Integrated Cryogenic Refrigeration System Design For Superconducting Magnetic Energy Storage Systems

by

Brian J. Bowers

B.S., Mechanical Engineering
University of Wisconsin-Platteville, 1994

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Mechanical Engineering

at the
Massachusetts Institute of Technology

June 1997

Signature of Author________________________________
Department of Mechanical Engineering
May 9, 1997

Certified by________________________________
Joseph L. Smith, Jr.
Collins Professor of Mechanical Engineering
Thesis Supervisor

Accepted by________________________________
Ain A. Sonin
Chairman of Departmental Graduate Committee
Department of Mechanical Engineering

Integrated Cryogenic Refrigeration System Design
For Superconducting Magnetic Energy Storage Systems

by

Brian J. Bowers

Submitted To the Department of Mechanical Engineering on May 9, 1997
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

ABSTRACT

The cryogenic refrigeration system is a significant part of any superconducting magnetic energy storage (SMES) system. Matching the designs of the magnet and refrigeration system could reduce the required cooling power and lead to a smaller, less expensive refrigeration system.

This study looked at the magnet/refrigerator interactions in the lowest temperature area of the SMES system since this area affects the performance of the entire refrigerator. Magnets made with cable-in-conduit conductors (CICCs) were considered.

Computer models were used to study the flow of helium through the flow passage of a cable-in-conduit conductor. A similarity relation was found, which can estimate the maximum amount of heat that can be removed for flow passages of different sizes under steady state conditions.

The transient response of a recirculator-loop cooling configuration was modeled. This scheme cools the magnet by circulating helium in a closed loop and rejecting heat to a buffer tank, which is cooled by the refrigerator. The buffer is a closed tank with two-phase helium that absorbs heat during magnet discharges to "average" the load on the refrigerator. Characteristics of the pump, heat exchangers, and buffer tank were included in the model. Several buffers were compared.

The modeling showed that the pumping power required for forced-flow cooling can significantly increase the refrigeration load. The pumping power can be reduced by re-cooling the helium before it enters the pump and by using a lower pressure in the system. High efficiency pumps and parallel flow cooling configurations can also decrease the pumping power and allow a smaller refrigerator to be used. Future SMES models should include pump characteristics to better estimate the system response and the size of the required refrigeration system.

Thesis Supervisor: Professor Joseph L. Smith, Jr.
Collins Professor of Mechanical Engineering
Department of Mechanical Engineering

Acknowledgments

I thank my advisor, Professor Joseph L. Smith, for his patient guidance and for giving me the opportunity to do graduate work at MIT. I also thank Westinghouse and the U. S. Navy for sponsoring this project and providing financial support.

I thank Dr. Joseph Minervini for filling in as my advisor last summer while Professor Smith took advantage of a research opportunity in Japan.

I express my sincere appreciation for Greg Nellis (now Dr. Nellis), a fellow graduate student who answered my numerous questions about thermodynamics and taught me computational techniques that I used throughout this project.

I thank Ashok Patel and Professor John Brisson for allowing me to use their computer.

I acknowledge Dr. Edward Ognibene, who started this project and passed it on to me.

I recognize Joel Schultz and Pei Wen Wang, of the MIT Plasma fusion center for providing me with memos and computer software.

I thank my friends in Ashdown House for supporting me and I thank my roommate, Li Ju, for putting up with the noisy fan on my computer as it ran night and day to perform the simulations. I thank Tom Burbine and Dhaya Lakshminarayanan for proofreading and for bringing excitement into the otherwise boring social life of a graduate dorm.

And I especially thank Doris Elsemiller, for her reminders and advice, for mailing my monthly reports, for giving me candy, and for her caring personality.

Table of Contents

List of Figures

List of Tables

Nomenclature

A	cross sectional area of flow
C	flow constant
Cp	specific heat
D_h	hydraulic diameter
E'	change of energy per time of the control volume
f	friction factor
g	gravitational acceleration
h	specific enthalpy
h_{fg}	heat of vaporization
L	length of flow passage
m	mass
m'	mass flow rate
n	number of nodes
P	absolute pressure
P_{min}	minimum allowable pressure in conductor flow passages
q'	rate of heat addition per unit length
Q	energy transfer
Q'	heat transfer per time
Q'_{load}	total rate of heat addition to buffer tank per time
Q'_{magnet}	total rate of heat removal from the magnet per time
Q'_{ss}	steady state cooling power provided by refrigerator
R_{He}	gas constant of helium
s	specific entropy
$Sgen$	entropy generation
T	temperature
T_{max}	maximum allowable helium temperature in conductor flow passages
T_{Buffer}	temperature of buffer tank
u	specific internal energy
ua	overall heat transfer coefficient per length of heat exchanger
w	work per unit mass
W_{AB}	total work done by A on B
w'	work transfer per time per length of conductor
W'	power (work transfer per time)
W'_{pump}	pumping power
v	velocity
V	volume
x	distance from inlet of flow passage
x_{init}	initial quality of helium in buffer tank
X	dimensionless distance from inlet of flow passage (x/L)
z	elevation

Γ	ratio of initial cold volume to initial warm volume in divided tank buffer concept
η	isentropic efficiency of pump
Θ	dimensionless temperature (T/T_{ref})
ν	specific volume
Π	dimensionless pressure (P/P_{ref})
ρ	density
ϑ	ratio of change in cold volume to original cold volume in divided tank buffer

