = zeros(1,inputs);
hxchgr_tube_thick = zeros(1,inputs);
hxchgr_plate_k = zeros(1,inputs);
hxchgr_plate_thick = zeros(1,inputs);
hxchgr_num_gaps = zeros(1,inputs);
for i=1:inputs
    if strcmp(Hxchgr_Type(i),'cc')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for j=1:Num_CC_Types
                if max(Load_Value_kW(i,:)) < CC_Capacity_kW(j) &&
hxchgr_capacity > CC_Capacity_kW(j)
                    hxchgr_capacity = CC_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = CC_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CC_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CC_hl_m(hxchgr_index);
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CC_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = CC_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CC_U(Hxchgr_Num(i));
```

```
            hxchgr_fluid_mfr = CC_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CC_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CC_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CC_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(Hxchgr_Num(i));
        end
    elseif strcmp(Hxchgr_Type(i),'50cc')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for j=1:Num_50_Series_Types
                if max(Load_Value_kW(i,:)) < CC_Capacity_kW(j) &&
hxchgr_capacity > CC_Capacity_kW(j)
                    hxchgr_capacity = CC_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = CC_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CC_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CC_hl_m(hxchgr_index);
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CC_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = CC_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CC_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = CC_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CC_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CC_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CC_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(Hxchgr_Num(i));
        end
```

```
    elseif strcmp(Hxchgr_Type(i),'60cc')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for
j=Num_50_Series_Types+1:Num_50_Series_Types+Num_60_Series_Types
                if max(Load_Value_kW(i,:)) < CC_Capacity_kW(j) &&
hxchgr_capacity > CC_Capacity_kW(j)
                    hxchgr_capacity = CC_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = CC_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CC_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CC_hl_m(hxchgr_index);
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CC_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = CC_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CC_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = CC_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CC_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CC_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CC_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(Hxchgr_Num(i));
        end
    elseif strcmp(Hxchgr_Type(i),'uc')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for
j=Num_50_Series_Types+Num_60_Series_Types+1:Num_50_Series_Types+Num_60_Series
_Types+Num_Unit_Cooler_Types
                if max(Load_Value_kW(i,:)) < CC_Capacity_kW(j) &&
hxchgr_capacity > CC_Capacity_kW(j)
                    hxchgr_capacity = CC_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
```

```
            hxchgr_U(i) = CC_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CC_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CC_hl_m(hxchgr_index);
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CC_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = CC_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CC_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = CC_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CC_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CC_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CC_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(Hxchgr_Num(i));
        end
    elseif strcmp(Hxchgr_Type(i),'oc')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for
j=Num_50_Series_Types+Num_60_Series_Types+Num_Unit_Cooler_Types+1:Num_CC_Type
s
                if max(Load_Value_kW(i,:)) < CC_Capacity_kW(j) &&
hxchgr_capacity > CC_Capacity_kW(j)
                    hxchgr_capacity = CC_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = CC_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CC_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CC_hl_m(hxchgr_index);
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CC_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = CC_Tube_k(hxchgr_index);
```

```
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CC_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = CC_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CC_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CC_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CC_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CC_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = 1500; %J/kg-K
            hxchgr_dim(i,:) = CC_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CC_Area_Pri_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CC_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CC_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CC_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CC_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CC_Tube_Thick_cm(Hxchgr_Num(i));
        end
    elseif strcmp(Hxchgr_Type(i),'fp')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for j=1:Num_FP_Types
                if max(Load_Value_kW(i,:)) < FP_Capacity_kW(j) &&
hxchgr_capacity > FP_Capacity_kW(j)
                    hxchgr_capacity = FP_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = FP_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = FP_Fluid_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = FP_Fluid_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = FP_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = FP_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = FP_hl_m(hxchgr_index);
            hxchgr_cp(i) = FP_Fluid_cp(hxchgr_index);
            hxchgr_dim(i,:) = FP_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = FP_Area_cm2(hxchgr_index);
            hxchgr_area_sec(i) = FP_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = FP_Fluid_hc(hxchgr_index);
            hxchgr_plate_k(i) = FP_Plate_k(hxchgr_index);
            hxchgr_plate_thick(i) = FP_Plate_Thick_cm(hxchgr_index);
            hxchgr_num_gaps(i) = FP_Num_Gaps(hxchgr_index);
        else
            hxchgr_U(i) = FP_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = FP_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = FP_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = FP_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = FP_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = FP_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = FP_Fluid_cp(Hxchgr_Num(i));
            hxchgr_dim(i,:) = FP_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = FP_Area_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = FP_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = FP_Fluid_hc(Hxchgr_Num(i));
```

```
                hxchgr_plate_k(i) = FP_Plate_k(Hxchgr_Num(i));
                hxchgr_plate_thick(i) = FP_Plate_Thick_cm(Hxchgr_Num(i));
                hxchgr_num_gaps(i) = FP_Num_Gaps(Hxchgr_Num(i));
            end
        elseif strcmp(Hxchgr_Type(i),'st')
            if isnan(Hxchgr_Num(i))
                %find closest hxchgr
                hxchgr_capacity = 100000000000000;
                for j=1:Num_ST_Types
                    if max(Load_Value_kW(i,:)) < ST_Capacity_kW(j) &&
hxchgr_capacity > ST_Capacity_kW(j)
                        hxchgr_capacity = ST_Capacity_kW(j);
                        hxchgr_index = j;
                    end
                end
                hxchgr_U(i) = ST_U(hxchgr_index);
                hxchgr_fluid_mfr(i) = ST_Fluid_Mfr_kgps(hxchgr_index);
                hxchgr_fluid_temp_in(i) = ST_Fluid_Temp_In_C(hxchgr_index);
                hxchgr_weight_wet(i) = ST_Weight_Wet_kg(hxchgr_index);
                hxchgr_weight_dry(i) = ST_Weight_Dry_kg(hxchgr_index);
                hxchgr_hl(i) = ST_hl_m(hxchgr_index);
                hxchgr_cp(i) = ST_Fluid_cp(hxchgr_index);
                hxchgr_dim(i,:) = ST_Dim_m(hxchgr_index,:);
                hxchgr_area_pri(i) = ST_Area_cm2(hxchgr_index);
                hxchgr_area_sec(i) = ST_Area_Sec_cm2(hxchgr_index);
                hxchgr_hc(i) = ST_Fluid_hc(hxchgr_index);
                hxchgr_tube_k(i) = ST_Tube_k(hxchgr_index);
                hxchgr_tube_diam(i) = ST_Tube_Diam_cm(hxchgr_index);
                hxchgr_tube_thick(i) = ST_Tube_Thick_cm(hxchgr_index);
            else
                hxchgr_U(i) = ST_U(Hxchgr_Num(i));
                hxchgr_fluid_mfr = ST_Fluid_Mfr_kgps(Hxchgr_Num(i));
                hxchgr_fluid_temp_in(i) = ST_Fluid_Temp_In_C(Hxchgr_Num(i));
                hxchgr_weight_wet(i) = ST_Weight_Wet_kg(Hxchgr_Num(i));
                hxchgr_weight_dry(i) = ST_Weight_Dry_kg(Hxchgr_Num(i));
                hxchgr_hl(i) = ST_hl_m(Hxchgr_Num(i));
                hxchgr_cp(i) = ST_Fluid_cp(Hxchgr_Num(i));
                hxchgr_dim(i,:) = ST_Dim_m(Hxchgr_Num(i),:);
                hxchgr_area_pri(i) = ST_Area_cm2(Hxchgr_Num(i));
                hxchgr_area_sec(i) = ST_Area_Sec_cm2(Hxchgr_Num(i));
                hxchgr_hc(i) = ST_Fluid_hc(Hxchgr_Num(i));
                hxchgr_tube_k(i) = ST_Tube_k(Hxchgr_Num(i));
                hxchgr_tube_diam(i) = ST_Tube_Diam_cm(Hxchgr_Num(i));
                hxchgr_tube_thick(i) = ST_Tube_Thick_cm(Hxchgr_Num(i));
            end
        elseif strcmp(Hxchgr_Type(i),'cp')
            if isnan(Hxchgr_Num(i))
                %find closest hxchgr
                hxchgr_capacity = 100000000000000;
                for j=1:Num_CP_Types
                    if max(Load_Value_kW(i,:)) < CP_Capacity_kW(j) &&
hxchgr_capacity > CP_Capacity_kW(j)
                        hxchgr_capacity = CP_Capacity_kW(j);
                        hxchgr_index = j;
                    end
```

```
            end
            hxchgr_U(i) = CP_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = CP_Mfr_Fluid_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = CP_Air_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = CP_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = CP_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = CP_hl_m(hxchgr_index);
            hxchgr_cp(i) = CP_Fluid_cp(hxchgr_index);
            hxchgr_dim(i,:) = CP_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = CP_Area_cm2(hxchgr_index);
            hxchgr_area_sec(i) = CP_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = CP_Air_hc(hxchgr_index);
            hxchgr_tube_k(i) = CP_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = CP_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = CP_Tube_Thick_cm(hxchgr_index);
            hxchgr_plate_k(i) = CP_Plate_k(hxchgr_index);
            hxchgr_plate_thick(i) = CP_Plate_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = CP_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = CP_Fluid_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = CP_Fluid_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = CP_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = CP_Weight_Dry_kg(Hxchgr_Num(i));
            hxchgr_hl(i) = CP_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = CP_Fluid_cp(Hxchgr_Num(i));
            hxchgr_dim(i,:) = CP_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = CP_Area_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = CP_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = CP_Air_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = CP_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = CP_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = CP_Tube_Thick_cm(Hxchgr_Num(i));
            hxchgr_plate_k(i) = CP_Plate_k(hxchgr_index);
            hxchgr_plate_thick(i) = CP_Plate_Thick_cm(hxchgr_index);
        end
    elseif strcmp(Hxchgr_Type(i),'o')
        if isnan(Hxchgr_Num(i))
            %find closest hxchgr
            hxchgr_capacity = 100000000000000;
            for j=1:Num_Other_Hxchgr_Types
                if max(Load_Value_kW(i,:)) < O_Capacity_kW(j) &&
hxchgr_capacity > O_Capacity_kW(j)
                    hxchgr_capacity = O_Capacity_kW(j);
                    hxchgr_index = j;
                end
            end
            hxchgr_U(i) = O_U(hxchgr_index);
            hxchgr_fluid_mfr(i) = O_Air_Mfr_kgps(hxchgr_index);
            hxchgr_fluid_temp_in(i) = O_Air_Temp_In_C(hxchgr_index);
            hxchgr_weight_wet(i) = O_Weight_Wet_kg(hxchgr_index);
            hxchgr_weight_dry(i) = O_Weight_Dry_kg(hxchgr_index);
            hxchgr_hl(i) = O_hl_m(hxchgr_index);
            hxchgr_cp(i) = O_Fluid_cp(hxchgr_index);
            hxchgr_dim(i,:) = O_Dim_m(hxchgr_index,:);
            hxchgr_area_pri(i) = O_Area_cm2(hxchgr_index);
```

```
            hxchgr_area_sec(i) = O_Area_Sec_cm2(hxchgr_index);
            hxchgr_hc(i) = O_Fluid_hc(hxchgr_index);
            hxchgr_tube_k(i) = O_Tube_k(hxchgr_index);
            hxchgr_tube_diam(i) = O_Tube_Diam_cm(hxchgr_index);
            hxchgr_tube_thick(i) = O_Tube_Thick_cm(hxchgr_index);
        else
            hxchgr_U(i) = O_U(Hxchgr_Num(i));
            hxchgr_fluid_mfr = O_Air_Mfr_kgps(Hxchgr_Num(i));
            hxchgr_fluid_temp_in(i) = O_Air_Temp_In_C(Hxchgr_Num(i));
            hxchgr_weight_wet(i) = O_Weight_Wet_kg(Hxchgr_Num(i));
            hxchgr_weight_dry(i) = O_Weight_Dry_kg(Hxhcgr_Num(i));
            hxchgr_hl(i) = O_hl_m(Hxchgr_Num(i));
            hxchgr_cp(i) = O_Fluid_cp(Hxchgr_Num(i));
            hxchgr_dim(i,:) = O_Dim_m(Hxchgr_Num(i),:);
            hxchgr_area_pri(i) = O_Area_cm2(Hxchgr_Num(i));
            hxchgr_area_sec(i) = O_Area_Sec_cm2(Hxchgr_Num(i));
            hxchgr_hc(i) = O_Fluid_hc(Hxchgr_Num(i));
            hxchgr_tube_k(i) = O_Tube_k(Hxchgr_Num(i));
            hxchgr_tube_diam(i) = O_Tube_Diam_cm(Hxchgr_Num(i));
            hxchgr_tube_thick(i) = O_Tube_Thick_cm(Hxchgr_Num(i));
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design SW aux system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('A notional SW auxiliary system is provided, which provides the SW
needed for heat rejection of the AC units.\n')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define SW pumps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_pump_dim = [1 1 1];
SW_pump_loc = [0.3*LOA 0.8*beam/2 eng_deck_ht_above_keel+SW_pump_dim(3)/2;
    0.3*LOA -0.8*beam/2 eng_deck_ht_above_keel+SW_pump_dim(3)/2;
    -0.3*LOA 0.8*beam/2 eng_deck_ht_above_keel+SW_pump_dim(3)/2;
    -0.3*LOA -0.8*beam/2 eng_deck_ht_above_keel+SW_pump_dim(3)/2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define SW valves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_valve_loc = zeros(4,2,3);
for i=1:4
    SW_valve_loc(i,:,:) = [SW_pump_loc(i,1)-2 SW_pump_loc(i,2)
SW_pump_loc(i,3);
        SW_pump_loc(i,1)-1 SW_pump_loc(i,2) SW_pump_loc(i,3)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define geometry of SW aux risers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if piping_config == 1
    port_header_deck_ht = 5.2;
    stbd_header_deck_ht = 10.2;
```

```
end
SW_risers = zeros(4,6,3);
for i=1:4
    if i==1 || i==3
        sign=1;
        riser_ht = port_header_deck_ht - 1;
    else
        sign = -1;
        riser_ht = stbd_header_deck_ht - 1;
    end
    SW_risers(i,:,:) = [SW_pump_loc(i,1)-3 SW_pump_loc(i,2) 0;
        SW_pump_loc(i,1)-3 SW_pump_loc(i,2) SW_pump_loc(i,3);
        SW_pump_loc(i,1) SW_pump_loc(i,2) SW_pump_loc(i,3);
        SW_pump_loc(i,1) SW_pump_loc(i,2) SW_pump_loc(i,3)+3;
        SW_pump_loc(i,1) SW_pump_loc(i,2)+sign*0.15*beam/2
SW_pump_loc(i,3)+3;
        SW_pump_loc(i,1) SW_pump_loc(i,2)+sign*0.15*beam/2 riser_ht];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define SW cross connect valves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_cc_valve_loc = [0.25*LOA 0 (SW_risers(1,6,3)+SW_risers(2,6,3))/2;
    -0.25*LOA 0 (SW_risers(1,6,3)+SW_risers(2,6,3))/2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define geometry of SW cross-connects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_cross_connects = zeros(2,4,3);

SW_cross_connects(1,:,:) = [0.25*LOA SW_risers(1,6,2) SW_risers(1,6,3);
    0.25*LOA 0 SW_risers(1,6,3);
    0.25*LOA 0 SW_risers(2,6,3);
    0.25*LOA SW_risers(2,6,2) SW_risers(2,6,3)];
SW_cross_connects(2,:,:) = [-0.25*LOA SW_risers(1,6,2) SW_risers(1,6,3);
    -0.25*LOA 0 SW_risers(1,6,3);
    -0.25*LOA 0 SW_risers(2,6,3);
    -0.25*LOA SW_risers(2,6,2) SW_risers(2,6,3)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define geometry of SW mains
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_mains = zeros(2,2,3);

SW_mains(1,:,:) = [0.35*LOA SW_risers(1,6,2) SW_risers(1,6,3);
    -0.475*LOA SW_risers(1,6,2) SW_risers(1,6,3)];
SW_mains(2,:,:) = [0.35*LOA SW_risers(2,6,2) SW_risers(2,6,3);
    -0.475*LOA SW_risers(2,6,2) SW_risers(2,6,3)];

figure(5)
hold on
for i=1:4

plot3(SW_risers(i,:,1),SW_risers(i,:,2),SW_risers(i,:,3),'b','Linewidth',2)
end
```

```
for i=1:2

plot3(SW_cross_connects(i,:,1),SW_cross_connects(i,:,2),SW_cross_connects(i,:
,3),'b','Linewidth',2)
end
for i=1:2
    plot3(SW_mains(i,:,1),SW_mains(i,:,2),SW_mains(i,:,3),'b','Linewidth',2)
end
axis equal
xlabel('Longitudinal Axis [m]')
ylabel('Transverse Axis [m]')
zlabel('Vertical Axis [m]')
title('3D AUX SW Mains Layout')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine other connections to SW system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
shaft_bearing = 0;
reply = input('Do you want to account for the shaft bearing? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    shaft_bearing = 1;
    shaft_bearing_loc = [-0.4*LOA 0 1+eng_deck_ht_above_keel];
    shaft_bearing_gpm = 2;
    reply = input('The default gmp flow rate through the shaft bearing is 2
gpm. Do you want to change it? [y/n]: ','s');
    if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
        shaft_bearing_gpm = input('Please enter the shaft bearing flow rate
in gpm. ');
    end
    fprintf('The default shaft bearing location is:\n')
    shaft_bearing_loc
    reply = input('Do you want to change it? [y/n]: ','s');
    if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
        fprintf('Please enter the shaft bearing location.\n');
        fprintf('Example: [-45 0 3]\n')
        shaft_bearing_loc = input('Shaft bearing location: ');
    end
end

SW_hxchgrs = 0;
reply = input('Do you want to account for other connections to the SW system?
[y/n]: ','s');
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    SW_hxchgrs = input('How many other connections? ');
    SW_hxchgr_loc = zeros(SW_hxchgrs,3);
    SW_hxchgr_gpm = zeros(SW_hxchgrs,3);
    for i=1:SW_hxchgrs
        fprintf('Please enter location %d\n',i)
        fprintf('Example: [20 3 10]\n')
        input_loc = input('Location: ');
        fprintf('Please enter flow rate %d in gpm\n',i)
        input_flow = input('Flow rate: ');
```

```
        SW_hxchgr_loc(i,:) = input_loc;
        SW_hxchgr_gpm(i) = input_flow;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design SW piping system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_piping = zeros(sum(chillers)+shaft_bearing+SW_hxchgrs,6,3);

for i=1:sum(chillers)
    if chiller_loc(i,2)>0 %chiller on port side
        SW_piping(i,:,:) = [chiller_loc(i,1)+chiller_dim(1)/2
SW_risers(1,6,2) SW_risers(1,6,3);
            chiller_loc(i,1)+chiller_dim(1)/2 SW_risers(1,6,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)+chiller_dim(1)/2 chiller_loc(i,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2 chiller_loc(i,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)+chiller_dim(2)/2+1.5 chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)+chiller_dim(2)/2+1.5 0];
    else %chiller on stbd side
        SW_piping(i,:,:) = [chiller_loc(i,1)+chiller_dim(1)/2
SW_risers(2,6,2) SW_risers(2,6,3);
            chiller_loc(i,1)+chiller_dim(1)/2 SW_risers(2,6,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)+chiller_dim(1)/2 chiller_loc(i,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2 chiller_loc(i,2)
chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2 chiller_loc(i,2)-
chiller_dim(2)/2-1.5 chiller_loc(i,3)-chiller_dim(3)*0.25;
            chiller_loc(i,1)-chiller_dim(1)/2 chiller_loc(i,2)-
chiller_dim(2)/2-1.5 0];
    end
end

if shaft_bearing == 1
    SW_piping((sum(chillers)+1),:,:) = [shaft_bearing_loc(1) SW_risers(1,6,2)
SW_risers(1,6,3);
            shaft_bearing_loc(1) SW_risers(1,6,2) shaft_bearing_loc(3);
            shaft_bearing_loc(1) shaft_bearing_loc(2) shaft_bearing_loc(3);
            shaft_bearing_loc(1) shaft_bearing_loc(2) shaft_bearing_loc(3);
            shaft_bearing_loc(1) shaft_bearing_loc(2) shaft_bearing_loc(3);
            shaft_bearing_loc(1) shaft_bearing_loc(2) shaft_bearing_loc(3)];
end

if SW_hxchgrs>=0
    for i=1:sum(SW_hxchgrs)
        if SW_hxchgr_loc(i,2)>0 %SW hxchgr on port side
            SW_piping(i+shaft_bearing+sum(chillers),:,:) =
[SW_hxchgr_loc(i,1)+1 SW_risers(1,6,2) SW_risers(1,6,3);
```

```
                SW_hxchgr_loc(i,1)+1 SW_risers(1,6,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)+1 SW_hxchgr_loc(i,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)+1 SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)+1 0];
        else %chiller on stbd side
            SW_piping(i+shaft_bearing+sum(chillers),:,:) =
[SW_hxchgr_loc(i,1)+1 SW_risers(2,6,2) SW_risers(2,6,3);
                SW_hxchgr_loc(i,1)+1 SW_risers(2,6,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)+1 SW_hxchgr_loc(i,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2) SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)-1 SW_hxchgr_loc(i,3);
                SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)-1 0];
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Locate SW segregation valves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SW_seg_valve_loc = zeros(sum(chillers),3,3);

for i=1:sum(chillers)
    if chiller_loc(i,2)>0 %chiller on port side
        SW_seg_valve_loc(i,1,:) = [chiller_loc(i,1)+chiller_dim(1)/2
(SW_risers(1,6,2)+chiller_loc(i,2))/2 chiller_loc(i,3)-chiller_dim(3)*0.25];
        SW_seg_valve_loc(i,2,:) = [chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)+chiller_dim(2)/2+0.5 chiller_loc(i,3)-chiller_dim(3)*0.25];
        SW_seg_valve_loc(i,3,:) = [chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)+chiller_dim(2)/2+1 chiller_loc(i,3)-chiller_dim(3)*0.25];
    else
        SW_seg_valve_loc(i,1,:) = [chiller_loc(i,1)+chiller_dim(1)/2
(SW_risers(2,6,2)+chiller_loc(i,2))/2 chiller_loc(i,3)-chiller_dim(3)*0.25];
        SW_seg_valve_loc(i,2,:) = [chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)-chiller_dim(2)/2-0.5 chiller_loc(i,3)-chiller_dim(3)*0.25];
        SW_seg_valve_loc(i,3,:) = [chiller_loc(i,1)-chiller_dim(1)/2
chiller_loc(i,2)-chiller_dim(2)/2-1 chiller_loc(i,3)-chiller_dim(3)*0.25];
    end
end

for i=1:sum(SW_hxchgrs)
    if SW_hxchgr_loc(i,2)>0 %SW hxchgr on port side
        SW_seg_valve_loc(i,1,:) = [SW_hxchgr_loc(i,1)+1 SW_risers(1,6,2)-1
SW_hxchgr_loc(i,3)];
        SW_seg_valve_loc(i,2,:) = [SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)+1
1];
        SW_seg_valve_loc(i,3,:) = [SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)+1
0.5];
    else %SW hxchgr on stbd side
        SW_seg_valve_loc(i,1,:) = [SW_hxchgr_loc(i,1)+1 SW_risers(2,6,2)+1
SW_hxchgr_loc(i,3)];
        SW_seg_valve_loc(i,2,:) = [SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)-1
1];
        SW_seg_valve_loc(i,3,:) = [SW_hxchgr_loc(i,1)-1 SW_hxchgr_loc(i,2)-1
0.5];
```

```
        end
    end

    figure(6)
    hold on
    for i=1:4

    plot3(SW_risers(i,:,1),SW_risers(i,:,2),SW_risers(i,:,3),'b','Linewidth',2)
    end
    for i=1:2

    plot3(SW_cross_connects(i,:,1),SW_cross_connects(i,:,2),SW_cross_connects(i,:
    ,3),'b','Linewidth',2)
    end
    for i=1:2
        plot3(SW_mains(i,:,1),SW_mains(i,:,2),SW_mains(i,:,3),'b','Linewidth',2)
    end
    for i=1:(sum(chillers)+sum(SW_hxchgrs)+shaft_bearing)
        plot3(SW_piping(i,:,1),SW_piping(i,:,2),SW_piping(i,:,3),'Color',[0 0
    0.5],'Linewidth',1.5)
    end
    axis equal
    xlabel('Longitudinal Axis [m]')
    ylabel('Transverse Axis [m]')
    zlabel('Vertical Axis [m]')
    title('3D AUX SW Mains Layout')

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Plot 3D layout w/ mains, branches, loads and valves
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure(7)
    hold on
    for i=1:4 %plot SW risers

    plot3(SW_risers(i,:,1),SW_risers(i,:,2),SW_risers(i,:,3),'c','Linewidth',2)
    end
    for i=1:2 %plot SW cross connects

    plot3(SW_cross_connects(i,:,1),SW_cross_connects(i,:,2),SW_cross_connects(i,:
    ,3),'c','Linewidth',2)
    end
    for i=1:2 %plot SW mains
        plot3(SW_mains(i,:,1),SW_mains(i,:,2),SW_mains(i,:,3),'c','Linewidth',2)
    end
    for i=1:(sum(chillers)+sum(SW_hxchgrs)+shaft_bearing) %plot SW piping
        plot3(SW_piping(i,:,1),SW_piping(i,:,2),SW_piping(i,:,3),'Color',[0 0
    0.9],'Linewidth',1.5)
    end
    for i=1:sum(chillers) %plot mains

    plot3(header_loc_s(i,:,1),header_loc_s(i,:,2),header_loc_s(i,:,3),'b','Linewi
    dth',1.5)

    plot3(header_loc_r(i,:,1),header_loc_r(i,:,2),header_loc_r(i,:,3),'r','Linewi
    dth',1.5)
```

```
plot3(header_loc_s_alt(i,:,1),header_loc_s_alt(i,:,2),header_loc_s_alt(i,:,3)
,'b','Linewidth',1.5)

plot3(header_loc_r_alt(i,:,1),header_loc_r_alt(i,:,2),header_loc_r_alt(i,:,3)
,'r','Linewidth',1.5)
end
for i=1:sum(chillers) %plot recirc line
    plot3(recirc_line(i,:,1),recirc_line(i,:,2),recirc_line(i,:,3),'b')
end
if piping_config == 2 %plot athwartship cc piping for double mains
    plot3(cc1_loc_s(:,1),cc1_loc_s(:,2),cc1_loc_s(:,3),'b','Linewidth',1.5)
    plot3(cc2_loc_s(:,1),cc2_loc_s(:,2),cc2_loc_s(:,3),'b','Linewidth',1.5)
    plot3(cc1_loc_r(:,1),cc1_loc_r(:,2),cc1_loc_r(:,3),'r','Linewidth',1.5)
    plot3(cc2_loc_r(:,1),cc2_loc_r(:,2),cc2_loc_r(:,3),'r','Linewidth',1.5)
end
plot3([LOA/2 LOA/2 -LOA/2 -LOA/2 LOA/2],[beam/2 -beam/2 -beam/2 beam/2
beam/2],[0 0 0 0 0]) %plot ship boundaries
chiller_vec_3D = [chiller_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    chiller_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    chiller_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
pump_vec_3D = [pump_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    pump_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    pump_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
seg_valve_vec_3D = [seg_valve_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1
-1];
    seg_valve_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    seg_valve_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
load_vec_3D(i,:,:) = [1/ft_per_m/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    1/ft_per_m/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    1/ft_per_m/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
for i=1:sum(chillers) %plot chillers and pumps

plot3(chiller_vec_3D(1,:)+chiller_loc(i,1),chiller_vec_3D(2,:)+chiller_loc(i,
2),chiller_vec_3D(3,:)+chiller_loc(i,3),'k','Linewidth',2)

plot3(pump_vec_3D(1,:)+pump_loc(i,1),pump_vec_3D(2,:)+pump_loc(i,2),pump_vec_
3D(3,:)+pump_loc(i,3),'k')
end
for i=1:4 %plot FM pumps

plot3(pump_vec_3D(1,:)+SW_pump_loc(i,1),pump_vec_3D(2,:)+SW_pump_loc(i,2),pum
p_vec_3D(3,:)+SW_pump_loc(i,3),'k')
end
for i=1:sum(chillers)
    for j=1:3 %plot sw seg valves

plot3(seg_valve_vec_3D(1,:)+SW_seg_valve_loc(i,j,1),seg_valve_vec_3D(2,:)+SW_
seg_valve_loc(i,j,2),seg_valve_vec_3D(3,:)+SW_seg_valve_loc(i,j,3),'k')
    end
end
for i=1:4
    for j=1:2
```

```
plot3(seg_valve_vec_3D(1,:)+SW_valve_loc(i,j,1),seg_valve_vec_3D(2,:)+SW_valv
e_loc(i,j,2),seg_valve_vec_3D(3,:)+SW_valve_loc(i,j,3),'k')
    end
end
for i=1:2

plot3(seg_valve_vec_3D(1,:)+SW_cc_valve_loc(i,1),seg_valve_vec_3D(2,:)+SW_cc_
valve_loc(i,2),seg_valve_vec_3D(3,:)+SW_cc_valve_loc(i,3),'k')
end
for i=1:length(seg_valve_loc) %plot header isolation valves

plot3(seg_valve_vec_3D(1,:)+seg_valve_loc(i,1),seg_valve_vec_3D(2,:)+seg_valv
e_loc(i,2),seg_valve_vec_3D(3,:)+seg_valve_loc(i,3),'k')
end
for i=1:inputs %plot branch piping
    plot3(branch_loc(:,1,1,i), branch_loc(:,1,2,i), branch_loc(:,1,3,i),'g')
    plot3(branch_loc(:,2,1,i), branch_loc(:,2,2,i), branch_loc(:,2,3,i),'r')
end
for i=1:inputs %plot heat load/hxchgr
    plot3(Load_Loc_m(i,1)+hxchgr_dim(i,1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -
1 -1 -1], ...
        Load_Loc_m(i,2)+hxchgr_dim(i,2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -
1 1 1], ...
        Load_Loc_m(i,3)+hxchgr_dim(i,3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1
1 -1],'k')
end
for i=1:inputs %plot branch gate & globe valves

plot3(load_vec_3D(1,:)+branch_gate_loc(1,1,1,i),load_vec_3D(2,:)+branch_gate_
loc(1,1,2,i),load_vec_3D(3,:)+branch_gate_loc(1,1,3,i),'m')

plot3(load_vec_3D(1,:)+branch_gate_loc(2,1,1,i),load_vec_3D(2,:)+branch_gate_
loc(2,1,2,i),load_vec_3D(3,:)+branch_gate_loc(2,1,3,i),'m')

plot3(load_vec_3D(1,:)+branch_globe_loc(1,1,1,i),load_vec_3D(2,:)+branch_glob
e_loc(1,1,2,i),load_vec_3D(3,:)+branch_globe_loc(1,1,3,i),'c')

plot3(load_vec_3D(1,:)+branch_gate_loc(1,2,1,i),load_vec_3D(2,:)+branch_gate_
loc(1,2,2,i),load_vec_3D(3,:)+branch_gate_loc(1,2,3,i),'m')

plot3(load_vec_3D(1,:)+branch_gate_loc(2,2,1,i),load_vec_3D(2,:)+branch_gate_
loc(2,2,2,i),load_vec_3D(3,:)+branch_gate_loc(2,2,3,i),'m')

plot3(load_vec_3D(1,:)+branch_globe_loc(1,2,1,i),load_vec_3D(2,:)+branch_glob
e_loc(1,2,2,i),load_vec_3D(3,:)+branch_globe_loc(1,2,3,i),'c')
end
axis equal
xlabel('Longitudinal Axis [m]')
ylabel('Transverse Axis [m]')
zlabel('Vertical Axis [m]')
title('3D CW System and AUX SW System Layout')

%% Output geometry to .mat file
save geometry
```

analysis.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cooling System Design Tool                             %
% Author: Ben Sanfiorenzo                                %
% Analysis module: Reads in excel data and user input %
% and creates the structure of the chilled water       %
% system. Provides 2D and 3D layout of CW structure.  %
% Last Modified: 5-13-13                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clc
clear all

%% Step 1: Load geometry data and plot CW structure

reply = input('Were modifications made to the geometry.mat file? [y/n]:
','s');
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    load analysis_interface
else
    load geometry
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot 3D layout w/ mains, branches, loads and valves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(7)
hold on
for i=1:4 %plot SW risers

plot3(SW_risers(i,:,1),SW_risers(i,:,2),SW_risers(i,:,3),'c','Linewidth',2)
end
for i=1:2 %plot SW cross connects

plot3(SW_cross_connects(i,:,1),SW_cross_connects(i,:,2),SW_cross_connects(i,:
,3),'c','Linewidth',2)
end
for i=1:2 %plot SW mains
    plot3(SW_mains(i,:,1),SW_mains(i,:,2),SW_mains(i,:,3),'c','Linewidth',2)
end
for i=1:(sum(chillers)+sum(SW_hxchgrs)+shaft_bearing) %plot SW piping
    plot3(SW_piping(i,:,1),SW_piping(i,:,2),SW_piping(i,:,3),'Color',[0 0
0.9],'Linewidth',1.5)
end
for i=1:sum(chillers) %plot mains

plot3(header_loc_s(i,:,1),header_loc_s(i,:,2),header_loc_s(i,:,3),'b','Linewi
dth',1.5)

plot3(header_loc_r(i,:,1),header_loc_r(i,:,2),header_loc_r(i,:,3),'r','Linewi
dth',1.5)
```

```
plot3(header_loc_s_alt(i,:,1),header_loc_s_alt(i,:,2),header_loc_s_alt(i,:,3)
,'b','Linewidth',1.5)

plot3(header_loc_r_alt(i,:,1),header_loc_r_alt(i,:,2),header_loc_r_alt(i,:,3)
,'r','Linewidth',1.5)
end
for i=1:sum(chillers) %plot recirc line
    plot3(recirc_line(i,:,1),recirc_line(i,:,2),recirc_line(i,:,3),'b')
end
if piping_config == 2 %plot athwartship cc piping for double mains
    plot3(cc1_loc_s(:,1),cc1_loc_s(:,2),cc1_loc_s(:,3),'b','Linewidth',1.5)
    plot3(cc2_loc_s(:,1),cc2_loc_s(:,2),cc2_loc_s(:,3),'b','Linewidth',1.5)
    plot3(cc1_loc_r(:,1),cc1_loc_r(:,2),cc1_loc_r(:,3),'r','Linewidth',1.5)
    plot3(cc2_loc_r(:,1),cc2_loc_r(:,2),cc2_loc_r(:,3),'r','Linewidth',1.5)
end
plot3([LOA/2 LOA/2 -LOA/2 -LOA/2 LOA/2],[beam/2 -beam/2 -beam/2 beam/2
beam/2],[0 0 0 0 0]) %plot ship boundaries
chiller_vec_3D = [chiller_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    chiller_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    chiller_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
pump_vec_3D = [pump_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    pump_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    pump_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
seg_valve_vec_3D = [seg_valve_dim(1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1
-1];
    seg_valve_dim(2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    seg_valve_dim(3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
load_vec_3D(i,:,:) = [1/ft_per_m/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 -1 -1];
    1/ft_per_m/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 1 1];
    1/ft_per_m/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1]];
for i=1:sum(chillers) %plot chillers and pumps

plot3(chiller_vec_3D(1,:)+chiller_loc(i,1),chiller_vec_3D(2,:)+chiller_loc(i,
2),chiller_vec_3D(3,:)+chiller_loc(i,3),'k','Linewidth',2)

plot3(pump_vec_3D(1,:)+pump_loc(i,1),pump_vec_3D(2,:)+pump_loc(i,2),pump_vec_
3D(3,:)+pump_loc(i,3),'k')
end
for i=1:4 %plot FM pumps

plot3(pump_vec_3D(1,:)+SW_pump_loc(i,1),pump_vec_3D(2,:)+SW_pump_loc(i,2),pum
p_vec_3D(3,:)+SW_pump_loc(i,3),'k')
end
for i=1:sum(chillers)
    for j=1:3 %plot sw seg valves

plot3(seg_valve_vec_3D(1,:)+SW_seg_valve_loc(i,j,1),seg_valve_vec_3D(2,:)+SW_
seg_valve_loc(i,j,2),seg_valve_vec_3D(3,:)+SW_seg_valve_loc(i,j,3),'k')
    end
end
for i=1:4
    for j=1:2
```

```
plot3(seg_valve_vec_3D(1,:)+SW_valve_loc(i,j,1),seg_valve_vec_3D(2,:)+SW_valv
e_loc(i,j,2),seg_valve_vec_3D(3,:)+SW_valve_loc(i,j,3),'k')
    end
end
for i=1:2

plot3(seg_valve_vec_3D(1,:)+SW_cc_valve_loc(i,1),seg_valve_vec_3D(2,:)+SW_cc_
valve_loc(i,2),seg_valve_vec_3D(3,:)+SW_cc_valve_loc(i,3),'k')
end
for i=1:length(seg_valve_loc) %plot header isolation valves

plot3(seg_valve_vec_3D(1,:)+seg_valve_loc(i,1),seg_valve_vec_3D(2,:)+seg_valv
e_loc(i,2),seg_valve_vec_3D(3,:)+seg_valve_loc(i,3),'k')
end
for i=1:inputs %plot branch piping
    plot3(branch_loc(:,1,1,i), branch_loc(:,1,2,i), branch_loc(:,1,3,i),'g')
    plot3(branch_loc(:,2,1,i), branch_loc(:,2,2,i), branch_loc(:,2,3,i),'r')
end
for i=1:inputs %plot heat load/hxchgr
    plot3(Load_Loc_m(i,1)+hxchgr_dim(i,1)/2*[1 1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -
1 -1 -1], ...
        Load_Loc_m(i,2)+hxchgr_dim(i,2)/2*[1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 -1 -
1 1 1], ...
        Load_Loc_m(i,3)+hxchgr_dim(i,3)/2*[-1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1
1 -1],'k')
end
for i=1:inputs %plot branch gate & globe valves

plot3(load_vec_3D(1,:)+branch_gate_loc(1,1,1,i),load_vec_3D(2,:)+branch_gate_
loc(1,1,2,i),load_vec_3D(3,:)+branch_gate_loc(1,1,3,i),'m')

plot3(load_vec_3D(1,:)+branch_gate_loc(2,1,1,i),load_vec_3D(2,:)+branch_gate_
loc(2,1,2,i),load_vec_3D(3,:)+branch_gate_loc(2,1,3,i),'m')

plot3(load_vec_3D(1,:)+branch_globe_loc(1,1,1,i),load_vec_3D(2,:)+branch_glob
e_loc(1,1,2,i),load_vec_3D(3,:)+branch_globe_loc(1,1,3,i),'c')

plot3(load_vec_3D(1,:)+branch_gate_loc(1,2,1,i),load_vec_3D(2,:)+branch_gate_
loc(1,2,2,i),load_vec_3D(3,:)+branch_gate_loc(1,2,3,i),'m')

plot3(load_vec_3D(1,:)+branch_gate_loc(2,2,1,i),load_vec_3D(2,:)+branch_gate_
loc(2,2,2,i),load_vec_3D(3,:)+branch_gate_loc(2,2,3,i),'m')

plot3(load_vec_3D(1,:)+branch_globe_loc(1,2,1,i),load_vec_3D(2,:)+branch_glob
e_loc(1,2,2,i),load_vec_3D(3,:)+branch_globe_loc(1,2,3,i),'c')
end
axis equal
xlabel('Longitudinal Axis')
ylabel('Transverse Axis')
title('3D Mains Layout')

%% Step 2: Initial guess at branch diameters, branch velocities, and branch
mass flow rates based on Q
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define thickness and diameter of copper alloy pipe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Class 200 Type I
Outer_diams_class_200 = [.25 .5 .54 .675 .84 1.050 1.315 1.66 1.9 2.375 ...
    2.875 3.5 4 4.5 5 5.563 6.625 7.625 8.625 9.625 10.75 12.75]; %inches
Thickness_class_200 = [0.035 0.035 0.065 0.065 0.065 0.065 0.065 0.072 0.072
...
    0.083 0.083 0.095 0.095 0.109 0.120 0.125 0.134 0.134 0.148 0.187 0.187
...
    0.250]; %inches
Inner_diams_class_200 = Outer_diams_class_200 - Thickness_class_200; %inches
Inner_diams_class_200_SI = Inner_diams_class_200*2.54/100; % meters
Thickness_class_200_SI = Thickness_class_200*2.54/100; % meters

%Class 200 Type II
Outer_diams_class_200_II = [.25 .5 .54 .675 .84 1.05 1.315 1.66 1.9 2.375 ...
    2.875 3.5 4 4.5 5 5.563 6.625 7.625 8.625 9.625 10.75 12.75]; %inches
Thickness_class_200_II = [.092 .198 .376 .483 .613 .779 .989 1.39 1.6 2.32
...
    2.82 3.94 4.51 5.83 7.12 8.28 10.6 12.2 15.3 21.5 24 38]; %inches
Inner_diams_class_200_II = Outer_diams_class_200_II - Thickness_class_200_II;
%inches
Inner_diams_class_200_II_SI = Inner_diams_class_200_II*2.54/100; % meters
Thickness_class_200_II_SI = Thickness_class_200_II*2.54/100; % meters

%Class 700
Outer_diams_class_700 = [.5 .54 .675 .84 1.050 1.315 1.66 1.9 2.375 ...
    2.875 3.5 4 4.5 5 5.563 6.625 7.625 8.625 9.625 10.75 12.75 14 15 16];
%inches
Thickness_class_700 = [.065 .065 .072 .072 .083 .095 .095 .109 .12 .134 .165
...
    .18 .203 .203 .22 .259 .284 .34 .34 .38 .454 .473 .503 .534]; %inches
Inner_diams_class_700 = Outer_diams_class_700 - Thickness_class_700; %inches
Inner_diams_class_700_SI = Inner_diams_class_700*2.54/100; % meters
Thickness_class_700_SI = Thickness_class_700*2.54/100; % meters

%Class 1650
Outer_diams_class_1650 = [.5 .54 .675 .75 .84 1 1.050 1.25 1.315 1.5 1.66 1.9
2 2.375 ...
    2.5 2.875 3.5 4 4.5 5 5.563 6.625 7.625 8.625 9.625 10.75 12.75]; %inches
Thickness_class_1650 = [.035 .042 .049 .058 .058 .072 .083 .095 .095 .109 .12
...
    .134 .148 .165 .18 .203 .25 .284 .34 .38 .425 .457 .526 .595 .664 .741
.879]; %inches
Inner_diams_class_1650 = Outer_diams_class_1650 - Thickness_class_1650;
%inches
Inner_diams_class_1650_SI = Inner_diams_class_1650*2.54/100; % meters
Thickness_class_1650_SI = Thickness_class_1650*2.54/100; % meters

%Class 3300
Outer_diams_class_3300 = [.125 .25 .375 .405 .5 .54 .675 .75 .84 1 1.050 1.25
...
```

```
    1.315 1.5 1.66 1.9 2 2.375 2.5 2.875 3.5]; %inches
Thickness_class_3300 = [.028 .035 .049 .058 .072 .072 .095 .109 .12 .134 .148
...
    .165 .18 .203 .22 .25 .284 .34 .34 .38 .458]; %inches
Inner_diams_class_3300 = Outer_diams_class_3300 - Thickness_class_3300;
%inches
Inner_diams_class_3300_SI = Inner_diams_class_3300*2.54/100; % meters
Thickness_class_3300_SI = Thickness_class_3300*2.54/100; % meters

%Class 6000
Outer_diams_class_6000 = [.125 .25 .375 .405 .5 .54 .675 .75 .84 1 1.050 1.25
...
    1.315 1.5 1.66 1.9 2 2.375 2.5 2.875 3.5]; %inches
Thickness_class_6000 = [.028 .058 .083 .095 .12 .12 .148 .165 .203 .22 .238
...
    .284 .3 .34 .38 .425 .454 .52 .547 .63 .76]; %inches
Inner_diams_class_6000 = Outer_diams_class_6000 - Thickness_class_6000;
%inches
Inner_diams_class_6000_SI = Inner_diams_class_6000*2.54/100; % meters
Thickness_class_6000_SI = Thickness_class_6000*2.54/100; % meters

%Telec_b = 100*ones(1,inputs); %Celcius - initial iteration assumed 100C
helec = 80*ones(1,inputs);%240*ones(1,inputs); %???
Tcold = 6.6; %Celsius = 43.88F could be as high as 47F (8.3C)
Copper_type = 1; %Choices: 1:=90-10, 2:=70-30, 3:=pure
Class_type = 2; %Choices: 1:=200, 2:=700, 3:=1650, 4:=3300, 5:=6000
%90-10 only in 200, 700; 70-30 in the rest

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determine thermal conductivity of copper alloy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Copper_type == 1)
    kcopper = 50; %90-10 copper-nickel alloy
elseif (Copper_type == 2)
    kcopper = 10; %70-30 copper-nickel alloy
elseif (Copper_type == 3)
    kcopper = 386; % pure copper
else
    kcopper = 30; %80-20 copper-nickel alloy
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Preallocate variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
D_b = zeros(1,inputs);
D_SI_b = zeros(1,inputs);
V_b = zeros(1,inputs);
V_SI_b = zeros(1,inputs);
A_b = zeros(1,inputs);
V_flow_rate_b = zeros(1,inputs);
mass_flow_rate_b = zeros(1,inputs);
thickness_b = zeros(1,inputs);
hc_b = zeros(1,inputs);
Thot_b = zeros(1,inputs);
```

```
Tave_b = zeros(1,inputs);
T1_b = zeros(1,inputs);
T2_b = zeros(1,inputs);
Telec_b = zeros(1,inputs);
Q_per_m = zeros(1,inputs);
Q_per_l = zeros(1,inputs);
Telec_b_ave = zeros(1,inputs);
delta_T_sec = zeros(1,inputs);
Telec_b_in = zeros(1,inputs);
length = zeros(1,inputs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine approximate velocity, mass flow rate, branch thickness,
% Reynolds number, convective heat transfer coefficient and
% approximate temperatures for branches independent of network
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
K = 4.5;
C = 4;
Q = max(Load_Value_kW')'*1000;
for i=1:inputs
    D_b(i) = ((4*K*Q(i)/C/pi()*0.133680556/60/.2931/12000/12^0.5)^0.4)*12;
%in inches
    D_SI_b(i) = D_b(i)/12/ft_per_m; %diameter in meters
    if (D_SI_b(i)<0.015)
        D_SI_b(i) = 0.015; %assumed minimum diameter of 15mm
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine branch thickness and actual diameter
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (Class_type == 1 && Copper_type == 1)
        if D_SI_b(i) < Inner_diams_class_200_SI(1)
            D_SI_b(i) = Inner_diams_class_200_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_200_SI))
            if ((D_SI_b(i) < Inner_diams_class_200_SI(j)) && (D_SI_b(i) >
Inner_diams_class_200_SI(j-1)))
                D_SI_b(i) = Inner_diams_class_200_SI(j);
                thickness_b(i) = Thickness_class_200_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_200_SI(max(size(Inner_diams_class_200_SI)))
            D_SI_b(i) =
Inner_diams_class_200_SI(max(size(Inner_diams_class_200_SI)));
            thickness_b(i) =
Thickness_class_200_SI(max(size(Inner_diams_class_200_SI)));
        end
    elseif (Class_type == 1 && Copper_type == 2)
        if D_SI_b(i) < Inner_diams_class_200_II_SI(1)
            D_SI_b(i) = Inner_diams_class_200_II_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_200_II_SI))
            if ((D_SI_b(i) < Inner_diams_class_200_II_SI(j)) && (D_SI_b(i) >
Inner_diams_class_200_II_SI(j-1)))
```

```
                D_SI_b(i) = Inner_diams_class_200_II_SI(j);
                thickness_b(i) = Thickness_class_200_II_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)))
            D_SI_b(i) =
Inner_diams_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)));
            thickness_b(i) =
Thickness_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)));
        end
    elseif (Class_type == 2 && Copper_type == 1)
        if D_SI_b(i) < Inner_diams_class_700_SI(1)
            D_SI_b(i) = Inner_diams_class_700_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_700_SI))
            if ((D_SI_b(i) < Inner_diams_class_700_SI(j)) && (D_SI_b(i) >
Inner_diams_class_700_SI(j-1)))
                D_SI_b(i) = Inner_diams_class_700_SI(j);
                thickness_b(i) = Thickness_class_700_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_700_SI(max(size(Inner_diams_class_700_SI)))
            D_SI_b(i) =
Inner_diams_class_700_SI(max(size(Inner_diams_class_700_SI)));
            thickness_b(i) =
Thickness_class_700_SI(max(size(Inner_diams_class_700_SI)));
        end
    elseif (Class_type == 3 && Copper_type == 1)
        if D_SI_b(i) < Inner_diams_class_1650_SI(1)
            D_SI_b(i) = Inner_diams_class_1650_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_1650_SI))
            if ((D_SI_b(i) < Inner_diams_class_1650_SI(j)) && (D_SI_b(i) >
Inner_diams_class_1650_SI(j-1)))
                D_SI_b(i) = Inner_diams_class_1650_SI(j);
                thickness_b(i) = Thickness_class_1650_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_1650_SI(max(size(Inner_diams_class_1650_SI)))
            D_SI_b(i) =
Inner_diams_class_1650_SI(max(size(Inner_diams_class_1650_SI)));
            thickness_b(i) =
Thickness_class_1650_SI(max(size(Inner_diams_class_1650_SI)));
        end
    elseif (Class_type == 4 && Copper_type == 1)
        if D_SI_b(i) < Inner_diams_class_3300_SI(1)
            D_SI_b(i) = Inner_diams_class_3300_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_3300_SI))
            if ((D_SI_b(i) < Inner_diams_class_3300_SI(j)) && (D_SI_b(i) >
Inner_diams_class_3300_SI(j-1)))
                D_SI_b(i) = Inner_diams_class_3300_SI(j);
```

```
                thickness_b(i) = Thickness_class_3300_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_3300_SI(max(size(Inner_diams_class_3300_SI)))
            D_SI_b(i) =
Inner_diams_class_3300_SI(max(size(Inner_diams_class_3300_SI)));
            thickness_b(i) =
Thickness_class_3300_SI(max(size(Inner_diams_class_3300_SI)));
        end
    elseif (Class_type == 5 && Copper_type == 1)
        if D_SI_b(i) < Inner_diams_class_6000_SI(1)
            D_SI_b(i) = Inner_diams_class_6000_SI(1);
        end
        for j = 2:max(size(Inner_diams_class_6000_SI))
            if ((D_SI_b(i) < Inner_diams_class_6000_SI(j)) && (D_SI_b(i) >
Inner_diams_class_6000_SI(j-1)))
                D_SI_b(i) = Inner_diams_class_6000_SI(j);
                thickness_b(i) = Thickness_class_6000_SI(j);
            end
        end
        if D_SI_b(i) >
Inner_diams_class_6000_SI(max(size(Inner_diams_class_6000_SI)))
            D_SI_b(i) =
Inner_diams_class_6000_SI(max(size(Inner_diams_class_6000_SI)));
            thickness_b(i) =
Thickness_class_6000_SI(max(size(Inner_diams_class_6000_SI)));
        end
    end
    D_b(i) = D_SI_b(i)*12*ft_per_m;
    V_b(i) = (C*D_b(i)^0.5); %in ft/sec
    V_SI_b(i) = V_b(i)/ft_per_m; %in m/s
    A_b(i) = (pi()*D_SI_b(i)^2)/4; %cross-sectional area
    V_flow_rate_b(i) = A_b(i)*V_SI_b(i);
    mass_flow_rate_b(i) = rho*V_flow_rate_b(i); %mass flow rate kg/m^3

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate temperatures
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    hc_b(i) = calc_hc(D_SI_b(i),V_SI_b(i),k,nu,rho,cp);
    Thot_b(i) = Q(i)/(mass_flow_rate_b(i)*cp)+Tcold; %Celsius
    Tave_b(i) = (Tcold+Thot_b(i))/2;
    T1_b(i) = Tave_b(i) + Q(i)*(hxchgr_area_pri(i)*0.0001*hc_b(i))^-1; %Inner
wall temp
    if strcmp(Hxchgr_Type(i),'fp')
        T2_b(i) = T1_b(i) +
Q(i)*hxchgr_plate_thick(i)/100*(hxchgr_area_pri(i)*0.0001*hxchgr_plate_k(i))^
-1; %Outer wall temp
    else
        Q_per_l(i) =
Q(i)*hxchgr_tube_diam(i)*pi()/100/(hxchgr_area_pri(i)*0.0001);
        T2_b(i) = T1_b(i) +
Q_per_l(i)*log((hxchgr_tube_diam(i)/2+hxchgr_tube_thick(i))/(hxchgr_tube_diam
(i)/2))/(2*pi()*kcopper); %Outer wall temp
    end
```

```
    Telec_b_ave(i) = (T2_b(i) +
Q(i)/(hxchgr_area_sec(i)*0.0001*hxchgr_hc(i))); %Secondary fluid average temp
    delta_T_sec(i) = Q(i)/hxchgr_fluid_mfr(i)/hxchgr_cp(i);
    Telec_b_in(i) = Telec_b_ave(i)+delta_T_sec(i)/2;
    Telec_b(i) = Telec_b_ave(i)-delta_T_sec(i)/2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot temperatures as a function of branch index (unordered)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Tcold_b = Tcold*ones(1,inputs);
plot(Tcold_b,'b')
hold on
plot(Tave_b,'g')
plot(Thot_b,'r')
plot(T1_b,'c')
plot(T2_b,'m')
plot(Telec_b_ave,'y')
plot(Telec_b_in,'k')
plot(Telec_b,'b:')
legend('T_c_o_l_d_ _c_w','T_a_v_e_ _c_w','T_h_o_t_
_c_w','T_1','T_2','T_a_v_e_ _s_e_c','T_i_n_ _s_e_c','T_o_u_t_ _s_e_c')
xlabel('Branch index (unordered)')
ylabel('T(C)')
title('Initial Static Temperatures as a Function of Branch Index
(unordered)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Display pipe characteristics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Pipe Characteristics Estimation\n')
for i=1:inputs
    fprintf('Load: %3.0f  Q(W): %10.4f  Diameter(m): %6.5f  Velocity(m/sec):
%6.4f  Mass flow rate(kg/s): %6.4f  Thot(C): %7.4f  Telec(C): %8.4f\n' ...
        ,i, Q(i), D_SI_b(i) ,V_SI_b(i) ,mass_flow_rate_b(i), Thot_b(i),
Telec_b(i))
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine bends in branches
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bends_90_b = zeros(2,inputs);
for i=1:inputs
    for j=1:9
        for k=1:2
            if branch_loc(j,k,1,i)~=branch_loc(j+1,k,1,i) ||
branch_loc(j,k,2,i)~=branch_loc(j+1,k,2,i) ||
branch_loc(j,k,3,i)~=branch_loc(j+1,k,3,i)
                bends_90_b(k,i) = bends_90_b(k,i)+1;
            end
        end
    end
    for j=1:2
        if bends_90_b(j,i) > 0
```

```
            bends_90_b(j,i) = bends_90_b(j,i)-1;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine number of gate valves per branch
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gate_valve_b = zeros(1,inputs);
globe_valve_b = zeros(1,inputs);
for i=1:inputs
    for j=1:2
        if branch_gate_loc(j,1,i)~=0 || branch_gate_loc(j,2,i)~=0 ||
branch_gate_loc(j,3,i)~=0
            gate_valve_b(i) = gate_valve_b(i)+1;
        end
    end
    if branch_globe_loc(1,1,i)~=0 || branch_globe_loc(1,2,i)~=0 ||
branch_globe_loc(1,3,i)~=0
        globe_valve_b(i) = globe_valve_b(i)+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine darcy friction factor for each branch
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
epsilon = 0.00005;
f_b = zeros(1,inputs);
for i=1:inputs
    f_b(i) = friction_factor(D_SI_b(i),V_SI_b(i),k,nu,epsilon,rho,cp);
end

%% Step 3: Determine network segments

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count_5=0;
size_header = size(header_loc_s); %number of header risers
size_seg_valve_loc = size(seg_valve_loc); %number of seg valves in header
curr_header_pt = [0 0 0]; %keeps track of current location in header
next_header_pt = [0 0 0]; %keeps track of next bend in header
Pressure = 50*ones(size_header(1),2,1); %Pressure stored as a vector for each
header riser [riser#, 1=cw 2=ccw, pressure vector]
Location_x = zeros(size_header(1),2,1); %Location as stored as a vector for
each point pressure is calculated [riser#, 1=cw 2=ccw, location vector]
branch_order = zeros(size_header(1),2,inputs); %[riser#, cw/ccw, branch#]
header_1 = [0 0 0];
header_2 = [0 0 0];
length_h = zeros(size_header(1),2,inputs);
bends_90_h = zeros(size_header(1),2,inputs);
gate_valve_h = zeros(size_header(1),2,inputs);
dPdX = zeros(size_header(1),2,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine corners in header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if piping_config == 1
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Note:Only includes fwd most point in header and aft most point in
    % header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    header_corner = [header_loc_s(1,6,1) header_loc_s(1,6,2)
header_loc_s(1,6,3);
        header_loc_s(size_header(1),6,1) header_loc_s(size_header(1),6,2)
header_loc_s(size_header(1),6,3)];
else
    if piping_double_config == 1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Note: Includes four corners of header as well as corners associated
        % with cross connects connecting the port and starboard supply
        % headers
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        size_header_loc_s_alt = size(header_loc_s_alt);
        header_corner = [header_loc_s(1,7,1) header_loc_s(1,7,2)
header_loc_s(1,7,3);
            header_loc_s(1,8,1) header_loc_s(1,8,2) header_loc_s(1,8,3);
            header_loc_s(2,8,1) header_loc_s(2,8,2) header_loc_s(2,8,3);
            header_loc_s(2,7,1) header_loc_s(2,7,2) header_loc_s(2,7,3);
            header_loc_s_alt(size_header_loc_s_alt(1),2,1)
header_loc_s_alt(size_header_loc_s_alt(1),2,2)
header_loc_s_alt(size_header_loc_s_alt(1),2,3);
            header_loc_s_alt(size_header_loc_s_alt(1),3,1)
header_loc_s_alt(size_header_loc_s_alt(1),3,2)
header_loc_s_alt(size_header_loc_s_alt(1),3,3);
            header_loc_s_alt(size_header_loc_s_alt(1)-1,3,1)
header_loc_s_alt(size_header_loc_s_alt(1)-1,3,2)
header_loc_s_alt(size_header_loc_s_alt(1)-1,3,3);
            header_loc_s_alt(size_header_loc_s_alt(1)-1,2,1)
header_loc_s_alt(size_header_loc_s_alt(1)-1,2,2)
header_loc_s_alt(size_header_loc_s_alt(1)-1,2,3)]; %in cw order
    else
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Note: Icludes the header bends specified by the user as well as the
        % corners associated with the cross connects connecting the port
        % and starboard supply headers
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
size_header_corner = size(header_corner); %determines how many corners are in
the header
header_loc_s_x = zeros(1,size_header(1));
header_loc_s_x_order = zeros(1,size_header(1));
header_loc_s_x_index = 1;
dPdX_header_loc_s_index = zeros(1,size_header(1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine pressure drop as a function of distance
% Order and determine branch lengths and header segment lengths
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This section of code only works for double piping simple layout.
% Need to add code to account for other two layouts. Requires too much time
% at the moment. Will come back if time remains, but need to prove rest of
% analysis program with at least one layout beforehand.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:size_header(1) %each riser section
    for n=1:2 %1=cw, 2=ccw

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Define variables - reset for each riser section and for cw/ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        dPdX_index = 2; %keeps track of next index for Pressure, Location_x,
and Location vectors
        length_h_index = 1;
        bends_90_h_index = 1;
        gate_valve_h_index = 1;
        header_loc_s_direction_queue_rm = 0;
        header_loc_s_direction_index = 1;
        branch_queue_rm = 0; %vector used to store index of branches which
are accounted for
        branch_queue_index = 1; %integer which increments to keep track of
branch_queue_rm length
        valve_queue_rm = 0; %vector used to store index of seg valves in
header which are accounted for
        valve_queue_index = 1; %integer which increments to keep track of
valve_queue_rm length
        branch_order_index = 1; %integer which keeps track of index of branch
order matrix
        curr_header_pt(1) = header_loc_s(i,6,1); %initialize curr_header_pt x
        curr_header_pt(2) = header_loc_s(i,6,2); %initialize curr_header_pt y
        curr_header_pt(3) = header_loc_s(i,6,3); %initialize curr_header_pt z
        header_1 = curr_header_pt;
        for j=1:size_header_corner(1)+1 %compute for each segnemt from
current point to next corner for each corner
            count = 1; %count number of loops - delete
            if n==1 %going cw
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Refine next_header_pt depending on which header is
                % considered
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if j==size_header_corner(1)+1
                    next_header_pt(1)=header_loc_s(i,6,1);
                    next_header_pt(2)=header_loc_s(i,6,2);
                    next_header_pt(3)=header_loc_s(i,6,3);
                else
                    if mod(i,2)==1
                        header_corner_index=j;
                    else
                        if j<=4
                            header_corner_index=j+4;
                        else
                            header_corner_index=j-4;
                        end
                    end
```

```
                    next_header_pt(1) = header_corner(header_corner_index,1);
%initialize next_header_pt x
                    next_header_pt(2) = header_corner(header_corner_index,2);
%initialize next_header_pt y
                    next_header_pt(3) = header_corner(header_corner_index,3);
%initialize next_header_pt z
                end
            else %going ccw
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Refine next_header_pt depending on which header is
considered
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if j==size_header_corner(1)+1
                    next_header_pt(1)=header_loc_s(i,6,1);
                    next_header_pt(2)=header_loc_s(i,6,2);
                    next_header_pt(3)=header_loc_s(i,6,3);
                else
                    if mod(i,2)==1
                        header_corner_index=size_header_corner(1)+1-j;
%change index from 1->8 to 8->1
                    else
                        if j<=4
                            header_corner_index=size_header_corner(1)-j-3;
%change index from 5,6,7,8,1,2,3,4 to 4,3,2,1,8,7,6,5
                        else
                            header_corner_index=size_header_corner(1)-j+5;
                        end
                    end
                    next_header_pt(1) = header_corner(header_corner_index,1);
%initialize next_header_pt x
                    next_header_pt(2) = header_corner(header_corner_index,2);
%initialize next_header_pt y
                    next_header_pt(3) = header_corner(header_corner_index,3);
%initialize next_header_pt z
                end
            end
            header_direction = next_header_pt-curr_header_pt; %determine
direction moving in header
            flag = true;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Find next branch or valve between header corners until header
            % corner is reached
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            while flag == true %header corner not reached yet
                header_direction(1) = next_header_pt(1)-curr_header_pt(1);
%curr_header_pt is initially riser location, then updated to current point of
valve or branch
                header_direction(2) = next_header_pt(2)-curr_header_pt(2);
                header_direction(3) = next_header_pt(3)-curr_header_pt(3);
                count = count+1;
                if header_direction(1)>0
                    direction = 1; %+x
                    sign=1;
                elseif header_direction(1)<0
```

```
                direction = 2; %-x
                sign=-1;
            elseif header_direction(2)>0
                direction = 3; %+y
                sign=1;
            elseif header_direction(2)<0
                direction = 4; %-y
                sign=-1;
            elseif header_direction(3)>0
                direction = 5; %+z
                sign=1;
            elseif header_direction(3)<0
                direction = 6; %-z
                sign=-1;
            else
                direction = 7; %no change
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Determine which is the closest header
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            closest_header_loc_s_direction = sign*[1000 1000 1000];
            if i==1 && n==1 %determine header_loc_s_x values
                for q=1:size_header(1)
                    header_loc_s_direction_flag = 1;
                    for r=1:max(size(header_loc_s_direction_queue_rm))
                        if q==header_loc_s_direction_queue_rm(r)
                            header_loc_s_direction_flag = 0; %not in
queue
                        end
                    end
                    header_loc_s_direction = [0 0 0];
                    header_loc_s_direction(1) = header_loc_s(q,6,1)-
curr_header_pt(1);
                    header_loc_s_direction(2) = header_loc_s(q,6,2)-
curr_header_pt(2);
                    header_loc_s_direction(3) = header_loc_s(q,6,3)-
curr_header_pt(3);
                    if direction == 1
                        if header_loc_s_direction(2)==0 &&
header_loc_s_direction(3)==0 && ...
                                (header_direction(1)-
header_loc_s_direction(1))>=0 && header_loc_s_direction(1)>=0 &&
header_loc_s_direction_flag==1
                            if (header_direction(1)-
header_loc_s_direction(1))>=(header_direction(1)-
closest_header_loc_s_direction(1))
                                closest_header_loc_s_direction =
header_loc_s_direction;
                                closest_header_loc_s_index = q;
                            end
                        end
                    elseif direction == 2
                        if header_loc_s_direction(2)==0 &&
header_loc_s_direction(3)==0 && ...
```

```
                                    (header_direction(1)-
header_loc_s_direction(1))<=0 && header_loc_s_direction(1)<=0 &&
header_loc_s_direction_flag == 1
                                if (header_direction(1)-
header_loc_s_direction(1))<=(header_direction(1)-
closest_header_loc_s_direction(1))
                                    closest_header_loc_s_direction =
header_loc_s_direction;
                                    closest_header_loc_s_index = q;
                                end
                            end
                        elseif direction == 3
                            if header_loc_s_direction(1)==0 &&
header_loc_s_direction(3)==0 && ...
                                    (header_direction(2)-
header_loc_s_direction(2))>=0 && header_loc_s_direction(2)>=0 &&
header_loc_s_direction_flag == 1
                                if (header_direction(2)-
header_loc_s_direction(2))>=(header_direction(2)-
closest_header_loc_s_direction(2))
                                    closest_header_loc_s_direction =
header_loc_s_direction;
                                    closest_header_loc_s_index = q;
                                end
                            end
                        elseif direction == 4
                            if header_loc_s_direction(1)==0 &&
header_loc_s_direction(3)==0 && ...
                                    (header_direction(2)-
header_loc_s_direction(2))<=0 && header_loc_s_direction(2)<=0 &&
header_loc_s_direction_flag == 1
                                if (header_direction(2)-
header_loc_s_direction(2))<=(header_direction(2)-
closest_header_loc_s_direction(2))
                                    closest_header_loc_s_direction =
header_loc_s_direction;
                                    closest_header_loc_s_index = q;
                                end
                            end
                        elseif direction == 5
                            if header_loc_s_direction(1)==0 &&
header_loc_s_direction(2)==0 && ...
                                    (header_direction(3)-
header_loc_s_direction(3))>=0 && header_loc_s_direction(3)>=0 &&
header_loc_s_direction_flag == 1
                                if (header_direction(3)-
header_loc_s_direction(3))>=(header_direction(3)-
closest_header_loc_s_direction(3))
                                    closest_header_loc_s_direction =
header_loc_s_direction;
                                    closest_header_loc_s_index = q;
                                end
                            end
                        elseif direction == 6
```

```
                        if header_loc_s_direction(1)==0 &&
header_loc_s_direction(2)==0 && ...
                                (header_direction(3)-
header_loc_s_direction(3))<=0 && header_loc_s_direction(3)<=0 &&
header_loc_s_direction_flag == 1
                            if (header_direction(3)-
header_loc_s_direction(3))<=(header_direction(3)-
closest_header_loc_s_direction(3))
                                closest_header_loc_s_direction =
header_loc_s_direction;
                                closest_header_loc_s_index = q;
                            end
                        end
                    end
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Determine which is the closest branch
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            closest_branch_direction = sign*[1000 1000 1000];
            for k=1:inputs
                branch_flag = 1;
                for m=1:max(size(branch_queue_rm))
                    if k==branch_queue_rm(m)
                        branch_flag = 0; %not in queue
                    end
                end
                branch_direction = [0 0 0];
                branch_direction(1) = branch_loc(1,1,1,k)-
curr_header_pt(1);
                branch_direction(2) = branch_loc(1,1,2,k)-
curr_header_pt(2);
                branch_direction(3) = branch_loc(1,1,3,k)-
curr_header_pt(3);
                if direction == 1
                    if branch_direction(2)==0 && branch_direction(3)==0
&& (header_direction(1)-branch_direction(1))>=0 && branch_direction(1)>=0 &&
branch_flag == 1
                        if (header_direction(1)-
branch_direction(1))>=(header_direction(1)-closest_branch_direction(1))
                            closest_branch_direction = branch_direction;
                            closest_branch_index = k;
                        end
                    end
                elseif direction == 2
                    if branch_direction(2)==0 && branch_direction(3)==0
&& (header_direction(1)-branch_direction(1))<=0 && branch_direction(1)<=0 &&
branch_flag == 1
                        if (header_direction(1)-
branch_direction(1))<=(header_direction(1)-closest_branch_direction(1))
                            closest_branch_direction = branch_direction;
                            closest_branch_index = k;
                        end
                    end
```

```
                    elseif direction == 3
                        if branch_direction(1)==0 && branch_direction(3)==0
&& (header_direction(2)-branch_direction(2))>=0 && branch_direction(2)>=0 &&
branch_flag == 1
                            if (header_direction(2)-
branch_direction(2))>=(header_direction(2)-closest_branch_direction(2))
                                closest_branch_direction = branch_direction;
                                closest_branch_index = k;
                            end
                        end
                    elseif direction == 4
                        if branch_direction(1)==0 && branch_direction(3)==0
&& (header_direction(2)-branch_direction(2))<=0 && branch_direction(2)<=0 &&
branch_flag == 1
                            if (header_direction(2)-
branch_direction(2))<=(header_direction(2)-closest_branch_direction(2))
                                closest_branch_direction = branch_direction;
                                closest_branch_index = k;
                            end
                        end
                    elseif direction == 5
                        if branch_direction(1)==0 && branch_direction(2)==0
&& (header_direction(3)-branch_direction(3))>=0 && branch_direction(3)>=0 &&
branch_flag == 1
                            if (header_direction(3)-
branch_direction(3))>=(header_direction(3)-closest_branch_direction(3))
                                closest_branch_direction = branch_direction;
                                closest_branch_index = k;
                            end
                        end
                    elseif direction == 6
                        if branch_direction(1)==0 && branch_direction(2)==0
&& (header_direction(3)-branch_direction(3))<=0 && branch_direction(3)<=0 &&
branch_flag == 1
                            if (header_direction(3)-
branch_direction(3))<=(header_direction(3)-closest_branch_direction(3))
                                closest_branch_direction = branch_direction;
                                closest_branch_index = k;
                            end
                        end
                    else
                    end
                end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Determine which is the closest valve
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                closest_valve_direction = sign*[1000 1000 1000];
                valve_direction = [0 0 0];
                for k=1:size_seg_valve_loc(1)
                    valve_flag = 1;
                    for m=1:max(size(valve_queue_rm))
                        if k==valve_queue_rm(m)
                            valve_flag = 0; %not in queue
                        end
```

```
                        end
                        valve_direction(1) = seg_valve_loc(k,1)-
curr_header_pt(1);
                        valve_direction(2) = seg_valve_loc(k,2)-
curr_header_pt(2);
                        valve_direction(3) = seg_valve_loc(k,3)-
curr_header_pt(3);
                        if direction == 1
                            if valve_direction(2)==0 && valve_direction(3)==0 &&
(header_direction(1)-valve_direction(1))>=0 && valve_direction(1)>=0 &&
valve_flag == 1
                                if (header_direction(1)-
valve_direction(1))>=(header_direction(1)-closest_valve_direction(1))
                                    closest_valve_direction = valve_direction;
                                    closest_valve_index = k;
                                end
                            end
                        elseif direction == 2
                            if valve_direction(2)==0 && valve_direction(3)==0 &&
(header_direction(1)-valve_direction(1))<=0 && valve_direction(1)<=0 &&
valve_flag == 1
                                if (header_direction(1)-
valve_direction(1))<=(header_direction(1)-closest_valve_direction(1))
                                    closest_valve_direction = valve_direction;
                                    closest_valve_index = k;
                                end
                            end
                        elseif direction == 3
                            if valve_direction(1)==0 && valve_direction(3)==0 &&
(header_direction(2)-valve_direction(2))>=0 && valve_direction(2)>=0 &&
valve_flag == 1
                                if (header_direction(2)-
valve_direction(2))>=(header_direction(2)-closest_valve_direction(2))
                                    closest_valve_direction = valve_direction;
                                    closest_valve_index = k;
                                end
                            end
                        elseif direction == 4
                            if valve_direction(1)==0 && valve_direction(3)==0 &&
(header_direction(2)-valve_direction(2))<=0 && valve_direction(2)<=0 &&
valve_flag == 1
                                if (header_direction(2)-
valve_direction(2))<=(header_direction(2)-closest_valve_direction(2))
                                    closest_valve_direction = valve_direction;
                                    closest_valve_index = k;
                                end
                            end
                        elseif direction == 5
                            if valve_direction(1)==0 && valve_direction(2)==0 &&
(header_direction(3)-valve_direction(3))>=0 && valve_direction(3)>=0 &&
valve_flag == 1
                                if (header_direction(3)-
valve_direction(3))>=(header_direction(3)-closest_valve_direction(3))
                                    closest_valve_direction = valve_direction;
                                    closest_valve_index = k;
```

```
                        end
                    end
                elseif direction == 6
                    if valve_direction(1)==0 && valve_direction(2)==0 &&
(header_direction(3)-valve_direction(3))<=0 && valve_direction(3)<=0 &&
valve_flag == 1
                        if (header_direction(3)-
valve_direction(3))<=(header_direction(3)-closest_valve_direction(3))
                            closest_valve_direction = valve_direction;
                            closest_valve_index = k;
                        end
                    end
                else
                    fprintf('Error with pipe geometry!!!\n') %Something
went wrong - Points not orthogonal???
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Is there a branch or a valve within the remaining segment
of
            % header?
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            if (closest_valve_direction(1) == sign*1000 &&
closest_valve_direction(2) == sign*1000 && closest_valve_direction(3) ==
sign*1000 ...
                && closest_branch_direction(1) == sign*1000 &&
closest_branch_direction(2) == sign*1000 && closest_branch_direction(3) ==
sign*1000) %account for riser later
                curr_header_pt = next_header_pt;
                flag = false;
                count_5 = count_5+1;

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Add location_x right before bend, increment
                % dPdX_index, define dPdX attributed to friction
                % add location_x right at bend, increment
                % dPdX_index, define dPdX attributed to bend friction
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if n==1 %going cw location_x propogates positively
                    Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)+sum(abs(header_direction)); %Point right before
bend
                else %going ccw location_x propogates negatively
                    Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)-sum(abs(header_direction)); %Point right before
bend
                end
                dPdX(i,n,dPdX_index) = 1;%friction
                dPdX_index = dPdX_index+1;
                Location_x(i,n,dPdX_index) = Location_x(i,n,dPdX_index-
1)+0; %Point right after bend
                dPdX(i,n,dPdX_index) = 4; %bend friction
                dPdX_index = dPdX_index+1;
```

```
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % Account for pressure due to changes in height
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    Pressure_height_h(i,n,dPdX_index-2) = -
62.31/144*header_direction(3);

                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % Determine bends in branches
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    if bends_90_h_index<=inputs
                        bends_90_h(i,n,bends_90_h_index) =
bends_90_h(i,bends_90_h_index)+1;
                    end
                else
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % Determine if next object is a branch or a bend or
                    % a riser
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    if ((direction == 1 && (closest_branch_direction(1) <
closest_valve_direction(1)) && (closest_branch_direction(1) <
closest_header_loc_s_direction(1)))||...
                            (direction == 2 && (closest_branch_direction(1) >
closest_valve_direction(1)) && (closest_branch_direction(1) >
closest_header_loc_s_direction(1)))||...
                            (direction == 3 && (closest_branch_direction(2) <
closest_valve_direction(2)) && (closest_branch_direction(2) <
closest_header_loc_s_direction(2)))||...
                            (direction == 4 && (closest_branch_direction(2) >
closest_valve_direction(2)) && (closest_branch_direction(2) >
closest_header_loc_s_direction(2)))||...
                            (direction == 5 && (closest_branch_direction(3) <
closest_valve_direction(3)) && (closest_branch_direction(3) <
closest_header_loc_s_direction(3)))||...
                            (direction == 6 && (closest_branch_direction(3) >
closest_valve_direction(3)) && (closest_branch_direction(3) >
closest_header_loc_s_direction(3)))) %branch is next closest object

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % Keep track of branch order by index
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        branch_order(i,n,branch_order_index) =
closest_branch_index; %matrix which keeps track of branch order in cw
direction for each header
                        branch_order_index = branch_order_index + 1;
%increment branch_order_index

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % Add location_x right before branch, increment
                        % dPdX_index, define dPdX attributed to friction
                        % add location_x right at branch, increment
                        % dPdX_index, define dPdX attributed to entrance
                        % effects
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                        if n==1 %going cw
                            Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)+sum(abs(closest_branch_direction)); %Point right
before branch
                        else %going ccw
                            Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)-sum(abs(closest_branch_direction)); %Point right
before branch
                        end
                        dPdX(i,n,dPdX_index) = 1;%friction
                        dPdX_index = dPdX_index+1;
                        Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)+0; %Point right after branch
                        dPdX(i,n,dPdX_index) = 2;%entrance effect
                        dPdX_index = dPdX_index+1;

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % Account for pressure due to changes in height
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Pressure_height_h(i,n,dPdX_index) = -
62.31/144*closest_branch_direction(3);

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % Redefine variable values
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        closest_direction = closest_branch_direction;
                        curr_header_pt =
branch_loc(1,1,:,closest_branch_index);
                        branch_queue_rm(branch_queue_index) =
closest_branch_index;
                        test(closest_branch_index) = 0;
                        branch_queue_index = branch_queue_index+1;

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        % Get header length
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        header_2 = curr_header_pt;
                        if flag == true
                            length_h(i,n,length_h_index) = sqrt((header_2(1)-
header_1(1))^2+(header_2(2)-header_1(2))^2+(header_2(3)-header_1(3))^2);
                        else
                            flag = true;
                            length_h_index=length_h_index-1;
                            length_h(i,n,length_h_index) =
length_h(i,n,length_h_index)+sqrt((header_2(1)-header_1(1))^2+(header_2(2)-
header_1(2))^2+(header_2(3)-header_1(3))^2);
                        end
                        length_h_index=length_h_index+1;
                        header_1=header_2;
                        bends_90_h_index = bends_90_h_index+1;
                        gate_valve_h_index = gate_valve_h_index+1;
                    elseif ((direction == 1 && closest_valve_direction(1) <
closest_header_loc_s_direction(1))||...
```

```
                              (direction == 2 && closest_valve_direction(1) >
closest_header_loc_s_direction(1))||...
                              (direction == 3 && closest_valve_direction(2) <
closest_header_loc_s_direction(2))||...
                              (direction == 4 && closest_valve_direction(2) >
closest_header_loc_s_direction(2))||...
                              (direction == 5 && closest_valve_direction(3) <
closest_header_loc_s_direction(3))||...
                              (direction == 6 && closest_valve_direction(3) >
closest_header_loc_s_direction(3))) %branch is next closest object
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         % Add location_x right before valve, increment
                         % dPdX_index, define dPdX attributed to friction
                         % add location_x right at valve, increment
                         % dPdX_index, define dPdX attributed to valve
friction
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         if n==1 %cw
                             Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)+sum(abs(closest_valve_direction)); %Point right
before valve
                         else %ccw
                             Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)-sum(abs(closest_valve_direction)); %Point right
before valve
                         end
                         dPdX(i,n,dPdX_index) = 1;%friction
                         dPdX_index = dPdX_index+1;
                         Location_x(i,n,dPdX_index) =
Location_x(i,n,dPdX_index-1)+0; %Point right after valve
                         dPdX(i,n,dPdX_index) = 3; %valve friction
                         dPdX_index = dPdX_index+1;

                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         % Account for pressure due to changes in height
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         Pressure_height_h(i,n,dPdX_index-2) = -
62.31/144*closest_valve_direction(3);

                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         % Redefine variable values
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         closest_direction = closest_valve_direction;
                         curr_header_pt =
seg_valve_loc(closest_valve_index,:);
                         valve_queue_rm(valve_queue_index) =
closest_valve_index;
                         valve_queue_index = valve_queue_index+1;

                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         % Determine index of valve locations in header
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         if gate_valve_h_index<=inputs
```

```
                            gate_valve_h(i,n,bends_90_h_index) =
gate_valve_h(i,n,gate_valve_h_index)+1;
                        end
                    elseif i==1 %header riser next object
                        header_loc_s_x(header_loc_s_x_index) =
Location_x(1,1,dPdX_index-1)+sum(abs(closest_header_loc_s_direction));
                        header_loc_s_x_order(header_loc_s_x_index) =
closest_header_loc_s_index;
                        header_loc_s_direction_queue_rm(header_loc_s_x_index)
= closest_header_loc_s_index;
                        dPdX_header_loc_s_index(header_loc_s_x_index) =
dPdX_index-1;
                        header_loc_s_x_index = header_loc_s_x_index+1;
                    end
                end
                if count > 190
                    flag = false;
                    force_escape = true;
                end
            end %while
        end %for j=1:size_header_corner(1)
    end %for n=1:2
end %for i=1:size_header(1)

%% Step 4: Refining branch velocities and mass flow rates using network
analysis accounting for bends, friction, and valves

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate mfr_total
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mfr_total = 0;
for i=1:inputs
    mfr_total = mfr_total + mass_flow_rate_b(i);
end
V_SI_h = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate area_b_unordered, Q_total
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
area_b_unordered = zeros(1,inputs);
D_SI_b_unordered = D_SI_b;
Q_total = 0;
for i=1:inputs
    area_b_unordered(i) = pi()/4*D_SI_b_unordered(i)^2;
    Q_total = Q_total + Q(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate D_h, D_SI_h
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
D_h = 1.1*((4*K*Q_total/C/pi()*0.133680556/60/.2931/12000/12^0.5)^0.4)*12;
%in inches
D_SI_h = D_h/12/ft_per_m; %diameter in meters

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Determine header thickness (thickness_h) and actual diameter (D_SI_h)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Class_type == 1 && Copper_type == 1)
    if D_SI_h < Inner_diams_class_200_SI(1)
        D_SI_h = Inner_diams_class_200_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_200_SI))
        if ((D_SI_h < Inner_diams_class_200_SI(j)) && (D_SI_h >
Inner_diams_class_200_SI(j-1)))
            D_SI_h = Inner_diams_class_200_SI(j);
            thickness_h = Thickness_class_200_SI(j);
        end
    end
    if D_SI_h > Inner_diams_class_200_SI(max(size(Inner_diams_class_200_SI)))
        D_SI_h =
Inner_diams_class_200_SI(max(size(Inner_diams_class_200_SI)));
        thickness_h =
Thickness_class_200_SI(max(size(Inner_diams_class_200_SI)));
    end
elseif (Class_type == 1 && Copper_type == 2)
    if D_SI_h < Inner_diams_class_200_II_SI(1)
        D_SI_h = Inner_diams_class_200_II_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_200_II_SI))
        if ((D_SI_h < Inner_diams_class_200_II_SI(j)) && (D_SI_h >
Inner_diams_class_200_II_SI(j-1)))
            D_SI_h = Inner_diams_class_200_II_SI(j);
            thickness_h = Thickness_class_200_II_SI(j);
        end
    end
    if D_SI_h >
Inner_diams_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)))
        D_SI_h =
Inner_diams_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)));
        thickness_h =
Thickness_class_200_II_SI(max(size(Inner_diams_class_200_II_SI)));
    end
elseif (Class_type == 2 && Copper_type == 1)
    if D_SI_h < Inner_diams_class_700_SI(1)
        D_SI_h = Inner_diams_class_700_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_700_SI))
        if ((D_SI_h < Inner_diams_class_700_SI(j)) && (D_SI_h >
Inner_diams_class_700_SI(j-1)))
            D_SI_h = Inner_diams_class_700_SI(j);
            thickness_h = Thickness_class_700_SI(j);
        end
    end
    if D_SI_h > Inner_diams_class_700_SI(max(size(Inner_diams_class_700_SI)))
        D_SI_h =
Inner_diams_class_700_SI(max(size(Inner_diams_class_700_SI)));
        thickness_h =
Thickness_class_700_SI(max(size(Inner_diams_class_700_SI)));
    end
elseif (Class_type == 3 && Copper_type == 1)
```

```
    if D_SI_h < Inner_diams_class_1650_SI(1)
        D_SI_h = Inner_diams_class_1650_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_1650_SI))
        if ((D_SI_h < Inner_diams_class_1650_SI(j)) && (D_SI_h >
Inner_diams_class_1650_SI(j-1)))
            D_SI_h = Inner_diams_class_1650_SI(j);
            thickness_h = Thickness_class_1650_SI(j);
        end
    end
    if D_SI_h >
Inner_diams_class_1650_SI(max(size(Inner_diams_class_1650_SI)))
        D_SI_h =
Inner_diams_class_1650_SI(max(size(Inner_diams_class_1650_SI)));
        thickness_h =
Thickness_class_1650_SI(max(size(Inner_diams_class_1650_SI)));
    end
elseif (Class_type == 4 && Copper_type == 1)
    if D_SI_h < Inner_diams_class_3300_SI(1)
        D_SI_h = Inner_diams_class_3300_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_3300_SI))
        if ((D_SI_h < Inner_diams_class_3300_SI(j)) && (D_SI_h >
Inner_diams_class_3300_SI(j-1)))
            D_SI_h = Inner_diams_class_3300_SI(j);
            thickness_h = Thickness_class_3300_SI(j);
        end
    end
    if D_SI_h >
Inner_diams_class_3300_SI(max(size(Inner_diams_class_3300_SI)))
        D_SI_h =
Inner_diams_class_3300_SI(max(size(Inner_diams_class_3300_SI)));
        thickness_h =
Thickness_class_3300_SI(max(size(Inner_diams_class_3300_SI)));
    end
elseif (Class_type == 5 && Copper_type == 1)
    if D_SI_h < Inner_diams_class_6000_SI(1)
        D_SI_h = Inner_diams_class_6000_SI(1);
    end
    for j = 2:max(size(Inner_diams_class_6000_SI))
        if ((D_SI_h < Inner_diams_class_6000_SI(j)) && (D_SI_h >
Inner_diams_class_6000_SI(j-1)))
            D_SI_h = Inner_diams_class_6000_SI(j);
            thickness_h = Thickness_class_6000_SI(j);
        end
    end
    if D_SI_h >
Inner_diams_class_6000_SI(max(size(Inner_diams_class_6000_SI)))
        D_SI_h =
Inner_diams_class_6000_SI(max(size(Inner_diams_class_6000_SI)));
        thickness_h =
Thickness_class_6000_SI(max(size(Inner_diams_class_6000_SI)));
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resize V_SI_b
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_temp_unordered = zeros(1,inputs);
V_SI_b_unordered = V_SI_b;
for i=1:inputs
    V_SI_b_temp_unordered(i) = V_SI_b_unordered(i);
end
clear V_SI_b_unordered;
V_SI_b_unordered = zeros(size_header(1),2,inputs);
for m=1:size_header(1)
    for n=1:2
        for i=1:inputs
            V_SI_b_unordered(m,n,i) = V_SI_b_temp_unordered(i);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Order Q, length_b, D_SI_b, and area_b_ordered
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Q_ordered = zeros(size_header(1),2,inputs);
length_b_ordered = zeros(2,size_header(1),2,inputs);
D_SI_b_ordered = zeros(size_header(1),2,inputs);
area_b_ordered = zeros(size_header(1),2,inputs);
for m=1:size_header(1)
    for n=1:2
        for i=1:inputs
            Q_ordered(m,n,i) = Q(branch_order(m,n,i));
            D_SI_b_ordered(m,n,i) = D_SI_b(branch_order(m,n,i));
            for j=1:2
                length_b_ordered(j,m,n,i) = length_b(j,branch_order(m,n,i));
%double check mode 2
            end
            area_b_ordered(m,n,i) = area_b_unordered(branch_order(m,n,i));
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recalculate area_h with new D_SI_h
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
area_h = pi()*(D_SI_h/2)^2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
velocity_delta = 10*ones(size_header(1),2);
velocity_old = zeros(size_header(1),2);
r_d = 3*ones(size_header(1),2,inputs); %assume r/d=3

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize unordered variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
length_b_unordered = length_b;
K_loss_hx_b_unordered = zeros(size_header(1),2,inputs);
f_b_unordered = zeros(size_header(1),2,inputs);
K_loss_b_unordered = zeros(size_header(1),2,inputs);
K_loss_friction_b_unordered = zeros(size_header(1),2,inputs);
K_loss_bend_90_b_unordered = zeros(size_header(1),2,inputs);
K_loss_gate_b_unordered = zeros(size_header(1),2,inputs);
K_loss_globe_b_unordered = zeros(size_header(1),2,inputs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize ordered variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
length_h_ordered = length_h;
V_SI_h_ordered = 1.5*ones(size_header(1),2,inputs); %initial guess at header
velocities
K_loss_h_ordered = zeros(size_header(1),2,inputs);
K_loss_friction_h_ordered = zeros(size_header(1),2,inputs);
K_loss_bend_90_h_ordered = zeros(size_header(1),2,inputs);
K_loss_gate_h_ordered = zeros(size_header(1),2,inputs);
K_loss_globe_h_ordered = zeros(size_header(1),2,inputs);
K_loss_check_h_ordered = zeros(size_header(1),2,inputs);
f_h_ordered = zeros(size_header(1),2,inputs);
K_loss_rh_ordered = zeros(size_header(1),2,inputs);
K_loss_friction_rh_ordered = zeros(size_header(1),2,inputs);
K_loss_bend_90_rh_ordered = zeros(size_header(1),2,inputs);
K_loss_gate_rh_ordered = zeros(size_header(1),2,inputs);
K_loss_globe_rh_ordered = zeros(size_header(1),2,inputs);
K_h_A_h_2 = zeros(size_header(1),2,inputs);
K_b_A_b_2 = zeros(size_header(1),2,inputs);
K_A_eq = zeros(size_header(1),2,inputs);
V_SI_b_ordered = zeros(size_header(1),2,inputs);
K_loss_b_ordered = zeros(size_header(1),2,inputs);
mfr_h_ordered = zeros(size_header(1),2,inputs);
mfr_b_ordered = zeros(size_header(1),2,inputs);
V_b_ordered = zeros(size_header(1),2,inputs);
V_h_ordered = zeros(size_header(1),2,inputs);
hc_b_ordered = zeros(size_header(1),2,inputs);
Thot_b_ordered = zeros(size_header(1),2,inputs);
Tave_b_ordered = zeros(size_header(1),2,inputs);
T1_b_ordered = zeros(size_header(1),2,inputs);
Q_per_l_ordered = zeros(size_header(1),2,inputs);
T2_b_ordered = zeros(size_header(1),2,inputs);
Telec_b_ave_ordered = zeros(size_header(1),2,inputs);
delta_T_sec_ordered = zeros(size_header(1),2,inputs);
Telec_b_in_ordered = zeros(size_header(1),2,inputs);
Telec_b_ordered = zeros(size_header(1),2,inputs);

for m=1:size_header(1)
    for n=1:2
        velocity_old(m,n) = V_SI_b_unordered(m,n,branch_order(m,n,i));
    end
end

for m=1:size_header(1)
```

```
    for n=1:2

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Perform loop until difference in previously and current velocity is
        % negligible, i.e., the velocity converges
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        while velocity_delta(m,n) > 10^-8

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Determine K_loss_hx_b_unordered
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                velocity_fraction = 0; %determine velocity within hxchgr and
scale accordingly
                K_loss_hx_b_unordered(m,n,i) =
hxchgr_hl(i)*2*g_mps2/1.3716^2; %guess headloss of 0.06m per 1kW, velocity of
4.5ft/s=1.3716m/s and subsequent K_loss_hx_b
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_loss_b due to friction, bends, valves
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                f_b_unordered(m,n,i) =
friction_factor(D_SI_b_unordered(i),V_SI_b_unordered(m,n,i),k,nu,epsilon,rho,
cp);

K_loss_friction_b_unordered(m,n,i)=f_b_unordered(m,n,i)*length_b_unordered(1,
i)/D_SI_b_unordered(i); %due to pipe length
                K_loss_bend_90_b_unordered(m,n,i) =
bends_90_b(1,i)*(f_b_unordered(m,n,i)*pi()/2*r_d(i)+(0.10+2.4*f_b_unordered(m
,n,i))*sin(pi()/4) ...

+6.6*f_b_unordered(m,n,i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi
()))); %due to 90 bends
                K_loss_gate_b_unordered(m,n,i) = gate_valve_b(i)*0.2; %due to
gate valves
                K_loss_globe_b_unordered(m,n,i) = globe_valve_b(i)*3.5; %due
to globe valves
                K_loss_b_unordered(m,n,i) =
K_loss_friction_b_unordered(m,n,i)+K_loss_bend_90_b_unordered(m,n,i)+...

K_loss_gate_b_unordered(m,n,i)+K_loss_globe_b_unordered(m,n,i)+K_loss_hx_b_un
ordered(i);
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_loss_h due to friction, bends, valves
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs

f_h_ordered(m,n,i)=friction_factor(D_SI_h,V_SI_h_ordered(m,n,i),k,nu,epsilon,
rho,cp);
```

```
K_loss_friction_h_ordered(m,n,i)=f_h_ordered(m,n,i)*length_h_ordered(m,n,i)/D
_SI_h; %due to pipe length based on first branch Darcy friction factor
                K_loss_bend_90_h_ordered(m,n,i) =
bends_90_h(m,n,i)*(f_h_ordered(m,n,i)*pi()/2*r_d(i)+(0.10+2.4*f_h_ordered(m,n
,i))*sin(pi()/4) ...

+6.6*f_h_ordered(m,n,i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(m,n,i)^(4*pi()/2/
pi())); %due to 90 bends
                K_loss_gate_h_ordered(m,n,i) = gate_valve_h(m,n,i)*0.2;
            %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
            %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
                K_loss_h_ordered(m,n,i) =
K_loss_friction_h_ordered(m,n,i)+K_loss_bend_90_h_ordered(m,n,i)+K_loss_gate_
h_ordered(m,n,i);%+ ...
            %        K_loss_globe_h(i)+K_loss_check_h(i);
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_loss_rh due to friction, bends, valves
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
            %    K_loss_friction_rh(i)=f_b(1)*length_rh(i)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
            %    K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_b(1)*pi()/2*r_d(i)+(0.10+2.4*f_b(1))*sin(pi()/4) ...
            %
+6.6*f_b(1)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
            %    K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
            %    K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
            %    K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_gate_rh(i)+K_loss_globe_rh(
i);
            K_loss_rh_ordered(m,n,i) = K_loss_h_ordered(m,n,i); %assume same
loss coefficient for supply and return header segments
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                order = branch_order(m,n,i);
                K_h_A_h_2(m,n,i) =
(K_loss_h_ordered(m,n,i)+K_loss_rh_ordered(m,n,i))/area_h^2;
                K_loss_b_ordered(m,n,i) = K_loss_b_unordered(m,n,order);
                K_b_A_b_2(m,n,i) =
K_loss_b_ordered(m,n,i)/(area_b_ordered(m,n,i))^2;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_A_eq
```

```
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=inputs:-1:1
                if i==inputs
                    K_A_eq(m,n,i)=K_b_A_b_2(m,n,i);
                else
                    K_A_eq(m,n,i) =
(1/(1/K_b_A_b_2(m,n,i)^0.5+1/(K_A_eq(m,n,i+1)+K_h_A_h_2(m,n,i+1))^0.5))^2;
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Initialize variables
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            mfr_left = mfr_total;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Determine branch and header velocities
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                order = branch_order(m,n,i);
                mfr_h_ordered(m,n,i) = mfr_left;
                mfr_b_ordered(m,n,i) =
mfr_left*(K_A_eq(m,n,i)/K_b_A_b_2(m,n,i))^0.5;
                mfr_left = mfr_left - mfr_b_ordered(m,n,i);
                if i == inputs
                    velocity_old(m,n) = V_SI_b_ordered(m,n,i);
                end
                V_SI_b_ordered(m,n,i) =
mfr_b_ordered(m,n,i)/rho/area_b_ordered(m,n,i);
                V_b_ordered(m,n,i) = V_SI_b_ordered(m,n,i)*ft_per_m;
                V_SI_h_ordered(m,n,i) = mfr_h_ordered(m,n,i)/rho/area_h;
                V_h_ordered(m,n,i) = V_SI_h_ordered(m,n,i)*ft_per_m;
                if i == inputs
                    velocity_delta(m,n) = abs(V_SI_b_ordered(m,n,i)-
velocity_old(m,n));
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate temperatures
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                order = branch_order(m,n,i);
                hc_b_ordered(m,n,i) =
calc_hc(D_SI_b_ordered(m,n,i),V_SI_b_ordered(m,n,i),k,nu,rho,cp);
                Thot_b_ordered(m,n,i) =
Q_ordered(m,n,i)/(mfr_b_ordered(m,n,i)*cp)+Tcold; %Celsius
                Tave_b_ordered(m,n,i) = (Tcold+Thot_b_ordered(m,n,i))/2;
                T1_b_ordered(m,n,i) = Tave_b_ordered(m,n,i) +
Q_ordered(m,n,i)*(hxchgr_area_pri(order)*0.0001*hc_b_ordered(m,n,i))^-1;
%Inner wall temp
                if strcmp(Hxchgr_Type(order),'fp')
```

```
                    T2_b_ordered(m,n,i) = T1_b_ordered(m,n,i) +
Q_ordered(m,n,i)*hxchgr_plate_thick(order)/100*(hxchgr_area_pri(order)*0.0001
*hxchgr_plate_k(order))^-1; %Inner wall temp
                else
                    Q_per_l_ordered(m,n,i) =
Q_ordered(m,n,i)*hxchgr_tube_diam(order)*pi()/100/(hxchgr_area_pri(order)*0.0
001);
                    T2_b_ordered(m,n,i) = T1_b_ordered(m,n,i) +
Q_per_l_ordered(m,n,i)*log((hxchgr_tube_diam(order)/2+hxchgr_tube_thick(order
))/(hxchgr_tube_diam(order)/2))/(2*pi()*kcopper); %Outer wall temp
                end
                Telec_b_ave_ordered(m,n,i) = (T2_b_ordered(m,n,i) +
Q_ordered(m,n,i)/(hxchgr_area_sec(order)*0.0001*hxchgr_hc(order)));
%Electrical component temp
                delta_T_sec_ordered(m,n,i) =
Q_ordered(m,n,i)/hxchgr_fluid_mfr(order)/hxchgr_cp(order);
                Telec_b_in_ordered(m,n,i) =
Telec_b_ave_ordered(m,n,i)+delta_T_sec_ordered(m,n,i)/2;
                Telec_b_ordered(m,n,i) = Telec_b_ave_ordered(m,n,i)-
delta_T_sec_ordered(m,n,i)/2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Display refined velocities, etc.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        fprintf('Second Step: Refined Velocities\n')
        for i=1:inputs
            fprintf('Load: %2.0f %2.0f %3.0f  Q(W): %10.4f  Diameter(m):
%6.5f  Velocity(m/sec): %6.4f  Mass flow rate(kg/s): %6.4f  Thot(C): %7.4f
Telec(C): %8.4f\n' ...
                ,m, n, i, Q_ordered(m,n,i), D_SI_b_ordered(m,n,i)
,V_SI_b_ordered(m,n,i) ,mfr_b_ordered(m,n,i), Thot_b_ordered(m,n,i),
Telec_b_ordered(m,n,i))
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine least and greatest branch velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lowest_vel = 10000*ones(size_header(1),2);
greatest_vel = zeros(size_header(1),2);
for m=1:size_header(1)
    for n=1:2
        for i=1:inputs
            %lowest_index = 1;
            %greatest_index = 1;
            if V_SI_b_ordered(m,n,i)<lowest_vel(m,n)
                lowest_vel(m,n) = V_SI_b_ordered(m,n,i);
                %lowest_index = i;
            elseif V_SI_b_ordered(m,n,i)>greatest_vel(m,n)
                greatest_vel(m,n) = V_SI_b_ordered(m,n,i);
                %greatest_index = i;
            end
```

```
        end
    end
end
lowest_vel
greatest_vel

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define V_SI_b_2 and V_SI_h_1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_2 = zeros(1,inputs);
V_SI_h_1 = zeros(1,inputs);
for i=1:inputs
    V_SI_b_2(i) = V_SI_b_ordered(1,1,i);
    V_SI_h_1(i) = V_SI_h_ordered(1,1,i);
end

%% Step 5: Account for entrance, exit effects with refined velocities,
K_loss, f_b, f_h for each riser going cw and ccw

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
K_loss_entrance_b_ordered = zeros(size_header(1),2,inputs);
K_loss_exit_b_ordered = zeros(size_header(1),2,inputs);
K_loss_b_in_ordered = zeros(size_header(1),2,inputs);
r_d3 = 0.1;
K_loss_entrance_h_ordered = zeros(size_header(1),2,inputs);
K_loss_exit_h_ordered = zeros(size_header(1),2,inputs);
K_loss_entrance_rh_ordered = zeros(size_header(1),2,inputs);
K_loss_exit_rh_ordered = zeros(size_header(1),2,inputs);
f_b_ordered = zeros(size_header(1),2,inputs);
K_loss_friction_b_ordered = zeros(size_header(1),2,inputs);
K_loss_bend_90_b_ordered = zeros(size_header(1),2,inputs);
K_loss_gate_b_ordered = zeros(size_header(1),2,inputs);
K_loss_globe_b_ordered = zeros(size_header(1),2,inputs);
K_loss_b_ordered = zeros(size_header(1),2,inputs);

for m=1:size_header(1)
    for n=1:2
        velocity_delta(m,n) = 10;
        velocity_old(m,n) = V_SI_b_ordered(m,n,inputs);
        counter = 0;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Refine velocities
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        while counter<10%(velocity_delta(m,n) > 0.000001)
            counter = counter+1;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate loss coefficient for branches due to friction, bends,
            % valves, entrance and exit effects (in order wrt header)
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
```

```
                order = branch_order(m,n,i);
                f_b_ordered(m,n,i) =
friction_factor(D_SI_b_ordered(m,n,i),V_SI_b_ordered(m,n,i),k,nu,epsilon,rho,
cp);

K_loss_friction_b_ordered(m,n,i)=f_b_ordered(m,n,i)*length_b_ordered(1,m,n,i)
/D_SI_b_ordered(m,n,i); %due to pipe length
                K_loss_bend_90_b_ordered(m,n,i) =
bends_90_b(1,order)*(f_b_ordered(m,n,i)*pi()/2*r_d(m,n,i)+(0.10+2.4*f_b_order
ed(m,n,i))*sin(pi()/4) ...

+6.6*f_b_ordered(m,n,i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(m,n,i)^(4*pi()/2/
pi())); %due to 90 bends
                K_loss_gate_b_ordered(m,n,i) = gate_valve_b(order)*0.2; %due
to gate valves
                K_loss_globe_b_ordered(m,n,i) = globe_valve_b(order)*3.5;
%due to globe valves
                K_loss_b_ordered(m,n,i) =
K_loss_friction_b_ordered(m,n,i)+K_loss_bend_90_b_ordered(m,n,i)+K_loss_gate_
b_ordered(m,n,i)+...

K_loss_globe_b_ordered(m,n,i)+K_loss_hx_b_unordered(order);

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Calculate entrance and exit effects for branch
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                Keq = 0.57-1.07*r_d3^0.5-2.13*r_d3+8.24*r_d3^1.5-
8.48*r_d3^2+2.9*r_d3^2.5;
                K_loss_entrance_b_ordered(m,n,i) = (0.81-
1.13*mfr_h_ordered(m,n,i)/mfr_b_ordered(m,n,i) + ...

mfr_h_ordered(m,n,i)^2/mfr_b_ordered(m,n,i)^2)*D_SI_b_ordered(m,n,i)^4/D_SI_h
^4 + ...
                    1.12*D_SI_b_ordered(m,n,i)/D_SI_h-
1.08*D_SI_b_ordered(m,n,i)^3/D_SI_h^3 + Keq;%due to entrance; assume r/d3 =
0.1
                Cyc = 1-0.25*(D_SI_b_ordered(m,n,i)/D_SI_h)^1.3-(0.11*r_d3-
0.65*r_d3^2+0.83*r_d3^3)*D_SI_b_ordered(m,n,i)^2/D_SI_h^2;
                Cxc = 0.08+0.56*r_d3-1.75*r_d3^2+1.83*r_d3^3;
                Cm = 0.23+1.46*r_d3-2.75*r_d3^2+1.65*r_d3^3;
                K_loss_exit_b_ordered(m,n,i) = 2*Cyc-
1+D_SI_b_ordered(m,n,i)^4/D_SI_h^4*(2*(Cxc-1)+...
                    2*(2-Cxc-Cm)*mfr_h_ordered(m,n,i)/mfr_b_ordered(m,n,i)-
0.92*mfr_h_ordered(m,n,i)^2/mfr_b_ordered(m,n,i)^2);%due to exit; assume r/d3
= 0.1

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Calculate K_loss_b and K_loss_b_in
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                K_loss_b_ordered(m,n,i) =
K_loss_friction_b_ordered(m,n,i)+K_loss_bend_90_b_ordered(m,n,i)+K_loss_gate_
b_ordered(m,n,i) ...

+K_loss_globe_b_ordered(m,n,i)+K_loss_hx_b_unordered(order)+K_loss_entrance_b
_ordered(m,n,i)+K_loss_exit_b_ordered(m,n,i);
```

```
                K_loss_b_in_ordered(m,n,i) =
K_loss_friction_b_ordered(m,n,i)/2+K_loss_bend_90_b_ordered(m,n,i)/2+K_loss_g
ate_b_ordered(m,n,i)/2 ...

+K_loss_globe_b_ordered(m,n,i)*0+K_loss_hx_b_unordered(order)*0+K_loss_entran
ce_b_ordered(m,n,i)+K_loss_exit_b_ordered(m,n,i)*0;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate loss coefficient for supply header due to friction,
bends,
            % valves, entrance and exit effects
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                f_h_ordered(m,n,i) =
friction_factor(D_SI_h,V_SI_h_ordered(m,n,i),k,nu,epsilon,rho,cp);

K_loss_friction_h_ordered(m,n,i)=f_h_ordered(m,n,i)*length_h_ordered(m,n,i)/D
_SI_h; %due to pipe length based on revised Darcy friction factor
                K_loss_bend_90_h_ordered(m,n,i) =
bends_90_h(m,n,i)*(f_h_ordered(m,n,i)*pi()/2*r_d(m,n,i)+(0.10+2.4*f_h_ordered
(m,n,i))*sin(pi()/4) ...

+6.6*f_h_ordered(m,n,i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(m,n,i)^(4*pi()/2/
pi())); %due to 90 bends
                K_loss_gate_h_ordered(m,n,i) = gate_valve_h(m,n,i)*0.2;
                %K_loss_globe_h_ordered(i) = globe_valve_h(i)*3.5;
                %K_loss_check_h_ordered(i) = check_valve_h(i)*2;

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Calculate entrance effects for header segments
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if i==inputs
                    K_loss_entrance_h_ordered(m,n,i) = 0;
                else
                    K_loss_entrance_h_ordered(m,n,i) = 0.62-
0.98*mfr_h_ordered(m,n,i)/mfr_h_ordered(m,n,i+1)+ ...

0.36*(mfr_h_ordered(m,n,i)/mfr_h_ordered(m,n,i+1))^2+0.03*(mfr_h_ordered(m,n,
i+1)/mfr_h_ordered(m,n,i))^6;
                end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Calculate K_loss_h
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                K_loss_h_ordered(m,n,i) =
K_loss_friction_h_ordered(m,n,i)+K_loss_bend_90_h_ordered(m,n,i)+ ...

K_loss_gate_h_ordered(m,n,i)+K_loss_entrance_h_ordered(m,n,i);
%K_loss_globe_h(i)+K_loss_check_h(i);
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate K_loss_rh due to friction, bends, valves
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            %K_loss_friction_rh(i)=f_h(i)*length_rh(i)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
            %K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_h(i)*pi()/2*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/4) ...
            %
+6.6*f_h(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
            %K_loss_bend_180_rh(i) =
bends_180_rh(i)*(f_h(i)*pi()*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/2) ...
            %
+6.6*f_h(i)*((sin(pi()/2))^0.5+sin(pi()/2))/r_d(i)^(4*pi()/pi())); %due to
180 bends
            %K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
            %K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate exit effects for header segments
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            if i==inputs
                K_loss_entrance_rh_ordered(m,n,i) = 0;
            else
                K_loss_entrance_rh_ordered(m,n,i) = 0.62-
0.98*mfr_h_ordered(m,n,i)/mfr_h_ordered(m,n,i+1)+...

0.36*(mfr_h_ordered(m,n,i)/mfr_h_ordered(m,n,i+1))^2+0.03*(mfr_h_ordered(m,n,
i+1)/mfr_h_ordered(m,n,i))^6; %exit
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %Calculate K_loss_h
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            K_loss_rh_ordered(m,n,i) = K_loss_h_ordered(m,n,i)-
K_loss_entrance_h_ordered(m,n,i)+K_loss_entrance_rh_ordered(m,n,i);
            %K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_bend_180_rh(i)+K_loss_gate_
rh(i)+ ...
            %    K_loss_globe_rh(i)+K_loss_entrance_rh(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            K_h_A_h_2(m,n,i) =
(K_loss_h_ordered(m,n,i)+K_loss_rh_ordered(m,n,i))/area_h^2;
            K_b_A_b_2(m,n,i) =
K_loss_b_ordered(m,n,i)/area_b_ordered(m,n,i)^2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_eq
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
            for i=inputs:-1:1
                if i==inputs
                    K_A_eq(m,n,i)=K_b_A_b_2(m,n,i);
                else
                    K_A_eq(m,n,i) =
(1/(1/K_b_A_b_2(m,n,i)^0.5+1/(K_A_eq(m,n,i+1)+K_h_A_h_2(m,n,i+1))^0.5))^2;
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Initialize variables
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            mfr_left = mfr_total;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Determine branch and header velocities
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                order = branch_order(m,n,i);
                mfr_h_ordered(m,n,i) = mfr_left;
                mfr_b_ordered(m,n,i) =
mfr_left*(K_A_eq(m,n,i)/K_b_A_b_2(m,n,i))^0.5;
                mfr_left = mfr_left - mfr_b_ordered(m,n,i);
                if i == inputs
                    velocity_old(m,n) = V_SI_b_ordered(m,n,i);
                end
                V_SI_b_ordered(m,n,i) =
mfr_b_ordered(m,n,i)/rho/area_b_ordered(m,n,i);
                V_b_ordered(m,n,i) = V_SI_b_ordered(m,n,i)*ft_per_m;
                V_SI_h_ordered(m,n,i) = mfr_h_ordered(m,n,i)/rho/area_h;
                V_h_ordered(m,n,i) = V_SI_h_ordered(m,n,i)*ft_per_m;
                if i == inputs
                    velocity_delta(m,n) = abs(V_SI_b_ordered(m,n,i)-
velocity_old(m,n));
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate temperatures
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                order = branch_order(m,n,i);
                hc_b_ordered(m,n,i) =
calc_hc(D_SI_b_ordered(m,n,i),V_SI_b_ordered(m,n,i),k,nu,rho,cp);
                Thot_b_ordered(m,n,i) =
Q_ordered(m,n,i)/(mfr_b_ordered(m,n,i)*cp)+Tcold; %Celsius
                Tave_b_ordered(m,n,i) = (Tcold+Thot_b_ordered(m,n,i))/2;
                T1_b_ordered(m,n,i) = Tave_b_ordered(m,n,i) +
Q_ordered(m,n,i)*(hxchgr_area_pri(order)*0.0001*hc_b_ordered(m,n,i))^-1;
%Inner wall temp
                if strcmp(Hxchgr_Type(order),'fp')
                    T2_b_ordered(m,n,i) = T1_b_ordered(m,n,i) +
Q_ordered(m,n,i)*hxchgr_plate_thick(order)/100*(hxchgr_area_pri(order)*0.0001
*hxchgr_plate_k(order))^-1; %Inner wall temp
```

```
                else
                    Q_per_l_ordered(m,n,i) =
Q_ordered(m,n,i)*hxchgr_tube_diam(order)*pi()/100/(hxchgr_area_pri(order)*0.0
001);
                    T2_b_ordered(m,n,i) = T1_b_ordered(m,n,i) +
Q_per_l_ordered(m,n,i)*log((hxchgr_tube_diam(order)/2+hxchgr_tube_thick(order
))/(hxchgr_tube_diam(order)/2))/(2*pi()*kcopper); %Outer wall temp
                end
                Telec_b_ave_ordered(m,n,i) = (T2_b_ordered(m,n,i) +
Q_ordered(m,n,i)/(hxchgr_area_sec(order)*0.0001*hxchgr_hc(order)));
%Electrical component temp
                delta_T_sec_ordered(m,n,i) =
Q_ordered(m,n,i)/hxchgr_fluid_mfr(order)/hxchgr_cp(order);
                Telec_b_in_ordered(m,n,i) =
Telec_b_ave_ordered(m,n,i)+delta_T_sec_ordered(m,n,i)/2;
                Telec_b_ordered(m,n,i) = Telec_b_ave_ordered(m,n,i)-
delta_T_sec_ordered(m,n,i)/2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Display refined velocities, etc.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        fprintf('Third Step: Entrance and exit effects\n')
        for i=1:inputs
            fprintf('Load: %2.0f %2.0f %3.0f  Q(W): %10.4f  Diameter(m):
%6.5f  Velocity(m/sec): %6.4f  Mass flow rate(kg/s): %6.4f  Thot(C): %7.4f
Telec(C): %8.4f\n' ...
                ,m, n, i, Q_ordered(m,n,i), D_SI_b_ordered(m,n,i)
,V_SI_b_ordered(m,n,i) ,mfr_b_ordered(m,n,i), Thot_b_ordered(m,n,i),
Telec_b_ordered(m,n,i))
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define V_SI_b_1, V_SI_b_2, V_SI_b_3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_1 = V_SI_b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define V_SI_b_3 and V_SI_h_2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_3 = zeros(1,inputs);
V_SI_h_2 = zeros(1,inputs);
for i=1:inputs
    V_SI_b_3(i) = V_SI_b_ordered(1,1,i);
    V_SI_h_2(i) = V_SI_h_ordered(1,1,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Show changes in temperature for each step in the program with flow
% initialted from the riser going cw
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
input_vec = 1:inputs;
branch_limit = 9/3.28084*ones(1,inputs);
header_limit = 12/3.28084*ones(1,inputs);
figure(7)
subplot(2,1,1)
plot(input_vec,V_SI_b_1,'r')
hold on
plot(input_vec,V_SI_b_2,'g')
plot(input_vec,V_SI_b_3,'k')
plot(input_vec,branch_limit,'m')
title('Chilled Water Velocity in Branch Piping as a Function of Branch
Junction Index')
xlabel('Branch Index')
ylabel('Branch Velocity (m/s)')
legend('Initial','Intermediate','Final','Limit')
subplot(2,1,2)
plot(input_vec,V_SI_h_1,'r')
hold on
plot(input_vec,V_SI_h_2,'k')
plot(input_vec,header_limit,'m')
title('Chilled Water Velocity in Supply Header as a Function of Branch
Junction Index')
xlabel('Branch Index')
ylabel('Header Velocity (m/s)')
legend('Intermediate','Final','Limit')

%% Step 6: Determine pressure drop as a function of distance for each riser
going cw and ccw

rho_w = 62.421; %[lb/ft^3]
g_fps2 = 32.1740; %[ft/sec^2]
lbm_per_kg = 2.20462;
lb_p_in2_to_pa = 6894.76;
dPdX_mfr_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_mfr_h_2 = zeros(size_header(1),2,max(size(dPdX)));
dPdX_K_loss_friction_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_K_loss_bend_90_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_K_loss_valve_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_fric_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_bend_h = zeros(size_header(1),2,max(size(dPdX)));
dPdX_valve_h = zeros(size_header(1),2,max(size(dPdX)));
for m=1:size_header(1)
    for n=1:2
        mfr_h_index = 1;
        fric_h_index = 1;
        bend_h_index = 1;
        for i=1:max(size(dPdX))
            if dPdX(m,n,i)==2 %branch
                mfr_h_index = mfr_h_index+1;
                dPdX_K_loss_friction_h(m,n,i) = 0;
                dPdX_K_loss_bend_90_h(m,n,i) = 0;
                dPdX_K_loss_valve_h(m,n,i) = 0;
            elseif dPdX(m,n,i)==1 %friction
                if mfr_h_index <= inputs
```

```
                    dPdX_K_loss_friction_h(m,n,i) = 
friction_factor(D_SI_h,V_SI_h_ordered(m,n,mfr_h_index),k,nu,epsilon,rho,cp)*.
..
                        abs(Location_x(m,n,i)-Location_x(m,n,i-1))/D_SI_h; 
%due to pipe length based on revised Darcy friction factor;
                else
                    dPdX_K_loss_friction_h(m,n,i) = 0;
                end
                dPdX_K_loss_bend_90_h(m,n,i) = 0;
                dPdX_K_loss_valve_h(m,n,i) = 0;
            elseif dPdX(m,n,i)==3 %valve
                if mfr_h_index <= inputs
                    dPdX_K_loss_valve_h(m,n,i) = 0.2;
                else
                    dPdX_K_loss_friction_h(m,n,i) = 0;
                end
                dPdX_K_loss_friction_h(m,n,i) = 0;
                dPdX_K_loss_bend_90_h(m,n,i) = 0;
            elseif dPdX(m,n,i)==4 %bend
                dPdX_K_loss_friction_h(m,n,i) = 0;
                dPdX_r_d = 3;
                if mfr_h_index <= inputs
                    dPdX_f_h = 
friction_factor(D_SI_h,V_SI_h_ordered(m,n,mfr_h_index),k,nu,epsilon,rho,cp);
                    dPdX_K_loss_bend_90_h(m,n,i) = 
1*(dPdX_f_h*pi()/2*dPdX_r_d+(0.10+2.4*dPdX_f_h)*sin(pi()/4) ...

+6.6*dPdX_f_h*((sin(pi()/4))^0.5+sin(pi()/4))/dPdX_r_d^(4*pi()/2/pi())); %due 
to 90 bends;
                else
                    dPdX_K_loss_bend_90_h(m,n,i) = 0;
                end
                dPdX_K_loss_friction_h(m,n,i) = 0;
                dPdX_K_loss_valve_h(m,n,i) = 0;
            else
                dPdX_K_loss_friction_h(m,n,i) = 0;
                dPdX_K_loss_bend_90_h(m,n,i) = 0;
                dPdX_K_loss_valve_h(m,n,i) = 0;
            end
            if mfr_h_index <= inputs-1
                dPdX_mfr_h(m,n,i) = mfr_h_ordered(m,n,mfr_h_index);
                dPdX_mfr_h_2(m,n,i) = mfr_h_ordered(m,n,mfr_h_index+1);
            elseif mfr_h_index == inputs
                dPdX_mfr_h(m,n,i) = mfr_h_ordered(m,n,mfr_h_index);
                dPdX_mfr_h_2(m,n,i) = 0;
            else
                dPdX_mfr_h(m,n,i) = 0;
                dPdX_mfr_h_2(m,n,i) = 0;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert mfr_h to wfr_h
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m=1:size_header(1)
    for n=1:2
        for i=1:max(size(dPdX_mfr_h))
            dPdX_wfr_h(m,n,i) = dPdX_mfr_h(m,n,i)*lbm_per_kg;
            dPdX_wfr_h_2(m,n,i) = dPdX_mfr_h_2(m,n,i)*lbm_per_kg;
        end
    end
end

Pressure_SI = zeros(size_header(1),2,max(size(Pressure)));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine pressures at locations along Location_x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m=1:size_header(1)
    for n=1:2
        for i=1:max(size(Location_x))
            dPdX_fric_h(m,n,i) =
(dPdX_K_loss_friction_h(m,n,i)*dPdX_wfr_h(m,n,i)^2)/(288*(area_h*3.28084^2)^2
*rho_w*g_fps2);
            dPdX_bend_h(m,n,i) =
(dPdX_K_loss_bend_90_h(m,n,i)*dPdX_wfr_h(m,n,i)^2)/(288*(area_h*3.28084^2)^2*
rho_w*g_fps2);
            dPdX_valve_h(m,n,i) =
(dPdX_K_loss_valve_h(m,n,i)*dPdX_wfr_h(m,n,i)^2)/(288*(area_h*3.28084^2)^2*rh
o_w*g_fps2);
            if dPdX(m,n,i)==2 %branch
                dPdX_entrance_h(m,n,i) =
(dPdX_wfr_h_2(m,n,i)^2)/(288*(area_h*3.28084^2)^2*rho_w*g_fps2)*...
                    (1.62-0.98*dPdX_wfr_h(m,n,i)/dPdX_wfr_h_2(m,n,i)-
0.64*dPdX_wfr_h(m,n,i)^2/dPdX_wfr_h_2(m,n,i)^2+0.03*dPdX_wfr_h_2(m,n,i)^6/dPd
X_wfr_h(m,n,i)^6);
                if isnan(dPdX_entrance_h(m,n,i))
                    dPdX_entrance_h(m,n,i) = 0;
                end
            else
                dPdX_entrance_h(m,n,i) = 0;
            end
            dPdX_total_h(m,n,i) =
dPdX_fric_h(m,n,i)+dPdX_bend_h(m,n,i)+dPdX_valve_h(m,n,i)+dPdX_entrance_h(m,n
,i);
            if i<max(size(Location_x))
                Pressure(m,n,i+1) = Pressure(m,n,i)-dPdX_total_h(m,n,i);
            end
        end
        %temp_pressure = 0;
        %for i=1:max(size(Pressure))-1
        %    temp_pressure = temp_pressure+Pressure_height_h(m,n,i);
        %    Pressure(m,n,i) = Pressure(m,n,i)+temp_pressure;
        %end

        for i=1:max(size(Pressure))
            Pressure_SI(m,n,i) = Pressure(m,n,i)*lb_p_in2_to_pa;
        end
        %figure(8)
```

```
            %plot(Location_x(m,n,:),Pressure_SI(m,n,:),'r')
            %hold on
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Preallocate variables
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Location_1 = zeros(1,max(size(dPdX)));
    dPdX_1 = zeros(1,max(size(dPdX)));
    Pressure_height_h1 = zeros(1,max(size(dPdX)));
    dPdX_mfr_h11 = zeros(1,max(size(dPdX)));
    dPdX_mfr_h12 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_friction_h1 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_bend_90_h1 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_valve_h1 = zeros(1,max(size(dPdX)));
    dPdX_entrance_h1 = zeros(1,max(size(dPdX)));
    dPdX_total_h1 = zeros(1,max(size(dPdX)));
    for i=1:max(size(dPdX))
        Location_1(i) = Location_x(1,1,i);
        dPdX_1(i) = dPdX(1,1,i);
        dPdX_mfr_h11(i) = dPdX_mfr_h(1,1,i);
        dPdX_mfr_h12(i) = dPdX_mfr_h_2(1,1,i);
        dPdX_K_loss_friction_h1(i) = dPdX_K_loss_friction_h(1,1,i);
        dPdX_K_loss_bend_90_h1(i) = dPdX_K_loss_bend_90_h(1,1,i);
        dPdX_K_loss_valve_h1(i) = dPdX_K_loss_valve_h(1,1,i);
        dPdX_entrance_h1(i) = dPdX_entrance_h(1,1,i);
        dPdX_total_h1(i) = dPdX_total_h(1,1,i);
    end
    for i=1:(max(size(dPdX))-1)
        Pressure_height_h1(i) = Pressure_height_h(1,1,i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Preallocate variables
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Location_2 = zeros(1,max(size(dPdX)));
    dPdX_2 = zeros(1,max(size(dPdX)));
    Pressure_height_h2 = zeros(1,max(size(dPdX)));
    dPdX_mfr_h21 = zeros(1,max(size(dPdX)));
    dPdX_mfr_h22 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_friction_h2 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_bend_90_h2 = zeros(1,max(size(dPdX)));
    dPdX_K_loss_valve_h2 = zeros(1,max(size(dPdX)));
    dPdX_entrance_h2 = zeros(1,max(size(dPdX)));
    dPdX_total_h2 = zeros(1,max(size(dPdX)));
    for i=1:max(size(dPdX))
        Location_2(i) = Location_x(1,2,i);
        dPdX_2(i) = dPdX(1,2,i);
        dPdX_mfr_h21(i) = dPdX_mfr_h(1,2,i);
        dPdX_mfr_h22(i) = dPdX_mfr_h_2(1,2,i);
        dPdX_K_loss_friction_h2(i) = dPdX_K_loss_friction_h(1,2,i);
        dPdX_K_loss_bend_90_h2(i) = dPdX_K_loss_bend_90_h(1,2,i);
        dPdX_K_loss_valve_h2(i) = dPdX_K_loss_valve_h(1,2,i);
        dPdX_entrance_h2(i) = dPdX_entrance_h(1,2,i);
```

```
        dPdX_total_h2(i) = dPdX_total_h(1,2,i);
    end
    for i=1:(max(size(dPdX))-1)
        Pressure_height_h2(i) = Pressure_height_h(1,2,i);
    end
    %% Step 7: Find stagnation points

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Preallocate variables
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    size_Pressure_SI = size(Pressure_SI);
    size_dPdX_header_loc_s_index = size(dPdX_header_loc_s_index);
    min_difference_pressure = 1000000000000*ones(1,size_header(1));
    min_pressure = zeros(1,size_header(1));
    min_location = zeros(1,size_header(1));
    index_diff = zeros(1,size_header(1));

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine index, location and minimum pressures between risers
    % (stagnation points)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:size_Pressure_SI(3)
        for j=1:size_dPdX_header_loc_s_index(2);
            if j==1 %j==1
                if i<=dPdX_header_loc_s_index(2)
                    difference_pressure = abs(Pressure_SI(1,1,i)-
    Pressure_SI(2,2,dPdX_header_loc_s_index(2)-i+1));
                    if min_difference_pressure(1)>difference_pressure
                        min_difference_pressure(1) = difference_pressure;
                        min_pressure(1) = Pressure_SI(1,1,i);
                        min_location(1) = Location_x(1,1,i);
                        index_diff(1) = i;
                    end
                end
            elseif j>=2 && (j<=size_dPdX_header_loc_s_index(2)/2) %j=2:4
                if (dPdX_header_loc_s_index(j)<i) &&
    (i<=dPdX_header_loc_s_index(j+1))
                    difference_pressure = abs(Pressure_SI((j-1)*2,1,i-
    dPdX_header_loc_s_index(j))-Pressure_SI(j*2,2,dPdX_header_loc_s_index(j+1)-
    i+1));
                    if min_difference_pressure(j)>difference_pressure
                        min_difference_pressure(j) = difference_pressure;
                        min_pressure(j) = Pressure_SI((j-1)*2,1,i-
    dPdX_header_loc_s_index(j)+1);
                        min_location(j) = Location_x(1,1,i);
                        index_diff(j) = i;
                    end
                end
            elseif (size_dPdX_header_loc_s_index(2)/2<j) &&
    (j<=size_dPdX_header_loc_s_index(2)/2+1) %j=5
                if (dPdX_header_loc_s_index(j)<i) &&
    (i<=dPdX_header_loc_s_index(j+1))
                    difference_pressure =
    abs(Pressure_SI((size_dPdX_header_loc_s_index(2)-j)*2+2,1,i-
```

```
dPdX_header_loc_s_index(j))-Pressure_SI((size_dPdX_header_loc_s_index(2)-j-
1)*2+3,2,dPdX_header_loc_s_index(j+1)-i+1));
                if min_difference_pressure(j)>difference_pressure
                    min_difference_pressure(j) = difference_pressure;
                    min_pressure(j) =
Pressure_SI((size_dPdX_header_loc_s_index(2)-j)*2+2,1,i-
dPdX_header_loc_s_index(j)+1);
                    min_location(j) = Location_x(1,1,i);
                    index_diff(j) = i;
                end
            end
        elseif (size_dPdX_header_loc_s_index(2)/2+1<j) &&
(j<=size_dPdX_header_loc_s_index(2)-1) %j=6:7
            if (dPdX_header_loc_s_index(j)<i) &&
(i<=dPdX_header_loc_s_index(j+1))
                difference_pressure =
abs(Pressure_SI((size_dPdX_header_loc_s_index(2)-j)*2+3,1,i-
dPdX_header_loc_s_index(j))-Pressure_SI((size_dPdX_header_loc_s_index(2)-j-
1)*2+3,2,dPdX_header_loc_s_index(j+1)-i+1));
                if min_difference_pressure(j)>difference_pressure
                    min_difference_pressure(j) = difference_pressure;
                    min_pressure(j) =
Pressure_SI((size_dPdX_header_loc_s_index(2)-j)*2+3,1,i-
dPdX_header_loc_s_index(j)+1);
                    min_location(j) = Location_x(1,1,i);
                    index_diff(j) = i;
                end
            end
        elseif j==size_dPdX_header_loc_s_index(2) %j==8
            if dPdX_header_loc_s_index(j)<i
                difference_pressure = abs(Pressure_SI(3,1,i-
dPdX_header_loc_s_index(j))-Pressure_SI(1,2,size_Pressure_SI(3)-i+1));
                if min_difference_pressure(j)>difference_pressure
                    min_difference_pressure(j) = difference_pressure;
                    min_pressure(j) = Pressure_SI(3,1,i-
dPdX_header_loc_s_index(j)+1);
                    min_location(j) = Location_x(1,1,i);
                    index_diff(j) = i;
                end
            end
        end
    end
end
%%
Pressure_SI_temp_11 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_12 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_21 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_22 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_31 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_32 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_41 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_42 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_51 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_52 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_61 = zeros(1,max(size(Pressure_SI)));
```

```
Pressure_SI_temp_62 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_71 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_72 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_81 = zeros(1,max(size(Pressure_SI)));
Pressure_SI_temp_82 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_11 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_12 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_21 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_22 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_31 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_32 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_41 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_42 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_51 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_52 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_61 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_62 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_71 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_72 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_81 = zeros(1,max(size(Pressure_SI)));
Location_x_temp_82 = zeros(1,max(size(Pressure_SI)));
for i=1:max(size(dPdX))
    if sum(chillers)>=2
        Pressure_SI_temp_11(i) = Pressure_SI(1,1,i);
        Pressure_SI_temp_12(i) = Pressure_SI(1,2,i);
        Pressure_SI_temp_21(i) = Pressure_SI(2,1,i);
        Pressure_SI_temp_22(i) = Pressure_SI(2,2,i);
    end
    if sum(chillers)>=3
        Pressure_SI_temp_31(i) = Pressure_SI(3,1,i);
        Pressure_SI_temp_32(i) = Pressure_SI(3,2,i);
    end
    if sum(chillers)>=4
        Pressure_SI_temp_41(i) = Pressure_SI(4,1,i);
        Pressure_SI_temp_42(i) = Pressure_SI(4,2,i);
    end
    if sum(chillers)>=5
        Pressure_SI_temp_51(i) = Pressure_SI(5,1,i);
        Pressure_SI_temp_52(i) = Pressure_SI(5,2,i);
    end
    if sum(chillers)>=6
        Pressure_SI_temp_61(i) = Pressure_SI(6,1,i);
        Pressure_SI_temp_62(i) = Pressure_SI(6,2,i);
    end
    if sum(chillers)>=7
        Pressure_SI_temp_71(i) = Pressure_SI(7,1,i);
        Pressure_SI_temp_72(i) = Pressure_SI(7,2,i);
    end
    if sum(chillers)>=8
        Pressure_SI_temp_81(i) = Pressure_SI(8,1,i);
        Pressure_SI_temp_82(i) = Pressure_SI(8,2,i);
    end
    for j=1:size_header(1)
        if header_loc_s_x_order(j)==1
            Location_x_temp_11(i) = Location_x(1,1,i)+header_loc_s_x(j);
```

```
            Location_x_temp_12(i) = Location_x(1,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==2
            Location_x_temp_21(i) = Location_x(2,1,i)+header_loc_s_x(j);
            Location_x_temp_22(i) = Location_x(2,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==3
            Location_x_temp_31(i) = Location_x(3,1,i)+header_loc_s_x(j);
            Location_x_temp_32(i) = Location_x(3,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==4
            Location_x_temp_41(i) = Location_x(4,1,i)+header_loc_s_x(j);
            Location_x_temp_42(i) = Location_x(4,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==5
            Location_x_temp_51(i) = Location_x(5,1,i)+header_loc_s_x(j);
            Location_x_temp_52(i) = Location_x(5,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==6
            Location_x_temp_61(i) = Location_x(6,1,i)+header_loc_s_x(j);
            Location_x_temp_62(i) = Location_x(6,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==7
            Location_x_temp_71(i) = Location_x(7,1,i)+header_loc_s_x(j);
            Location_x_temp_72(i) = Location_x(7,2,i)+header_loc_s_x(j);
        elseif header_loc_s_x_order(j)==8
            Location_x_temp_81(i) = Location_x(8,1,i)+header_loc_s_x(j);
            Location_x_temp_82(i) = Location_x(8,2,i)+header_loc_s_x(j);
        end
    end
end
figure(8)
    plot(Location_x_temp_11,Pressure_SI_temp_11,'r')
hold on
    plot(Location_x_temp_12,Pressure_SI_temp_12,'r')
    plot(Location_x_temp_21,Pressure_SI_temp_21,'b')
    plot(Location_x_temp_22,Pressure_SI_temp_22,'b')
if sum(chillers)>=3
    plot(Location_x_temp_31,Pressure_SI_temp_31,'g')
    plot(Location_x_temp_32,Pressure_SI_temp_32,'g')
end
if sum(chillers)>=4
    plot(Location_x_temp_41,Pressure_SI_temp_41,'c')
    plot(Location_x_temp_42,Pressure_SI_temp_42,'c')
end
if sum(chillers)>=5
    plot(Location_x_temp_51,Pressure_SI_temp_51,'k')
    plot(Location_x_temp_52,Pressure_SI_temp_52,'k')
end
if sum(chillers)>=6
    plot(Location_x_temp_61,Pressure_SI_temp_61,'y')
    plot(Location_x_temp_62,Pressure_SI_temp_62,'y')
end
if sum(chillers)>=7
    plot(Location_x_temp_71,Pressure_SI_temp_71,'m')
    plot(Location_x_temp_72,Pressure_SI_temp_72,'m')
end
if sum(chillers)>=8
    plot(Location_x_temp_81,Pressure_SI_temp_81,'k:')
    plot(Location_x_temp_82,Pressure_SI_temp_82,'k:')
end
```

```
xlabel('Distance along Supply Header [m]')
ylabel('Pressure [Pa]')
title('Pressure as a Function of Location in Supply Header')

figure(9)
plot(Location_x_temp_11,Pressure_SI_temp_11,'r')
xlabel('Distance along Supply Header [m]')
ylabel('Pressure [Pa]')
title('Pressure as a Function of Location in Supply Header')
hold on
Location_x_temp_12 =
Location_x_temp_12+abs(Location_x_temp_12(max(size(Location_x_temp_12))));
plot(Location_x_temp_12,Pressure_SI_temp_12,'b')
axis tight
legend('Clockwise flow','Counterclockwise flow')


%% Step 8: Final calculation of velocities and mass flow rates using network
analysis with system split up into sections at stagnation points

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert dPdX_header_loc_s_index to branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
riser_count_index = 1;
stag_count_index = 1;
for i=1:max(size(dPdX))
    if dPdX(1,1,i)==2 %branch
        count = count+1;
        if riser_count_index <= max(size(dPdX_header_loc_s_index))
            if i>=dPdX_header_loc_s_index(riser_count_index)
                riser_branch_index(riser_count_index) = count;
                riser_count_index = riser_count_index+1;
            end
        end
        if stag_count_index <= max(size(index_diff))
            if i>=index_diff(stag_count_index)
                stag_branch_index(stag_count_index) = count;
                stag_count_index = stag_count_index+1;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
velocity_delta_seg = 10*ones(1,size_header(1));
velocity_old_seg = zeros(1,size_header(1));
V_SI_h_seg = 1.5*ones(1,inputs+1); %initial guess at header velocities
f_b_seg = zeros(1,inputs);
K_loss_b_seg = zeros(1,inputs);
K_loss_friction_b_seg = zeros(1,inputs);
K_loss_bend_90_b_seg = zeros(1,inputs);
```

```
K_loss_gate_b_seg = zeros(1,inputs);
K_loss_globe_b_seg = zeros(1,inputs);
r_d_seg = 3*ones(1,inputs+1); %assume r/d=3
K_loss_h_seg = zeros(1,inputs+1);
K_loss_friction_h_seg = zeros(1,inputs+1);
K_loss_bend_90_h_seg = zeros(1,inputs+1);
K_loss_gate_h_seg = zeros(1,inputs+1);
K_loss_globe_h_seg = zeros(1,inputs+1);
K_loss_check_h_seg = zeros(1,inputs+1);
f_h_seg = zeros(1,inputs+1);
K_loss_rh_seg = zeros(1,inputs+1);
K_loss_friction_rh = zeros(size_header(1),2,inputs);
K_loss_bend_90_rh = zeros(size_header(1),2,inputs);
K_loss_gate_rh = zeros(size_header(1),2,inputs);
K_loss_globe_rh = zeros(size_header(1),2,inputs);
K_h_A_h_2_seg = zeros(1,inputs+1);
K_b_A_b_2_seg = zeros(1,inputs);
K_A_eq_seg = zeros(size_header(1),3,inputs);
mfr_h = zeros(size_header(1),2,inputs);
mfr_b = zeros(size_header(1),2,inputs);
V_b = zeros(size_header(1),2,inputs);
V_h = zeros(size_header(1),2,inputs);
mfr_total_seg = zeros(3,size_header(1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate total mfr's for each segment going cw and ccw
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:size_header(1)
    if i==1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=1:stag_branch_index(1)%j=1:(stag_branch_index(1)-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 1-15
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(1))))/2; %half of branch
15

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=(stag_branch_index(max(size(stag_branch_index)))+1):inputs
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 164:180
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

    elseif 1<i && i<size_header(1)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=riser_branch_index(i):stag_branch_index(i)%j=riser_branch_index(i):stag_bra
nch_index(i)-1
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 38:60
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i))))/2; %half of branch
60

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=stag_branch_index(i-1)+1:riser_branch_index(i)-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 16:37
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i-1))))/2; %half of
branch 15

    elseif i== size_header(1)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=riser_branch_index(max(size(riser_branch_index))):stag_branch_index(max(siz
e(stag_branch_index)))
%j=riser_branch_index(max(size(riser_branch_index))):(stag_branch_index(max(s
ize(stag_branch_index)))-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 154:163
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=stag_branch_index(max(size(stag_branch_index))-
1)+1:riser_branch_index(max(size(riser_branch_index)))-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 148:153
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))-1))))/2; %half of branch 147
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sum up mass flow rate going cw and ccw to give mass flow rate exiting
```

```
% each riser
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:size_header(1)
    mfr_total_seg(3,i) = mfr_total_seg(1,i)+mfr_total_seg(2,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resize and re-order V_SI_b and store in V_SI_b_seg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_seg = zeros(1,inputs);
for m=1:inputs
    V_SI_b_seg(m) = V_SI_b_1(branch_order(1,1,m));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterate through loop a predetermined number of times, modifying the
% branch diameters to satisfy the velocity limits set forth by NAVSEA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
while count<10
    count=count+1;

    if count == 1 %use estimated V_SI_b_seg to begin iterative process and
only consider friction bends and valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_b_seg due to friction, bends, valves for branches
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
            K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2;
%due to gate valves
            K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5;
%due to globe valves
            K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_h_seg due to friction, bends, valves for supply
header
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
```

```
f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,1,i)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,1,i)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,1,i)*0.2;
        %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %        K_loss_globe_h(i)+K_loss_check_h(i);
        end
        for i=inputs+1

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
        %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %        K_loss_globe_h(i)+K_loss_check_h(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_rh_seg due to friction, bends, valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs+1
        %    K_loss_friction_rh(i)=f_b(1)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %    K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_b(1)*pi()/2*r_d(i)+(0.10+2.4*f_b(1))*sin(pi()/4) ...
        %
+6.6*f_b(1)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %    K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %    K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
```

```
        %    K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_gate_rh(i)+K_loss_globe_rh(
i);
            K_loss_rh_seg(i) = K_loss_h_seg(i); %assume same loss coefficient
for supply and return header segments
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_b/A_b^2 and K_h/A_h^2 for branches and header segments
        % respectively
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
        end
        for i=1:inputs+1
            K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2 = zeros(1,inputs);
        for i=1:size_header(1)
            if i==1
                for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for
j=stag_branch_index(max(size(stag_branch_index)))+2:inputs %165:180
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=stag_branch_index(i) %15
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %1:14
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            else
                for j=stag_branch_index(i-1)+1 %16
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=stag_branch_index(i) %60
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59:38
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
```

```
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2_oa = zeros(1,size_header(1));
        for i=1:size_header(1)
            if i==1
                K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(riser_branch_index(i))^0.5))^2;
            else
                K_A_2_oa(i) = (1/(1/K_A_2(riser_branch_index(i)-
1)^0.5+1/K_A_2(riser_branch_index(i))^0.5))^2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_oa = zeros(2,size_header(1)); %cw=1, ccw=2
        for i=1:size_header(1)
            if i==1
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
            else
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)-1))^0.5;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_temp
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_b = zeros(1,inputs);
        mfr_seg_temp = zeros(1,inputs);
        for i=1:size_header(1)
            if i==1
                for j=riser_branch_index(i):stag_branch_index(i) %1:15
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for
j=stag_branch_index(max(size(stag_branch_index)))+1:inputs %164:180
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
                end
            else
                for j=riser_branch_index(i):stag_branch_index(i) %38:60
```

```
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for j=stag_branch_index(i-1)+1:riser_branch_index(i)-1 %16:37
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(riser_branch_index(i)-1)/K_A_2(j))^0.5;
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_b
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:size_header(1)
            if i==1
                for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for
j=stag_branch_index(max(size(stag_branch_index)))+2:inputs %165:180
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=stag_branch_index(i) %15
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %14:1
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            else
                for j=stag_branch_index(i-1)+1 %16
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=stag_branch_index(i) %60
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59;38
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_h
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_h = zeros(1,inputs);
        for i=1:size_header(1)
            if i==1
                for j=stag_branch_index(i) %15
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %14;1
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
```

```
                    end
                    for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                        mfr_seg_h(j) = mfr_seg_b(j);
                    end
                    for
j=stag_branch_index(max(size(stag_branch_index)))+2:inputs %165:180
                        mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                    end
                else
                    for j=stag_branch_index(i) %60
                        mfr_seg_h(j) = mfr_seg_b(j);
                    end
                    for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59:38
                        mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
                    end
                    for j=stag_branch_index(i-1)+1 %16
                        mfr_seg_h(j) = mfr_seg_b(j);
                    end
                    for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                        mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                    end
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate V_SI_b_seg
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate V_SI_h_seg
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:inputs
                V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate loss coefficient for branches due to friction, bends,
        % valves, entrance and exit effects (in order wrt header)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_loss_entrance_b_seg = zeros(1,inputs);
        K_loss_exit_b_seg = zeros(1,inputs);
        for i=1:inputs
            f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
```

```
        K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2; %due to
gate valves
        K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5; %due
to globe valves
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));

        Cyc(i) = 1-0.25*(D_SI_b(branch_order(1,1,i))/D_SI_h)^1.3-(0.11*r_d3-
0.65*r_d3^2+0.83*r_d3^3)*D_SI_b(branch_order(1,1,i))^2/D_SI_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance and exit effects for branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Keq = 0.57-1.07*r_d3^0.5-2.13*r_d3+8.24*r_d3^1.5-
8.48*r_d3^2+2.9*r_d3^2.5;
    Cxc = 0.08+0.56*r_d3-1.75*r_d3^2+1.83*r_d3^3;
    Cm = 0.23+1.46*r_d3-2.75*r_d3^2+1.65*r_d3^3;
    for j=1:size_header(1)
        if j==1
            for i=riser_branch_index(j):stag_branch_index(j) %cw 1:15
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=inputs:-1:stag_branch_index(max(size(stag_branch_index)))+1
%ccw 180:164
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        else
```

```
            for i=riser_branch_index(j):stag_branch_index(j) %cw 38:60
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=riser_branch_index(j)-1:-1:stag_branch_index(j-1)+1 %ccw
37:16
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_loss_b_seg and K_loss_b_in_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_b_in_seg = zeros(1,inputs);
    for i=1:inputs
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+K_loss_exit_b_seg(i);
        K_loss_b_in_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+0*K_loss_exit_b_seg(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_b_seg(i) <= 0
            K_loss_b_seg(i) = 0.01; %negligible loss coefficient
        end
        if K_loss_b_in_seg(i) <= 0
```

```
            K_loss_b_in_seg(i) = 0.01; %negligible loss coefficient
        end
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate loss coefficient for supply header due to friction, bends,
    % valves, entrance and exit effects
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_h_seg = zeros(1,inputs);
    for i=1:inputs
        f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
        K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
        K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
    %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
    %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves considered
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:size_header(1)
        if j==1
            for i=stag_branch_index(j) %cw 15
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=riser_branch_index(j):stag_branch_index(j)-1 %cw 1:14
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for i=stag_branch_index(max(size(stag_branch_index)))+1 %ccw 164
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=inputs:-1:stag_branch_index(max(size(stag_branch_index)))+2
%ccw 180:165
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        else
            for i=stag_branch_index(j) %cw 60
                K_loss_entrance_h_seg(i) = 0;
            end
```

```
            for i=riser_branch_index(j):stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for i=stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=riser_branch_index(j)-1:-1:stag_branch_index(j-1)+2 %ccw
37:17
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        end
    end
    for i=1:inputs
        K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i)+K_loss_
entrance_h_seg(i);%+ ...
        %    K_loss_globe_h(i)+K_loss_check_h(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_h_seg(i) <= 0
            K_loss_h_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate K_loss_rh due to friction, bends, valves
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_rh_seg = zeros(1,inputs);
    for i=1:inputs
        %K_loss_friction_rh(i)=f_h(i)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_h(i)*pi()/2*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/4) ...
        %
+6.6*f_h(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %K_loss_bend_180_rh(i) =
bends_180_rh(i)*(f_h(i)*pi()*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/2) ...
        %
+6.6*f_h(i)*((sin(pi()/2))^0.5+sin(pi()/2))/r_d(i)^(4*pi()/pi())); %due to
180 bends
        %K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
    end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:size_header(1)
        if j==1
            for i=stag_branch_index(j) %cw 15
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=riser_branch_index(j):stag_branch_index(j)-1 %cw 1:14
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for i=stag_branch_index(max(size(stag_branch_index)))+1 %ccw 164
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=inputs:-1:stag_branch_index(max(size(stag_branch_index)))+2
%ccw 180:165
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        else
            for i=stag_branch_index(j) %cw 60
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=riser_branch_index(j):stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for i=stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=riser_branch_index(j)-1:-1:stag_branch_index(j-1)+2 %ccw
37:17
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        end
    end
    for i=1:inputs
        K_loss_rh_seg(i) = K_loss_h_seg(i)-
K_loss_entrance_h_seg(i)+K_loss_entrance_rh_seg(i);
        %K_loss_rh_seg(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_bend_180_rh(i)+K_loss_gate_
rh(i)+ ...
```

```
        %    K_loss_globe_rh(i)+K_loss_entrance_rh(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_rh_seg(i) <= 0
            K_loss_rh_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
    end
    for i=1:inputs+1
        K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2 = zeros(1,inputs);
    for i=1:size_header(1)
        if i==1
            for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=stag_branch_index(max(size(stag_branch_index)))+2:inputs
%165:180
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=stag_branch_index(i) %15
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %1:14
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        else
            for j=stag_branch_index(i-1)+1 %16
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=stag_branch_index(i) %60
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
```

```
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59:38
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2_oa = zeros(1,size_header(1));
    for i=1:size_header(1)
        if i==1
            K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(riser_branch_index(i))^0.5))^2;
        else
            K_A_2_oa(i) = (1/(1/K_A_2(riser_branch_index(i)-
1)^0.5+1/K_A_2(riser_branch_index(i))^0.5))^2;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_oa = zeros(2,size_header(1)); %cw=1, ccw=2
    for i=1:size_header(1)
        if i==1
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
        else
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(riser_branch_index(i)-1))^0.5;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_temp
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_b = zeros(1,inputs);
    mfr_seg_temp = zeros(1,inputs);
    for i=1:size_header(1)
        if i==1
            for j=riser_branch_index(i):stag_branch_index(i) %1:15
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for j=stag_branch_index(max(size(stag_branch_index)))+1:inputs
%164:180
```

```
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
            end
        else
            for j=riser_branch_index(i):stag_branch_index(i) %38:60
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for j=stag_branch_index(i-1)+1:riser_branch_index(i)-1 %16:37
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(riser_branch_index(i)-1)/K_A_2(j))^0.5;
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:size_header(1)
        if i==1
            for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=stag_branch_index(max(size(stag_branch_index)))+2:inputs
%165:180
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
            end
            for j=stag_branch_index(i) %15
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %14:1
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        else
            for j=stag_branch_index(i-1)+1 %16
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
            end
            for j=stag_branch_index(i) %60
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59;38
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_h
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_h = zeros(1,inputs);
    for i=1:size_header(1)
        if i==1
```

```
            for j=stag_branch_index(i) %15
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %14;1
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=stag_branch_index(max(size(stag_branch_index)))+1 %164
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=stag_branch_index(max(size(stag_branch_index)))+2:inputs
%165:180
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        else
            for j=stag_branch_index(i) %60
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=stag_branch_index(i)-1:-1:riser_branch_index(i) %59:38
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=stag_branch_index(i-1)+1 %16
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=stag_branch_index(i-1)+2:riser_branch_index(i)-1 %17:37
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_b_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_h_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Code for simple network example
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %temp = zeros(3,size_header(1));
    %for i=1:size_header(1)
    %    if i==1
    %        for j=stag_branch_index(i):-1:riser_branch_index(i) %15:1
    %            temp(1,i) = temp(1,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %           temp(3,i) = temp(3,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %        end
```

```
    %        for j=stag_branch_index(max(size(stag_branch_index)))+1:inputs
%164:180
    %            temp(2,i) = temp(2,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %            temp(3,i) = temp(3,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %        end
    %    else
    %        for j=stag_branch_index(i):-1:riser_branch_index(i) %60:39
%60:38
    %            temp(1,i) = temp(1,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %            temp(3,i) = temp(3,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %            %temp(1) =
temp(1)+1/(K_b_A_b_2_seg(j)+K_h_A_h_2_seg(j))^0.5;
    %            %temp(3) =
temp(3)+1/(K_b_A_b_2_seg(j)+K_h_A_h_2_seg(j))^0.5;
    %        end
    %        for j=stag_branch_index(i-1)+1:riser_branch_index(i)-1 %16:37
    %            temp(2,i) = temp(2,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %            temp(3,i) = temp(3,i)+1/(K_b_A_b_2_seg(j))^0.5;
    %            %temp(2) =
temp(2)+1/(K_b_A_b_2_seg(j)+K_h_A_h_2_seg(j))^0.5;
    %           %temp(3) = temp(3)+1/(K_b_A_b_2_seg(j)+K_h_A_h_2_seg(j))^0.5;
    %        end
    %    end
    %end
    %  %K_A_2_oa = zeros(3,size_header(1));
   % K_A_2_oa = (1./temp).^2;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %mfr_b_seg = zeros(1,inputs);
    %mfr_b_seg_temp = zeros(2,size_header(1));
    %for i=1:size_header(1)
    %    mfr_b_seg_temp(1,i) =
(mfr_total_seg(1,i)+mfr_total_seg(2,i))*(K_A_2_oa(3,i)/K_A_2_oa(1,i))^0.5;
    %    mfr_b_seg_temp(2,i) =
(mfr_total_seg(1,i)+mfr_total_seg(2,i))*(K_A_2_oa(3,i)/K_A_2_oa(2,i))^0.5;
    %end
    %for i=1:size_header(1)
    %    if i==1
    %        for j=stag_branch_index(i):-1:riser_branch_index(i) %15:1
    %            mfr_b_seg(j) =
mfr_b_seg_temp(1,i)*(K_A_2_oa(1,i)/K_b_A_b_2_seg(j))^0.5;
    %        end
    %        for j=stag_branch_index(max(size(stag_branch_index)))+1:inputs
%164:180
    %            mfr_b_seg(j) =
mfr_b_seg_temp(2,i)*(K_A_2_oa(2,i)/K_b_A_b_2_seg(j))^0.5;
    %        end
    %    else
    %        for j=stag_branch_index(i):-1:riser_branch_index(i) %60:38
    %            mfr_b_seg(j) =
mfr_b_seg_temp(1,i)*(K_A_2_oa(1,i)/K_b_A_b_2_seg(j))^0.5;
    %        end
    %        for j=stag_branch_index(i-1)+1:riser_branch_index(i)-1 %16:37
```

```
    %           mfr_b_seg(j) =
mfr_b_seg_temp(2,i)*(K_A_2_oa(2,i)/K_b_A_b_2_seg(j))^0.5;
    %         end
    %     end
    %end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine least and greatest branch velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_vel_b = min(V_SI_b_seg)
max_vel_b = max(V_SI_b_seg)
min_vel_h = min(V_SI_h_seg)
V_SI_h_seg(181) = 0;
max_vel_h = max(V_SI_h_seg)


%% Step 9: Calculate branch inlet temperatures

Tcold_delta = zeros(1,inputs);
Tcold_delta_cum = zeros(1,inputs);
Tcold_delta_b = zeros(1,inputs);
Tcold = (44-32)*5/9;
g_mps2 = 9.81*ft_per_m;

for i=1:inputs
    H_l_h(i) = K_loss_h_seg(i)*V_SI_h_seg(i)^2/2/g_mps2;
    Tcold_delta(i) = (H_l_h(i)/778.169/1.0025)*10/18;
    for j=i:inputs
        Tcold_delta_cum(j) = Tcold_delta_cum(j)+Tcold_delta(i);
    end
end

Thot_delta_b = zeros(1,inputs);
for i=1:inputs
    H_l_b_in(i) = K_loss_b_in_seg(i)*V_SI_b_seg(i)^2/2/g_mps2;
    H_l_b(i) = K_loss_b_seg(i)*V_SI_b_seg(i)^2/2/g_mps2;
    Tcold_delta_b(i) = H_l_b_in(i)/778.169262/1.0025*10/18;
    Thot_delta_b(i) = H_l_b(i)/778.169262/1.0025*10/18;
end

Tcold_h = zeros(1,inputs);
Tcold_b = zeros(1,inputs);
Thot_h = zeros(1,inputs);
Thot_b = zeros(1,inputs);
for i=1:(inputs)
    Tcold_h(i) = Tcold + Tcold_delta_cum(i);
    Tcold_b(i) = Tcold_h(i) + Tcold_delta_b(i);
    Thot_b(i) = Tcold_h(i) + Thot_delta_b(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate temperatures
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:inputs
    order = branch_order(1,1,i);
    hc_b_seg(i) = calc_hc(D_SI_b_ordered(1,1,i),V_SI_b_seg(i),k,nu,rho,cp);
    Thot_b_seg(i) = Q_ordered(1,1,i)/(mfr_seg_b(i)*cp)+Tcold_b(i); %Celsius
    Tave_b_seg(i) = (Tcold_b(i)+Thot_b_seg(i))/2;
    T1_b_seg(i) = Tave_b_seg(i) +
Q_ordered(1,1,i)*(hxchgr_area_pri(order)*0.0001*hc_b_ordered(1,1,i))^-1;
%Inner wall temp
    if strcmp(Hxchgr_Type(order),'fp')
        T2_b_seg(i) = T1_b_seg(i) +
Q_ordered(1,1,i)*hxchgr_plate_thick(order)/100*(hxchgr_area_pri(order)*0.0001
*hxchgr_plate_k(order))^-1; %Inner wall temp
    else
        Q_per_l_seg(i) =
Q_ordered(1,1,i)*hxchgr_tube_diam(order)*pi()/100/(hxchgr_area_pri(order)*0.0
001);
        T2_b_seg(i) = T1_b_seg(i) +
Q_per_l_seg(i)*log((hxchgr_tube_diam(order)/2+hxchgr_tube_thick(order))/(hxch
gr_tube_diam(order)/2))/(2*pi()*kcopper); %Outer wall temp
    end
    Telec_b_ave_seg(i) = (T2_b_seg(i) +
Q_ordered(1,1,i)/(hxchgr_area_sec(order)*0.0001*hxchgr_hc(order)));
%Electrical component temp
    delta_T_sec_seg(i) =
Q_ordered(1,1,i)/hxchgr_fluid_mfr(order)/hxchgr_cp(order);
    Telec_b_in_seg(i) = Telec_b_ave_seg(i)+delta_T_sec_seg(i)/2;
    Telec_b_seg(i) = Telec_b_ave_seg(i)-delta_T_sec_seg(i)/2;
end

fprintf('Fifth Step: Refined Inlet Temperatures\n')
for i=1:inputs
    fprintf('Load: %3.0f  Q(W): %10.4f  Diameter(m): %6.5f  Velocity(m/sec):
%6.4f  Mass flow rate(kg/s): %6.4f  Thot(C): %7.4f  Telec(C): %8.4f\n' ...
        ,i,Q(branch_order(1,i)), D_SI_b(branch_order(1,i)) ,V_SI_b_seg(i)
,mfr_seg_b(i), Thot_b_seg(i), Telec_b_seg(i))
end

%% Step 10: Determine chiller capacity needed and select chillers

Thot_h = Thot_h+273.15
Thot_b = Thot_b+273.15
for i=inputs-1:-1:1
    %Thot_h(i) = (Thot_h(i+1)*mfr
    %Thot_h(i) =
(Thot_h(i+1)*mfr_h(i+1)+Thot_b(i)*mfr_b_seg(i))/(mfr_h(i+1)+mfr_b_seg(i))+...
     %   K_loss_rh(i)*V_b(i+1)^2/2/g_mps2/778.169/1.0025*10/18;
end
temp_mfr = zeros(1,180);
for j=1:size_header(1)
    if j==1
        for i=stag_branch_index(j) %15
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i);
        end
```

```
        for i=(stag_branch_index(j)-1):-1:riser_branch_index(j) %ccw14:1
            temp_mfr(i) = temp_mfr(i+1)+Thot_b_seg(i)*mfr_seg_b(i);
        end
        for i=(stag_branch_index(max(size(stag_branch_index)))+1) %164
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i);
        end
        for i=(stag_branch_index(max(size(stag_branch_index)))+2):inputs %cw
164:180
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i)+temp_mfr(i-1);
        end
    else
        for i=stag_branch_index(j) % 64
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i);
        end
        for i=(stag_branch_index(j)-1):-1:riser_branch_index(j) %ccw 59:38
            temp_mfr(i) = temp_mfr(i+1)+Thot_b_seg(i)*mfr_seg_b(i);
        end
        for i=stag_branch_index(j-1)+1
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i);
        end
        for i=(stag_branch_index(j-1)+2):(riser_branch_index(j)-1) %cw 16:37
            temp_mfr(i) = Thot_b_seg(i)*mfr_seg_b(i)+temp_mfr(i-1);
        end
    end
end
temp_mfr_riser = zeros(1,size_header(1));
Tchiller_hot = zeros(1,size_header(1));
Tchiller_mfr = zeros(1,size_header(1));
for i=1:size_header(1)
    if i==1
        temp_mfr_riser(i) = temp_mfr(1)+temp_mfr(inputs);
        Tchiller_hot(i) = temp_mfr_riser(i)/(mfr_seg_h(1)+mfr_seg_h(inputs));
        Tchiller_mfr(i) = mfr_seg_h(1)+mfr_seg_h(inputs);
    else
        temp_mfr_riser(i) =
temp_mfr(riser_branch_index(i))+temp_mfr(riser_branch_index(i)-1);
        Tchiller_hot(i) =
temp_mfr_riser(i)/(mfr_seg_h(riser_branch_index(i))+mfr_seg_h(riser_branch_in
dex(i)-1));
        Tchiller_mfr(i) =
mfr_seg_h(riser_branch_index(i))+mfr_seg_h(riser_branch_index(i)-1);
    end
end
Tchiller_hot
Tchiller_mfr

Tchiller_delta = zeros(1,size_header(1));
Tchiller_cap_kW = zeros(1,size_header(1));
for i=1:size_header(1)
    Tchiller_delta(i) = Tchiller_hot(i)-Tcold;
    Tchiller_cap_kW(i) = Tchiller_mfr(i)*Tchiller_delta(i)*cp/1000; %kg-K/sec
end

Tchiller_cap_tons = Tchiller_cap_kW*0.284345136; %in tons
```

```
fprintf('\n\n--------------------------------------------------------------------------------------\n')
fprintf('Report 1: Minimum Chiller Capacity\n')
fprintf('-----------------------------------------------------------------------------------\n')
for i=1:size_header(1)
    fprintf('Chiller %d  Chiller Capacity(tons):  %10.4f     Chiller Capacity(kW):  %10.4f\n', i, Tchiller_cap_tons(i),Tchiller_cap_kW(i))
end
fprintf('-----------------------------------------------------------------------------------\n')
fprintf('Total      Chiller Capacity(tons):  %10.4f     Chiller Capacity(kW): %10.4f\n', sum(Tchiller_cap_tons),sum(Tchiller_cap_kW))

% average chiller capacity must be greater than max chiller capacity above and
% satisfy N-1 criterion, i.e., N-1 chillers have adequate capacity to meet all
% cooling needs and must be greater than max chiller capacity above
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine minimum chiller size
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Min_chillers_operational = size_header(1)-1;
Min_chiller_cap_kW =
max(max(Tchiller_cap_kW),sum(Tchiller_cap_kW)/Min_chillers_operational);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select chillers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
chiller_capacity = 10*10^20;
chiller_index = 1;
chiller_dim = zeros(1,3);
chiller_out_temp = 0;
chiller_P = zeros(1,3);
chiller_T = zeros(1,3);
chiller_weight = 0;
flag = false;
if strcmp(chiller_type,'d') %default
    if Num_C_Chiller_Types > 0
        for j=1:Num_C_Chiller_Types
            if Min_chiller_cap_kW < C_Chiller_Capacity_kW(j) &&
chiller_capacity > C_Chiller_Capacity_kW(j)
                chiller_capacity = C_Chiller_Capacity_kW(j);
                chiller_index = j;
                flag = true;
            end
        end
        if flag == true
            chiller_dim = C_Chiller_Dim_m(chiller_index,:);
            chiller_out_temp = C_Chiller_Out_Temp_C(chiller_index);
            chiller_P = C_Chiller_P_MPa(chiller_index,:);
            chiller_T = C_Chiller_T_C(chiller_index,:);
            chiller_weight = C_Chiller_Weight_kg(chiller_index);
```

```
            chiller_refrig_type = C_Chiller_Type(chiller_index);
            flag = false;
        end
    end
    if Num_R_Chiller_Types > 0
        for j=1:Num_R_Chiller_Types
            if Min_chiller_cap_kW < R_Chiller_Capacity_kW(j) &&
chiller_capacity > R_Chiller_Capacity_kW(j)
                chiller_capacity = R_Chiller_Capacity_kW(j);
                chiller_index = j;
                flag = true;
            end
        end
        if flag == true
            chiller_dim = R_Chiller_Dim_m(chiller_index,:);
            chiller_out_temp = R_Chiller_Out_Temp_C(chiller_index);
            chiller_P = R_Chiller_P_MPa(chiller_index,:);
            chiller_T = R_Chiller_T_C(chiller_index,:);
            chiller_weight = R_Chiller_Weight_kg(chiller_index);
            chiller_refrig_type = R_Chiller_Type(chiller_index);
            flag = false;
        end
    end
    if Num_S_Chiller_Types > 0
        for j=1:Num_S_Chiller_Types
            if Min_chiller_cap_kW < S_Chiller_Capacity_kW(j) &&
chiller_capacity > S_Chiller_Capacity_kW(j)
                chiller_capacity = S_Chiller_Capacity_kW(j);
                chiller_index = j;
                flag = true;
            end
        end
        if flag == true
            chiller_dim = S_Chiller_Dim_m(chiller_index,:);
            chiller_out_temp = S_Chiller_Out_Temp_C(chiller_index);
            chiller_P = S_Chiller_P_MPa(chiller_index,:);
            chiller_T = S_Chiller_T_C(chiller_index,:);
            chiller_weight = S_Chiller_Weight_kg(chiller_index);
            chiller_refrig_type = S_Chiller_Type(chiller_index);
            flag = false;
        end
    end
    if Num_O_Chiller_Types > 0
        for j=1Num_O_Chiller_Types
            if Min_chiller_cap_kW < O_Chiller_Capacity_kW(j) &&
chiller_capacity > O_Chiller_Capacity_kW(j)
                chiller_capacity = O_Chiller_Capacity_kW(j);
                chiller_index = j;
                flag = true;
            end
        end
        if flag == true
            chiller_dim = O_Chiller_Dim_m(chiller_index,:);
            chiller_out_temp = O_Chiller_Out_Temp_C(chiller_index);
            chiller_P = O_Chiller_P_MPa(chiller_index,:);
```

```
                chiller_T = O_Chiller_T_C(chiller_index,:);
                chiller_weight = O_Chiller_Weight_kg(chiller_index);
                chiller_refrig_type = O_Chiller_Type(chiller_index);
                flag = false;
            end
        end
    elseif strcmp(chiller_type,'c') %centrifugal
        for j=1:Num_C_Chiller_Types
            if Min_chiller_cap < C_Chiller_Capacity_kW(j) && chiller_capacity >
    C_Chiller_Capacity_kW(j)
                chiller_capacity = C_Chiller_Capacity_kW(j);
                chiller_index = j;
            end
        end
    elseif strcmp(chiller_type,'s') %screw
        if Num_S_Chiller_Types > 0
            for j=1:Num_S_Chiller_Types
                if Min_chiller_cap_kW < S_Chiller_Capacity_kW(j) &&
    chiller_capacity > S_Chiller_Capacity_kW(j)
                    chiller_capacity = S_Chiller_Capacity_kW(j);
                    chiller_index = j;
                    flag = true;
                end
            end
            if flag == true
                chiller_dim = S_Chiller_Dim_m(chiller_index,:);
                chiller_out_temp = S_Chiller_Out_Temp_C(chiller_index);
                chiller_P = S_Chiller_P_MPa(chiller_index,:);
                chiller_T = S_Chiller_T_C(chiller_index,:);
                chiller_weight = S_Chiller_Weight_kg(chiller_index);
                chiller_refrig_type = S_Chiller_Type(chiller_index);
                flag = false;
            end
        end
    elseif strcmp(chiller_type,'r') %reciprocating
        if Num_R_Chiller_Types > 0
            for j=1:Num_R_Chiller_Types
                if Min_chiller_cap_kW < R_Chiller_Capacity_kW(j) &&
    chiller_capacity > R_Chiller_Capacity_kW(j)
                    chiller_capacity = R_Chiller_Capacity_kW(j);
                    chiller_index = j;
                    flag = true;
                end
            end
            if flag == true
                chiller_dim = R_Chiller_Dim_m(chiller_index,:);
                chiller_out_temp = R_Chiller_Out_Temp_C(chiller_index);
                chiller_P = R_Chiller_P_MPa(chiller_index,:);
                chiller_T = R_Chiller_T_C(chiller_index,:);
                chiller_weight = R_Chiller_Weight_kg(chiller_index);
                chiller_refrig_type = R_Chiller_Type(chiller_index);
                flag = false;
            end
        end
    else %other
```

```
    if Num_O_Chiller_Types > 0
        for j=1Num_O_Chiller_Types
            if Min_chiller_cap_kW < O_Chiller_Capacity_kW(j) &&
chiller_capacity > O_Chiller_Capacity_kW(j)
                chiller_capacity = O_Chiller_Capacity_kW(j);
                chiller_index = j;
                flag = true;
            end
        end
        if flag == true
            chiller_dim = O_Chiller_Dim_m(chiller_index,:);
            chiller_out_temp = O_Chiller_Out_Temp_C(chiller_index);
            chiller_P = O_Chiller_P_MPa(chiller_index,:);
            chiller_T = O_Chiller_T_C(chiller_index,:);
            chiller_weight = O_Chiller_Weight_kg(chiller_index);
            chiller_refrig_type = O_Chiller_Type(chiller_index);
            flag = false;
        end
    end
end

fprintf('\n\n-------------------------------------------------------------
--------------------\n')
fprintf('Report 2: Default Chillers Selected\n')
fprintf('----------------------------------------------------------------
----------------\n')
for i=1:size_header(1)
    fprintf('Chiller %d  Chiller Capacity(tons):  %10.4f     Chiller
Capacity(kW):  %10.4f\n', i, chiller_capacity*0.284345136,chiller_capacity)
end
fprintf('----------------------------------------------------------------
----------------\n')
fprintf('Total      Chiller Capacity(tons):  %10.4f     Chiller Capacity(kW):
%10.4f\n',
chiller_capacity*sum(chillers)*0.284345136,chiller_capacity*sum(chillers))
fprintf('Capacity Installed/Minimum Capacity Required:
%4.2f\n',chiller_capacity*sum(chillers)/sum(Tchiller_cap_kW))
chillers_reqd = ceil(sum(Tchiller_cap_kW)/chiller_capacity);
fprintf('Minimum number of chillers needed to meet maximum heat load demands:
%d \n',chillers_reqd)

%% Step 11: Expansion tank sizing

temp = 1; %0=false 1=true
if temp==1
    pump_time = 30; %seconds
else
    pump_time = 10; %seconds
end
Q_cw = mfr_total/1000*262.4*60; %capacity of the pump [gal/min]
V_o = pump_time/60*Q_cw; %operating water capacity of tank [gal]
H_t = 15*ft_per_m; %max vertical distance [ft] - change: find highest point
in system
P_c = 5+0.433527*H_t; %expansion tank charging pressure [lbs/in^2]
```

```
V_t_1 = 1.1*(15*V_o/P_c + V_o); %total expansion tank capacity method 1 [gal]
V_t_1 = V_t_1/262.4; % [m^3]

rho_cold = 1000; %T=273.15K=0C=32F%T=6.6C 999.41
rho_hot = 988.31; %T=322.0389K=48.8889C=120.0000F
water_vol_cold = 0;
water_vol_hot = 0;
for i=1:inputs
    water_vol_cold = water_vol_cold +
length_b_ordered(1,1,i)*D_SI_b_ordered(i)^2*pi()/4;
    water_vol_cold = water_vol_cold + length_h(1,1,i)*D_SI_h^2*pi()/4;
    water_vol_hot = water_vol_hot +
length_b_ordered(1,1,i)*D_SI_b_ordered(i)^2*pi()/4*rho_cold/rho_hot;
    water_vol_hot = water_vol_hot +
length_h(1,1,i)*D_SI_h^2*pi()/4*rho_cold/rho_hot;
end
water_vol_delta = water_vol_hot - water_vol_cold;

V_e = 1.1*water_vol_delta+(rho_cold/rho_hot-1)*V_o/262.4; %total expanded
water volume [m^3]
V_t_2 = 1.1*(V_e+V_o/262.4); %[m^3]

V_t = max([V_t_1 V_t_2]); %total expansion tank volume [m^3]

tank_thickness = 0.004; %assume tank thickness=4mm
tank_radius = (V_t/2/pi())^(1/3); %[m]
tank_height = 2*tank_radius;
tank_density = 7860; %kg/m^3
tank_weight =
0.004*(2*pi()*tank_radius^2+tank_height*pi()*2*tank_radius)*tank_density;
%[kg]
tank_instr_weight = 50; %estimate[kg]
cw_tank_weight = pi()*tank_radius^2*tank_height*rho; %assume tank 100% full

fprintf('\n\n-----------------------------------------------------------------
----------------------\n')
fprintf('Report 3: Expansion Tank Sizing\n')
fprintf('-------------------------------------------------------------------
-----------------\n')
fprintf('Expansion Tank Height(m):     %6.6f  \nExpansion Tank Radius(m):
%6.6f  \nExpansion Tank Thickness(mm): %6.6f\n', ...
    tank_height, tank_radius, tank_thickness*1000)
fprintf('-------------------------------------------------------------------
-----------------\n')

%% Step 12: Model SW System
size_sw_mains = size(SW_mains);
num_sw_mains = size_sw_mains(1);
size_sw_risers = size(SW_risers);
num_sw_risers = size_sw_risers(1);
size_sw_cc = size(SW_cross_connects);
num_sw_cc = size_sw_cc(1);
size_sw_piping = size(SW_piping);
num_sw_piping = size_sw_piping(1);
```

```
D_SI_sw_piping = zeros(1,num_sw_piping);
thickness_sw_piping = zeros(1,num_sw_piping);

if strcmp(chiller_refrig_type,'R134a')
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Set inlet and outlet seawater temperautres
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    sw_temp_in = (95-32)*5/9;
    sw_temp_out = (105-32)*5/9;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Verify outlet condenser temperature of the refrigerant is greater
    % than the inlet seawater temperature into the condenser
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if sw_temp_in>chiller_T(3)
        fprintf('SW inlet temperature is greater than refrigerant outlet
temperature\n')
        fprintf('Verify the SW inlet temperature\n')
        sw_temp_in = input('SW inlet temperaure (C): ');
        fprintf('Verify the condenser outlet temperature of the
refrigerant\n')
        chiller_T(3) = input('Condenser oultet temperautre of the
refrigerant: ');
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Find enthalpies of pressures and temperatures
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    h1 = calc_h_sat(chiller_T(1),R134a_Sat_T_C,R134a_Sat_hg);
    h2 =
calc_h_SHV(chiller_T(2),chiller_P(2),R134a_SHV_T_C,R134a_SHV_P_MPa,R134a_SHV_
h);
    h3 = calc_h_sat(chiller_T(3),R134a_Sat_T_C,R134a_Sat_hf);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Find mfr_refrig
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_refrig = chiller_capacity*1000/abs(h1-h3);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Find compressor power
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Q_comp = mfr_refrig*abs(h2-h1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Find heat rejected to sw
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Q_cond = mfr_refrig*abs(h3-h2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Use LMTD to find sw out temp assume sw in temp=95F and LMTD=10C
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    LMTD_assumption = 10; %C
    LMTD_temp = LMTD(chiller_T(2),sw_temp_out,chiller_T(3),sw_temp_in)
```

```
        while abs(LMTD_temp-LMTD_assumption)>0.01
            if LMTD_temp>LMTD_assumption
                sw_temp_out = sw_temp_out + 0.01;
            else
                sw_temp_out = sw_temp_out - 0.01;
            end
            LMTD_temp = LMTD(chiller_T(2),sw_temp_out,chiller_T(3),sw_temp_in);
        end
        sw_out_F = sw_temp_out*9/5+32;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Determine mfr of the seawater
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        cp_sw = 3993;
        sw_chillers_mfr = Q_cond/(cp_sw*abs(sw_temp_out-sw_temp_in))
    elseif strcmp(chiller_refrig_type,'R404a')
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Set inlet and outlet seawater temperautres
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        sw_temp_in = (95-32)*5/9;
        sw_temp_out = (105-32)*5/9;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Verify outlet condenser temperature of the refrigerant is greater
        % than the inlet seawater temperature into the condenser
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if sw_temp_in>chiller_T(3)
            fprintf('SW inlet temperature is greater than refrigerant outlet
temperature\n')
            fprintf('Verify the SW inlet temperature\n')
            sw_temp_in = input('SW inlet temperaure (C): ');
            fprintf('Verify the condenser outlet temperature of the
refrigerant\n')
            chiller_T(3) = input('Condenser oultet temperautre of the
refrigerant: ');
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Find enthalpies of pressures and temperatures
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        h1 = calc_h_sat(chiller_T(1),R404a_Sat_T_C,R404a_Sat_hg);
        h2 =
calc_h_SHV(chiller_T(2),chiller_P(2),R404a_SHV_T_C,R404a_SHV_P_MPa,R404a_SHV_
h);
        h3 = calc_h_sat(chiller_T(3),R404a_Sat_T_C,R404a_Sat_hf);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Find mfr_refrig
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_refrig = chiller_capacity*1000/abs(h1-h3);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Find compressor power
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    Q_comp = mfr_refrig*abs(h2-h1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Find heat rejected to sw
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Q_cond = mfr_refrig*abs(h3-h2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Use LMTD to find sw out temp assume sw in temp=95F and LMTD=10C
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    LMTD_assumption = 10; %C
    LMTD_temp = LMTD(chiller_T(2),sw_temp_out,chiller_T(3),sw_temp_in);
    while abs(LMTD_temp-LMTD_assumption)>0.01
        if LMTD_temp>LMTD_assumption
            sw_temp_out = sw_temp_out + 0.01;
        else
            sw_temp_out = sw_temp_out - 0.01;
        end
        LMTD_temp = LMTD(chiller_T(2),sw_temp_out,chiller_T(3),sw_temp_in);
    end
    sw_out_F = sw_temp_out*9/5+32;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine mfr of the seawater
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    cp_sw = 3993; %cp for sw temp of 95F - could modify this to call a
function which determines cp based on sw temp
    sw_chillers_mfr = Q_cond/(cp_sw*abs(sw_temp_out-sw_temp_in));
else
    fprintf('The refrigerant type is not within the CSDT database. Please
input the \n')
    fprintf('mass flow rate of the seawater across the chiller\n')
    sw_chillers_mfr = input('SW mass flow rate [kg/s]: ');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine expected mfr limits for a given cross-sectional area and
% velocity limit
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:sum(chillers) %sw connection to chillers
    if sw_chillers_mfr <= 0.3276 %kg/s
        D_SI_sw_piping(i) = 0.5/12/3.28084;
        thickness_sw_piping(i) = 0.035/12/3.28084;
    elseif (0.3276 < sw_chillers_mfr) && (sw_chillers_mfr <= 0.6237) %kg/s
        D_SI_sw_piping(i) = 0.75/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (0.6237 < sw_chillers_mfr) && (sw_chillers_mfr <= 1.1718) %kg/s
        D_SI_sw_piping(i) = 1/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (1.1718 < sw_chillers_mfr) && (sw_chillers_mfr <= 2.1987) %kg/s
        D_SI_sw_piping(i) = 1.25/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (2.1987 < sw_chillers_mfr) && (sw_chillers_mfr <= 3.1374) %kg/s
        D_SI_sw_piping(i) = 1.5/12/3.28084;
```

```
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (3.1374 < sw_chillers_mfr) && (sw_chillers_mfr <= 5.5692) %kg/s
        D_SI_sw_piping(i) = 2/12/3.28084;
        thickness_sw_piping(i) = 0.072/12/3.28084;
    elseif (5.5692 < sw_chillers_mfr) && (sw_chillers_mfr <= 9.261) %kg/s
        D_SI_sw_piping(i) = 2.5/12/3.28084;
        thickness_sw_piping(i) = 0.083/12/3.28084;
    elseif (9.261 < sw_chillers_mfr) && (sw_chillers_mfr <= 15.372) %kg/s
        D_SI_sw_piping(i) = 3/12/3.28084;
        thickness_sw_piping(i) = 0.083/12/3.28084;
    elseif (15.372 < sw_chillers_mfr) && (sw_chillers_mfr <= 21.924) %kg/s
        D_SI_sw_piping(i) = 3.5/12/3.28084;
        thickness_sw_piping(i) = 0.095/12/3.28084;
    elseif (21.924 < sw_chillers_mfr) && (sw_chillers_mfr <= 29.106) %kg/s
        D_SI_sw_piping(i) = 4/12/3.28084;
        thickness_sw_piping(i) = 0.095/12/3.28084;
    elseif (29.106 < sw_chillers_mfr) && (sw_chillers_mfr <= 50.022) %kg/s
        D_SI_sw_piping(i) = 5/12/3.28084;
        thickness_sw_piping(i) = 0.120/12/3.28084;
    else
        D_SI_sw_piping(i) = 12.0/12/3.28084;
        thickness_sw_piping(i) = 0.134/12/3.28084;
    end
end

sw_shaft_bearing_mfr = 0;
if shaft_bearing == 1 %sw connection to shaft bearing
    sw_shaft_bearing_mfr = shaft_bearing_gpm*0.063; %kg/s
    i = sum(chillers)+1;
    if sw_shaft_bearing_mfr <= 0.3276 %kg/s
        D_SI_sw_piping(i) = 0.5/12/3.28084;
        thickness_sw_piping(i) = 0.035/12/3.28084;
    elseif (0.3276 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
0.6237) %kg/s
        D_SI_sw_piping(i) = 0.75/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (0.6237 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
1.1718) %kg/s
        D_SI_sw_piping(i) = 1/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (1.1718 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
2.1987) %kg/s
        D_SI_sw_piping(i) = 1.25/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (2.1987 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
3.1374) %kg/s
        D_SI_sw_piping(i) = 1.5/12/3.28084;
        thickness_sw_piping(i) = 0.065/12/3.28084;
    elseif (3.1374 < sw_shaft_bearing_mfr)  (sw_shaft_bearing_mfr <= 5.5692)
%kg/s
        D_SI_sw_piping(i) = 2/12/3.28084;
        thickness_sw_piping(i) = 0.072/12/3.28084;
    elseif (5.5692 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <= 9.261)
%kg/s
        D_SI_sw_piping(i) = 2.5/12/3.28084;
```

```
        thickness_sw_piping(i) = 0.083/12/3.28084;
    elseif (9.261 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <= 15.372)
%kg/s
        D_SI_sw_piping(i) = 3/12/3.28084;
        thickness_sw_piping(i) = 0.083/12/3.28084;
    elseif (15.372 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
21.924) %kg/s
        D_SI_sw_piping(i) = 3.5/12/3.28084;
        thickness_sw_piping(i) = 0.095/12/3.28084;
    elseif (21.924 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
29.106) %kg/s
        D_SI_sw_piping(i) = 4/12/3.28084;
        thickness_sw_piping(i) = 0.095/12/3.28084;
    elseif (29.106 < sw_shaft_bearing_mfr) && (sw_shaft_bearing_mfr <=
50.022) %kg/s
        D_SI_sw_piping(i) = 5/12/3.28084;
        thickness_sw_piping(i) = 0.120/12/3.28084;
    else
        D_SI_sw_piping(i) = 12.0/12/3.28084;
        thickness_sw_piping(i) = 0.134/12/3.28084;
    end
end

sw_hxchgr_mfr = 0;
if SW_hxchgrs > 0 %sw connection to SW/XX hxchgrs
    sw_hxchgr_mfr = zeros(1,SW_hxchgrs);
    for i=1:SW_hxchgrs
        sw_hxchgr_mfr(i) = SW_hxchgr_gpm(i)*0.063; %kg/s
        if sw_hxchgr_mfr(i) <= 0.3276 %kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 0.5/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.035/12/3.28084;
        elseif (0.3276 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 0.6237)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 0.75/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.065/12/3.28084;
        elseif (0.6237 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 1.1718)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 1/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.065/12/3.28084;
        elseif (1.1718 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 2.1987)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 1.25/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.065/12/3.28084;
        elseif (2.1987 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 3.1374)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 1.5/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.065/12/3.28084;
        elseif (3.1374 < sw_hxchgr_mfr(i)) && ( sw_hxchgr_mfr(i) <= 5.5692)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 2/12/3.28084;
```

```
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.072/12/3.28084;
        elseif (5.5692 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 9.261)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 2.5/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.083/12/3.28084;
        elseif (9.261 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 15.372)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 3/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.083/12/3.28084;
        elseif (15.372 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 21.924)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 3.5/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.095/12/3.28084;
        elseif (21.924 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 29.106)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 4/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.095/12/3.28084;
        elseif (29.106 < sw_hxchgr_mfr(i)) && (sw_hxchgr_mfr(i) <= 50.022)
%kg/s
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 5/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.120/12/3.28084;
        else
            D_SI_sw_piping(i+sum(chillers)+shaft_bearing) = 12.0/12/3.28084;
            thickness_sw_piping(i+sum(chillers)+shaft_bearing) =
0.134/12/3.28084;
        end
    end
end

sw_mains_mfr =
0.5*(sum(chillers)*sw_chillers_mfr+sum(sw_hxchgr_mfr)+sw_shaft_bearing_mfr);
if sw_mains_mfr <= 0.3276 %kg/s
    D_SI_sw_mains = 0.5/12/3.28084;
    D_SI_sw_risers = 0.5/12/3.28084;
    D_SI_sw_cc = 0.5/12/3.28084;
    thickness_sw_mains = 0.035/12/3.28084;
    thickness_sw_risers = 0.035/12/3.28084;
    thickness_sw_cc = 0.035/12/3.28084;
elseif (0.3276 < sw_mains_mfr) && (sw_mains_mfr <= 0.6237) %kg/s
    D_SI_sw_mains = 0.75/12/3.28084;
    D_SI_sw_risers = 0.75/12/3.28084;
    D_SI_sw_cc = 0.75/12/3.28084;
    thickness_sw_mains = 0.065/12/3.28084;
    thickness_sw_risers = 0.065/12/3.28084;
    thickness_sw_cc = 0.065/12/3.28084;
elseif (0.6237 < sw_mains_mfr) && (sw_mains_mfr <= 1.1718) %kg/s
    D_SI_sw_mains = 1/12/3.28084;
    D_SI_sw_risers = 1/12/3.28084;
    D_SI_sw_cc = 1/12/3.28084;
```

```
        thickness_sw_mains = 0.065/12/3.28084;
        thickness_sw_risers = 0.065/12/3.28084;
        thickness_sw_cc = 0.065/12/3.28084;
    elseif (1.1718 < sw_mains_mfr) && (sw_mains_mfr <= 2.1987) %kg/s
        D_SI_sw_mains = 1.25/12/3.28084;
        D_SI_sw_risers = 1.25/12/3.28084;
        D_SI_sw_cc = 1.25/12/3.28084;
        thickness_sw_mains = 0.065/12/3.28084;
        thickness_sw_risers = 0.065/12/3.28084;
        thickness_sw_cc = 0.065/12/3.28084;
    elseif (2.1987 < sw_mains_mfr) && (sw_mains_mfr <= 3.1374) %kg/s
        D_SI_sw_mains = 1.5/12/3.28084;
        D_SI_sw_risers = 1.5/12/3.28084;
        D_SI_sw_cc = 1.5/12/3.28084;
        thickness_sw_mains = 0.065/12/3.28084;
        thickness_sw_risers = 0.065/12/3.28084;
        thickness_sw_cc = 0.065/12/3.28084;
    elseif (3.1374 < sw_mains_mfr) && (sw_mains_mfr <= 5.5692) %kg/s
        D_SI_sw_mains = 2/12/3.28084;
        D_SI_sw_risers = 2/12/3.28084;
        D_SI_sw_cc = 2/12/3.28084;
        thickness_sw_mains = 0.072/12/3.28084;
        thickness_sw_risers = 0.072/12/3.28084;
        thickness_sw_cc = 0.072/12/3.28084;
    elseif (5.5692 < sw_mains_mfr) && (sw_mains_mfr <= 9.261) %kg/s
        D_SI_sw_mains = 2.5/12/3.28084;
        D_SI_sw_risers = 2.5/12/3.28084;
        D_SI_sw_cc = 2.5/12/3.28084;
        thickness_sw_mains = 0.083/12/3.28084;
        thickness_sw_risers = 0.083/12/3.28084;
        thickness_sw_cc = 0.083/12/3.28084;
    elseif (9.261 < sw_mains_mfr) && (sw_mains_mfr <= 15.372) %kg/s
        D_SI_sw_mains = 3/12/3.28084;
        D_SI_sw_risers = 3/12/3.28084;
        D_SI_sw_cc = 3/12/3.28084;
        thickness_sw_mains = 0.083/12/3.28084;
        thickness_sw_risers = 0.083/12/3.28084;
        thickness_sw_cc = 0.083/12/3.28084;
    elseif (15.372 < sw_mains_mfr) && (sw_mains_mfr <= 21.924) %kg/s
        D_SI_sw_mains = 3.5/12/3.28084;
        D_SI_sw_risers = 3.5/12/3.28084;
        D_SI_sw_cc = 3.5/12/3.28084;
        thickness_sw_mains = 0.095/12/3.28084;
        thickness_sw_risers = 0.095/12/3.28084;
        thickness_sw_cc = 0.095/12/3.28084;
    elseif (21.924 < sw_mains_mfr) && (sw_mains_mfr <= 29.106) %kg/s
        D_SI_sw_mains = 4/12/3.28084;
        D_SI_sw_risers = 4/12/3.28084;
        D_SI_sw_cc = 4/12/3.28084;
        thickness_sw_mains = 0.095/12/3.28084;
        thickness_sw_risers = 0.095/12/3.28084;
        thickness_sw_cc = 0.095/12/3.28084;
    elseif (29.106 < sw_mains_mfr) && (sw_mains_mfr <= 50.022) %kg/s
        D_SI_sw_mains = 5/12/3.28084;
        D_SI_sw_risers = 5/12/3.28084;
```

```
    D_SI_sw_cc = 5/12/3.28084;
    thickness_sw_mains = 0.12/12/3.28084;
    thickness_sw_risers = 0.12/12/3.28084;
    thickness_sw_cc = 0.12/12/3.28084;
else
    D_SI_sw_mains = 6/12/3.28084;
    D_SI_sw_risers = 6/12/3.28084;
    D_SI_sw_cc = 6/12/3.28084;
    thickness_sw_mains = 0.134/12/3.28084;
    thickness_sw_risers = 0.134/12/3.28084;
    thickness_sw_cc = 0.134/12/3.28084;
end

%% Step 12: Weight analysis - Calculate total weight and center of gravity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine weights for piping and lagging for branches and headers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pipe_density = 1000*0.323/2204.62262*(12*ft_per_m)^3; %kg/m^3
pipe_b_weight = 0;
pipe_b_CG = [0 0 0]; %[LCG VCG TCG]

lagging_density = 1000*5/2204.62262*ft_per_m^3; %kg/m^3
lagging_thickness = 0.75/12/ft_per_m; %3/4 inches
lagging_b_weight = 0;
lagging_h_weight = 0;

cw_b_weight = 0;
cw_h_weight = 0;

for i=1:inputs
    pipe_b_weight = pipe_b_weight +
(length_b(1,i)+length_b(2,i))*((D_SI_b(i)+thickness_b(i))^2*pi()/4-
D_SI_b(i)^2*pi()/4)*pipe_density;
    cw_b_weight = cw_b_weight + length_b(i)*D_SI_b(i)^2*pi()/4*rho;
    lagging_b_weight = lagging_b_weight +
(length_b(1,i)+length_b(2,i))*((D_SI_b(i)+thickness_b(i)+lagging_thickness)^2
*pi()/4-...
        (D_SI_b(i)+thickness_b(i))^2*pi()/4)*lagging_density;
end

size_header_loc_s = size(header_loc_s);
length_h_s = zeros(size_header_loc_s(1),size_header_loc_s(2)-1);
pipe_h_CG = [0 0 0];
pipe_h_weight = 0;
for i=1:size_header_loc_s(1)
    for j=1:(size_header_loc_s(2)-1)
        length_h_s(i,j) = sqrt((header_loc_s(i,j,1)-
header_loc_s(i,j+1,1))^2+(header_loc_s(i,j,2)-header_loc_s(i,j+1,2))^2+...
            (header_loc_s(i,j,3)-header_loc_s(i,j+1,3))^2);
        pipe_h_weight = pipe_h_weight +
length_h_s(i,j)*((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        cw_h_weight = cw_h_weight + length_h_s(i,j)*D_SI_h^2*pi()/4*rho;
        lagging_h_weight = lagging_h_weight +
length_h_s(i,j)*((D_SI_h+thickness_h+lagging_thickness)^2*pi()/4-...
```

```
            (D_SI_h+thickness_h)^2*pi()/4)*lagging_density;
        pipe_h_CG(1) =
pipe_h_CG(1)+(header_loc_s(i,j,1)+header_loc_s(i,j+1,1))/2*length_h_s(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(2) =
pipe_h_CG(2)+(header_loc_s(i,j,2)+header_loc_s(i,j+1,2))/2*length_h_s(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(3) =
pipe_h_CG(3)+(header_loc_s(i,j,3)+header_loc_s(i,j+1,3))/2*length_h_s(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
    end
end
size_header_loc_s_alt = size(header_loc_s_alt);
length_h_s_alt = zeros(size_header_loc_s_alt(1),size_header_loc_s_alt(2)-1);
for i=1:size_header_loc_s_alt(1)
    for j=1:(size_header_loc_s_alt(2)-1)
        length_h_s_alt(i,j) = sqrt((header_loc_s_alt(i,j,1)-
header_loc_s_alt(i,j+1,1))^2+(header_loc_s_alt(i,j,2)-
header_loc_s_alt(i,j+1,2))^2+...
            (header_loc_s_alt(i,j,3)-header_loc_s_alt(i,j+1,3))^2);
        pipe_h_weight = pipe_h_weight +
length_h_s_alt(i,j)*((D_SI_h+thickness_h)^2*pi()/4-
D_SI_h^2*pi()/4)*pipe_density;
        cw_h_weight = cw_h_weight + length_h_s_alt(i,j)*D_SI_h^2*pi()/4*rho;
        lagging_h_weight = lagging_h_weight +
length_h_s_alt(i,j)*((D_SI_h+thickness_h+lagging_thickness)^2*pi()/4-...
            (D_SI_h+thickness_h)^2*pi()/4)*lagging_density;
        pipe_h_CG(1) =
pipe_h_CG(1)+(header_loc_s_alt(i,j,1)+header_loc_s_alt(i,j+1,1))/2*length_h_s
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(2) =
pipe_h_CG(2)+(header_loc_s_alt(i,j,2)+header_loc_s_alt(i,j+1,2))/2*length_h_s
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(3) =
pipe_h_CG(3)+(header_loc_s_alt(i,j,3)+header_loc_s_alt(i,j+1,3))/2*length_h_s
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
    end
end
size_header_loc_r = size(header_loc_r);
length_h_r = zeros(size_header_loc_r(1),size_header_loc_r(2)-1);
for i=1:size_header_loc_r(1)
    for j=1:(size_header_loc_r(2)-1)
        length_h_r(i,j) = sqrt((header_loc_r(i,j,1)-
header_loc_r(i,j+1,1))^2+(header_loc_r(i,j,2)-header_loc_r(i,j+1,2))^2+...
            (header_loc_r(i,j,3)-header_loc_r(i,j+1,3))^2);
        pipe_h_weight = pipe_h_weight +
length_h_r(i,j)*((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        cw_h_weight = cw_h_weight + length_h_r(i,j)*D_SI_h^2*pi()/4*rho;
```

```
        lagging_h_weight = lagging_h_weight +
length_h_r(i,j)*((D_SI_h+thickness_h+lagging_thickness)^2*pi()/4-...
            (D_SI_h+thickness_h)^2*pi()/4)*lagging_density;
        pipe_h_CG(1) =
pipe_h_CG(1)+(header_loc_r(i,j,1)+header_loc_r(i,j+1,1))/2*length_h_r(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(2) =
pipe_h_CG(2)+(header_loc_r(i,j,2)+header_loc_r(i,j+1,2))/2*length_h_r(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(3) =
pipe_h_CG(3)+(header_loc_r(i,j,3)+header_loc_r(i,j+1,3))/2*length_h_r(i,j)*..
.
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
    end
end
size_header_loc_r_alt = size(header_loc_r_alt);
length_h_r_alt = zeros(size_header_loc_r_alt(1),size_header_loc_r_alt(2)-1);
for i=1:size_header_loc_r_alt(1)
    for j=1:(size_header_loc_r_alt(2)-1)
        length_h_r_alt(i,j) = sqrt((header_loc_r_alt(i,j,1)-
header_loc_r_alt(i,j+1,1))^2+(header_loc_r_alt(i,j,2)-
header_loc_r_alt(i,j+1,2))^2+...
            (header_loc_r_alt(i,j,3)-header_loc_r_alt(i,j+1,3))^2);
        pipe_h_weight = pipe_h_weight +
length_h_r_alt(i,j)*((D_SI_h+thickness_h)^2*pi()/4-
D_SI_h^2*pi()/4)*pipe_density;
        cw_h_weight = cw_h_weight + length_h_r_alt(i,j)*D_SI_h^2*pi()/4*rho;
        lagging_h_weight = lagging_h_weight +
length_h_r_alt(i,j)*((D_SI_h+thickness_h+lagging_thickness)^2*pi()/4-...
            (D_SI_h+thickness_h)^2*pi()/4)*lagging_density;
        pipe_h_CG(1) =
pipe_h_CG(1)+(header_loc_r_alt(i,j,1)+header_loc_r_alt(i,j+1,1))/2*length_h_r
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(2) =
pipe_h_CG(2)+(header_loc_r_alt(i,j,2)+header_loc_r_alt(i,j+1,2))/2*length_h_r
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
        pipe_h_CG(3) =
pipe_h_CG(3)+(header_loc_r_alt(i,j,3)+header_loc_r_alt(i,j+1,3))/2*length_h_r
_alt(i,j)*...
            ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
    end
end

length_h_cc1_s = sqrt((cc1_loc_s(1,1)-cc1_loc_s(2,1))^2+(cc1_loc_s(1,2)-
cc1_loc_s(2,2))^2+(cc1_loc_s(1,3)-cc1_loc_s(2,3))^2);
length_h_cc2_s = sqrt((cc2_loc_s(1,1)-cc2_loc_s(2,1))^2+(cc2_loc_s(1,2)-
cc2_loc_s(2,2))^2+(cc2_loc_s(1,3)-cc2_loc_s(2,3))^2);
length_h_cc1_r = sqrt((cc1_loc_r(1,1)-cc1_loc_r(2,1))^2+(cc1_loc_r(1,2)-
cc1_loc_r(2,2))^2+(cc1_loc_r(1,3)-cc1_loc_r(2,3))^2);
length_h_cc2_r = sqrt((cc2_loc_r(1,1)-cc2_loc_r(2,1))^2+(cc2_loc_r(1,2)-
cc2_loc_r(2,2))^2+(cc2_loc_r(1,3)-cc2_loc_r(2,3))^2);
```

```
length_h_cc = length_h_cc1_s+length_h_cc2_s+length_h_cc1_r+length_h_cc2_r;
pipe_h_weight = pipe_h_weight + length_h_cc*((D_SI_h+thickness_h)^2*pi()/4-
D_SI_h^2*pi()/4)*pipe_density;
cw_h_weight = cw_h_weight + length_h_cc*D_SI_h^2*pi()/4*rho;
lagging_h_weight = lagging_h_weight +
length_h_cc*((D_SI_h+thickness_h+lagging_thickness)^2*pi()/4-...
    (D_SI_h+thickness_h)^2*pi()/4)*lagging_density;
pipe_h_CG(1) =
pipe_h_CG(1)+(cc1_loc_s(1,1)+cc1_loc_s(2,1)+cc2_loc_s(1,1)+cc2_loc_s(2,1)+...

cc1_loc_r(1,1)+cc2_loc_r(2,1)+cc2_loc_r(1,1)+cc2_loc_r(2,1))/8*length_h_cc1_s
*...
    ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
pipe_h_CG(2) =
pipe_h_CG(2)+(cc1_loc_s(1,2)+cc1_loc_s(2,2)+cc2_loc_s(1,2)+cc2_loc_s(2,2)+...

cc1_loc_r(1,2)+cc2_loc_r(2,2)+cc2_loc_r(1,2)+cc2_loc_r(2,2))/8*length_h_cc1_s
*...
    ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
pipe_h_CG(3) =
pipe_h_CG(3)+(cc1_loc_s(1,3)+cc1_loc_s(2,3)+cc2_loc_s(1,3)+cc2_loc_s(2,3)+...

cc1_loc_r(1,3)+cc2_loc_r(2,3)+cc2_loc_r(1,3)+cc2_loc_r(2,3))/8*length_h_cc1_s
*...
    ((D_SI_h+thickness_h)^2*pi()/4-D_SI_h^2*pi()/4)*pipe_density;
pipe_h_CG = pipe_h_CG/pipe_h_weight;
cw_h_CG = pipe_h_CG;
lagging_h_CG = pipe_h_CG;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine CG for piping and lagging
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
length_branch_seg = zeros(inputs,2,10);
weight_branch = zeros(inputs,2,10);
CG_branch = zeros(inputs,2,9,3);

for i=1:inputs
    for j=1:2
        for k=1:9
            length_branch_seg(i,j,k)=sqrt((branch_loc(k,j,1,i)-
branch_loc(k+1,j,1,i))^2+...
                (branch_loc(k,j,2,i)-branch_loc(k+1,j,2,i))^2+...
                (branch_loc(k,j,3,i)-branch_loc(k+1,j,3,i))^2);

weight_branch(i,j,k)=length_branch_seg(i,j,k)*((D_SI_b(i)+thickness_b(i))^2*p
i()/4-D_SI_b(i)^2*pi()/4)*pipe_density;
            CG_branch(i,j,k,1) =
(branch_loc(k,j,1,i)+branch_loc(k+1,j,1,i))/2;
            CG_branch(i,j,k,2) =
(branch_loc(k,j,2,i)+branch_loc(k+1,j,2,i))/2;
            CG_branch(i,j,k,3) =
(branch_loc(k,j,3,i)+branch_loc(k+1,j,3,i))/2;
        end
    end
end
```

```
for i=1:inputs
    for j=1:2
        for k=1:9
            pipe_b_CG(1) =
CG_branch(i,j,k,1)*weight_branch(i,j,k)+pipe_b_CG(1);
            pipe_b_CG(2) =
CG_branch(i,j,k,2)*weight_branch(i,j,k)+pipe_b_CG(2);
            pipe_b_CG(3) =
CG_branch(i,j,k,3)*weight_branch(i,j,k)+pipe_b_CG(3);
        end
    end
end
pipe_b_CG = pipe_b_CG/pipe_b_weight;
lagging_b_CG = pipe_b_CG;
cw_b_CG = pipe_b_CG;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine pipe weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pipe_weight = pipe_b_weight+pipe_h_weight;
pipe_CG = [0 0 0]; %[LCG VCG TCG]
pipe_CG(1) =
(pipe_b_CG(1)*pipe_b_weight+pipe_h_CG(1)*pipe_h_weight)/pipe_weight;
pipe_CG(2) =
(pipe_b_CG(2)*pipe_b_weight+pipe_h_CG(2)*pipe_h_weight)/pipe_weight;
pipe_CG(3) =
(pipe_b_CG(3)*pipe_b_weight+pipe_h_CG(3)*pipe_h_weight)/pipe_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine lagging weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lagging_weight = lagging_b_weight+lagging_h_weight;
lagging_CG = [0 0 0]; %[LCG VCG TCG]
lagging_CG(1) =
(lagging_b_CG(1)*lagging_b_weight+lagging_h_CG(1)*lagging_h_weight)/lagging_w
eight;
lagging_CG(2) =
(lagging_b_CG(2)*lagging_b_weight+lagging_h_CG(2)*lagging_h_weight)/lagging_w
eight;
lagging_CG(3) =
(lagging_b_CG(3)*lagging_b_weight+lagging_h_CG(3)*lagging_h_weight)/lagging_w
eight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define gate valve and globe valve weights for various sizes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gate_valve_b_weight = 0;
gate_valve_h_weight = 0;
globe_valve_b_weight = 0;
globe_valve_h_weight = 0;
check_valve_b_weight = 0;
check_valve_h_weight = 0;
check_valve_b_CG = [0 0 0]; %[LCG VCG TCG]
check_valve_h_CG = [0 0 0]; %[LCG VCG TCG]
```

```
globe_valve_b_CG = [0 0 0]; %[LCG VCG TCG]
globe_valve_h_CG = [0 0 0]; %[LCG VCG TCG]
gate_valve_b_CG = [0 0 0]; %[LCG VCG TCG]
gate_valve_h_CG = [0 0 0]; %[LCG VCG TCG]
Valve_diams_class_150 = [0.5 0.75 1 1.5 2 3 4 5 6 8 10 12 14 16 18 20
24];%inches
Gate_valve_weight_class_150 = [3.2 4.2 5.8 11 15.4 35 50 70 80 135 185 ...
    280 395 530 670 775 1150]; %kg
Globe_valve_weight_class_150 = [3.1 4 5.7 10.6 15.4 35 55 80 98 165 305 ...
    425 590 830 1040 1260 1700]; %kg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define check valve weights for various sizes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Check_valve_diams_class_150 = [2 2.5 3 4 5 6 8 10 12 14 16 18 20 24];%inches
Check_valve_weight_class_150 = [13 17 24 36 57 62 96 158 238 324 483 548 782
1150]; %kg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine gate valve and globe valve weights, LCG, VCG, and TCG for
branches
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:inputs
    if vital == 1
        if D_SI_b(i)<Valve_diams_class_150(1)/12/ft_per_m
            gate_valve_b_weight =
gate_valve_b_weight+gate_valve_b(i)*Gate_valve_weight_class_150(1)*2;
            globe_valve_b_weight =
globe_valve_b_weight+globe_valve_b(i)*Globe_valve_weight_class_150(1)*2;
            gate_valve_b_CG(1) = gate_valve_b_CG(1) +
(branch_gate_loc(1,1,1,i)+branch_gate_loc(1,2,1,i)+...

branch_gate_loc(2,1,1,i)+branch_gate_loc(2,1,1,i))*Gate_valve_weight_class_15
0(1);
            gate_valve_b_CG(2) = gate_valve_b_CG(2) +
(branch_gate_loc(1,1,3,i)+branch_gate_loc(1,2,3,i)+...

branch_gate_loc(2,1,3,i)+branch_gate_loc(2,1,3,i))*Gate_valve_weight_class_15
0(1);
            gate_valve_b_CG(3) = gate_valve_b_CG(3) +
(branch_gate_loc(1,1,2,i)+branch_gate_loc(1,2,2,i)+...

branch_gate_loc(2,1,2,i)+branch_gate_loc(2,1,2,i))*Gate_valve_weight_class_15
0(1);
            globe_valve_b_CG(1) = globe_valve_b_CG(1) +
(branch_globe_loc(1,1,1,i)+branch_globe_loc(1,2,1,i))*Globe_valve_weight_clas
s_150(1);
            globe_valve_b_CG(2) = globe_valve_b_CG(2) +
(branch_globe_loc(1,1,3,i)+branch_globe_loc(1,2,3,i))*Globe_valve_weight_clas
s_150(1);
            globe_valve_b_CG(3) = globe_valve_b_CG(3) +
(branch_globe_loc(1,1,2,i)+branch_globe_loc(1,2,2,i))*Globe_valve_weight_clas
s_150(1);
        else
            for j=1:max(size(Valve_diams_class_150))-1
```

```
                if (Valve_diams_class_150(j)/12/ft_per_m < D_SI_b(i)) &&
(D_SI_b(i) <= Valve_diams_class_150(j+1)/12/ft_per_m)
                    gate_valve_b_weight =
gate_valve_b_weight+gate_valve_b(i)*Gate_valve_weight_class_150(j+1)*2;
                    globe_valve_b_weight =
globe_valve_b_weight+globe_valve_b(i)*Globe_valve_weight_class_150(j+1)*2;
                    gate_valve_b_CG(1) = gate_valve_b_CG(1) +
(branch_gate_loc(1,1,1,i)+branch_gate_loc(1,2,1,i)+...

branch_gate_loc(2,1,1,i)+branch_gate_loc(2,1,1,i))*Gate_valve_weight_class_15
0(j+1);
                    gate_valve_b_CG(2) = gate_valve_b_CG(2) +
(branch_gate_loc(1,1,3,i)+branch_gate_loc(1,2,3,i)+...

branch_gate_loc(2,1,3,i)+branch_gate_loc(2,1,3,i))*Gate_valve_weight_class_15
0(j+1);
                    gate_valve_b_CG(3) = gate_valve_b_CG(3) +
(branch_gate_loc(1,1,2,i)+branch_gate_loc(1,2,2,i)+...

branch_gate_loc(2,1,2,i)+branch_gate_loc(2,1,2,i))*Gate_valve_weight_class_15
0(j+1);
                    globe_valve_b_CG(1) = globe_valve_b_CG(1) +
(branch_globe_loc(1,1,1,i)+branch_globe_loc(1,2,1,i))*Globe_valve_weight_clas
s_150(j+1);
                    globe_valve_b_CG(2) = globe_valve_b_CG(2) +
(branch_globe_loc(1,1,3,i)+branch_globe_loc(1,2,3,i))*Globe_valve_weight_clas
s_150(j+1);
                    globe_valve_b_CG(3) = globe_valve_b_CG(3) +
(branch_globe_loc(1,1,2,i)+branch_globe_loc(1,2,2,i))*Globe_valve_weight_clas
s_150(j+1);
                end
            end
        end
    else
        if D_SI_b(i)<Valve_diams_class_150(1)/12/ft_per_m
            gate_valve_b_weight =
gate_valve_b_weight+gate_valve_b(i)*Gate_valve_weight_class_150(1);
            globe_valve_b_weight =
globe_valve_b_weight+globe_valve_b(i)*Globe_valve_weight_class_150(1);
            gate_valve_b_CG(1) = gate_valve_b_CG(1) +
(branch_gate_loc(1,1,1,i)+branch_gate_loc(2,1,1,i))*Gate_valve_weight_class_1
50(1);
            gate_valve_b_CG(2) = gate_valve_b_CG(2) +
(branch_gate_loc(1,1,3,i)+branch_gate_loc(2,1,3,i))*Gate_valve_weight_class_1
50(1);
            gate_valve_b_CG(3) = gate_valve_b_CG(3) +
(branch_gate_loc(1,1,2,i)+branch_gate_loc(2,1,2,i))*Gate_valve_weight_class_1
50(1);
            globe_valve_b_CG(1) = globe_valve_b_CG(1) +
branch_globe_loc(1,1,1,i)*Globe_valve_weight_class_150(1);
            globe_valve_b_CG(2) = globe_valve_b_CG(2) +
branch_globe_loc(1,1,3,i)*Globe_valve_weight_class_150(1);
            globe_valve_b_CG(3) = globe_valve_b_CG(3) +
branch_globe_loc(1,1,2,i)*Globe_valve_weight_class_150(1);
        else
```

```
            for j=1:max(size(Valve_diams_class_150))-1
                if (Valve_diams_class_150(j)/12/ft_per_m < D_SI_b(i)) &&
(D_SI_b(i) <= Valve_diams_class_150(j+1)/12/ft_per_m)
                    gate_valve_b_weight =
gate_valve_b_weight+gate_valve_b(i)*Gate_valve_weight_class_150(j+1);
                    globe_valve_b_weight =
globe_valve_b_weight+globe_valve_b(i)*Globe_valve_weight_class_150(j+1);
                    gate_valve_b_CG(1) = gate_valve_b_CG(1) +
(branch_gate_loc(1,1,1,i)+branch_gate_loc(2,1,1,i))*Gate_valve_weight_class_1
50(j+1);
                    gate_valve_b_CG(2) = gate_valve_b_CG(2) +
(branch_gate_loc(1,1,2,i)+branch_gate_loc(2,1,2,i))*Gate_valve_weight_class_1
50(j+1);
                    gate_valve_b_CG(3) = gate_valve_b_CG(3) +
(branch_gate_loc(1,1,3,i)+branch_gate_loc(2,1,3,i))*Gate_valve_weight_class_1
50(j+1);
                    globe_valve_b_CG(1) = globe_valve_b_CG(1) +
branch_globe_loc(1,1,1,i)*Globe_valve_weight_class_150(j+1);
                    globe_valve_b_CG(2) = globe_valve_b_CG(2) +
branch_globe_loc(1,1,2,i)*Globe_valve_weight_class_150(j+1);
                    globe_valve_b_CG(3) = globe_valve_b_CG(3) +
branch_globe_loc(1,1,3,i)*Globe_valve_weight_class_150(j+1);
                end
            end
        end
    end
end
gate_valve_b_CG = gate_valve_b_CG/gate_valve_b_weight;
globe_valve_b_CG = globe_valve_b_CG/globe_valve_b_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine gate valve and globe valve weights for header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if D_SI_h<Valve_diams_class_150(1)/12/ft_per_m
    for i=1:max(size(seg_valve_loc))
        gate_valve_h_weight =
gate_valve_h_weight+Gate_valve_weight_class_150(1);
        gate_valve_h_CG(1) = gate_valve_h_CG(1) +
seg_valve_loc(i,1)*Gate_valve_weight_class_150(1);
        gate_valve_h_CG(2) = gate_valve_h_CG(2) +
seg_valve_loc(i,2)*Gate_valve_weight_class_150(1);
        gate_valve_h_CG(3) = gate_valve_h_CG(3) +
seg_valve_loc(i,3)*Gate_valve_weight_class_150(1);
    end
else
    for j=1:max(size(Valve_diams_class_150))-1
        if (Valve_diams_class_150(j)/12/ft_per_m < D_SI_h) && (D_SI_h <=
Valve_diams_class_150(j+1)/12/ft_per_m)
            for i=1:max(size(seg_valve_loc))
                gate_valve_h_weight =
gate_valve_h_weight+Gate_valve_weight_class_150(j+1);
                gate_valve_h_CG(1) = gate_valve_h_CG(1) +
seg_valve_loc(i,1)*Gate_valve_weight_class_150(j+1);
                gate_valve_h_CG(2) = gate_valve_h_CG(2) +
seg_valve_loc(i,2)*Gate_valve_weight_class_150(j+1);
```

```
                gate_valve_h_CG(3) = gate_valve_h_CG(3) +
seg_valve_loc(i,3)*Gate_valve_weight_class_150(j+1);
            end
        end
    end
end
gate_valve_h_CG = gate_valve_h_CG/gate_valve_h_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine check valve weights for header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if D_SI_h<Check_valve_diams_class_150(1)/12/ft_per_m
    for i=1:max(size(chiller_loc))
        check_valve_h_weight =
check_valve_h_weight+size_header(1)*Check_valve_weight_class_150(1);
        check_valve_h_CG(1) = check_valve_h_CG(1) +
chiller_loc(i,1)*Check_valve_weight_class_150(1);
        check_valve_h_CG(2) = check_valve_h_CG(2) +
chiller_loc(i,2)*Check_valve_weight_class_150(1);
        check_valve_h_CG(3) = check_valve_h_CG(3) +
chiller_loc(i,3)*Check_valve_weight_class_150(1);
    end
else
    for j=1:max(size(Check_valve_diams_class_150))-1
        if (Check_valve_diams_class_150(j)/12/ft_per_m < D_SI_h) && (D_SI_h
<= Check_valve_diams_class_150(j+1)/12/ft_per_m)
            for i=1:sum(chillers)
                check_valve_h_weight =
check_valve_h_weight+size_header(1)*Check_valve_weight_class_150(j+1);
                check_valve_h_CG(1) = check_valve_h_CG(1) +
chiller_loc(i,1)*Check_valve_weight_class_150(j+1);
                check_valve_h_CG(2) = check_valve_h_CG(2) +
chiller_loc(i,2)*Check_valve_weight_class_150(j+1);
                check_valve_h_CG(3) = check_valve_h_CG(3) +
chiller_loc(i,3)*Check_valve_weight_class_150(j+1);
            end
        end
    end
end
check_valve_h_CG = check_valve_h_CG/check_valve_h_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine valve weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
globe_valve_weight = globe_valve_b_weight+globe_valve_h_weight;
globe_valve_CG = [0 0 0]; %[LCG VCG TCG]
globe_valve_CG(1) =
(globe_valve_b_CG(1)*globe_valve_b_weight+globe_valve_h_CG(1)*globe_valve_h_w
eight)/globe_valve_weight;
globe_valve_CG(2) =
(globe_valve_b_CG(2)*globe_valve_b_weight+globe_valve_h_CG(2)*globe_valve_h_w
eight)/globe_valve_weight;
globe_valve_CG(3) =
(globe_valve_b_CG(3)*globe_valve_b_weight+globe_valve_h_CG(3)*globe_valve_h_w
eight)/globe_valve_weight;
```

```
gate_valve_weight = gate_valve_b_weight+gate_valve_h_weight;
gate_valve_CG = [0 0 0]; %[LCG VCG TCG]
gate_valve_CG(1) =
(gate_valve_b_CG(1)*gate_valve_b_weight+gate_valve_h_CG(1)*gate_valve_h_weigh
t)/gate_valve_weight;
gate_valve_CG(2) =
(gate_valve_b_CG(2)*gate_valve_b_weight+gate_valve_h_CG(2)*gate_valve_h_weigh
t)/gate_valve_weight;
gate_valve_CG(3) =
(gate_valve_b_CG(3)*gate_valve_b_weight+gate_valve_h_CG(3)*gate_valve_h_weigh
t)/gate_valve_weight;

check_valve_weight = check_valve_b_weight+check_valve_h_weight;
check_valve_CG = [0 0 0]; %[LCG VCG TCG]
check_valve_CG(1) =
(check_valve_b_CG(1)*check_valve_b_weight+check_valve_h_CG(1)*check_valve_h_w
eight)/check_valve_weight;
check_valve_CG(2) =
(check_valve_b_CG(2)*check_valve_b_weight+check_valve_h_CG(2)*check_valve_h_w
eight)/check_valve_weight;
check_valve_CG(3) =
(check_valve_b_CG(3)*check_valve_b_weight+check_valve_h_CG(3)*check_valve_h_w
eight)/check_valve_weight;

valve_weight = globe_valve_weight+gate_valve_weight+check_valve_weight;
valve_CG = [0 0 0]; %[LCG VCG TCG]
valve_CG(1) =
(globe_valve_CG(1)*globe_valve_weight+gate_valve_CG(1)*gate_valve_weight+chec
k_valve_CG(1)*check_valve_weight)/valve_weight;
valve_CG(2) =
(globe_valve_CG(2)*globe_valve_weight+gate_valve_CG(2)*gate_valve_weight+chec
k_valve_CG(2)*check_valve_weight)/valve_weight;
valve_CG(3) =
(globe_valve_CG(3)*globe_valve_weight+gate_valve_CG(3)*gate_valve_weight+chec
k_valve_CG(3)*check_valve_weight)/valve_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine chiller weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
chiller_CG = [0 0 0]; %[LCG VCG TCG]
num_chillers = sum(chillers);
chiller_weight_total = 0;
for i=1:num_chillers
    chiller_weight_total = chiller_weight_total + chiller_weight;
    chiller_CG(1) = chiller_CG(1) + chiller_weight*chiller_loc(i,1);
    chiller_CG(2) = chiller_CG(2) + chiller_weight*chiller_loc(i,2);
    chiller_CG(3) = chiller_CG(3) + chiller_weight*chiller_loc(i,3);
end
chiller_CG = chiller_CG/chiller_weight_total;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine hxchgr weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
hxchgr_weight = 0;
hxchgr_CG = [0 0 0];
cw_hxchgr_weight = 0;
cw_hxchgr_CG = [0 0 0];
for i=1:inputs
    hxchgr_weight = hxchgr_weight + hxchgr_weight_dry(i);
    hxchgr_CG(1) = hxchgr_CG(1) + hxchgr_weight_dry(i)*Load_Loc_m(i,1);
    hxchgr_CG(2) = hxchgr_CG(2) + hxchgr_weight_dry(i)*Load_Loc_m(i,2);
    hxchgr_CG(3) = hxchgr_CG(3) + hxchgr_weight_dry(i)*Load_Loc_m(i,3);
    cw_hxchgr_weight = cw_hxchgr_weight + hxchgr_weight_wet(i) -
hxchgr_weight_dry(i);
    cw_hxchgr_CG(1) = cw_hxchgr_CG(1) + (hxchgr_weight_wet(i) -
hxchgr_weight_dry(i))*Load_Loc_m(i,1);
    cw_hxchgr_CG(2) = cw_hxchgr_CG(2) + (hxchgr_weight_wet(i) -
hxchgr_weight_dry(i))*Load_Loc_m(i,2);
    cw_hxchgr_CG(3) = cw_hxchgr_CG(3) + (hxchgr_weight_wet(i) -
hxchgr_weight_dry(i))*Load_Loc_m(i,3);
end
hxchgr_CG = hxchgr_CG/hxchgr_weight;
cw_hxchgr_CG = cw_hxchgr_CG/cw_hxchgr_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine tank weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tank_CG = chiller_CG;
total_tank_weight = tank_weight*num_chillers;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine tank instr weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tank_instr_CG = tank_CG;
total_tank_instr_weight = tank_instr_weight*num_chillers;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine pump weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pump_cw_CG = [0 0 0];
pump_cw_weight = 1200; %revise
pump_cw_weight_total = 0;
for i=1:num_chillers
    pump_cw_weight_total = pump_cw_weight_total + pump_cw_weight;
    pump_cw_CG(1) = pump_cw_CG(1)+pump_cw_weight*pump_loc(i,1);
    pump_cw_CG(2) = pump_cw_CG(2)+pump_cw_weight*pump_loc(i,2);
    pump_cw_CG(3) = pump_cw_CG(3)+pump_cw_weight*pump_loc(i,3);
end
pump_cw_CG = pump_cw_CG/pump_cw_weight_total;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine bracket weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hangar_b_lb_per_ft = zeros(1,inputs);
for i=1:inputs
    if D_SI_b(i) <= 0.25/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1161;
```

```
    elseif 0.25/12/3.28084 < D_SI_b(i) <= 0.375/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1182;
    elseif 0.375/12/3.28084 < D_SI_b(i) <= 0.5/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1213;
    elseif 0.5/12/3.28084 < D_SI_b(i) <= 0.75/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1677;
    elseif 0.75/12/3.28084 < D_SI_b(i) <= 1/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1444;
    elseif 1/12/3.28084 < D_SI_b(i) <= 1.25/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1514;
    elseif 1.25/12/3.28084 < D_SI_b(i) <= 1.5/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1584;
    elseif 1.5/12/3.28084 < D_SI_b(i) <= 2/12/3.28084
        hangar_b_lb_per_ft(i) = 0.1231;
    elseif 2/12/3.28084 < D_SI_b(i) <= 2.5/12/3.28084
        hangar_b_lb_per_ft(i) = 0.2624;
    elseif 2.5/12/3.28084 < D_SI_b(i) <= 3/12/3.28084
        hangar_b_lb_per_ft(i) = 0.2798;
    elseif 3/12/3.28084 < D_SI_b(i) <= 3.5/12/3.28084
        hangar_b_lb_per_ft(i) = 0.2938;
    elseif 3.5/12/3.28084 < D_SI_b(i) <= 4/12/3.28084
        hangar_b_lb_per_ft(i) = 0.3902;
    elseif 4/12/3.28084 < D_SI_b(i) <= 5/12/3.28084
        hangar_b_lb_per_ft(i) = 0.2848;
    elseif 5/12/3.28084 < D_SI_b(i) <= 6/12/3.28084
        hangar_b_lb_per_ft(i) = 0.4952;
    elseif 6/12/3.28084 < D_SI_b(i) <= 8/12/3.28084
        hangar_b_lb_per_ft(i) = 0.5784;
    elseif 8/12/3.28084 < D_SI_b(i) <= 10/12/3.28084
        hangar_b_lb_per_ft(i) = 0.8453;
    elseif 10/12/3.28084 < D_SI_b(i) <= 12/12/3.28084
        hangar_b_lb_per_ft(i) = 0.8233;
    elseif 12/12/3.28084 < D_SI_b(i) <= 14/12/3.28084
        hangar_b_lb_per_ft(i) = 1.0456;
    elseif 14/12/3.28084 < D_SI_b(i) <= 16/12/3.28084
        hangar_b_lb_per_ft(i) = 1.0302;
    elseif 16/12/3.28084 < D_SI_b(i) <= 18/12/3.28084
        hangar_b_lb_per_ft(i) = 1.2802;
    elseif 18/12/3.28084 < D_SI_b(i) <= 20/12/3.28084
        hangar_b_lb_per_ft(i) = 1.2664;
    elseif 20/12/3.28084 < D_SI_b(i) <= 22/12/3.28084
        hangar_b_lb_per_ft(i) = 1.5139;
    else
        hangar_b_lb_per_ft(i) = 1.5014;
    end
end

bracket_b_weight = zeros(2,inputs);
for i=1:inputs
    for j=1:2
        bracket_b_weight(j,i) =
hangar_b_lb_per_ft(i)*length_b(j,i)/2.20462*3.28084;%kg
    end
end
bracket_b_weight_total = sum(sum(bracket_b_weight));
```

```
bracket_b_CG = pipe_b_CG;

if D_SI_h <= 0.25/12/3.28084
        hangar_h_lb_per_ft = 0.1161;
    elseif 0.25/12/3.28084 < D_SI_h <= 0.375/12/3.28084
        hangar_h_lb_per_ft = 0.1182;
    elseif 0.375/12/3.28084 < D_SI_h <= 0.5/12/3.28084
        hangar_h_lb_per_ft = 0.1213;
    elseif 0.5/12/3.28084 < D_SI_h <= 0.75/12/3.28084
        hangar_h_lb_per_ft = 0.1677;
    elseif 0.75/12/3.28084 < D_SI_h <= 1/12/3.28084
        hangar_h_lb_per_ft = 0.1444;
    elseif 1/12/3.28084 < D_SI_h <= 1.25/12/3.28084
        hangar_h_lb_per_ft = 0.1514;
    elseif 1.25/12/3.28084 < D_SI_h <= 1.5/12/3.28084
        hangar_h_lb_per_ft = 0.1584;
    elseif 1.5/12/3.28084 < D_SI_h <= 2/12/3.28084
        hangar_h_lb_per_ft = 0.1231;
    elseif 2/12/3.28084 < D_SI_h <= 2.5/12/3.28084
        hangar_h_lb_per_ft = 0.2624;
    elseif 2.5/12/3.28084 < D_SI_h <= 3/12/3.28084
        hangar_h_lb_per_ft = 0.2798;
    elseif 3/12/3.28084 < D_SI_h <= 3.5/12/3.28084
        hangar_h_lb_per_ft = 0.2938;
    elseif 3.5/12/3.28084 < D_SI_h <= 4/12/3.28084
        hangar_h_lb_per_ft = 0.3902;
    elseif 4/12/3.28084 < D_SI_h <= 5/12/3.28084
        hangar_h_lb_per_ft = 0.2848;
    elseif 5/12/3.28084 < D_SI_h <= 6/12/3.28084
        hangar_h_lb_per_ft = 0.4952;
    elseif 6/12/3.28084 < D_SI_h <= 8/12/3.28084
        hangar_h_lb_per_ft = 0.5784;
    elseif 8/12/3.28084 < D_SI_h <= 10/12/3.28084
        hangar_h_lb_per_ft = 0.8453;
    elseif 10/12/3.28084 < D_SI_h <= 12/12/3.28084
        hangar_h_lb_per_ft = 0.8233;
    elseif 12/12/3.28084 < D_SI_h <= 14/12/3.28084
        hangar_h_lb_per_ft = 1.0456;
    elseif 14/12/3.28084 < D_SI_h <= 16/12/3.28084
        hangar_h_lb_per_ft = 1.0302;
    elseif 16/12/3.28084 < D_SI_h <= 18/12/3.28084
        hangar_h_lb_per_ft = 1.2802;
    elseif 18/12/3.28084 < D_SI_h <= 20/12/3.28084
        hangar_h_lb_per_ft = 1.2664;
    elseif 20/12/3.28084 < D_SI_h <= 22/12/3.28084
        hangar_h_lb_per_ft = 1.5139;
    else
        hangar_h_lb_per_ft = 1.5014;
end
length_h_total =
sum(sum(length_h_s))+sum(sum(length_h_s_alt))+sum(sum(length_h_r))+sum(sum(le
ngth_h_r_alt))+length_h_cc;
bracket_h_weight = hangar_h_lb_per_ft/2.20462*3.28084*length_h_total;
bracket_h_CG = pipe_h_CG;
```

```
bracket_weight = bracket_b_weight_total + bracket_h_weight;
bracket_CG = [0 0 0];
bracket_CG(1) =
(bracket_b_CG(1)*bracket_b_weight_total+bracket_h_CG(1)*bracket_h_weight)/bra
cket_weight;
bracket_CG(2) =
(bracket_b_CG(2)*bracket_b_weight_total+bracket_h_CG(2)*bracket_h_weight)/bra
cket_weight;
bracket_CG(3) =
(bracket_b_CG(3)*bracket_b_weight_total+bracket_h_CG(3)*bracket_h_weight)/bra
cket_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine chilled water weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cw_CG = [0 0 0];
cw_weight = cw_b_weight+cw_h_weight+cw_hxchgr_weight+cw_tank_weight;
cw_CG(1) =
(cw_b_weight*cw_b_CG(1)+cw_h_weight*cw_h_CG(1)+cw_hxchgr_weight*cw_hxchgr_CG(
1)+cw_tank_weight*tank_CG(1))/cw_weight;
cw_CG(2) =
(cw_b_weight*cw_b_CG(2)+cw_h_weight*cw_h_CG(2)+cw_hxchgr_weight*cw_hxchgr_CG(
2)+cw_tank_weight*tank_CG(2))/cw_weight;
cw_CG(3) =
(cw_b_weight*cw_b_CG(3)+cw_h_weight*cw_h_CG(3)+cw_hxchgr_weight*cw_hxchgr_CG(
3)+cw_tank_weight*tank_CG(3))/cw_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine pump weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pump_sw_weight = 1500; %revise
pump_sw_weight_total = 0;
pump_sw_CG = [0 0 0];
size_sw_pumps = size(SW_pump_loc);
num_sw_pumps = size_sw_pumps(1);
for i=1:num_sw_pumps
    pump_sw_weight_total = pump_sw_weight_total + pump_sw_weight;
    pump_sw_CG(1) = pump_sw_CG(1)+pump_sw_weight*SW_pump_loc(i,1);
    pump_sw_CG(2) = pump_sw_CG(2)+pump_sw_weight*SW_pump_loc(i,2);
    pump_sw_CG(3) = pump_sw_CG(3)+pump_sw_weight*SW_pump_loc(i,3);
end
pump_sw_CG = pump_sw_CG/pump_sw_weight_total;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine sea water pipe weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pipe_sw_CG = [0 0 0];
pipe_sw_weight = 0;
length_sw_piping = zeros(num_sw_piping,size_sw_piping(2)-1);
for i=1:num_sw_piping
    for j=1:(size_sw_piping(2)-1)
        length_sw_piping(i,j) = sqrt((SW_piping(i,j,1)-
SW_piping(i,j+1,1))^2+(SW_piping(i,j,2)-SW_piping(i,j+1,2))^2+...
            (SW_piping(i,j,3)-SW_piping(i,j+1,3))^2);
```

```
        pipe_sw_weight = pipe_sw_weight +
length_sw_piping(i,j)*pipe_density*((D_SI_sw_piping(i)+thickness_sw_piping(i)
)^2-...
            D_SI_sw_piping(i)^2)*pi()/4;
        pipe_sw_CG(1) = pipe_sw_CG(1) +
(SW_piping(i,j,1)+SW_piping(i,j+1,1))/2*length_sw_piping(i,j)*pipe_density*..
.
            ((D_SI_sw_piping(i)+thickness_sw_piping(i))^2-
D_SI_sw_piping(i)^2)*pi()/4;
        pipe_sw_CG(2) = pipe_sw_CG(2) +
(SW_piping(i,j,2)+SW_piping(i,j+1,2))/2*length_sw_piping(i,j)*pipe_density*..
.
            ((D_SI_sw_piping(i)+thickness_sw_piping(i))^2-
D_SI_sw_piping(i)^2)*pi()/4;
        pipe_sw_CG(3) = pipe_sw_CG(3) +
(SW_piping(i,j,3)+SW_piping(i,j+1,3))/2*length_sw_piping(i,j)*pipe_density*..
.
            ((D_SI_sw_piping(i)+thickness_sw_piping(i))^2-
D_SI_sw_piping(i)^2)*pi()/4;
    end
end

length_sw_mains = zeros(1,num_sw_mains);
for i=1:num_sw_mains
    for j=1:(size_sw_mains(2)-1)
        length_sw_mains(i,j) = sqrt((SW_mains(i,j,1)-
SW_mains(i,j+1,1))^2+(SW_mains(i,j,2)-SW_mains(i,j+1,2))^2+...
            (SW_mains(i,j,3)-SW_mains(i,j+1,3))^2);
        pipe_sw_weight = pipe_sw_weight +
length_sw_mains(i,j)*pipe_density*((D_SI_sw_mains+thickness_sw_mains)^2-...
            D_SI_sw_mains^2)*pi()/4;
        pipe_sw_CG(1) = pipe_sw_CG(1) +
(SW_mains(i,j,1)+SW_mains(i,j+1,1))/2*length_sw_mains(i,j)*pipe_density*...
            ((D_SI_sw_mains+thickness_sw_mains)^2-D_SI_sw_mains^2)*pi()/4;
        pipe_sw_CG(2) = pipe_sw_CG(2) +
(SW_mains(i,j,2)+SW_mains(i,j+1,2))/2*length_sw_mains(i,j)*pipe_density*...
            ((D_SI_sw_mains+thickness_sw_mains)^2-D_SI_sw_mains^2)*pi()/4;
        pipe_sw_CG(3) = pipe_sw_CG(3) +
(SW_mains(i,j,3)+SW_mains(i,j+1,3))/2*length_sw_mains(i,j)*pipe_density*...
            ((D_SI_sw_mains+thickness_sw_mains)^2-D_SI_sw_mains^2)*pi()/4;
    end
end

length_sw_risers = zeros(1,num_sw_risers);
for i=1:num_sw_risers
    for j=1:(size_sw_risers(2)-1)
        length_sw_risers(i,j) = sqrt((SW_risers(i,j,1)-
SW_risers(i,j+1,1))^2+(SW_risers(i,j,2)-SW_risers(i,j+1,2))^2+...
            (SW_risers(i,j,3)-SW_risers(i,j+1,3))^2);
        pipe_sw_weight = pipe_sw_weight +
length_sw_risers(i,j)*pipe_density*((D_SI_sw_risers+thickness_sw_risers)^2-
...
            D_SI_sw_risers^2)*pi()/4;
```

```
        pipe_sw_CG(1) = pipe_sw_CG(1) +
(SW_risers(i,j,1)+SW_risers(i,j+1,1))/2*length_sw_risers(i,j)*pipe_density*..
.
            ((D_SI_sw_risers+thickness_sw_risers)^2-D_SI_sw_risers^2)*pi()/4;
        pipe_sw_CG(2) = pipe_sw_CG(2) +
(SW_risers(i,j,2)+SW_risers(i,j+1,2))/2*length_sw_risers(i,j)*pipe_density*..
.
            ((D_SI_sw_risers+thickness_sw_risers)^2-D_SI_sw_risers^2)*pi()/4;
        pipe_sw_CG(3) = pipe_sw_CG(3) +
(SW_risers(i,j,3)+SW_risers(i,j+1,3))/2*length_sw_risers(i,j)*pipe_density*..
.
            ((D_SI_sw_risers+thickness_sw_risers)^2-D_SI_sw_risers^2)*pi()/4;
    end
end

length_sw_cc = zeros(1,num_sw_cc);
for i=1:num_sw_cc
    for j=1:(size_sw_cc(2)-1)
        length_sw_cc(i,j) = sqrt((SW_cross_connects(i,j,1)-
SW_cross_connects(i,j+1,1))^2+(SW_cross_connects(i,j,2)-
SW_cross_connects(i,j+1,2))^2+...
            (SW_cross_connects(i,j,3)-SW_cross_connects(i,j+1,3))^2);
        pipe_sw_weight = pipe_sw_weight +
length_sw_cc(i,j)*pipe_density*((D_SI_sw_cc+thickness_sw_cc)^2-...
            D_SI_sw_cc^2)*pi()/4;
        pipe_sw_CG(1) = pipe_sw_CG(1) +
(SW_cross_connects(i,j,1)+SW_cross_connects(i,j+1,1))/2*length_sw_cc(i,j)*pip
e_density*...
            ((D_SI_sw_cc+thickness_sw_cc)^2-D_SI_sw_cc^2)*pi()/4;
        pipe_sw_CG(2) = pipe_sw_CG(2) +
(SW_cross_connects(i,j,2)+SW_cross_connects(i,j+1,2))/2*length_sw_cc(i,j)*pip
e_density*...
            ((D_SI_sw_cc+thickness_sw_cc)^2-D_SI_sw_cc^2)*pi()/4;
        pipe_sw_CG(3) = pipe_sw_CG(3) +
(SW_cross_connects(i,j,3)+SW_cross_connects(i,j+1,3))/2*length_sw_cc(i,j)*pip
e_density*...
            ((D_SI_sw_cc+thickness_sw_cc)^2-D_SI_sw_cc^2)*pi()/4;
    end
end
pipe_sw_CG = pipe_sw_CG/pipe_sw_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine sea water valve weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gate_valve_sw_mains_CG = [0 0 0];
gate_valve_sw_cc_CG = [0 0 0];
gate_valve_sw_mains_weight = 0;
gate_valve_sw_cc_weight = 0;
size_sw_mains_gate_valves = size(SW_valve_loc);
size_sw_cc_gate_valves = size(SW_cc_valve_loc);
if D_SI_sw_mains<Valve_diams_class_150(1)/12/ft_per_m
    for i=1:size_sw_mains_gate_valves(1)
        for j=1:size_sw_mains_gate_valves(2)
            gate_valve_sw_mains_weight =
gate_valve_sw_mains_weight+Gate_valve_weight_class_150(1);
```

```
            gate_valve_sw_mains_CG(1) = gate_valve_sw_mains_CG(1) +
SW_valve_loc(i,j,1)*Gate_valve_weight_class_150(1);
            gate_valve_sw_mains_CG(2) = gate_valve_sw_mains_CG(2) +
SW_valve_loc(i,j,2)*Gate_valve_weight_class_150(1);
            gate_valve_sw_mains_CG(3) = gate_valve_sw_mains_CG(3) +
SW_valve_loc(i,j,3)*Gate_valve_weight_class_150(1);
        end
    end
    for i=1:size_sw_cc_gate_valves(1)
        gate_valve_sw_cc_weight =
gate_valve_sw_cc_weight+Gate_valve_weight_class_150(1);
        gate_valve_sw_cc_CG(1) = gate_valve_sw_cc_CG(1) +
SW_cc_valve_loc(i,1)*Gate_valve_weight_class_150(1);
        gate_valve_sw_cc_CG(2) = gate_valve_sw_cc_CG(2) +
SW_cc_valve_loc(i,2)*Gate_valve_weight_class_150(1);
        gate_valve_sw_cc_CG(3) = gate_valve_sw_cc_CG(3) +
SW_cc_valve_loc(i,3)*Gate_valve_weight_class_150(1);
    end
else
    for j=1:max(size(Valve_diams_class_150))-1
        if (Valve_diams_class_150(j)/12/ft_per_m < D_SI_sw_mains) &&
(D_SI_sw_mains <= Valve_diams_class_150(j+1)/12/ft_per_m)
            for i=1:size_sw_mains_gate_valves(1)
                for k=1:size_sw_mains_gate_valves(2)
                    gate_valve_sw_mains_weight =
gate_valve_sw_mains_weight+Gate_valve_weight_class_150(j+1); %j or j+1
                    gate_valve_sw_mains_CG(1) = gate_valve_sw_mains_CG(1) +
SW_valve_loc(i,k,1)*Gate_valve_weight_class_150(j+1);
                    gate_valve_sw_mains_CG(2) = gate_valve_sw_mains_CG(2) +
SW_valve_loc(i,k,2)*Gate_valve_weight_class_150(j+1);
                    gate_valve_sw_mains_CG(3) = gate_valve_sw_mains_CG(3) +
SW_valve_loc(i,k,3)*Gate_valve_weight_class_150(j+1);
                end
            end
            for i=1:size_sw_cc_gate_valves(1)
                gate_valve_sw_cc_weight =
gate_valve_sw_cc_weight+Gate_valve_weight_class_150(j+1);
                gate_valve_sw_cc_CG(1) = gate_valve_sw_cc_CG(1) +
SW_cc_valve_loc(i,1)*Gate_valve_weight_class_150(j+1);
                gate_valve_sw_cc_CG(2) = gate_valve_sw_cc_CG(2) +
SW_cc_valve_loc(i,2)*Gate_valve_weight_class_150(j+1);
                gate_valve_sw_cc_CG(3) = gate_valve_sw_cc_CG(3) +
SW_cc_valve_loc(i,3)*Gate_valve_weight_class_150(j+1);
            end
        end
    end
end

gate_valve_sw_piping_weight = 0;
gate_valve_sw_piping_CG = [0 0 0];
size_sw_piping_gate_valves = size(SW_seg_valve_loc);
for i=1:size_sw_piping_gate_valves(1)
    for j=1:size_sw_piping_gate_valves(2)
        if D_SI_sw_piping(i)<Valve_diams_class_150(1)/12/ft_per_m
```

```
            gate_valve_sw_piping_weight =
gate_valve_sw_piping_weight+Gate_valve_weight_class_150(1);
            gate_valve_sw_piping_CG(1) = gate_valve_sw_piping_CG(1) +
SW_seg_valve_loc(i,j,1)*Gate_valve_weight_class_150(1);
            gate_valve_sw_piping_CG(2) = gate_valve_sw_piping_CG(2) +
SW_seg_valve_loc(i,j,2)*Gate_valve_weight_class_150(1);
            gate_valve_sw_piping_CG(3) = gate_valve_sw_piping_CG(3) +
SW_seg_valve_loc(i,j,3)*Gate_valve_weight_class_150(1);
        end
        for k=1:max(size(Valve_diams_class_150))-1
            if (Valve_diams_class_150(k)/12/ft_per_m < D_SI_sw_piping(i)) &&
(D_SI_sw_piping(i) <= Valve_diams_class_150(j+1)/12/ft_per_m)
                gate_valve_sw_piping_weight =
gate_valve_sw_piping_weight+Gate_valve_weight_class_150(j+1); %j or j+1
                gate_valve_sw_piping_CG(1) = gate_valve_sw_piping_CG(1) +
SW_seg_valve_loc(i,j,1)*Gate_valve_weight_class_150(j+1);
                gate_valve_sw_piping_CG(2) = gate_valve_sw_piping_CG(2) +
SW_seg_valve_loc(i,j,2)*Gate_valve_weight_class_150(j+1);
                gate_valve_sw_piping_CG(3) = gate_valve_sw_piping_CG(3) +
SW_seg_valve_loc(i,j,3)*Gate_valve_weight_class_150(j+1);
            end
        end
    end
end

valve_sw_weight =
gate_valve_sw_cc_weight+gate_valve_sw_mains_weight+gate_valve_sw_piping_weigh
t;
valve_sw_CG =
(gate_valve_sw_cc_CG+gate_valve_sw_mains_CG+gate_valve_sw_piping_CG)/valve_sw
_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine sea water bracket weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bracket_weight = 0;
hangar_sw_b_lb_per_ft = zeros(1,inputs);
for i=1:num_sw_piping
    if D_SI_sw_piping(i) <= 0.25/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1161;
    elseif 0.25/12/3.28084 < D_SI_sw_piping(i) <= 0.375/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1182;
    elseif 0.375/12/3.28084 < D_SI_sw_piping(i) <= 0.5/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1213;
    elseif 0.5/12/3.28084 < D_SI_sw_piping(i) <= 0.75/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1677;
    elseif 0.75/12/3.28084 < D_SI_sw_piping(i) <= 1/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1444;
    elseif 1/12/3.28084 < D_SI_sw_piping(i) <= 1.25/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1514;
    elseif 1.25/12/3.28084 < D_SI_sw_piping(i) <= 1.5/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1584;
    elseif 1.5/12/3.28084 < D_SI_sw_piping(i) <= 2/12/3.28084
        hangar_sw_b_lb_per_ft(i) = 0.1231;
    elseif 2/12/3.28084 < D_SI_sw_piping(i) <= 2.5/12/3.28084
```

```
            hangar_sw_b_lb_per_ft(i) = 0.2624;
        elseif 2.5/12/3.28084 < D_SI_sw_piping(i) <= 3/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.2798;
        elseif 3/12/3.28084 < D_SI_sw_piping(i) <= 3.5/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.2938;
        elseif 3.5/12/3.28084 < D_SI_sw_piping(i) <= 4/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.3902;
        elseif 4/12/3.28084 < D_SI_sw_piping(i) <= 5/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.2848;
        elseif 5/12/3.28084 < D_SI_sw_piping(i) <= 6/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.4952;
        elseif 6/12/3.28084 < D_SI_sw_piping(i) <= 8/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.5784;
        elseif 8/12/3.28084 < D_SI_sw_piping(i) <= 10/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.8453;
        elseif 10/12/3.28084 < D_SI_sw_piping(i) <= 12/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 0.8233;
        elseif 12/12/3.28084 < D_SI_sw_piping(i) <= 14/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 1.0456;
        elseif 14/12/3.28084 < D_SI_sw_piping(i) <= 16/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 1.0302;
        elseif 16/12/3.28084 < D_SI_sw_piping(i) <= 18/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 1.2802;
        elseif 18/12/3.28084 < D_SI_sw_piping(i) <= 20/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 1.2664;
        elseif 20/12/3.28084 < D_SI_sw_piping(i) <= 22/12/3.28084
            hangar_sw_b_lb_per_ft(i) = 1.5139;
        else
            hangar_sw_b_lb_per_ft(i) = 1.5014;
        end
    end

    bracket_sw_weight = 0;
    for i=1:num_sw_piping
        bracket_sw_weight = bracket_sw_weight +
    hangar_sw_b_lb_per_ft(i)*length_sw_piping(i)/2.20462*3.28084;%kg
    end

    if D_SI_sw_mains <= 0.25/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1161;
        elseif 0.25/12/3.28084 < D_SI_sw_mains <= 0.375/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1182;
        elseif 0.375/12/3.28084 < D_SI_sw_mains <= 0.5/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1213;
        elseif 0.5/12/3.28084 < D_SI_sw_mains <= 0.75/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1677;
        elseif 0.75/12/3.28084 < D_SI_sw_mains <= 1/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1444;
        elseif 1/12/3.28084 < D_SI_sw_mains <= 1.25/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1514;
        elseif 1.25/12/3.28084 < D_SI_sw_mains <= 1.5/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1584;
        elseif 1.5/12/3.28084 < D_SI_sw_mains <= 2/12/3.28084
            hangar_sw_h_lb_per_ft = 0.1231;
        elseif 2/12/3.28084 < D_SI_sw_mains <= 2.5/12/3.28084
```

```
            hangar_sw_h_lb_per_ft = 0.2624;
        elseif 2.5/12/3.28084 < D_SI_sw_mains <= 3/12/3.28084
            hangar_sw_h_lb_per_ft = 0.2798;
        elseif 3/12/3.28084 < D_SI_sw_mains <= 3.5/12/3.28084
            hangar_sw_h_lb_per_ft = 0.2938;
        elseif 3.5/12/3.28084 < D_SI_sw_mains <= 4/12/3.28084
            hangar_sw_h_lb_per_ft = 0.3902;
        elseif 4/12/3.28084 < D_SI_sw_mains <= 5/12/3.28084
            hangar_sw_h_lb_per_ft = 0.2848;
        elseif 5/12/3.28084 < D_SI_sw_mains <= 6/12/3.28084
            hangar_sw_h_lb_per_ft = 0.4952;
        elseif 6/12/3.28084 < D_SI_sw_mains <= 8/12/3.28084
            hangar_sw_h_lb_per_ft = 0.5784;
        elseif 8/12/3.28084 < D_SI_sw_mains <= 10/12/3.28084
            hangar_sw_h_lb_per_ft = 0.8453;
        elseif 10/12/3.28084 < D_SI_sw_mains <= 12/12/3.28084
            hangar_sw_h_lb_per_ft = 0.8233;
        elseif 12/12/3.28084 < D_SI_sw_mains <= 14/12/3.28084
            hangar_sw_h_lb_per_ft = 1.0456;
        elseif 14/12/3.28084 < D_SI_sw_mains <= 16/12/3.28084
            hangar_sw_h_lb_per_ft = 1.0302;
        elseif 16/12/3.28084 < D_SI_sw_mains <= 18/12/3.28084
            hangar_sw_h_lb_per_ft = 1.2802;
        elseif 18/12/3.28084 < D_SI_sw_mains <= 20/12/3.28084
            hangar_sw_h_lb_per_ft = 1.2664;
        elseif 20/12/3.28084 < D_SI_sw_mains <= 22/12/3.28084
            hangar_sw_h_lb_per_ft = 1.5139;
        else
            hangar_sw_h_lb_per_ft = 1.5014;
    end
    bracket_sw_weight =
    bracket_sw_weight+hangar_sw_h_lb_per_ft/2.20462*3.28084*...

    (sum(sum(length_sw_mains))+sum(sum(length_sw_risers))+sum(sum(length_sw_cc)))
    ;
    bracket_sw_CG = pipe_sw_CG;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine sea water weight, LCG, VCG, and TCG
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    sw_density = 1029; %kg/m^3
    sw_weight = 0;
    for i=1:num_sw_piping
        for j=1:(size_sw_piping(2)-1)
            sw_weight = sw_weight +
    length_sw_piping(i,j)*sw_density*D_SI_sw_piping(i)^2*pi()/4;
        end
    end
    for i=1:num_sw_mains
        for j=1:(size_sw_mains(2)-1)
            sw_weight = sw_weight +
    length_sw_mains(i,j)*sw_density*D_SI_sw_mains^2*pi()/4;
        end
    end
    for i=1:num_sw_risers
```

```
    for j=1:(size_sw_risers(2)-1)
        sw_weight = sw_weight +
length_sw_risers(i,j)*sw_density*D_SI_sw_risers^2*pi()/4;
    end
end
for i=1:num_sw_cc
    for j=1:(size_sw_cc(2)-1)
        sw_weight = sw_weight +
length_sw_cc(i,j)*sw_density*D_SI_sw_cc^2*pi()/4;
    end
end
sw_CG = pipe_sw_CG;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine total weight, LCG, VCG, and TCG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CW_weight_total =
pipe_weight+lagging_weight+valve_weight+chiller_weight_total+total_tank_weigh
t+pump_cw_weight_total...
    +bracket_weight+total_tank_instr_weight+cw_weight+hxchgr_weight;
CW_CG_total = [0 0 0];
CW_CG_total(1) =
(pipe_CG(1)*pipe_weight+lagging_CG(1)*lagging_weight+valve_CG(1)*valve_weight
+ ...

chiller_CG(1)*chiller_weight_total+tank_CG(1)*total_tank_weight+pump_cw_CG(1)
*pump_cw_weight_total+bracket_CG(1)*bracket_weight+ ...

tank_instr_CG(1)*total_tank_instr_weight+cw_CG(1)*cw_weight+hxchgr_CG(1)*hxch
gr_weight)/CW_weight_total;
CW_CG_total(2) =
(pipe_CG(2)*pipe_weight+lagging_CG(2)*lagging_weight+valve_CG(2)*valve_weight
+ ...

chiller_CG(2)*chiller_weight_total+tank_CG(2)*total_tank_weight+pump_cw_CG(2)
*pump_cw_weight_total+bracket_CG(2)*bracket_weight+ ...

tank_instr_CG(2)*total_tank_instr_weight+cw_CG(2)*cw_weight+hxchgr_CG(2)*hxch
gr_weight)/CW_weight_total;
CW_CG_total(3) =
(pipe_CG(3)*pipe_weight+lagging_CG(3)*lagging_weight+valve_CG(3)*valve_weight
+ ...

chiller_CG(3)*chiller_weight_total+tank_CG(3)*total_tank_weight+pump_cw_CG(3)
*pump_cw_weight_total+bracket_CG(3)*bracket_weight+ ...

tank_instr_CG(3)*total_tank_instr_weight+cw_CG(3)*cw_weight+hxchgr_CG(3)*hxch
gr_weight)/CW_weight_total;

SW_weight_total =
pipe_sw_weight+valve_sw_weight+pump_sw_weight_total+bracket_sw_weight+sw_weig
ht;
SW_CG_total = [0 0 0];
```

```
SW_CG_total(1) =
(pipe_sw_weight*pipe_sw_CG(1)+valve_sw_weight*valve_sw_CG(1)+pump_sw_weight_t
otal*pump_sw_CG(1)+...
    bracket_sw_weight*bracket_sw_CG(1)+sw_weight*sw_CG(1))/SW_weight_total;
SW_CG_total(2) =
(pipe_sw_weight*pipe_sw_CG(2)+valve_sw_weight*valve_sw_CG(2)+pump_sw_weight_t
otal*pump_sw_CG(2)+...
    bracket_sw_weight*bracket_sw_CG(2)+sw_weight*sw_CG(2))/SW_weight_total;
SW_CG_total(3) =
(pipe_sw_weight*pipe_sw_CG(3)+valve_sw_weight*valve_sw_CG(3)+pump_sw_weight_t
otal*pump_sw_CG(3)+...
    bracket_sw_weight*bracket_sw_CG(3)+sw_weight*sw_CG(3))/SW_weight_total;

total_weight = CW_weight_total + SW_weight_total;
total_CG(1) =
(CW_weight_total*CW_CG_total(1)+SW_weight_total*SW_CG_total(1))/total_weight;
total_CG(2) =
(CW_weight_total*CW_CG_total(2)+SW_weight_total*SW_CG_total(2))/total_weight;
total_CG(3) =
(CW_weight_total*CW_CG_total(3)+SW_weight_total*SW_CG_total(3))/total_weight;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Margin
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Please enter the weight margin for the CW and SW systems (enter as a
decimal).\n')
margin = input('Weight margin: ');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Print weight report
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('\n\n-------------------------------------------------------------------
---------\n')
fprintf('Report 4: CW/SW Weight Summary\n')
fprintf('---------------------------------------------------------------------
-----\n')
fprintf('Item                    Weight (MT)       LCG (m)       TCG (m)       VCG
(m)\n')
fprintf('CW System:                %10.4f   %10.4f   %10.4f   %10.4f\n',
CW_weight_total/1000, CW_CG_total(1), CW_CG_total(2), CW_CG_total(3))
fprintf('  Pipe:                   %10.4f   %10.4f   %10.4f   %10.4f\n',
pipe_weight/1000, pipe_CG(1), pipe_CG(2), pipe_CG(3))
fprintf('    Main:                 %10.4f   %10.4f   %10.4f   %10.4f\n',
pipe_h_weight/1000, pipe_h_CG(1), pipe_h_CG(2), pipe_h_CG(3))
fprintf('    Branch:               %10.4f   %10.4f   %10.4f   %10.4f\n',
pipe_b_weight/1000, pipe_b_CG(1), pipe_b_CG(2), pipe_b_CG(3))
fprintf('  Lagging:                %10.4f   %10.4f   %10.4f   %10.4f\n',
lagging_weight/1000, lagging_CG(1), lagging_CG(2), lagging_CG(3))
fprintf('    Main:                 %10.4f   %10.4f   %10.4f   %10.4f\n',
lagging_h_weight/1000, lagging_h_CG(1), lagging_h_CG(2), lagging_h_CG(3))
fprintf('    Branch:               %10.4f   %10.4f   %10.4f   %10.4f\n',
lagging_b_weight/1000, lagging_b_CG(1), lagging_b_CG(2), lagging_b_CG(3))
fprintf('  Valves:                 %10.4f   %10.4f   %10.4f   %10.4f\n',
valve_weight/1000, valve_CG(1), valve_CG(2), valve_CG(3))
```

```
fprintf('    Globe:              %10.4f   %10.4f   %10.4f   %10.4f\n',
globe_valve_weight/1000, globe_valve_CG(1), globe_valve_CG(2),
globe_valve_CG(3))
fprintf('      Main:             %10.4f    %10.4f    %10.4f    %10.4f\n',
globe_valve_h_weight/1000, globe_valve_h_CG(1), globe_valve_h_CG(2),
globe_valve_h_CG(3))
fprintf('      Branch:           %10.4f    %10.4f    %10.4f    %10.4f\n',
globe_valve_b_weight/1000, globe_valve_b_CG(1), globe_valve_b_CG(2),
globe_valve_b_CG(3))
fprintf('    Gate:               %10.4f   %10.4f   %10.4f   %10.4f\n',
gate_valve_weight/1000, gate_valve_CG(1), gate_valve_CG(2), gate_valve_CG(3))
fprintf('      Main:             %10.4f    %10.4f    %10.4f    %10.4f\n',
gate_valve_h_weight/1000, gate_valve_h_CG(1), gate_valve_h_CG(2),
gate_valve_h_CG(3))
fprintf('      Branch:           %10.4f    %10.4f    %10.4f    %10.4f\n',
gate_valve_b_weight/1000, gate_valve_b_CG(1), gate_valve_b_CG(2),
gate_valve_b_CG(3))
fprintf('    Check:              %10.4f   %10.4f   %10.4f   %10.4f\n',
check_valve_weight/1000, check_valve_CG(1), check_valve_CG(2),
check_valve_CG(3))
fprintf('      Main:             %10.4f    %10.4f    %10.4f    %10.4f\n',
check_valve_h_weight/1000, check_valve_h_CG(1), check_valve_h_CG(2),
check_valve_h_CG(3))
fprintf('      Branch:           %10.4f    %10.4f    %10.4f    %10.4f\n',
check_valve_b_weight/1000, check_valve_b_CG(1), check_valve_b_CG(2),
check_valve_b_CG(3))
fprintf('  Chillers:             %10.4f   %10.4f   %10.4f   %10.4f\n',
chiller_weight_total/1000, chiller_CG(1), chiller_CG(2), chiller_CG(3))
fprintf('  Expansion tanks:      %10.4f   %10.4f   %10.4f   %10.4f\n',
total_tank_weight/1000, tank_CG(1), tank_CG(2), tank_CG(3))
fprintf('  Pumps:                %10.4f   %10.4f   %10.4f   %10.4f\n',
pump_cw_weight_total/1000, pump_cw_CG(1), pump_cw_CG(2), pump_cw_CG(3))
fprintf('  Brackets:             %10.4f   %10.4f   %10.4f   %10.4f\n',
bracket_weight/1000, bracket_CG(1), bracket_CG(2), bracket_CG(3))
fprintf('  Instrumentation:      %10.4f   %10.4f   %10.4f   %10.4f\n',
total_tank_instr_weight/1000, tank_instr_CG(1), tank_instr_CG(2),
tank_instr_CG(3))
fprintf('  Chilled water:        %10.4f   %10.4f   %10.4f   %10.4f\n',
cw_weight/1000, cw_CG(1), cw_CG(2), cw_CG(3))
fprintf('  Heat Exchangers:      %10.4f   %10.4f   %10.4f   %10.4f\n',
hxchgr_weight/1000, hxchgr_CG(1), hxchgr_CG(2), hxchgr_CG(3))
fprintf('SW System:            %10.4f   %10.4f   %10.4f   %10.4f\n',
SW_weight_total/1000, SW_CG_total(1), SW_CG_total(2), SW_CG_total(3))
fprintf('  Pipe:                 %10.4f   %10.4f   %10.4f   %10.4f\n',
pipe_sw_weight/1000, pipe_sw_CG(1), pipe_sw_CG(2), pipe_sw_CG(3))
fprintf('  Valves:               %10.4f   %10.4f   %10.4f   %10.4f\n',
valve_sw_weight/1000, valve_sw_CG(1), valve_sw_CG(2), valve_sw_CG(3))
fprintf('  Pumps:                %10.4f   %10.4f   %10.4f   %10.4f\n',
pump_sw_weight_total/1000, pump_sw_CG(1), pump_sw_CG(2), pump_sw_CG(3))
fprintf('  Brackets:             %10.4f   %10.4f   %10.4f   %10.4f\n',
bracket_sw_weight/1000, bracket_sw_CG(1), bracket_sw_CG(2), bracket_sw_CG(3))
fprintf('  Salt water:           %10.4f   %10.4f   %10.4f   %10.4f\n',
sw_weight/1000, sw_CG(1), sw_CG(2), sw_CG(3))
fprintf('-----------------------------------------------------------------
-----\n')
```

```
fprintf('Total:                  %10.4f   %10.4f   %10.4f   %10.4f\n',
total_weight/1000, total_CG(1), total_CG(2), total_CG(3))
fprintf('Margin:                 %10.4f   %10.4f   %10.4f   %10.4f\n',
total_weight*margin/1000, total_CG(1), total_CG(2), total_CG(3))
fprintf('-------------------------------------------------------------------
-----\n')
fprintf('Total with margin:  %10.4f   %10.4f   %10.4f   %10.4f\n',
total_weight*(1+margin)/1000, total_CG(1), total_CG(2), total_CG(3))
fprintf('\n')

save analysis

%% Step 13 Static temperature analysis
fprintf('The static analysis module provides a means to determine the
temperature\n')
fprintf('at certain locations over the entire system for a single chiller
line-up\n')
fprintf('which is specified by the user through the use of an excel
spreadsheet.\n\n')

fprintf('The possible load conditions are: \n')
Condition_Labels
fprintf('Of the above load conditions, which do you want to analyze when
performing\n')
fprintf('the static analysis?\n')
load_condition = menu('Select the load
condition','Shore','Design','Cruise','Battle');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input File
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'SteadyState.xlsx';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clear input file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear_vars = NaN(1000,12);
xlswrite(filename,clear_vars,1,'B11');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Order Load_Name, Q, Load_Value_kW
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Load_Name_Ordered = Load_Name;
Q_Ordered = zeros(size(Q));
Load_Value_kW_Ordered = zeros(size(Load_Value_kW));
for i=1:inputs
    Load_Name_Ordered(i) = Load_Name(branch_order(1,1,i));
    Q_Ordered(i) = Q(branch_order(1,1,i));
    Load_Value_kW_Ordered(i,:) = Load_Value_kW(branch_order(1,1,i),:);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set initial conditions in Excel Sheet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
load_number = [1:inputs]';
xlswrite(filename,load_number,1,'B11');
xlswrite(filename,Load_Name_Ordered,1,'C11');
xlswrite(filename,Q_Ordered/1000,1,'D11');
if load_condition == 1
    xlswrite(filename,Load_Value_kW_Ordered(:,1),1,'E11');
elseif load_condition == 2
    xlswrite(filename,Load_Value_kW_Ordered(:,2),1,'E11');
elseif load_condition == 3
   xlswrite(filename,Load_Value_kW_Ordered(:,3),1,'E11');
elseif load_condition == 4
   xlswrite(filename,Load_Value_kW_Ordered(:,4),1,'E11');
else
    fprintf('Error selecting load condition\n')
end

chiller_number = [1:num_chillers]';
xlswrite(filename,chiller_number,1,'G11');
xlswrite(filename,chiller_loc,1,'H11');

fprintf('Please open up the Excel file SteadyState.xlsx and provide the heat
load values before\n')
fprintf('and after the transient and the chiller configuration before and
after the transient\n')
fprintf('before proceeding through the analysis module.\n')

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in values from Excel Sheet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[num,txt] = xlsread(filename,'SteadyState');
static_Q = num(1:inputs,4);
static_chiller_status = txt(10:num_chillers+9,10);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert static_riser_branch_index from riser_branch_index and determine
% number of chillers in operation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
static_num_chillers = 0;
static_riser_count_index = 1;
static_riser_branch_index = 0;
for i=1:num_chillers
    if strcmp(static_chiller_status(i),'on')
        static_num_chillers = static_num_chillers+1;
        static_riser_branch_index(static_riser_count_index) =
riser_branch_index(i);
        static_riser_count_index = static_riser_count_index+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine total mass flow rates between risers of operational
chillers/pumps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
static_mfr_total = zeros(1,static_num_chillers);
for i=1:static_num_chillers-1
    for j=static_riser_branch_index(i):static_riser_branch_index(i+1)-1
        static_mfr_total(i) = static_mfr_total(i)+mfr_b_ordered(1,1,j);
    end
end
for i=static_riser_branch_index(static_num_chillers):inputs
    static_mfr_total(static_num_chillers) =
static_mfr_total(static_num_chillers)+mfr_b_ordered(1,1,i);
end
if static_riser_branch_index(1)~=1
    for i=1:static_riser_branch_index(1)-1
        static_mfr_total(static_num_chillers) =
static_mfr_total(static_num_chillers)+mfr_b_ordered(1,1,i);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find branch index corresponding to half-flow between segments (these are
% initial guesses at stagnation points)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
static_mfr_temp = zeros(1,static_num_chillers);
static_mfr_temp_index = static_riser_branch_index;
for i=1:static_num_chillers-1
    for j=static_riser_branch_index(i):static_riser_branch_index(i+1)-1
        if static_mfr_temp(i)*2 < static_mfr_total(i)
            static_mfr_temp(i) = static_mfr_temp(i)+mfr_b_ordered(1,1,j);
            static_mfr_temp_index(i) = static_mfr_temp_index(i)+1;
        end
    end
end
for i=static_riser_branch_index(static_num_chillers):inputs
    if static_mfr_temp(static_num_chillers)*2 <
static_mfr_total(static_num_chillers)
        static_mfr_temp(static_num_chillers) =
static_mfr_temp(static_num_chillers)+mfr_b_ordered(1,1,i);
        static_mfr_temp_index(static_num_chillers) =
static_mfr_temp_index(static_num_chillers)+1;
    end
end
if static_riser_branch_index(1)~=1
    if static_mfr_total(static_num_chillers)*2 <
static_mfr_total(static_num_chillers)
        static_mfr_temp_index(static_num_chillers)=1;
    end
    for i=1:static_riser_branch_index(1)-1
        if static_mfr_total(static_num_chillers)*2 <
static_mfr_total(static_num_chillers)
            static_mfr_total(static_num_chillers) =
static_mfr_total(static_num_chillers)+mfr_b_ordered(1,1,i);
            static_mfr_temp_index(static_num_chillers) =
static_mfr_temp_index(static_num_chillers)+1;
        end
    end
end
```

```
static_stag_branch_index = static_mfr_temp_index;

mfr_total_seg = zeros(3,static_num_chillers);
for i=1:static_num_chillers
    mfr_total_seg(1,i) = static_mfr_temp(i);
    if i~=static_num_chillers
        mfr_total_seg(2,i+1) = static_mfr_total(i)-static_mfr_temp(i);
    else
        mfr_total_seg(2,1) = static_mfr_total(static_num_chillers)-
static_mfr_temp(static_num_chillers);
    end
end
mfr_total_seg(3,:) = mfr_total_seg(1,:)+mfr_total_seg(2,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resize and re-order V_SI_b and store in V_SI_b_seg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_seg = zeros(1,inputs);
for m=1:inputs
    V_SI_b_seg(m) = V_SI_b_1(branch_order(1,1,m));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterate through loop a predetermined number of times, modifying the
% branch diameters to satisfy the velocity limits set forth by NAVSEA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
while count<10
    count=count+1;

    if count == 1 %use estimated V_SI_b_seg to begin iterative process and
only consider friction bends and valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_b_seg due to friction, bends, valves for branches
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
            K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2;
%due to gate valves
            K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5;
%due to globe valves
```

```
            K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_h_seg due to friction, bends, valves for supply
header
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,1,i)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,1,i)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,1,i)*0.2;
        %     K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %     K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %         K_loss_globe_h(i)+K_loss_check_h(i);
        end
        for i=inputs+1

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
        %     K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %     K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %         K_loss_globe_h(i)+K_loss_check_h(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_rh_seg due to friction, bends, valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs+1
```

```
        %     K_loss_friction_rh(i)=f_b(1)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %     K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_b(1)*pi()/2*r_d(i)+(0.10+2.4*f_b(1))*sin(pi()/4) ...
        %
+6.6*f_b(1)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %     K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %     K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
        %     K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_gate_rh(i)+K_loss_globe_rh(
i);
            K_loss_rh_seg(i) = K_loss_h_seg(i); %assume same loss coefficient
for supply and return header segments
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_b/A_b^2 and K_h/A_h^2 for branches and header segments
        % respectively
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
        end
        for i=1:inputs+1
            K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2 = zeros(1,inputs);
        for i=1:static_num_chillers
            if i==1
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=static_stag_branch_index(i) %15
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %1:14
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            else
                for j=static_stag_branch_index(i-1)+1 %16
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
```

```
                end
                for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=static_stag_branch_index(i) %60
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59:38
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2_oa = zeros(1,static_num_chillers);
        for i=1:static_num_chillers
            if i==1
                K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(static_riser_branch_index(i))^0.5))^2;
            else
                K_A_2_oa(i) = (1/(1/K_A_2(static_riser_branch_index(i)-
1)^0.5+1/K_A_2(static_riser_branch_index(i))^0.5))^2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_oa = zeros(2,static_num_chillers); %cw=1, ccw=2
        for i=1:static_num_chillers
            if i==1
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
            else
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)-1))^0.5;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_temp
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_b = zeros(1,inputs);
        mfr_seg_temp = zeros(1,inputs);
```

```
        for i=1:static_num_chillers
            if i==1
                for
j=static_riser_branch_index(i):static_stag_branch_index(i) %1:15
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(static_riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1:inputs
%164:180
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
                end
            else
                for
j=static_riser_branch_index(i):static_stag_branch_index(i) %38:60
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(static_riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for j=static_stag_branch_index(i-
1)+1:static_riser_branch_index(i)-1 %16:37
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(static_riser_branch_index(i)-1)/K_A_2(j))^0.5;
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_b
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:static_num_chillers
            if i==1
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=static_stag_branch_index(i) %15
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %14:1
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            else
                for j=static_stag_branch_index(i-1)+1 %16
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
```

```
            end
            for j=static_stag_branch_index(i) %60
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59;38
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_h
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_h = zeros(1,inputs);
    for i=1:static_num_chillers
        if i==1
            for j=static_stag_branch_index(i) %15
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %14;1
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        else
            for j=static_stag_branch_index(i) %60
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59:38
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=static_stag_branch_index(i-1)+1 %16
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_b_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
```

```
            V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate V_SI_h_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate loss coefficient for branches due to friction, bends,
    % valves, entrance and exit effects (in order wrt header)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_b_seg = zeros(1,inputs);
    K_loss_exit_b_seg = zeros(1,inputs);
    for i=1:inputs
        f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
        K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2; %due to
gate valves
        K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5; %due
to globe valves
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));

        Cyc(i) = 1-0.25*(D_SI_b(branch_order(1,1,i))/D_SI_h)^1.3-(0.11*r_d3-
0.65*r_d3^2+0.83*r_d3^3)*D_SI_b(branch_order(1,1,i))^2/D_SI_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance and exit effects for branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Keq = 0.57-1.07*r_d3^0.5-2.13*r_d3+8.24*r_d3^1.5-
8.48*r_d3^2+2.9*r_d3^2.5;
    Cxc = 0.08+0.56*r_d3-1.75*r_d3^2+1.83*r_d3^3;
    Cm = 0.23+1.46*r_d3-2.75*r_d3^2+1.65*r_d3^3;
    for j=1:static_num_chillers
        if j==1
            for i=static_riser_branch_index(j):static_stag_branch_index(j)
%cw 1:15
```

```
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=inputs:-
1:static_stag_branch_index(max(size(static_stag_branch_index)))+1 %ccw
180:164
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        else
            for i=static_riser_branch_index(j):static_stag_branch_index(j)
%cw 38:60
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=static_riser_branch_index(j)-1:-
1:static_stag_branch_index(j-1)+1 %ccw 37:16
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
```

```
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_loss_b_seg and K_loss_b_in_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_b_in_seg = zeros(1,inputs);
    for i=1:inputs
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+K_loss_exit_b_seg(i);
        K_loss_b_in_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+0*K_loss_exit_b_seg(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_b_seg(i) <= 0
            K_loss_b_seg(i) = 0.01; %negligible loss coefficient
        end
        if K_loss_b_in_seg(i) <= 0
            K_loss_b_in_seg(i) = 0.01; %negligible loss coefficient
        end
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate loss coefficient for supply header due to friction, bends,
    % valves, entrance and exit effects
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_h_seg = zeros(1,inputs);
    for i=1:inputs
        f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
        K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
        K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
    %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
    %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves considered
    end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:static_num_chillers
        if j==1
            for i=static_stag_branch_index(j) %cw 15
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=static_riser_branch_index(j):static_stag_branch_index(j)-1
%cw 1:14
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for
i=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %ccw 164
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=inputs:-
1:static_stag_branch_index(max(size(static_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        else
            for i=static_stag_branch_index(j) %cw 60
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=static_riser_branch_index(j):static_stag_branch_index(j)-1
%cw 38:59
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for i=static_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=static_riser_branch_index(j)-1:-
1:static_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        end
    end
    for i=1:inputs
        K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i)+K_loss_
entrance_h_seg(i);%+ ...
```

```
        %    K_loss_globe_h(i)+K_loss_check_h(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_h_seg(i) <= 0
            K_loss_h_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate K_loss_rh due to friction, bends, valves
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_rh_seg = zeros(1,inputs);
    for i=1:inputs
        %K_loss_friction_rh(i)=f_h(i)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_h(i)*pi()/2*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/4) ...
        %
+6.6*f_h(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %K_loss_bend_180_rh(i) =
bends_180_rh(i)*(f_h(i)*pi()*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/2) ...
        %
+6.6*f_h(i)*((sin(pi()/2))^0.5+sin(pi()/2))/r_d(i)^(4*pi()/pi())); %due to
180 bends
        %K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:static_num_chillers
        if j==1
            for i=static_stag_branch_index(j) %cw 15
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=static_riser_branch_index(j):static_stag_branch_index(j)-1
%cw 1:14
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for
i=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %ccw 164
                K_loss_entrance_rh_seg(i) = 0;
            end
```

```
            for i=inputs:-
1:static_stag_branch_index(max(size(static_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        else
            for i=static_stag_branch_index(j) %cw 60
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=static_riser_branch_index(j):static_stag_branch_index(j)-1
%cw 38:59
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for i=static_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=static_riser_branch_index(j)-1:-
1:static_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        end
    end
    for i=1:inputs
        K_loss_rh_seg(i) = K_loss_h_seg(i)-
K_loss_entrance_h_seg(i)+K_loss_entrance_rh_seg(i);
        %K_loss_rh_seg(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_bend_180_rh(i)+K_loss_gate_
rh(i)+ ...
        %    K_loss_globe_rh(i)+K_loss_entrance_rh(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_rh_seg(i) <= 0
            K_loss_rh_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
```

```
        K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
    end
    for i=1:inputs+1
        K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2 = zeros(1,inputs);
    for i=1:static_num_chillers
        if i==1
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=static_stag_branch_index(i) %15
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %1:14
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        else
            for j=static_stag_branch_index(i-1)+1 %16
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=static_stag_branch_index(i) %60
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59:38
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2_oa = zeros(1,static_num_chillers);
```

```
    for i=1:static_num_chillers
        if i==1
            K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(static_riser_branch_index(i))^0.5))^2;
        else
            K_A_2_oa(i) = (1/(1/K_A_2(static_riser_branch_index(i)-
1)^0.5+1/K_A_2(static_riser_branch_index(i))^0.5))^2;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_oa = zeros(2,static_num_chillers); %cw=1, ccw=2
    for i=1:static_num_chillers
        if i==1
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
        else
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(static_riser_branch_index(i)-1))^0.5;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_temp
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_b = zeros(1,inputs);
    mfr_seg_temp = zeros(1,inputs);
    for i=1:static_num_chillers
        if i==1
            for j=static_riser_branch_index(i):static_stag_branch_index(i)
%1:15
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(static_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1:inputs
%164:180
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
            end
        else
            for j=static_riser_branch_index(i):static_stag_branch_index(i)
%38:60
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(static_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for j=static_stag_branch_index(i-
1)+1:static_riser_branch_index(i)-1 %16:37
```

```
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(static_riser_branch_index(i)-1)/K_A_2(j))^0.5;
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:static_num_chillers
            if i==1
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=static_stag_branch_index(i) %15
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %14:1
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            else
                for j=static_stag_branch_index(i-1)+1 %16
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=static_stag_branch_index(i) %60
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59;38
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_h
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_h = zeros(1,inputs);
        for i=1:static_num_chillers
            if i==1
                for j=static_stag_branch_index(i) %15
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
```

```
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %14;1
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+1 %164
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for
j=static_stag_branch_index(max(size(static_stag_branch_index)))+2:inputs
%165:180
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        else
            for j=static_stag_branch_index(i) %60
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=static_stag_branch_index(i)-1:-
1:static_riser_branch_index(i) %59:38
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=static_stag_branch_index(i-1)+1 %16
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=static_stag_branch_index(i-
1)+2:static_riser_branch_index(i)-1 %17:37
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_b_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_h_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine least and greatest branch velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_vel_b = min(V_SI_b_seg)
max_vel_b = max(V_SI_b_seg)
min_vel_h = min(V_SI_h_seg)
V_SI_h_seg(181) = 0;
```

```
max_vel_h = max(V_SI_h_seg)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine total head loss across pump
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
headloss_seg = zeros(1,static_num_chillers);
for i=1:static_num_chillers
    headloss_seg(i) =
((K_A_2_oa(i)/144/3.28084^4*(mfr_total_seg(3,i)*lbm_per_kg).^2/(2*rho_w*g_fps
2))+30)*2.3069;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select pump based off of pump head and mass flow rate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ph_op = max(headloss_seg);
for i=1:static_num_chillers
    if ph_op==headloss_seg(i)
        mfr_op = mfr_total_seg(3,i);
    end
end
mfr_total_seg
ratio_cw_total_mfr = zeros(1,static_num_chillers);
for i=1:static_num_chillers
    ratio_cw_total_mfr(i) = mfr_total_seg(3,i)/mfr_total_seg(1,i);
end

pump_curve = pump_curves(Pump_Head,Pump_Mfr,ph_op/3.28084,mfr_op);
mfr_total_seg(3,:) = polyval(pump_curve(2,:),headloss_seg/3.28084); %revised
mass flow rates based off of pump curve

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If stagnation points stay the same scale up/down mass flow rates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:static_num_chillers
    mfr_total_seg(1,i) = ratio_cw_total_mfr(i)*mfr_total_seg(3,i);
    mfr_total_seg(2,i) = mfr_total_seg(3,i)-mfr_total_seg(2,i);
end

%This block of code needs refinement. The mass flow rates jump too wildly
%based off of the pump curves. Need to determine pressure distribution
%based on refined mass flow rates and check pressure on either side of the
%guessed stagnation point. If one side is dominating, readjust stagnation
%point to allow for pressures to equal. This should be done iteratively
%with the entire process repeated several times to solve floating boundary
%condition (i.e. stagnation point). Not enough time to complete. Recommend
%for future work. Could also consider flow entering from both directions
%into a single branch and calculate loss coefficients from each
%contribution (cw flow and ccw flow into branch).
```

analysis2.m

```
%% Step 13 part a: Transient analysis - user input

load analysis
fprintf('The transient analysis module provides a means to determine the
temperature\n')
fprintf('at either a single location over a timespan specified by the user,
or of the\n')
fprintf('entire system at a time specified by the user.\n')
fprintf('The program requires the status of the system before and after the
transient\n')
fprintf('which is specified by the user through the use of an excel
spreadsheet.\n\n')

fprintf('The possible load conditions are: \n')
Condition_Labels
fprintf('Of the above load conditions, which do you want to analyze when
performing\n')
fprintf('the transient analysis?\n')
load_condition = menu('Select the load
condition','Shore','Design','Cruise','Battle');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input File
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'Transient.xlsx';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clear input file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear_vars = NaN(1000,12);
xlswrite(filename,clear_vars,1,'B12');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Order Load_Name, Q, Load_Value_kW
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Load_Name_Ordered = Load_Name;
Q_Ordered = zeros(size(Q));
Load_Value_kW_Ordered = zeros(size(Load_Value_kW));
for i=1:inputs
    Load_Name_Ordered(i) = Load_Name(branch_order(1,1,i));
    Q_Ordered(i) = Q(branch_order(1,1,i));
    Load_Value_kW_Ordered(i,:) = Load_Value_kW(branch_order(1,1,i),:);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set initial conditions in Excel Sheet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_number = [1:inputs]';
xlswrite(filename,load_number,1,'B12');
xlswrite(filename,Load_Name_Ordered,1,'C12');
xlswrite(filename,Q_Ordered/1000,1,'D12');
if load_condition == 1
```

```
    xlswrite(filename,Load_Value_kW_Ordered(:,1),1,'E12');
elseif load_condition == 2
    xlswrite(filename,Load_Value_kW_Ordered(:,2),1,'E12');
elseif load_condition == 3
   xlswrite(filename,Load_Value_kW_Ordered(:,3),1,'E12');
elseif load_condition == 4
   xlswrite(filename,Load_Value_kW_Ordered(:,4),1,'E12');
else
    fprintf('Error selecting load condition\n')
end

chiller_number = [1:num_chillers]';
xlswrite(filename,chiller_number,1,'H12');
xlswrite(filename,chiller_loc,1,'I12');

fprintf('Please open up the Excel file Transient.xlsx and provide the heat
load values before\n')
fprintf('and after the transient and the chiller configuration before and
after the transient\n')
fprintf('before proceeding through the analysis module.\n')

%% Step 13 part b: Transient analysis - initial pressures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in values from Excel Sheet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[num,txt] = xlsread(filename,'Transient');
transient_Q_init = num(1:inputs,4);
transient_Q_final = num(1:inputs,5);
transient_chiller_status_init = txt(11:num_chillers+10,11);
transient_chiller_status_final = txt(11:num_chillers+10,12);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine number of chillers in operation before and after transient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_num_chillers_init = 0;
transient_num_chillers_final = 0;
for i=1:num_chillers
    if strcmp(transient_chiller_status_init(i),'on')
        transient_num_chillers_init = transient_num_chillers_init+1;
    end
    if strcmp(transient_chiller_status_final(i),'on')
        transient_num_chillers_final = transient_num_chillers_final+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocate variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
size_Pressure_SI = size(Pressure_SI);
size_dPdX_header_loc_s_index = size(dPdX_header_loc_s_index);
transient_min_difference_pressure =
1000000000000*ones(1,transient_num_chillers_init);
transient_min_pressure = zeros(1,transient_num_chillers_init);
```

```
transient_min_location = zeros(1,transient_num_chillers_init);
transient_index_diff = zeros(1,transient_num_chillers_init);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine Pressure as a function of length along header for initial
% chiller configuration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_Pressure_SI_sum = zeros(1,size_Pressure_SI(3));
pressure_riser_index = 1;
riser_pressure = 0;
riser_location = 0;
for j=1:size_dPdX_header_loc_s_index(2)
    if strcmp(transient_chiller_status_init(j),'on')
        for k=1:size_Pressure_SI(3)
            if k>=dPdX_header_loc_s_index(j)
                transient_Pressure_SI_sum(k) =
transient_Pressure_SI_sum(k)+...
                    Pressure_SI(j,1,k-dPdX_header_loc_s_index(j)+1)+...
                    Pressure_SI(j,2,size_Pressure_SI(3)-(k-
dPdX_header_loc_s_index(j)));
            else
                transient_Pressure_SI_sum(k) = transient_Pressure_SI_sum(k) +
...
                    Pressure_SI(j,1,(size_Pressure_SI(3)+k-
dPdX_header_loc_s_index(j)+1))+...
                    Pressure_SI(j,2,size_Pressure_SI(3)-
(size_Pressure_SI(3)+k-dPdX_header_loc_s_index(j)));
            end
        end
    end
end
transient_Pressure_SI_sum =
transient_Pressure_SI_sum/transient_num_chillers_init;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the pressure and location of risers for chillers operational
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:size_dPdX_header_loc_s_index(2)
    if strcmp(transient_chiller_status_init(j),'on')
        for k=1:size_Pressure_SI(3)
            if k==dPdX_header_loc_s_index(j)
                riser_pressure(pressure_riser_index) =
transient_Pressure_SI_sum(k);
                riser_location(pressure_riser_index) = k;
                pressure_riser_index = pressure_riser_index+1;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert transient_riser_branch_index from riser_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_riser_count_index = 1;
```

```
transient_riser_branch_index = 0;
for i=1:size_header(1)
    if strcmp(transient_chiller_status_init(i),'on')
        transient_riser_branch_index(transient_riser_count_index) =
riser_branch_index(i);
        transient_riser_count_index = transient_riser_count_index+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set stag_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
size_riser_pressure = size(riser_pressure);
index_min_pressure_temp = 10000000000000000*ones(1,size_riser_pressure(2)+1);
index_min_loc_temp = ones(1,size_riser_pressure(2)+1);
index_riser_location = 1;
riser_location_temp=riser_location;
riser_location_temp(size_riser_pressure(2)+1)=size_Pressure_SI(3);
for i=1:size_Pressure_SI(3)
    if i < riser_location_temp(index_riser_location)
        if index_min_pressure_temp(index_riser_location) >
transient_Pressure_SI_sum(i)
            index_min_pressure_temp(index_riser_location) =
transient_Pressure_SI_sum(i);
            index_min_loc_temp(index_riser_location) = i;
        end
    else
        index_riser_location = index_riser_location+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine transient_riser_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
index_min_loc = ones(1,size_riser_pressure(2));
index_min_pressure = ones(1,size_riser_pressure(2));
if
index_min_pressure_temp(1)<index_min_pressure_temp(max(size(index_min_pressur
e_temp)))
    for i=1:size_riser_pressure(2)
        index_min_pressure(i)=index_min_pressure_temp(i);
        index_min_loc(i)=index_min_loc_temp(i);
    end
else
    for i=1:size_riser_pressure(2)
        index_min_pressure(i)=index_min_pressure_temp(i+1);
        index_min_loc(i)=index_min_loc_temp(i+1);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot pressure as a function of distanche along header with riser
% locations corresponding to operational chillers highlighted in red and
% stagnation points highlighted in green
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plot(transient_Pressure_SI_sum)
hold on
scatter(riser_location,riser_pressure,'r')
scatter(index_min_loc,index_min_pressure,'g')
xlabel('Index')
ylabel('Pressure')
title('Pressure Distribution')
legend('Pressure Distribution','Riser Location','Stagnation Point')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert index_min_loc to transient_stag_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
transient_stag_count_index = 1;
for i=1:size_Pressure_SI(3)
    if dPdX(1,1,i) == 2 %branch
        count=count+1;
        if transient_stag_count_index <= max(size(index_min_loc))
            if i>=index_min_loc(transient_stag_count_index)

transient_stag_branch_index(transient_stag_count_index)=count;
                transient_stag_count_index=transient_stag_count_index+1;
            end
        end
    end
end
for i=1:max(size(transient_stag_branch_index))
    if transient_stag_branch_index(i)==inputs
        transient_stag_branch_index(i)=inputs-1;
    end
end

%% Step 13 part c: Transient analysis - initial velocities and static
temperatures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
velocity_delta_seg = 10*ones(1,transient_num_chillers_init);
velocity_old_seg = zeros(1,transient_num_chillers_init);
V_SI_h_seg = 1.5*ones(1,inputs+1); %initial guess at header velocities
f_b_seg = zeros(1,inputs);
K_loss_b_seg = zeros(1,inputs);
K_loss_friction_b_seg = zeros(1,inputs);
K_loss_bend_90_b_seg = zeros(1,inputs);
K_loss_gate_b_seg = zeros(1,inputs);
K_loss_globe_b_seg = zeros(1,inputs);
r_d_seg = 3*ones(1,inputs+1); %assume r/d=3
K_loss_h_seg = zeros(1,inputs+1);
K_loss_friction_h_seg = zeros(1,inputs+1);
K_loss_bend_90_h_seg = zeros(1,inputs+1);
K_loss_gate_h_seg = zeros(1,inputs+1);
K_loss_globe_h_seg = zeros(1,inputs+1);
```

```
K_loss_check_h_seg = zeros(1,inputs+1);
f_h_seg = zeros(1,inputs+1);
K_loss_rh_seg = zeros(1,inputs+1);
K_loss_friction_rh = zeros(transient_num_chillers_init,2,inputs);
K_loss_bend_90_rh = zeros(transient_num_chillers_init,2,inputs);
K_loss_gate_rh = zeros(transient_num_chillers_init,2,inputs);
K_loss_globe_rh = zeros(transient_num_chillers_init,2,inputs);
K_h_A_h_2_seg = zeros(1,inputs+1);
K_b_A_b_2_seg = zeros(1,inputs);
K_A_eq_seg = zeros(transient_num_chillers_init,3,inputs);
mfr_h = zeros(transient_num_chillers_init,2,inputs);
mfr_b = zeros(transient_num_chillers_init,2,inputs);
V_b = zeros(transient_num_chillers_init,2,inputs);
V_h = zeros(transient_num_chillers_init,2,inputs);
mfr_total_seg = zeros(3,transient_num_chillers_init);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate total mfr's for each segment going cw and ccw
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:transient_num_chillers_init
    if i==1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=1:transient_stag_branch_index(1)%j=1:(stag_branch_index(1)-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 1-15
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(1))))/2; %half of branch
15

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=(transient_stag_branch_index(max(size(transient_stag_branch_index)))+1):inp
uts
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 164:180
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

    elseif 1<i && i<transient_num_chillers_init
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_riser_branch_index(i):transient_stag_branch_index(i)%j=riser_bran
ch_index(i):stag_branch_index(i)-1
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 38:60
```

```
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i))))/2; %half of branch
60

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 16:37
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i-1))))/2; %half of
branch 15

    elseif i==transient_num_chillers_init
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_riser_branch_index(max(size(transient_riser_branch_index))):trans
ient_stag_branch_index(max(size(transient_stag_branch_index)))
%j=riser_branch_index(max(size(riser_branch_index))):(stag_branch_index(max(s
ize(stag_branch_index)))-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 154:163
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_stag_branch_index(max(size(transient_stag_branch_index))-
1)+1:transient_riser_branch_index(max(size(transient_riser_branch_index)))-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 148:153
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))-1))))/2; %half of branch 147
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sum up mass flow rate going cw and ccw to give mass flow rate exiting
% each riser
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:transient_num_chillers_init
    mfr_total_seg(3,i) = mfr_total_seg(1,i)+mfr_total_seg(2,i);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resize and re-order V_SI_b and store in V_SI_b_seg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_seg = zeros(1,inputs);
for m=1:inputs
    V_SI_b_seg(m) = V_SI_b_1(branch_order(1,1,m));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterate through loop a predetermined number of times, modifying the
% branch diameters to satisfy the velocity limits set forth by NAVSEA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
while count<10
    count=count+1;

    if count == 1 %use estimated V_SI_b_seg to begin iterative process and
only consider friction bends and valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_b_seg due to friction, bends, valves for branches
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
            K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2;
%due to gate valves
            K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5;
%due to globe valves
            K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_h_seg due to friction, bends, valves for supply
header
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,1,i)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
```

```
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,1,i)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,1,i)*0.2;
        %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %        K_loss_globe_h(i)+K_loss_check_h(i);
        end
        for i=inputs+1

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
        %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %        K_loss_globe_h(i)+K_loss_check_h(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_rh_seg due to friction, bends, valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs+1
        %    K_loss_friction_rh(i)=f_b(1)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %    K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_b(1)*pi()/2*r_d(i)+(0.10+2.4*f_b(1))*sin(pi()/4) ...
        %
+6.6*f_b(1)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %    K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %    K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
        %    K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_gate_rh(i)+K_loss_globe_rh(
i);
            K_loss_rh_seg(i) = K_loss_h_seg(i); %assume same loss coefficient
for supply and return header segments
        end
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_b/A_b^2 and K_h/A_h^2 for branches and header segments
        % respectively
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
        end
        for i=1:inputs+1
            K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2 = zeros(1,inputs);
        for i=1:transient_num_chillers_init
            if i==1
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i) %15
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %1:14
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            else
                for j=transient_stag_branch_index(i-1)+1 %16
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i) %60
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            end
```

```
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2_oa = zeros(1,transient_num_chillers_init);
        for i=1:transient_num_chillers_init
            if i==1
                K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
            else
                K_A_2_oa(i) = (1/(1/K_A_2(transient_riser_branch_index(i)-
1)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_oa = zeros(2,transient_num_chillers_init); %cw=1, ccw=2
        for i=1:transient_num_chillers_init
            if i==1
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
            else
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)-
1))^0.5;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_temp
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_b = zeros(1,inputs);
        mfr_seg_temp = zeros(1,inputs);
        for i=1:transient_num_chillers_init
            if i==1
                for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %1:15
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1:input
s %164:180
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
                end
            else
```

```
                    for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %38:60
                        mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
                    end
                    for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1 %16:37
                        mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(transient_riser_branch_index(i)-1)/K_A_2(j))^0.5;
                    end
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate mfr_seg_b
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for i=1:transient_num_chillers_init
                if i==1
                    for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                        mfr_seg_b(j) = mfr_seg_temp(j);
                    end
                    for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                        mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                    end
                    for j=transient_stag_branch_index(i) %15
                        mfr_seg_b(j) = mfr_seg_temp(j);
                    end
                    for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14:1
                        mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                    end
                else
                    for j=transient_stag_branch_index(i-1)+1 %16
                        mfr_seg_b(j) = mfr_seg_temp(j);
                    end
                    for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                        mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                    end
                    for j=transient_stag_branch_index(i) %60
                        mfr_seg_b(j) = mfr_seg_temp(j);
                    end
                    for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59;38
                        mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                    end
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Calculate mfr_seg_h
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        mfr_seg_h = zeros(1,inputs);
        for i=1:transient_num_chillers_init
            if i==1
                for j=transient_stag_branch_index(i) %15
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14;1
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                end
            else
                for j=transient_stag_branch_index(i) %60
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
                end
                for j=transient_stag_branch_index(i-1)+1 %16
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate V_SI_b_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate V_SI_h_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Calculate loss coefficient for branches due to friction, bends,
    % valves, entrance and exit effects (in order wrt header)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_b_seg = zeros(1,inputs);
    K_loss_exit_b_seg = zeros(1,inputs);
    for i=1:inputs
        f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
        K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2; %due to
gate valves
        K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5; %due
to globe valves
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));

        Cyc(i) = 1-0.25*(D_SI_b(branch_order(1,1,i))/D_SI_h)^1.3-(0.11*r_d3-
0.65*r_d3^2+0.83*r_d3^3)*D_SI_b(branch_order(1,1,i))^2/D_SI_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance and exit effects for branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Keq = 0.57-1.07*r_d3^0.5-2.13*r_d3+8.24*r_d3^1.5-
8.48*r_d3^2+2.9*r_d3^2.5;
    Cxc = 0.08+0.56*r_d3-1.75*r_d3^2+1.83*r_d3^3;
    Cm = 0.23+1.46*r_d3-2.75*r_d3^2+1.65*r_d3^3;
    for j=1:transient_num_chillers_init
        if j==1
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j) %cw 1:15
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
```

```
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
180:164
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        else
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j) %cw 38:60
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+1 %ccw 37:16
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_loss_b_seg and K_loss_b_in_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_b_in_seg = zeros(1,inputs);
    for i=1:inputs
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...
```

```
K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+K_loss_exit_b_seg(i);
        K_loss_b_in_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+0*K_loss_exit_b_seg(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_b_seg(i) <= 0
            K_loss_b_seg(i) = 0.01; %negligible loss coefficient
        end
        if K_loss_b_in_seg(i) <= 0
            K_loss_b_in_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate loss coefficient for supply header due to friction, bends,
    % valves, entrance and exit effects
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_h_seg = zeros(1,inputs);
    for i=1:inputs
        f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
        K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
        K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
    %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
    %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves considered
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:transient_num_chillers_init
        if j==1
            for i=transient_stag_branch_index(j) %cw 15
                K_loss_entrance_h_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 1:14
```

```
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for
i=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
164
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        else
            for i=transient_stag_branch_index(j) %cw 60
                K_loss_entrance_h_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for i=transient_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        end
    end
    for i=1:inputs
        K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i)+K_loss_
entrance_h_seg(i);%+ ...
        %    K_loss_globe_h(i)+K_loss_check_h(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_h_seg(i) <= 0
            K_loss_h_seg(i) = 0.01; %negligible loss coefficient
        end
```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate K_loss_rh due to friction, bends, valves
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_rh_seg = zeros(1,inputs);
    for i=1:inputs
        %K_loss_friction_rh(i)=f_h(i)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_h(i)*pi()/2*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/4) ...
        %
+6.6*f_h(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %K_loss_bend_180_rh(i) =
bends_180_rh(i)*(f_h(i)*pi()*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/2) ...
        %
+6.6*f_h(i)*((sin(pi()/2))^0.5+sin(pi()/2))/r_d(i)^(4*pi()/pi())); %due to
180 bends
        %K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:transient_num_chillers_init
        if j==1
            for i=transient_stag_branch_index(j) %cw 15
                K_loss_entrance_rh_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 1:14
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for
i=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
164
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        else
            for i=transient_stag_branch_index(j) %cw 60
                K_loss_entrance_rh_seg(i) = 0;
```

```
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for i=transient_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        end
    end
    for i=1:inputs
        K_loss_rh_seg(i) = K_loss_h_seg(i)-
K_loss_entrance_h_seg(i)+K_loss_entrance_rh_seg(i);
        %K_loss_rh_seg(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_bend_180_rh(i)+K_loss_gate_
rh(i)+ ...
        %    K_loss_globe_rh(i)+K_loss_entrance_rh(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_rh_seg(i) <= 0
            K_loss_rh_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
    end
    for i=1:inputs+1
        K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2 = zeros(1,inputs);
```

```
    for i=1:transient_num_chillers_init
        if i==1
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i) %15
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %1:14
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        else
            for j=transient_stag_branch_index(i-1)+1 %16
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i) %60
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2_oa = zeros(1,transient_num_chillers_init);
    for i=1:transient_num_chillers_init
        if i==1
            K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
        else
            K_A_2_oa(i) = (1/(1/K_A_2(transient_riser_branch_index(i)-
1)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Calculate mfr_seg_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_oa = zeros(2,transient_num_chillers_init); %cw=1, ccw=2
    for i=1:transient_num_chillers_init
        if i==1
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
        else
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)-
1))^0.5;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_temp
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_b = zeros(1,inputs);
    mfr_seg_temp = zeros(1,inputs);
    for i=1:transient_num_chillers_init
        if i==1
            for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %1:15
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1:input
s %164:180
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
            end
        else
            for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %38:60
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1 %16:37
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(transient_riser_branch_index(i)-1)/K_A_2(j))^0.5;
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:transient_num_chillers_init
        if i==1
```

```
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
            end
            for j=transient_stag_branch_index(i) %15
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14:1
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        else
            for j=transient_stag_branch_index(i-1)+1 %16
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
            end
            for j=transient_stag_branch_index(i) %60
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59;38
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_h
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_h = zeros(1,inputs);
    for i=1:transient_num_chillers_init
        if i==1
            for j=transient_stag_branch_index(i) %15
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14;1
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
```

```
            end
        else
            for j=transient_stag_branch_index(i) %60
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=transient_stag_branch_index(i-1)+1 %16
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_b_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_h_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rename variable
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_h_seg_init = V_SI_h_seg;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine least and greatest branch velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_vel_b = min(V_SI_b_seg)
max_vel_b = max(V_SI_b_seg)
min_vel_h = min(V_SI_h_seg)
V_SI_h_seg(181) = 0;
max_vel_h = max(V_SI_h_seg)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine initial temperatures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Tcold_delta = zeros(1,inputs);
Tcold_delta_cum = zeros(1,inputs);
```

```
Tcold_delta_b = zeros(1,inputs);
Tcold = (44-32)*5/9;
g_mps2 = 9.81*ft_per_m;

for i=1:inputs
    H_l_h(i) = K_loss_h_seg(i)*V_SI_h_seg(i)^2/2/g_mps2;
    Tcold_delta(i) = (H_l_h(i)/778.169/1.0025)*10/18;
    for j=i:inputs
        Tcold_delta_cum(j) = Tcold_delta_cum(j)+Tcold_delta(i);
    end
end

Thot_delta_b = zeros(1,inputs);
for i=1:inputs
    H_l_b_in(i) = K_loss_b_in_seg(i)*V_SI_b_seg(i)^2/2/g_mps2;
    H_l_b(i) = K_loss_b_seg(i)*V_SI_b_seg(i)^2/2/g_mps2;
    Tcold_delta_b(i) = H_l_b_in(i)/778.169262/1.0025*10/18;
    Thot_delta_b(i) = H_l_b(i)/778.169262/1.0025*10/18;
end

Tcold_h = zeros(1,inputs);
Tcold_b = zeros(1,inputs);
Thot_h = zeros(1,inputs);
Thot_b = zeros(1,inputs);
for i=1:(inputs)
    Tcold_h(i) = Tcold + Tcold_delta_cum(i);
    Tcold_b(i) = Tcold_h(i) + Tcold_delta_b(i);
    Thot_b(i) = Tcold_h(i) + Thot_delta_b(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate temperatures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:inputs
    hc_b_seg(i) = calc_hc(D_SI_b_ordered(1,1,i),V_SI_b_seg(i),k,nu,rho,cp);
    Thot_b_seg(i) = Q_ordered(1,1,i)/(mfr_seg_b(i)*cp)+Tcold_b(i); %Celsius
    Tave_b_seg(i) = (Tcold_b(i)+Thot_b_seg(i))/2;
    T1_b_seg(i) = Tave_b_seg(i) +
Q_ordered(1,1,i)*(hxchgr_area_pri(order)*0.0001*hc_b_ordered(1,1,i))^-1;
%Inner wall temp
    Q_per_l_seg(i) =
Q_ordered(1,1,i)*D_SI_b_ordered(1,1,i)*pi()/(hxchgr_area_pri(order)*0.0001);
    T2_b_seg(i) = T1_b_seg(i) +
Q_per_l_seg(i)*log((D_SI_b_ordered(1,1,i)/2+thickness_b(branch_order(1,1,i)))
/((D_SI_b_ordered(1,1,i))/2))/(2*pi()*kcopper); %Outer wall temp
    Telec_b_ave_seg(i) = (T2_b_seg(i) +
Q_ordered(1,1,i)/(hxchgr_area_sec(branch_order(1,1,i))*0.0001*hxchgr_hc(branc
h_order(1,1,i)))); %Electrical component temp
    delta_T_sec_seg(i) =
Q(branch_order(1,1,i))/hxchgr_fluid_mfr(branch_order(1,1,i))/hxchgr_cp(branch
_order(1,1,i));
    Telec_b_in_seg(i) = Telec_b_ave_seg(i)+delta_T_sec_seg(i)/2;
    Telec_b_seg(i) = Telec_b_ave_seg(i)-delta_T_sec_seg(i)/2;
end
```

```
fprintf('Fifth Step: Refined Inlet Temperatures\n')
for i=1:inputs
    fprintf('Load: %3.0f  Q(W): %10.4f  Diameter(m): %6.5f  Velocity(m/sec):
%6.4f  Mass flow rate(kg/s): %6.4f  Thot(C): %7.4f  Telec(C): %8.4f\n' ...
        ,i,Q(branch_order(1,i)), D_SI_b(branch_order(1,i)) ,V_SI_b_seg(i)
,mfr_seg_b(i), Thot_b_seg(i), Telec_b_seg(i))
end
max(Thot_b_seg)
max(Telec_b_seg)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rename variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_stag_branch_index_init = transient_stag_branch_index;
transient_riser_branch_index_init = transient_riser_branch_index;

%% Step 13 part d: Transient analysis - final pressures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocate variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_min_difference_pressure =
1000000000000*ones(1,transient_num_chillers_final);
transient_min_pressure = zeros(1,transient_num_chillers_final);
transient_min_location = zeros(1,transient_num_chillers_final);
transient_index_diff = zeros(1,transient_num_chillers_final);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine Pressure as a function of length along header for initial
% chiller configuration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_Pressure_SI_sum = zeros(1,size_Pressure_SI(3));
pressure_riser_index = 1;
riser_pressure = 0;
riser_location = 0;
for j=1:size_dPdX_header_loc_s_index(2)
    if strcmp(transient_chiller_status_final(j),'on')
        for k=1:size_Pressure_SI(3)
            if k>=dPdX_header_loc_s_index(j)
                transient_Pressure_SI_sum(k) =
transient_Pressure_SI_sum(k)+...
                    Pressure_SI(j,1,k-dPdX_header_loc_s_index(j)+1)+...
                    Pressure_SI(j,2,size_Pressure_SI(3)-(k-
dPdX_header_loc_s_index(j)));
            else
                transient_Pressure_SI_sum(k) = transient_Pressure_SI_sum(k) +
...
                    Pressure_SI(j,1,(size_Pressure_SI(3)+k-
dPdX_header_loc_s_index(j)+1))+...
                    Pressure_SI(j,2,size_Pressure_SI(3)-
(size_Pressure_SI(3)+k-dPdX_header_loc_s_index(j)));
            end
        end
    end
```

```
end
transient_Pressure_SI_sum =
transient_Pressure_SI_sum/transient_num_chillers_final;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the pressure and location of risers for chillers operational
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:size_dPdX_header_loc_s_index(2)
    if strcmp(transient_chiller_status_final(j),'on')
        for k=1:size_Pressure_SI(3)
            if k==dPdX_header_loc_s_index(j)
                riser_pressure(pressure_riser_index) =
transient_Pressure_SI_sum(k);
                riser_location(pressure_riser_index) = k;
                pressure_riser_index = pressure_riser_index+1;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert transient_riser_branch_index from riser_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
transient_riser_count_index = 1;
transient_riser_branch_index = 0;
for i=1:size_header(1)
    if strcmp(transient_chiller_status_final(i),'on')
        transient_riser_branch_index(transient_riser_count_index) =
riser_branch_index(i);
        transient_riser_count_index = transient_riser_count_index+1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set stag_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
size_riser_pressure = size(riser_pressure);
index_min_pressure_temp = 10000000000000000*ones(1,size_riser_pressure(2)+1);
index_min_loc_temp = ones(1,size_riser_pressure(2)+1);
index_riser_location = 1;
riser_location_temp=riser_location;
riser_location_temp(size_riser_pressure(2)+1)=size_Pressure_SI(3);
for i=1:size_Pressure_SI(3)
    if i < riser_location_temp(index_riser_location)
        if index_min_pressure_temp(index_riser_location) >
transient_Pressure_SI_sum(i)
            index_min_pressure_temp(index_riser_location) =
transient_Pressure_SI_sum(i);
            index_min_loc_temp(index_riser_location) = i;
        end
    else
        index_riser_location = index_riser_location+1;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine transient_riser_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
index_min_loc = ones(1,size_riser_pressure(2));
index_min_pressure = ones(1,size_riser_pressure(2));
if
index_min_pressure_temp(1)<index_min_pressure_temp(max(size(index_min_pressur
e_temp)))
    for i=1:size_riser_pressure(2)
        index_min_pressure(i)=index_min_pressure_temp(i);
        index_min_loc(i)=index_min_loc_temp(i);
    end
else
    for i=1:size_riser_pressure(2)
        index_min_pressure(i)=index_min_pressure_temp(i+1);
        index_min_loc(i)=index_min_loc_temp(i+1);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot pressure as a function of distanche along header with riser
% locations corresponding to operational chillers highlighted in red and
% stagnation points highlighted in green
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plot(transient_Pressure_SI_sum)
hold on
scatter(riser_location,riser_pressure,'r')
scatter(index_min_loc,index_min_pressure,'g')
xlabel('Index')
ylabel('Pressure')
title('Pressure Distribution')
legend('Pressure Distribution','Riser Location','Stagnation Point')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convert index_min_loc to transient_stag_branch_index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
transient_stag_count_index = 1;
transient_stag_branch_index = 0;
for i=1:size_Pressure_SI(3)
    if dPdX(1,1,i) == 2 %branch
        count=count+1;
        if transient_stag_count_index <= max(size(index_min_loc))
            if i>=index_min_loc(transient_stag_count_index)

transient_stag_branch_index(transient_stag_count_index)=count;
                transient_stag_count_index=transient_stag_count_index+1;
            end
        end
    end
end
for i=1:max(size(transient_stag_branch_index))
    if transient_stag_branch_index(i)==inputs
```

```
        transient_stag_branch_index(i)=inputs-1;
    end
end
%% Step 13 part e: Transient analysis - final velocities

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
velocity_delta_seg = 10*ones(1,transient_num_chillers_final);
velocity_old_seg = zeros(1,transient_num_chillers_final);
V_SI_h_seg = 1.5*ones(1,inputs+1); %initial guess at header velocities
f_b_seg = zeros(1,inputs);
K_loss_b_seg = zeros(1,inputs);
K_loss_friction_b_seg = zeros(1,inputs);
K_loss_bend_90_b_seg = zeros(1,inputs);
K_loss_gate_b_seg = zeros(1,inputs);
K_loss_globe_b_seg = zeros(1,inputs);
r_d_seg = 3*ones(1,inputs+1); %assume r/d=3
K_loss_h_seg = zeros(1,inputs+1);
K_loss_friction_h_seg = zeros(1,inputs+1);
K_loss_bend_90_h_seg = zeros(1,inputs+1);
K_loss_gate_h_seg = zeros(1,inputs+1);
K_loss_globe_h_seg = zeros(1,inputs+1);
K_loss_check_h_seg = zeros(1,inputs+1);
f_h_seg = zeros(1,inputs+1);
K_loss_rh_seg = zeros(1,inputs+1);
K_loss_friction_rh = zeros(transient_num_chillers_final,2,inputs);
K_loss_bend_90_rh = zeros(transient_num_chillers_final,2,inputs);
K_loss_gate_rh = zeros(transient_num_chillers_final,2,inputs);
K_loss_globe_rh = zeros(transient_num_chillers_final,2,inputs);
K_h_A_h_2_seg = zeros(1,inputs+1);
K_b_A_b_2_seg = zeros(1,inputs);
K_A_eq_seg = zeros(transient_num_chillers_final,3,inputs);
mfr_h = zeros(transient_num_chillers_final,2,inputs);
mfr_b = zeros(transient_num_chillers_final,2,inputs);
V_b = zeros(transient_num_chillers_final,2,inputs);
V_h = zeros(transient_num_chillers_final,2,inputs);
mfr_total_seg = zeros(3,transient_num_chillers_final);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate total mfr's for each segment going cw and ccw
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:transient_num_chillers_final
    if i==1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=1:transient_stag_branch_index(1)%j=1:(stag_branch_index(1)-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 1-15
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(1))))/2; %half of branch
15
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=(transient_stag_branch_index(max(size(transient_stag_branch_index)))+1):inp
uts
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); % branches 164:180
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

    elseif 1<i && i<transient_num_chillers_final
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_riser_branch_index(i):transient_stag_branch_index(i)%j=riser_bran
ch_index(i):stag_branch_index(i)-1
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 38:60
        end
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i))))/2; %half of branch
60

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 16:37
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(i-1))))/2; %half of
branch 15

    elseif i==transient_num_chillers_final
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg cw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_riser_branch_index(max(size(transient_riser_branch_index))):trans
ient_stag_branch_index(max(size(transient_stag_branch_index)))
%j=riser_branch_index(max(size(riser_branch_index))):(stag_branch_index(max(s
ize(stag_branch_index)))-1)
            mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 154:163
        end
```

```
        %mfr_total_seg(1,i) = mfr_total_seg(1,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))))))/2; %half of branch 163

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_total_seg ccw
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for
j=transient_stag_branch_index(max(size(transient_stag_branch_index))-
1)+1:transient_riser_branch_index(max(size(transient_riser_branch_index)))-1
            mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,j)); %branches 148:153
        end
        %mfr_total_seg(2,i) = mfr_total_seg(2,i) +
mass_flow_rate_b(branch_order(1,1,(stag_branch_index(max(size(stag_branch_ind
ex))-1))))/2; %half of branch 147
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sum up mass flow rate going cw and ccw to give mass flow rate exiting
% each riser
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:transient_num_chillers_final
    mfr_total_seg(3,i) = mfr_total_seg(1,i)+mfr_total_seg(2,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resize and re-order V_SI_b and store in V_SI_b_seg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
V_SI_b_seg = zeros(1,inputs);
for m=1:inputs
    V_SI_b_seg(m) = V_SI_b_1(branch_order(1,1,m));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterate through loop a predetermined number of times, modifying the
% branch diameters to satisfy the velocity limits set forth by NAVSEA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
count = 0;
while count<10
    count=count+1;

    if count == 1 %use estimated V_SI_b_seg to begin iterative process and
only consider friction bends and valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_b_seg due to friction, bends, valves for branches
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered
```

```
K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
            K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2;
%due to gate valves
            K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5;
%due to globe valves
            K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_h_seg due to friction, bends, valves for supply
header
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,1,i)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,1,i)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,1,i)*0.2;
        %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %        K_loss_globe_h(i)+K_loss_check_h(i);
        end
        for i=inputs+1

f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
            K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due
to pipe length based on first branch Darcy friction factor
            K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
            K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
```

```
        %     K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
        %     K_loss_check_h(i) = check_valve_h(i)*2; %no check valves
considered
            K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i);%+ ...
        %         K_loss_globe_h(i)+K_loss_check_h(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_rh_seg due to friction, bends, valves
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs+1
        %     K_loss_friction_rh(i)=f_b(1)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %     K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_b(1)*pi()/2*r_d(i)+(0.10+2.4*f_b(1))*sin(pi()/4) ...
        %
+6.6*f_b(1)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %     K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %     K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
        %     K_loss_rh(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_gate_rh(i)+K_loss_globe_rh(
i);
            K_loss_rh_seg(i) = K_loss_h_seg(i); %assume same loss coefficient
for supply and return header segments
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_b/A_b^2 and K_h/A_h^2 for branches and header segments
        % respectively
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
        end
        for i=1:inputs+1
            K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2 = zeros(1,inputs);
        for i=1:transient_num_chillers_final
            if i==1
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
```

```
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i) %15
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %1:14
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            else
                for j=transient_stag_branch_index(i-1)+1 %16
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                    K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i) %60
                    K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                    K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_A_2_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_A_2_oa = zeros(1,transient_num_chillers_final);
        for i=1:transient_num_chillers_final
            if i==1
                K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
            else
                K_A_2_oa(i) = (1/(1/K_A_2(transient_riser_branch_index(i)-
1)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_oa
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_oa = zeros(2,transient_num_chillers_final); %cw=1, ccw=2
        for i=1:transient_num_chillers_final
            if i==1
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
```

```
            else
                mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
                mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)-
1))^0.5;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_temp
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_b = zeros(1,inputs);
        mfr_seg_temp = zeros(1,inputs);
        for i=1:transient_num_chillers_final
            if i==1
                for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %1:15
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1:input
s %164:180
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
                end
            else
                for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %38:60
                    mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
                end
                for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1 %16:37
                    mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(transient_riser_branch_index(i)-1)/K_A_2(j))^0.5;
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_b
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:transient_num_chillers_final
            if i==1
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
```

```
                for j=transient_stag_branch_index(i) %15
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14:1
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            else
                for j=transient_stag_branch_index(i-1)+1 %16
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
                end
                for j=transient_stag_branch_index(i) %60
                    mfr_seg_b(j) = mfr_seg_temp(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59;38
                    mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate mfr_seg_h
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mfr_seg_h = zeros(1,inputs);
        for i=1:transient_num_chillers_final
            if i==1
                for j=transient_stag_branch_index(i) %15
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14;1
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                end
            else
                for j=transient_stag_branch_index(i) %60
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
                end
```

```
                for j=transient_stag_branch_index(i-1)+1 %16
                    mfr_seg_h(j) = mfr_seg_b(j);
                end
                for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                    mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate V_SI_b_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate V_SI_h_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate loss coefficient for branches due to friction, bends,
    % valves, entrance and exit effects (in order wrt header)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_b_seg = zeros(1,inputs);
    K_loss_exit_b_seg = zeros(1,inputs);
    for i=1:inputs
        f_b_seg(i) =
friction_factor(D_SI_b(branch_order(1,1,i)),V_SI_b_seg(i),k,nu,epsilon,rho,cp
); %ordered

K_loss_friction_b_seg(i)=f_b_seg(i)*length_b(branch_order(1,1,i))/D_SI_b(bran
ch_order(1,1,i)); %due to pipe length
        K_loss_bend_90_b_seg(i) =
bends_90_b(1,branch_order(1,1,i))*(f_b_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_b
_seg(i))*sin(pi()/4) ...

+6.6*f_b_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_b_seg(i) = gate_valve_b(branch_order(1,1,i))*0.2; %due to
gate valves
        K_loss_globe_b_seg(i) = globe_valve_b(branch_order(1,1,i))*3.5; %due
to globe valves
        K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+K_loss_
globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i));
```

```
        Cyc(i) = 1-0.25*(D_SI_b(branch_order(1,1,i))/D_SI_h)^1.3-(0.11*r_d3-
0.65*r_d3^2+0.83*r_d3^3)*D_SI_b(branch_order(1,1,i))^2/D_SI_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance and exit effects for branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Keq = 0.57-1.07*r_d3^0.5-2.13*r_d3+8.24*r_d3^1.5-
8.48*r_d3^2+2.9*r_d3^2.5;
    Cxc = 0.08+0.56*r_d3-1.75*r_d3^2+1.83*r_d3^3;
    Cm = 0.23+1.46*r_d3-2.75*r_d3^2+1.65*r_d3^3;
    for j=1:transient_num_chillers_final
        if j==1
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j) %cw 1:15
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
180:164
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
            end
        else
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j) %cw 38:60
                K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                    +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
```

```
                    mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
                end
                for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+1 %ccw 37:16
                    K_loss_entrance_b_seg(i) = (0.81-
1.13*mfr_seg_h(i)/mfr_seg_b(i)+mfr_seg_h(i)^2/mfr_seg_b(i)^2)*D_SI_b(branch_o
rder(1,1,i))^4/D_SI_h^4 ...
                        +1.12*D_SI_b(branch_order(1,1,i))/D_SI_h-
1.08*D_SI_b(branch_order(1,1,i))^3/D_SI_h^3 + Keq;%due to entrance; assume
r/d3 = 0.1
                    K_loss_exit_b_seg(i) = 2*Cyc(i)-
1+D_SI_b(branch_order(1,1,i))^4/D_SI_h^4*(2*(Cxc-1)+2*(2-Cxc-
Cm)*mfr_seg_h(i)/mfr_seg_b(i)-0.92* ...
                        mfr_seg_h(i)^2/mfr_seg_b(i)^2);%due to exit; assume r/d3
= 0.1
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate K_loss_b_seg and K_loss_b_in_seg
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_loss_b_in_seg = zeros(1,inputs);
        for i=1:inputs
            K_loss_b_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+K_loss_exit_b_seg(i);
            K_loss_b_in_seg(i) =
K_loss_friction_b_seg(i)+K_loss_bend_90_b_seg(i)+K_loss_gate_b_seg(i)+ ...

K_loss_globe_b_seg(i)+K_loss_hx_b_unordered(branch_order(1,1,i))+K_loss_entra
nce_b_seg(i)+0*K_loss_exit_b_seg(i);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % To avoid getting imaginary velocities, ensure K_loss is positive
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:inputs
            if K_loss_b_seg(i) <= 0
                K_loss_b_seg(i) = 0.01; %negligible loss coefficient
            end
            if K_loss_b_in_seg(i) <= 0
                K_loss_b_in_seg(i) = 0.01; %negligible loss coefficient
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate loss coefficient for supply header due to friction, bends,
        % valves, entrance and exit effects
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        K_loss_entrance_h_seg = zeros(1,inputs);
```

```
    for i=1:inputs
        f_h_seg(i)=friction_factor(D_SI_h,V_SI_h_seg(i),k,nu,epsilon,rho,cp);
        K_loss_friction_h_seg(i)=f_h_seg(i)*length_h(1,2,1)/D_SI_h; %due to
pipe length based on first branch Darcy friction factor
        K_loss_bend_90_h_seg(i) =
bends_90_h(1,2,1)*(f_h_seg(i)*pi()/2*r_d_seg(i)+(0.10+2.4*f_h_seg(i))*sin(pi(
)/4) ...

+6.6*f_h_seg(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d_seg(i)^(4*pi()/2/pi()));
%due to 90 bends
        K_loss_gate_h_seg(i) = gate_valve_h(1,2,1)*0.2;
    %    K_loss_globe_h(i) = globe_valve_h(i)*3.5; %no globe valves
considered
    %    K_loss_check_h(i) = check_valve_h(i)*2; %no check valves considered
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:transient_num_chillers_final
        if j==1
            for i=transient_stag_branch_index(j) %cw 15
                K_loss_entrance_h_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 1:14
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
            for
i=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
164
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        else
            for i=transient_stag_branch_index(j) %cw 60
                K_loss_entrance_h_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+ ...
                    0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
```

```
            for i=transient_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_h_seg(i) = 0;
            end
            for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_h_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+ ...
                    0.03*(mfr_seg_h(i-1)/mfr_seg_h(i))^6; %revisit mfr_seg_h
indices
            end
        end
    end
    for i=1:inputs
        K_loss_h_seg(i) =
K_loss_friction_h_seg(i)+K_loss_bend_90_h_seg(i)+K_loss_gate_h_seg(i)+K_loss_
entrance_h_seg(i);%+ ...
        %    K_loss_globe_h(i)+K_loss_check_h(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_h_seg(i) <= 0
            K_loss_h_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate K_loss_rh due to friction, bends, valves
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_loss_entrance_rh_seg = zeros(1,inputs);
    for i=1:inputs
        %K_loss_friction_rh(i)=f_h(i)*length_rh(i)/D_SI_h; %due to pipe
length based on first branch Darcy friction factor
        %K_loss_bend_90_rh(i) =
bends_90_rh(i)*(f_h(i)*pi()/2*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/4) ...
        %
+6.6*f_h(i)*((sin(pi()/4))^0.5+sin(pi()/4))/r_d(i)^(4*pi()/2/pi())); %due to
90 bends
        %K_loss_bend_180_rh(i) =
bends_180_rh(i)*(f_h(i)*pi()*r_d(i)+(0.10+2.4*f_h(i))*sin(pi()/2) ...
        %
+6.6*f_h(i)*((sin(pi()/2))^0.5+sin(pi()/2))/r_d(i)^(4*pi()/pi())); %due to
180 bends
        %K_loss_gate_rh(i) = gate_valve_rh(i)*0.2;
        %K_loss_globe_rh(i) = globe_valve_rh(i)*3.5;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate exit effects for header segments
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate entrance effects for header segments
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:transient_num_chillers_final
        if j==1
            for i=transient_stag_branch_index(j) %cw 15
                K_loss_entrance_rh_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 1:14
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for
i=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %ccw
164
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=inputs:-
1:transient_stag_branch_index(max(size(transient_stag_branch_index)))+2 %ccw
180:165
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        else
            for i=transient_stag_branch_index(j) %cw 60
                K_loss_entrance_rh_seg(i) = 0;
            end
            for
i=transient_riser_branch_index(j):transient_stag_branch_index(j)-1 %cw 38:59
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i+1)+...

0.36*(mfr_seg_h(i)/mfr_seg_h(i+1))^2+0.03*(mfr_seg_h(i+1)/mfr_seg_h(i))^6;
%exit
            end
            for i=transient_stag_branch_index(j-1)+1 %ccw 16
                K_loss_entrance_rh_seg(i) = 0;
            end
            for i=transient_riser_branch_index(j)-1:-
1:transient_stag_branch_index(j-1)+2 %ccw 37:17
                K_loss_entrance_rh_seg(i) = 0.62-
0.98*mfr_seg_h(i)/mfr_seg_h(i-1)+...
                    0.36*(mfr_seg_h(i)/mfr_seg_h(i-1))^2+0.03*(mfr_seg_h(i-
1)/mfr_seg_h(i))^6; %exit
            end
        end
    end
    for i=1:inputs
        K_loss_rh_seg(i) = K_loss_h_seg(i)-
K_loss_entrance_h_seg(i)+K_loss_entrance_rh_seg(i);
```

```
        %K_loss_rh_seg(i) =
K_loss_friction_rh(i)+K_loss_bend_90_rh(i)+K_loss_bend_180_rh(i)+K_loss_gate_
rh(i)+ ...
        %    K_loss_globe_rh(i)+K_loss_entrance_rh(i);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % To avoid getting imaginary velocities, ensure K_loss is positive
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        if K_loss_rh_seg(i) <= 0
            K_loss_rh_seg(i) = 0.01; %negligible loss coefficient
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_b/A_b^2 and K_h/A_h^2 for branches
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        K_b_A_b_2_seg(i) =
K_loss_b_seg(i)/area_b_unordered(branch_order(1,1,i))^2;
    end
    for i=1:inputs+1
        K_h_A_h_2_seg(i) = (K_loss_h_seg(i)+K_loss_rh_seg(i))/area_h^2;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2 = zeros(1,inputs);
    for i=1:transient_num_chillers_final
        if i==1
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i) %15
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %1:14
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        else
            for j=transient_stag_branch_index(i-1)+1 %16
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
```

```
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                K_A_2(j) = (1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j-
1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i) %60
                K_A_2(j) = K_b_A_b_2_seg(j);% + K_h_A_h_2_seg(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                K_A_2(j) =
(1/(1/K_b_A_b_2_seg(j)^0.5+1/K_A_2(j+1)^0.5))^2;%+K_h_A_h_2_seg(j);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate K_A_2_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K_A_2_oa = zeros(1,transient_num_chillers_final);
    for i=1:transient_num_chillers_final
        if i==1
            K_A_2_oa(i) =
(1/(1/K_A_2(inputs)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
        else
            K_A_2_oa(i) = (1/(1/K_A_2(transient_riser_branch_index(i)-
1)^0.5+1/K_A_2(transient_riser_branch_index(i))^0.5))^2;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_oa
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_oa = zeros(2,transient_num_chillers_final); %cw=1, ccw=2
    for i=1:transient_num_chillers_final
        if i==1
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(inputs))^0.5;
        else
            mfr_seg_oa(1,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)))^0.5;
            mfr_seg_oa(2,i) =
mfr_total_seg(3,i)*(K_A_2_oa(i)/K_A_2(transient_riser_branch_index(i)-
1))^0.5;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_temp
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_b = zeros(1,inputs);
    mfr_seg_temp = zeros(1,inputs);
```

```
    for i=1:transient_num_chillers_final
        if i==1
            for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %1:15
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1:input
s %164:180
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(inputs)/K_A_2(j))^0.5;
            end
        else
            for
j=transient_riser_branch_index(i):transient_stag_branch_index(i) %38:60
                mfr_seg_temp(j) =
mfr_seg_oa(1,i)*(K_A_2(transient_riser_branch_index(i))/K_A_2(j))^0.5;
            end
            for j=transient_stag_branch_index(i-
1)+1:transient_riser_branch_index(i)-1 %16:37
                mfr_seg_temp(j) =
mfr_seg_oa(2,i)*(K_A_2(transient_riser_branch_index(i)-1)/K_A_2(j))^0.5;
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:transient_num_chillers_final
        if i==1
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
            end
            for j=transient_stag_branch_index(i) %15
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14:1
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        else
            for j=transient_stag_branch_index(i-1)+1 %16
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j-1);
```

```
            end
            for j=transient_stag_branch_index(i) %60
                mfr_seg_b(j) = mfr_seg_temp(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59;38
                mfr_seg_b(j) = mfr_seg_temp(j)-mfr_seg_temp(j+1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate mfr_seg_h
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    mfr_seg_h = zeros(1,inputs);
    for i=1:transient_num_chillers_final
        if i==1
            for j=transient_stag_branch_index(i) %15
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %14;1
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+1 %164
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for
j=transient_stag_branch_index(max(size(transient_stag_branch_index)))+2:input
s %165:180
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        else
            for j=transient_stag_branch_index(i) %60
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i)-1:-
1:transient_riser_branch_index(i) %59:38
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j+1);
            end
            for j=transient_stag_branch_index(i-1)+1 %16
                mfr_seg_h(j) = mfr_seg_b(j);
            end
            for j=transient_stag_branch_index(i-
1)+2:transient_riser_branch_index(i)-1 %17:37
                mfr_seg_h(j) = mfr_seg_b(j)+mfr_seg_h(j-1);
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_b_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
```

```
        V_SI_b_seg(i) =
mfr_seg_b(i)/area_b_unordered(branch_order(1,1,i))/rho;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate V_SI_h_seg
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        V_SI_h_seg(i) = mfr_seg_h(i)/area_h/rho;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine least and greatest branch velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_vel_b = min(V_SI_b_seg)
max_vel_b = max(V_SI_b_seg)
min_vel_h = min(V_SI_h_seg)
V_SI_h_seg(181) = 0;
max_vel_h = max(V_SI_h_seg)

%% Step 13 part f: Transient analysis - transient temperatures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define time step and annular segment granularity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_length_b = min(length_b(1,:));
mesh_b = floor(min_length_b/5*10)/10; %calculate mesh size such that there is
at least five segments in the shortest branch
mesh_b = min(mesh_b,1); %set mesh_b no larger than 1 meter

min_length_h = 10^10;
for i=1:inputs
    if min_length_h>length_h(1,1,i) && length_h(1,1,i)>0
        min_length_h = length_h(1,1,i);
    end
end
min_length_h;
mesh_h = floor(min_length_h/2*10)/10; %calculate mesh size such that there is
at least 2 segments in the shortest header segment
mesh_h = min(mesh_h,1); %set mesh_h no larger than 1 meter

timestep_b = mesh_b/max(V_SI_b_seg);
timestep_h = mesh_h/max(V_SI_h_seg);

timestep = min(timestep_b,timestep_h);
timestep = floor(timestep*10)/10;
if timestep == 0
    timestep = min(timestep_b,timestep_h);
    timestep = floor(timestep*100)/100; %maximum recommended timestep
end
if timestep == 0
    fprintf('Error: The minimum time step is less than a hundredth of a
second.\n')
```

```
end

fprintf('\nBased on the geometry of the chilled water system the recommended
mesh size \n')
fprintf('for the branch and header segments are %4.2f and %4.2f,
respectively.\n',mesh_b, mesh_h)
fprintf('The recommended time-step when analyzing the thermal transients is
%4.2f.\n',timestep)
fprintf('This should be considered an upper bound, else the response will
become unstable. The \n')
fprintf('time-step can be lowered, but will increase the computational time
and memory usage significantly.\n')
reply = 'n';
%reply = input('Do you wish to lower the time-step? [y/n]: ','s');
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('Please enter the time-step.\n')
    timestep = input('Time-step [s]: ');
end

time = 60; %total time of transient
remainder = mod(60,timestep);
time = time+remainder;
fprintf('The default time of the transient is %4.2f seconds.\n',time)
%reply = 'n';
reply = input('Do you wish to change it? [y/n]: ','s')
if isempty(reply)
    reply = 'y';
end
if strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('Please enter the time duration.\n')
    time = input('Time [s]: ');
end
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Segment the header pipe structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
header_index=1;
length_header_rev_cum = zeros(1,inputs+1);
length_header_rev = zeros(1,inputs+1);
for i=1:max(size(dPdX))
    if dPdX(1,1,i) == 2%branch
        length_header_rev_cum(header_index) = Location_x(1,1,i);
        header_index = header_index+1;
    end
end
length_header_rev_cum(header_index) = Location_x(1,1,max(size(dPdX)));
length_header_rev(1) = length_header_rev_cum(1);
for i=2:inputs+1
    length_header_rev(i) = length_header_rev_cum(i)-length_header_rev_cum(i-
1);
end
```

```
node_h_index = 0;
node_h_length = 0;
node_h_length_cum = 0;
node_h_junction_index = 0;
transient_riser_index_index = 1;
transient_stag_index_index = 1;
node_h_riser_index = 0;
node_h_stag_index = 0;
transient_riser_index_index_init = 1;
transient_stag_index_index_init = 1;
node_h_riser_index_init = 0;
node_h_stag_index_init = 0;
for i=1:inputs+1
    temp_var = floor(length_header_rev(i)/mesh_h);
    if temp_var > 0
        for j=1:temp_var
            node_h_index = node_h_index+1;
            node_h_length(node_h_index)=mesh_h;
            if node_h_index==1
                node_h_length_cum(node_h_index) = mesh_h;
            else
                node_h_length_cum(node_h_index) =
node_h_length_cum(node_h_index-1)+mesh_h;
            end
        end
    end
    temp_var_rem = length_header_rev(i) - temp_var*mesh_h;
    if temp_var_rem > 0
        if node_h_index==0
            node_h_index=1;
        end
        node_h_length(node_h_index)=node_h_length(node_h_index)+temp_var_rem;
        if node_h_index==1
            node_h_length_cum(node_h_index) = temp_var_rem;
        else
            node_h_length_cum(node_h_index) =
node_h_length_cum(node_h_index)+temp_var_rem;
        end
    end
    node_h_junction_index(i) = node_h_index;
    if i==transient_riser_branch_index(transient_riser_index_index)
        node_h_riser_index(transient_riser_index_index) = node_h_index-
temp_var;%-floor(temp_var/2);
        if transient_riser_index_index <
max(size(transient_riser_branch_index));
            transient_riser_index_index = transient_riser_index_index+1;
        end
    end
    if i==transient_stag_branch_index(transient_stag_index_index)
        node_h_stag_index(transient_stag_index_index) = node_h_index;
        if transient_stag_index_index <
max(size(transient_stag_branch_index));
            transient_stag_index_index = transient_stag_index_index+1;
        end
```

```
        end
        if i==transient_riser_branch_index_init(transient_riser_index_index_init)
            node_h_riser_index_init(transient_riser_index_index_init) =
node_h_index-temp_var;%-floor(temp_var/2);
            if transient_riser_index_index_init <
max(size(transient_riser_branch_index_init));
                transient_riser_index_index_init =
transient_riser_index_index_init+1;
            end
        end
        if i==transient_stag_branch_index_init(transient_stag_index_index_init)
            node_h_stag_index_init(transient_stag_index_index_init) =
node_h_index;
            if transient_stag_index_index_init <
max(size(transient_stag_branch_index_init));
                transient_stag_index_index_init =
transient_stag_index_index_init+1;
            end
        end
end
if node_h_riser_index(1)==0
    node_h_riser_index(1)=1;
end
if node_h_riser_index_init(1)==0
    node_h_riser_index_init(1)=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Segment the branch pipe structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_b_index = zeros(1,inputs);
node_b_length = zeros(inputs,1);
node_b_length_cum = zeros(inputs,1);
for i=1:inputs
    temp_var = floor(length_b_ordered(1,1,1,i)/mesh_b);
    if temp_var > 0
        for j=1:temp_var
            node_b_index(i) = node_b_index(i)+1;
            node_b_length(i,node_b_index(i))=mesh_b;
            if node_b_index(i)==1
                node_b_length_cum(i,node_b_index(i)) = mesh_b;
            else
                node_b_length_cum(i,node_b_index(i)) =
node_b_length_cum(i,node_b_index(i)-1)+mesh_b;
            end
        end
    end
    temp_var_rem = length_b_ordered(1,1,1,i) - temp_var*mesh_b;
    if temp_var_rem > 0

node_b_length(i,node_b_index(i))=node_b_length(i,node_b_index(i))+temp_var_re
m;
        if node_b_index(i)==1
            node_b_length_cum(i,node_b_index(i)) = temp_var_rem;
        else
```

```
            node_b_length_cum(i,node_b_index(i)) =
node_b_length_cum(i,node_b_index(i))+temp_var_rem;
        end
    end
end
size_node_b = size(node_b_length);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify location of heat exchanger - assume in center of branch piping
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_b_hxchgr = zeros(1,inputs);
node_b_vol_hxchgr = zeros(1,inputs);
for i=1:inputs
    node_b_hxchgr(i) = floor(node_b_index(i)/2);
    node_b_vol_hxchgr(i) = (hxchgr_weight_wet(i)-hxchgr_weight_dry(i))/rho;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify initial temp at each node
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_h_temp = Tcold*ones(1,node_h_index);

node_b_temp = zeros(size(node_b_length));
for i=1:inputs
    for j=1:node_b_hxchgr(i)-1
        node_b_temp(i,j)=Tcold;
    end
    for j=node_b_hxchgr(i):node_b_index(i)
        node_b_temp(i,j)=Thot_b_seg(i);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify final velocity at each node in header with positive clockwise
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_h_velocity = zeros(1,node_h_index);
node_h_velocity_init = zeros(1,node_h_index);
node_h_stag_index_index = 1;
node_h_riser_index_index = 1;
node_h_stag_index_index_init = 1;
node_h_riser_index_index_init = 1;
if node_h_riser_index(1)<node_h_stag_index(1)
    node_h_riser_index(max(size(node_h_riser_index))+1)=node_h_index;
    node_h_stag_index(max(size(node_h_stag_index))+1)=node_h_index+1;
else
    node_h_riser_index(max(size(node_h_riser_index))+1)=node_h_index+1;
    node_h_stag_index(max(size(node_h_stag_index))+1)=node_h_index;
end
if node_h_riser_index_init(1)<node_h_stag_index_init(1)

node_h_riser_index_init(max(size(node_h_riser_index_init))+1)=node_h_index;

node_h_stag_index_init(max(size(node_h_stag_index_init))+1)=node_h_index+1;
else
```

```
node_h_riser_index_init(max(size(node_h_riser_index_init))+1)=node_h_index+1;
    node_h_stag_index_init(max(size(node_h_stag_index_init))+1)=node_h_index;
end

for i=1:inputs
    if i==1 %consider first node
        for j=1:node_h_junction_index(i)-1
            if node_h_riser_index_index < max(size(node_h_riser_index))
                if j==node_h_riser_index(node_h_riser_index_index)
                    node_h_riser_index_index = node_h_riser_index_index+1;
                end
            end
            if node_h_stag_index_index < max(size(node_h_stag_index))
                if j==node_h_stag_index(node_h_stag_index_index)
                    node_h_stag_index_index = node_h_stag_index_index+1;
                end
            end
            if
node_h_riser_index(node_h_riser_index_index)>node_h_stag_index(node_h_stag_in
dex_index) %cw
                node_h_velocity(j)=V_SI_h_seg(i);
            else
                node_h_velocity(j)=-V_SI_h_seg(i);
            end
        end
    else
        if
node_h_riser_index(node_h_riser_index_index)>node_h_junction_index(i-
1)&&node_h_riser_index(node_h_riser_index_index)<node_h_junction_index(i)
            for j=node_h_junction_index(i-
1):node_h_riser_index(node_h_riser_index_index)-1
                if node_h_riser_index_index < max(size(node_h_riser_index))
                    if j==node_h_riser_index(node_h_riser_index_index)
                        node_h_riser_index_index =
node_h_riser_index_index+1;
                    end
                end
                if node_h_stag_index_index < max(size(node_h_stag_index))
                    if j == node_h_stag_index(node_h_stag_index_index)
                        node_h_stag_index_index = node_h_stag_index_index+1;
                    end
                end
                node_h_riser_index(node_h_riser_index_index);
                node_h_stag_index(node_h_stag_index_index);
                if
node_h_riser_index(node_h_riser_index_index)>node_h_stag_index(node_h_stag_in
dex_index) %cw
                    node_h_velocity(j)=V_SI_h_seg(i);
                else %ccw
                    node_h_velocity(j)=-V_SI_h_seg(i);
                end
            end
            for
j=node_h_riser_index(node_h_riser_index_index):node_h_junction_index(i)-1
```

```
                if node_h_riser_index_index < max(size(node_h_riser_index))
                    if j==node_h_riser_index(node_h_riser_index_index)
                        node_h_riser_index_index =
node_h_riser_index_index+1;
                    end
                end
                if node_h_stag_index_index < max(size(node_h_stag_index))
                    if j == node_h_stag_index(node_h_stag_index_index)
                        node_h_stag_index_index = node_h_stag_index_index+1;
                    end
                end
                if
node_h_riser_index(node_h_riser_index_index)>node_h_stag_index(node_h_stag_in
dex_index) %cw
                    node_h_velocity(j)=V_SI_h_seg(i);
                else %ccw
                    node_h_velocity(j)=-V_SI_h_seg(i);
                end
            end
        else
            for j=node_h_junction_index(i-1):node_h_junction_index(i)-1
                if node_h_riser_index_index < max(size(node_h_riser_index))
                    if j==node_h_riser_index(node_h_riser_index_index)
                        node_h_riser_index_index =
node_h_riser_index_index+1;
                    end
                end
                if node_h_stag_index_index < max(size(node_h_stag_index))
                    if j == node_h_stag_index(node_h_stag_index_index)
                        node_h_stag_index_index = node_h_stag_index_index+1;
                    end
                end
                if
node_h_riser_index(node_h_riser_index_index)>node_h_stag_index(node_h_stag_in
dex_index) %cw
                    node_h_velocity(j)=V_SI_h_seg(i);
                else %ccw
                    node_h_velocity(j)=-V_SI_h_seg(i);
                end
            end
        end
    end
end
for j=node_h_junction_index(inputs):node_h_index
    if node_h_velocity(node_h_junction_index(inputs)-1)<0
        node_h_velocity(j)=-V_SI_h_seg(inputs);
    else
        node_h_velocity(j)=V_SI_h_seg(inputs);
    end
end
for i=1:max(size(node_h_riser_index))-1
    node_h_riser_index_temp(i) = node_h_riser_index(i);
end
for i=1:max(size(node_h_stag_index))-1
    node_h_stag_index_temp(i) = node_h_stag_index(i);
```

```
end

for i=1:inputs
    if i==1 %consider first node
        for j=1:node_h_junction_index(i)-1
            if node_h_riser_index_index_init <
max(size(node_h_riser_index_init))
                if j==node_h_riser_index_init(node_h_riser_index_index_init)
                    node_h_riser_index_index_init =
node_h_riser_index_index_init+1;
                end
            end
            if node_h_stag_index_index_init <
max(size(node_h_stag_index_init))
                if j==node_h_stag_index_init(node_h_stag_index_index_init)
                    node_h_stag_index_index_init =
node_h_stag_index_index_init+1;
                end
            end
            if
node_h_riser_index_init(node_h_riser_index_index_init)>node_h_stag_index_init
(node_h_stag_index_index_init) %cw
                node_h_velocity_init(j)=V_SI_h_seg_init(i);
            else
                node_h_velocity_init(j)=-V_SI_h_seg_init(i);
            end
        end
    else
        for j=node_h_junction_index(i-1):node_h_junction_index(i)-1
            if node_h_riser_index_index_init <
max(size(node_h_riser_index_init))
                if j==node_h_riser_index_init(node_h_riser_index_index_init)
                    node_h_riser_index_index_init =
node_h_riser_index_index_init+1;
                end
            end
            if node_h_stag_index_index_init <
max(size(node_h_stag_index_init))
                if j == node_h_stag_index_init(node_h_stag_index_index_init)
                        node_h_stag_index_index_init =
node_h_stag_index_index_init+1;
                end
            end
            if
node_h_riser_index_init(node_h_riser_index_index_init)>node_h_stag_index_init
(node_h_stag_index_index_init) %cw
                node_h_velocity_init(j)=V_SI_h_seg_init(i);
            else %ccw
                node_h_velocity_init(j)=-V_SI_h_seg_init(i);
            end
        end
    end
end
for j=node_h_junction_index(inputs):node_h_index
    if node_h_velocity_init(node_h_junction_index(inputs)-1)<0
```

```
        node_h_velocity_init(j)=-V_SI_h_seg_init(inputs);
    else
        node_h_velocity_init(j)=V_SI_h_seg_init(inputs);
    end
end
for i=1:max(size(node_h_riser_index_init))-1
    node_h_riser_index_temp_init(i) = node_h_riser_index_init(i);
end
for i=1:max(size(node_h_stag_index_init))-1
    node_h_stag_index_temp_init(i) = node_h_stag_index_init(i);
end

clear node_h_riser_index node_h_stag_index node_h_riser_index_init
node_h_stag_index_init
node_h_riser_index = node_h_riser_index_temp
node_h_stag_index = node_h_stag_index_temp
node_h_riser_index_init = node_h_riser_index_temp_init
node_h_stag_index_init = node_h_stag_index_temp_init
node_h_riser_index_index = node_h_riser_index_index-1
node_h_stag_index_index = node_h_stag_index_index - 1
node_rh_velocity_init = node_h_velocity_init;
node_rh_velocity = node_h_velocity;
node_rh_index = node_h_index;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify final velocity at each node in each branch - only consider
% primary branches
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_b_velocity = zeros(size(node_b_length));
for i=1:inputs
    for j=1:node_b_index(i)
        node_b_velocity(i,j) = V_SI_b_seg(i);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine initial temperature in return header based on initial
% velocities and branch temperatures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_rh_temp = zeros(size(node_h_temp));
if node_h_stag_index_init(1) > node_h_riser_index_init(1)
    for k=1:transient_stag_index_index_init
        for i=transient_stag_branch_index_init(k):-
1:transient_riser_branch_index_init(k)
            if i==1
                for j=node_h_junction_index(i):-1:node_h_riser_index_init(k)
                   if i==transient_stag_branch_index_init(k)
                       node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                    else
                        if j==node_h_junction_index(i)

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...
```

```
node_rh_temp(j+1)*abs(node_rh_velocity_init(j))*area_h)/(node_b_velocity(i,no
de_b_index(i))*area_b_ordered(1,1,i)+...
                                        abs(node_rh_velocity_init(j))*area_h);
                            else
                                node_rh_temp(j)=node_rh_temp(j+1);
                            end
                        end
                    end
                else
                    if node_h_junction_index(i)==node_h_junction_index(i-1)
                        j=node_h_junction_index(i);
                        if i==transient_stag_branch_index_init(k)
                            node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                        else
                            if j==node_h_junction_index(i)
                                if node_rh_temp(j+1)>0

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...

node_rh_temp(j+1)*node_rh_velocity_init(j)*area_h)/(node_b_velocity(i,node_b_
index(i))*area_b_ordered(1,1,i)+...
                                            node_rh_velocity_init(j)*area_h);
                                end
                            end
                        end
                    else
                        for j=node_h_junction_index(i):-
1:node_h_junction_index(i-1)+1
                            if i==transient_stag_branch_index_init(k)
                                node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                            else
                                if j==node_h_junction_index(i)
                                    if node_rh_temp(j+1)>0

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...

node_rh_temp(j+1)*abs(node_rh_velocity_init(j))*area_h)/(node_b_velocity(i,no
de_b_index(i))*area_b_ordered(1,1,i)+...

abs(node_rh_velocity_init(j))*area_h);
                                    end
                                else
                                    if node_rh_temp(j+1)>0
                                        node_rh_temp(j)=node_rh_temp(j+1);
                                    end
                                end
                            end
                        end
                    end
                end
            end
            if k<transient_riser_index_index_init
```

```
            for
i=transient_stag_branch_index_init(k):transient_riser_branch_index_init(k+1)
                if node_h_junction_index(i)==node_h_junction_index(i+1)
                    j=node_h_junction_index(i);
                    if i==transient_stag_branch_index_init(k)
                        node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                    else
                        if j==node_h_junction_index(i)
                            if node_rh_temp(j+1)>0

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...

node_rh_temp(j+1)*node_rh_velocity_init(j)*area_h)/(node_b_velocity(i,node_b_
index(i))*area_b_ordered(1,1,i)+...
                                node_rh_velocity_init(j)*area_h);
                            end
                        end
                    end
                else
                    for
j=node_h_junction_index(i):node_h_junction_index(i+1)-1
                        if i==transient_stag_branch_index_init(k)
                            node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                        else
                            if j==node_h_junction_index(i)
                                if node_rh_temp(j-1)>0

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...
                                    node_rh_temp(j-
1)*abs(node_rh_velocity(j))*area_h)/(node_b_velocity(i,node_b_index(i))*area_
b_ordered(1,1,i)+...
                                    abs(node_rh_velocity(j))*area_h);
                                end
                            else
                                if node_rh_temp(j-1)>0
                                    node_rh_temp(j)=node_rh_temp(j-1);
                                end
                            end
                        end
                    end
                end
            end
        else
            for i=transient_stag_branch_index_init(k):inputs
                if i==inputs
                    for j=node_h_junction_index(i):node_h_index
                        if j==node_h_junction_index(i)

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...
                            node_rh_temp(j-
1)*abs(node_rh_velocity_init(j))*area_h)/(node_b_velocity(i,node_b_index(i))*
area_b_ordered(1,1,i)+...
```

```
                                        abs(node_rh_velocity_init(j))*area_h);
                                else
                                    node_rh_temp(j)=node_rh_temp(j-1);
                                end
                            end
                        else
                            for
j=node_h_junction_index(i):node_h_junction_index(i+1)-1
                                if i==transient_stag_branch_index_init(k)
                                    node_rh_temp(j)=node_b_temp(i,node_b_index(i));
                                else
                                    if j==node_h_junction_index(i)

node_rh_temp(j)=(node_b_temp(i,node_b_index(i))*node_b_velocity(i,node_b_inde
x(i))*area_b_ordered(1,1,i)+...
                                           node_rh_temp(j-
1)*abs(node_rh_velocity_init(j))*area_h)/(node_b_velocity(i,node_b_index(i))*
area_b_ordered(1,1,i)+...
                                            abs(node_rh_velocity_init(j))*area_h);
                                    else
                                        node_rh_temp(j)=node_rh_temp(j-1);
                                    end
                                end
                            end
                        end
                    end
                end
            end
    for i=1:(transient_riser_branch_index_init(1)-1)
        %do something
    end
else
    %do something
end
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine total number of increments in time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iterations = floor(time/timestep);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine volume and surface area of each node in header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
node_h_vol = zeros(1,node_h_index);
node_h_SA = zeros(1,node_h_index);
for x=1:node_h_index
    node_h_vol(x) = area_h*node_h_length(x);
    node_h_SA(x) = pi()*node_h_length(x)*D_SI_h;
end
node_rh_vol = node_h_vol;
node_rh_SA = node_h_SA;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine volume of each node in branches
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
node_b_vol = zeros(inputs,size_node_b(2));
node_b_SA = zeros(inputs,size_node_b(2));
for i=1:inputs
    for x=1:node_b_index(i)
        if x==node_b_hxchgr(i)
            node_b_vol(i,x) = node_b_vol_hxchgr(i);
        else
            node_b_vol(i,x) = area_b_ordered(1,1,i)*node_b_length(i,x);
            node_b_SA(i,x) = pi()*node_b_length(i,x)*D_SI_b_ordered(i);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocate/initialize variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
radius_h = D_SI_h/2;
radius_b = D_SI_b_ordered/2;
lagging_thickness = 0.01; % 1 cm lagging
klagging = 0.035; %lagging
T_amb = 20;

Q1_h = zeros(node_h_index,iterations);
Q2_h = zeros(node_h_index,iterations);
Qgen_h = zeros(node_h_index,iterations);
Qloss_h = zeros(node_h_index,iterations);
node_h_hc_air = zeros(node_h_index,iterations);
node_h_hc_cw = zeros(node_h_index,iterations);
node_h_U = zeros(node_h_index,iterations);
node_h_T = zeros(node_h_index,iterations);
dT_h = zeros(node_h_index,iterations);

Q1_rh = zeros(node_h_index,iterations);
Q2_rh = zeros(node_h_index,iterations);
Qgen_rh = zeros(node_h_index,iterations);
Qloss_rh = zeros(node_h_index,iterations);
node_rh_hc_air = zeros(node_h_index,iterations);
node_rh_hc_cw = zeros(node_h_index,iterations);
node_rh_U = zeros(node_h_index,iterations);
node_rh_T = zeros(node_h_index,iterations);
dT_rh = zeros(node_h_index,iterations);

Q1_b = zeros(size_node_b(1),size_node_b(2),iterations);
Q2_b = zeros(size_node_b(1),size_node_b(2),iterations);
Qloss_b = zeros(size_node_b(1),size_node_b(2),iterations);
Qgen_b = zeros(size_node_b(1),size_node_b(2),iterations);
node_b_hc_air = zeros(size_node_b(1),size_node_b(2),iterations);
node_b_hc_cw = zeros(size_node_b(1),size_node_b(2),iterations);
node_b_U = zeros(size_node_b(1),size_node_b(2),iterations);
node_b_T = zeros(size_node_b(1),size_node_b(2),iterations);
dT_b = zeros(size_node_b(1),size_node_b(2),iterations);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine new temperatures based on new heat loads and new velocities
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for t=1:iterations
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q1 at each node in header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
        if node_h_velocity(x)>0
            if t == 1
                if x == 1
                    Q1_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_temp(node_h_index)-node_h_temp(x));
                else
                    Q1_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_temp(x-1)-node_h_temp(x));
                end
            else
                if x == 1
                    Q1_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_T(node_h_index,t-1)-node_h_T(x,t-
1));
                else
                    Q1_h(x,t) = rho*area_h*node_h_velocity(x)*cp*(node_h_T(x-
1,t-1)-node_h_T(x,t-1));
                end
            end
        else
            Q1_h(x,t) = 0;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q1 at each node in return header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_rh_index
        if node_rh_velocity(x)>0
            if t == 1
                if x == node_rh_index
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_temp(1)-
node_rh_temp(node_rh_index));
                else
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_temp(x+1)-node_rh_temp(x));
                end
            else
               if x == node_rh_index
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_T(1,t-1)-node_h_T(node_rh_index,t-
1));
               else
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_T(x+1,t-1)-node_rh_T(x,t-1));
               end
            end
        else
```

```
            if t == 1
                if x == 1
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_temp(1)-
node_rh_temp(node_rh_index));
                else
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_temp(x)-node_rh_temp(x-1));
                end
            else
               if x == 1
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_T(1,t-1)-node_h_T(node_rh_index,t-
1));
               else
                    Q1_rh(x,t) =
rho*area_h*node_rh_velocity(x)*cp*(node_rh_T(x,t-1)-node_rh_T(x-1,t-1));
               end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q1 at each node in branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        for x=1:node_b_index(i)
            if t == 1
                if x == 1
                    Q1_b(i,x,t) =
rho*area_b_ordered(1,1,i)*node_b_velocity(i,x)*cp*(node_h_temp(node_h_junctio
n_index(i))-node_b_temp(i,x));
                else
                    Q1_b(i,x,t) =
rho*area_b_ordered(1,1,i)*node_b_velocity(i,x)*cp*(node_b_temp(i,x-1)-
node_b_temp(i,x));
                end
            else
                if x == 1
                    Q1_b(i,x,t) =
rho*area_b_ordered(1,1,i)*node_b_velocity(i,x)*cp*(node_h_T(node_h_junction_i
ndex(i),t-1)-node_b_T(i,x,t-1));
                else
                    Q1_b(i,x,t) =
rho*area_b_ordered(1,1,i)*node_b_velocity(i,x)*cp*(node_b_T(i,x-1,t-1)-
node_b_T(i,x,t-1));
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q2 at each node in header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
```

```
        if node_h_velocity(x)<0
            if t == 1
                if x == node_h_index
                    Q2_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_temp(node_h_index)-node_h_temp(1));
                else
                    Q2_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_temp(x)-node_h_temp(x+1));
                end
            else
                if x == node_h_index
                    Q2_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_T(node_h_index,t-1)-node_h_T(1,t-
1));
                else
                    Q2_h(x,t) =
rho*area_h*node_h_velocity(x)*cp*(node_h_T(x,t-1)-node_h_T(x+1,t-1));
                end
            end
        else
            Q2_h(x,t) = 0;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q2 at each node in return header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        x_h=node_h_junction_index(i);
        x_b=node_b_index(i);
        if t == 1
            Q2_rh(x_h,t) =
Q2_rh(x_h,t)+rho*area_b_ordered(1,1,x_b)*node_b_velocity(i,x_b)*cp*(node_b_te
mp(i,x_b)-node_rh_temp(x_h));
        else
            Q2_rh(x_h,t) =
Q2_rh(x_h,t)+rho*area_b_ordered(1,1,x_b)*node_b_velocity(i,x_b)*cp*(node_b_T(
i,x_b,t-1)-node_rh_T(x_h,t-1));
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Q2 at each node in branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %for i=1:inputs
    %    for x=1:node_b_index(i)
            %if t == 1
            %   if x == 1
            %        Q2_b(i,x,t) =
0;%rho*area_b_ordered(i)*node_b_velocity(i,x)*cp*(node_h_temp(node_h_index)-
node_b_temp(i,x));
            %    else
            %        Q2_b(i,x,t) =
0;%rho*area_b_ordered(i)*node_b_velocity(i,x)*cp*(node_b_temp(i,x-1)-
node_b_temp(i,x));
```

```
            %    end
            %else
            %    if x == 1
            %        Q2_b(i,x,t) =
0;%rho*area_b_ordered(i)*node_b_velocity(i,x)*cp*(node_h_T(node_h_index,t-1)-
node_b_T(i,x,t-1));
            %    else
            %        Q2_b(i,x,t) =
0;%rho*area_b_ordered(i)*node_b_velocity(i,x)*cp*(node_b_T(i,x-1,t-1)-
node_b_T(i,x,t-1));
            %    end
            %end
        %end
   % end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Qgen at each node in header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %if t==1
    %    Qgen_h(x,t) = 0;
    %else
    %    Qgen_h(x,t)=0;
    %end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Qgen at each node in branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        for x=node_b_hxchgr(i)
            Qgen_b(i,x,t) = transient_Q_final(i)*1000;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Qloss at each node in header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
        node_h_hc_cw(x,t) = calc_hc(D_SI_h,node_h_velocity(x),k,nu,rho,cp);
        node_h_hc_air(x,t) = 0.1128; %can use other method to determine this
later - Nusselt#?
        node_h_U(x,t) =
(1/node_h_hc_cw(x,t)+(radius_h)/klagging*log((lagging_thickness+radius_h+thic
kness_h)/radius_h)+...

radius_h/kcopper*log((thickness_h+radius_h)/radius_h)+radius_h/(radius_h+thic
kness_h+lagging_thickness)/node_h_hc_air(x,t))^-1;
        if t==1
            Qloss_h(x,t)=node_h_U(x,t)*node_h_SA(x)*(T_amb-node_h_temp(x));
        else
            Qloss_h(x,t)=node_h_U(x,t)*node_h_SA(x)*(T_amb-node_h_T(x,t-1));
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Specify Qloss at each node in return header
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
        node_rh_hc_cw(x,t) = calc_hc(D_SI_h,node_rh_velocity(x),k,nu,rho,cp);
        node_rh_hc_air(x,t) = 0.1128; %can use other method to determine this
later - Nusselt#?
        node_rh_U(x,t) =
(1/node_rh_hc_cw(x,t)+(radius_h)/klagging*log((lagging_thickness+radius_h+thi
ckness_h)/radius_h)+...

radius_h/kcopper*log((thickness_h+radius_h)/radius_h)+radius_h/(radius_h+thic
kness_h+lagging_thickness)/node_rh_hc_air(x,t))^-1;
        if t==1
            Qloss_rh(x,t)=node_rh_U(x,t)*node_rh_SA(x)*(T_amb-
node_rh_temp(x));
        else
            Qloss_rh(x,t)=node_rh_U(x,t)*node_rh_SA(x)*(T_amb-node_rh_T(x,t-
1));
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Specify Qloss at each node in branch
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:inputs
        for x=1:node_b_index(i)
           node_b_hc_cw(i,x,t) =
calc_hc(D_SI_b_ordered(i),node_b_velocity(i,x),k,nu,rho,cp);
            node_b_hc_air(i,x,t) = 0.1128; %can use other method to determine
this later - Nusselt#?
            node_b_U(i,x,t) =
(1/node_b_hc_cw(i,x,t)+(radius_b(i))/klagging*log((lagging_thickness+radius_b
(i)+thickness_b(branch_order(1,1,i)))/radius_b(i))+...

radius_b(i)/kcopper*log((thickness_b(branch_order(1,1,i))+radius_b(i))/radius
_b(i))+radius_b(i)/(radius_b(i)+thickness_b(branch_order(1,1,i))+lagging_thic
kness)/node_b_hc_air(i,x,t))^-1;
            if t==1
                Qloss_b(i,x,t)=node_b_U(i,x,t)*node_b_SA(i,x)*(T_amb-
node_b_temp(i,x));
            else
                Qloss_b(i,x,t)=node_b_U(i,x,t)*node_b_SA(i,x)*(T_amb-
node_b_T(i,x,t-1));
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Determine dT at each node
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
        dT_h(x,t) =
(Q1_h(x,t)+Q2_h(x,t)+Qgen_h(x,t)+Qloss_h(x,t))/(rho*node_h_vol(x)*cp)*timeste
p; %revise
    end
```

```
    for i=1:inputs
        for x=1:node_b_index(i)
            dT_b(i,x,t) =
(Q1_b(i,x,t)+Q2_b(i,x,t)+Qloss_b(i,x,t)+Qgen_b(i,x,t))/(rho*node_b_vol(i,x)*c
p)*timestep;
        end
    end
    for x=1:node_rh_index
        dT_rh(x,t) =
(Q1_rh(x,t)+Q2_rh(x,t)+Qgen_rh(x,t)+Qloss_rh(x,t))/(rho*node_rh_vol(x)*cp)*ti
mestep; %revise
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate temperatures after timestep
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for x=1:node_h_index
        if t==1
            node_h_T(x,t) = node_h_temp(x)+dT_h(x,t);
        else
            node_h_T(x,t) = node_h_T(x,t-1)+dT_h(x,t);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Specify boundary conditions
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=1:max(size(node_h_riser_index))
            if x==node_h_riser_index(j)
                node_h_T(x,t) = Tcold;
            end
        end
    end
    for i=1:inputs
        for x=1:node_b_index(i)
            if t==1
                node_b_T(i,x,t) = node_b_temp(i,x)+dT_b(i,x,t);
            else
                node_b_T(i,x,t) = node_b_T(i,x,t-1)+dT_b(i,x,t);
            end
        end
    end
    for x=1:node_rh_index
        if t==1
            node_rh_T(x,t) = node_rh_temp(x)+dT_rh(x,t);
        else
            node_rh_T(x,t) = node_rh_T(x,t-1)+dT_rh(x,t);
        end
    end
end
%% Step 13 part g: Transient analysis - plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot temperatures over time at a specified location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
node_rh_riser_index = node_h_riser_index;
node_rh_stag_index = node_h_stag_index;
node_rh_junction_index = node_h_junction_index;
time_axis = zeros(1,iterations);
for i=1:iterations
    time_axis(i)=i*timestep;
end

fprintf('Do you want to analyze the temperature as a function of time at a
specific \n')
reply = input('location within the chilled water system? [y/n]: ','s');

while strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('Please select the general location you wish to analyze from the
pop-up menu')
    pipe_type = menu('Choose a location','Supply Header','Return
Header','Branch');
    fprintf('\n')
    if pipe_type == 1 %Supply Header
        fprintf('The Supply Header is broken up into %d annular segments with
node 1 corresponding to the\n',node_h_index)
        fprintf('riser location of the forward-most chiller portside. The
indices are incremented clockwise \n')
        fprintf('along the length of the Supply Header.\n')
        fprintf('The indices for the riser locations are:\n')
        node_h_riser_index
        fprintf('The indices for the stagnation points are:\n')
        node_h_stag_index
        fprintf('The indices for the branch junctions are:\n')
        node_h_junction_index
        input_index = input('Please enter the supply header index you wish to
analyze: ');
        plot_var = zeros(1,iterations);
        for i=1:iterations
            plot_var(i) = node_h_T(input_index,i);
        end
        plot(time_axis,plot_var)
        xlabel('Time(sec)')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Time within Supply Header')
    elseif pipe_type == 2 %Return Header
        fprintf('The Return Header is broken up into %d annular segments with
node 1 corresponding to the\n',node_rh_index)
        fprintf('riser location of the forward-most chiller portside. The
indices are incremented clockwise\n')
        fprintf('along the length of the Return Header.\n')
        fprintf('The indices for the riser locations are:\n')
        node_rh_riser_index
        fprintf('The indices for the stagnation points are:\n')
        node_rh_stag_index
        fprintf('The indices for the branch junctions are:\n')
        node_rh_junction_index
        input_index = input('Please enter the return header index you wish to
analyze: ');
        plot_var = zeros(1,iterations);
```

```
        for i=1:iterations
            plot_var(i) = node_rh_T(input_index,i);
        end
        plot(time_axis,plot_var)
        xlabel('Time(sec)')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Time within Return Header')
    else %Branch
        fprintf('There are %d branches in the chilled water system.
\n',inputs)
        branch_var = input('Please enter the branch number you want to
analyze: ');
        fprintf('There are %d indices in branch %d. The indices are
incremented from supply to return.\n',node_b_index(branch_var),branch_var)
        fprintf('The heat exchanger is located at index
%d.\n',node_b_hxchgr(branch_var))
        input_index = input('Please enter the branch index you wish to
analyze: ');
        plot_var = zeros(1,iterations);
        for i=1:iterations
            plot_var(i) = node_b_T(branch_var,input_index,i);
        end
        plot(time_axis,plot_var)
        xlabel('Time(sec)')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Time within Branch')
    end
    fprintf('Do you want to analyze the temperature as a function of time at
another \n')
    reply = input('location within the chilled water system? [y/n]: ','s');
end
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot temperatures over distance at a specified time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('Do you want to analyze the temperature as a function of distance at
a specific \n')
reply = input('time within the chilled water system? [y/n]: ','s');

while strcmp(reply,'y') || strcmp(reply,'Y') || strcmp(reply,'yes')
    fprintf('Please select the general location you wish to analyze from the
pop-up menu')
    pipe_type = menu('Choose a location','Supply Header','Return
Header','Branch');
    fprintf('\n')
    if pipe_type == 1 %Supply Header
        fprintf('The total time analyzed is %d seconds with a timestep of
%d.\n',time,timestep)
        input_index = input('Please enter the time you wish to analyze [sec]:
');
        input_index = input_index/timestep;
        plot_var = zeros(1,node_h_index);
        for i=1:node_h_index
            plot_var(i) = node_h_T(i,input_index);
```

```
        end
        plot(1:node_h_index,plot_var)
        xlabel('Header Index')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Distance within Supply Header')
    elseif pipe_type == 2 %Return Header
        fprintf('The total time analyzed is %d seconds with a timestep of
%d.\n',time,timestep)
        input_index = input('Please enter the time you wish to analyze [sec]:
');
        input_index = input_index/timestep;
        plot_var = zeros(1,node_rh_index);
        for i=1:node_rh_index
            plot_var(i) = node_rh_T(i,input_index);
        end
        plot(1:node_rh_index,plot_var)
        xlabel('Return Header Index')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Distance within Return Header')
    else %Branch
        fprintf('There are %d branches in the chilled water system.
\n',inputs)
        branch_var = input('Please enter the branch number you want to
analyze: ');
        fprintf('The total time analyzed is %d seconds with a timestep of
%d.\n',time,timestep)
        input_index = input('Please enter the time you wish to analyze [sec]:
');
        input_index = input_index/timestep;
        plot_var = zeros(1,node_b_index(branch_var));
        for i=1:node_b_index(branch_var)
            plot_var(i) = node_b_T(branch_var,i,input_index);
        end
        plot(1:node_b_index(branch_var),plot_var)
        xlabel('Branch Index')
        ylabel('Temperature(C)')
        title('Temperature as a Function of Distance within Branch')
    end
    fprintf('Do you want to analyze the temperature as a function of distance
at another \n')
    reply = input('time within the chilled water system? [y/n]: ','s');
end

%%
f=2
plot_var1 = zeros(1,node_b_index(f));
plot_var2 = zeros(1,node_b_index(f));
plot_var3 = zeros(1,node_b_index(f));
plot_var4 = zeros(1,node_b_index(f));
plot_var5 = zeros(1,node_b_index(f));
for i=1:node_b_index(f)
    plot_var1(i) = node_b_T(f,i,1);
    plot_var2(i) = node_b_T(f,i,10);
    plot_var3(i) = node_b_T(f,i,30);
    plot_var4(i) = node_b_T(f,i,50);
```

```
    plot_var5(i) = node_b_T(f,i,100);
end
plot(1:node_b_index(f),plot_var1,'r')
hold on
plot(1:node_b_index(f),plot_var2,'g')
plot(1:node_b_index(f),plot_var3,'b')
plot(1:node_b_index(f),plot_var4,'c')
plot(1:node_b_index(f),plot_var5,'k')

%% Survivbility

% Add survivability code here

% User defined blast location and radius

% Determine heat exchangers located within blast radius

% Determine chillers/pumps located within blast radius

% Segment pipe into a series of line segments and see if either end of
% segment falls within blast radius - if it does the segment is damaged if
% not, calculate line perpendicular to line segment which crosses center of
% blast. If perpendicular line length is less than blast raius and line
% falls within segments, then segment is damaged.

% Determine connectivity of remaining heat exchangers to remaining chillers
% through undamaed piping

% Prioritize flow to vital loads for those with at least 1 flow path
% remaining. Then prioritize flow to non-vital loads.

% Print report of heat exchangerd damaged and heat loads which can not be
% cooled due to no connectivity and heat loads which can not be cooled due
% to lack of cooling
```

analysis_interface.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cooling System Design Tool                          %
% Author: Ben Sanfiorenzo                              %
% Analysis interface module: Loads data from          %
% geometry.mat and provides the user with the ability %
% to modify the contents of the data. User must have  %
% a detailed understanding of the variables and the   %
% code. The analysis interface module stores the      %
% modified data in the file analysis_interface.mat.   %
% Last Modified: 3-2-13                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear
load geometry
who
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User inserts code here to modify the geometry.mat file
% Ex: To modify the number of zones from 4 to 5
% zones = 5;
% zonal_boundaries = [40 20 0 -20 -40]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Insert code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
zones = 5;

save analysis_interface
```

Center for Ocean Engineering
Naval Construction & Engineering Program
Department of Mechanical Engineering

Massachusetts Institute of Technology
77-massachusetts Avenue, Building 5-317
Cambridge, Massachusetts 02139–4307

calc_h_sat.m

```
function outarg = calc_h_sat(T,R_Sat_T,R_Sat_hf_or_hg)
% Determines the enthalpy (kJ/kg) of a refrigerant given the pressure (MPa)
and
% temperature (C). The function uses a matrix containing enthalpies at
% temperatures and pressures in the saturated range. The function
% linearly interpolates the enthalpy value.

size_sat_T = max(size(R_Sat_T));
for i=1:(size_sat_T-1)
    if T == R_Sat_T(i)
        index_temp = i;
    elseif (R_Sat_T(i) < T) && (T < R_Sat_T(i+1))
        index_temp = i;
    end
end
h1 = R_Sat_hf_or_hg(index_temp);
h2 = R_Sat_hf_or_hg(index_temp+1);
T1 = R_Sat_T(index_temp);
T2 = R_Sat_T(index_temp+1);
outarg = h1+(T-T1)/(T2-T1)*(h2-h1);
```

calc_h_SHV.m

```
function outarg = calc_h_SHV(T,P,R_SHV_T,R_SHV_P,R_SHV_h)
% Determines the enthalpy (kJ/kg) of a refrigerant given the pressure (MPa)
and
% temperature (C). The function uses a matrix containing enthalpies at
% temperatures and pressures in the superheated vapor range. The function
% linearly interpolates the enthalpy value.

size_SHV_T = max(size(R_SHV_T));
size_SHV_P = max(size(R_SHV_P));
for i=1:(size_SHV_T-1)
    if T == R_SHV_T(i)
        index_temp = i;
        flag_temp = 0;
    elseif (R_SHV_T(i) < T) && (T < R_SHV_T(i+1))
        index_temp = i;
        flag_temp = 1;
    end
end
for i=1:(size_SHV_P-1)
    if P == R_SHV_P(i)
        index_pres = i;
        flag_pres = 0;
    elseif (R_SHV_P(i) < P) && (P < R_SHV_P(i+1))
        index_pres = i;
        flag_pres = 1;
    end
end

h1 = R_SHV_h(index_temp,index_pres);
h2 = R_SHV_h(index_temp,index_pres+1);
h3 = R_SHV_h(index_temp+1,index_pres);
h4 = R_SHV_h(index_temp+1,index_pres+1);
T1 = R_SHV_T(index_temp);
T2 = R_SHV_T(index_temp+1);
P1 = R_SHV_P(index_pres);
P2 = R_SHV_P(index_pres+1);
outarg = h1+(P-P1)/(P2-P1)*(h2-h1)+(T-T1)/(T2-T1)*(h1+(P-P1)/(P2-P1)*(h2-h1)-
(h3+(T-T1)/(T2-T1)*(h4-h3)));
end
```

calc_hc.m

```
function outarg = calc_hc(diameter,velocity,k,nu,rho,cp)
%Calculate convective heat transfer coefficient

%Determine flow regime in branch/header end
Re = velocity*diameter/nu; %250-laminar  10000-turbulent

%Calculate Darcy friction factor
if(Re < 250)
    outarg = 3.66*k/diameter; %laminar flow
else
    outarg =
0.023*(velocity^0.8)*(k^0.6)*((rho*cp)^0.4)/(diameter^0.2)/(nu^0.4);
%turbulent flow
end
```

friction_factor.m

```
function outarg = friction_factor(diameter,velocity,k,nu,epsilon,rho,cp)
%Calculate Darcy friction factor

%Determine flow regime in branch/header end
Re = velocity*diameter/nu; %250-laminar  10000-turbulent

%Calculate Darcy friction factor
if(Re < 250)
    outarg = 64/Re; %Darcy friction factor for laminar flow
else
    f_0 = 0.02;
    f_1 = (-2*log10(epsilon/(3.7*diameter)+2.51/(Re*f_0^0.5)))^-2;
    f_2 = (-2*log10(epsilon/(3.7*diameter)+2.51/(Re*f_1^0.5)))^-2;
    f_3 = (-2*log10(epsilon/(3.7*diameter)+2.51/(Re*f_2^0.5)))^-2;
    outarg = (-2*log10(epsilon/(3.7*diameter)+2.51/(Re*f_3^0.5)))^-2;
end
```

pump_curves.m

```
function outarg = pump_curves(pump_matrix,mfr_matrix,pump_head,mfr)
% Selects a pump and provides the pump curve for a given operating condition

size_matrix = size(pump_matrix);
pump_curve = zeros(size_matrix(1),3);
for i=1:size_matrix(1)
    pump_vector = [pump_matrix(i,1) pump_matrix(i,2) pump_matrix(i,3)
pump_matrix(i,4)];
    mfr_vector = [mfr_matrix(i,1) mfr_matrix(i,2) mfr_matrix(i,3)
mfr_matrix(i,4)];
    pump_curve1(i,:) = polyfit(mfr_vector,pump_vector,2);
    pump_curve2(i,:) = polyfit(pump_vector,mfr_vector,2);
end
selected = 1;
min_dist = 10000000000000;
for i=1:size_matrix(1)
    if pump_head < polyval(pump_curve1(i,:),mfr)
        if mfr < mfr_matrix(i,4)
            dist = polyval(pump_curve1(i,:),mfr)-pump_head;
            if dist<min_dist
                min_dist=dist;
                selected = i;
            end
        end
    end
end
outarg = [pump_curve1(selected,:);pump_curve2(selected,:)];
```