Δt	time step
ΔV	change in volume
Δx	length of control volume segment

$\left.\frac{da}{db}\right\|_c$	derivative of a with respect to b while holding c constant

Subscripts

1, 2	first and second conductors being compared for similarity OR initial and final conditions in divided buffer tank
A-D	locations in recirculator loop
A, B	compartments in divided tank buffer scheme
ambient	surrounding environment
Buffer	buffer tank
final	ending condition
HX	heat exchanger
in	inlet of magnet / amount that enters control volume
initial	starting condition
load	heat transfer between recirculator loop and buffer tank
out	outlet of magnet / amount that exits control volume
ref	reference condition
s	exit of pump if the pump were isentropic
t	current time step
t-Δt	previous time step
v	vapor
x	node at location x
x+Δx	next node after node at location x
x-Δx	node immediately before node at location x

1. INTRODUCTION

This chapter is divided into three sections. Section 1 contains a general description of a superconducting magnetic energy storage (SMES) system and the motivation behind this project. Section 2 describes the various interactions between the SMES magnet and refrigeration system. Section 3 describes the areas of the SMES system on which this study has focused.

1.1 BACKGROUND AND MOTIVATION BEHIND THIS PROJECT

Superconducting magnetic energy storage (SMES) systems offer a way to directly store electrical energy. An electromagnet made of superconducting material has essentially no electrical resistance at low temperatures. By connecting the two ends of the magnet together, a continuous loop can be made. Electrical energy put into this loop will circulate nearly indefinitely. Therefore, energy can be stored in the magnet to be immediately available for later use.

SMES systems are most suited for applications where a large amount of energy must be available on short notice. Applications for the power industry include helping to control frequency variations in power grids and providing extra energy to the grid during peak load times. Naval applications include electromagnetic aircraft launching (EMAL), electrodynamic arresting gear, auxiliary power units, and emergency power units.

SMES systems contain two major subsystems: the magnet and the refrigeration system. As mentioned, the magnet is used for storing the electrical energy. The refrigeration system is needed to ensure that the magnet remains cold enough to superconduct. In a very small system, the magnet could be cooled using liquid helium brought in from elsewhere and replenished as needed. However, the size of most real-life systems makes it both impractical and inefficient to cool in this manner. Therefore, most SMES systems include a cryogenic refrigerator to continuously supply cold helium to the magnet and re-cool the warm helium that exits from the magnet.

By considering the interaction of the magnet and refrigeration system, the overall performance of the SMES system may be improved. The SMES system could possibly be made smaller, more efficient, and more cost effective.

The interaction between the magnet and the refrigeration system has not been fully studied and is not completely understood. Previous SMES magnets were designed without full consideration of how the magnet design could be altered to better match the refrigeration system. Likewise, only a small emphasis has been placed on the task of devising a refrigeration system specifically matched to a SMES system.

This project was undertaken to study the interactions between SMES magnets and their refrigeration systems. The goals that were set for the project are:

1. To develop a methodology for the design of refrigeration systems for SMES that are fully integrated with the design of the complete SMES system

2. To define refrigeration system configurations that are most suitable for integration into various SMES magnet systems.

3. To identify refrigeration system components that offer the largest payoff in system performance for an investment in additional component development.

4. To develop ideas for improving refrigeration system components that offer large payoff for a SMES system.

1.2 COOLING OF SMES SYSTEMS WITH HELIUM REFRIGERATION

1.2.1 MAGNET/REFRIGERATOR INTERACTIONS

A SMES refrigeration system interacts with the magnet in several ways. These interactions are designed to keep the magnet cold while using a minimal amount of refrigeration power. The primary interaction is the direct cooling of the magnet's superconducting coils. This cooling can be done in several ways, but usually involves cold helium in direct contact with the superconductor. The secondary magnet/refrigerator interactions minimize the heat that leaks in from the outside environment. These secondary loads include cooling for current leads, heat shields, and structural supports.

The direct cooling of the superconductor in the magnet can be done using either pool cooling or forced flow. A pool cooled magnet is entirely submerged in a pool of liquid helium. Heat that is generated inside the magnet or that leaks in from the environment is rejected to the pool and causes the liquid helium to boil. The vapor that is produced is sent to the refrigeration system to be re-liquefied and is then returned to the pool. Forced flow cooling is typically used with magnets made from cable in conduit conductors (CICC). The superconducting cable is made of many strands of superconductor placed inside a conduit. Cold helium is forced through the conduit, where it flows in direct contact with the numerous superconducting strands.

Cooling of the current leads, heat shields, and structural supports are important to minimize the heat leak into the magnet. The current leads between the cold magnet and the room temperature electronics can be a source for a large heat leak. The high thermal conductivity of the metal in the current leads can create a large heat leak if the leads are not cooled. The most common lead cooling method uses some of the liquid helium produced by the refrigerator. The liquid is forced into the leads at the magnet side and exits toward the room temperature side, where it is returned to the refrigerator. Cooling of the heat shields is done to intercept some of the radiation heat leak into the magnet. One or more shields are used to provide intermediate temperature surfaces between room temperature and the magnet temperature. These shields can be cooled by the boil-off vapor or can be specifically matched to intermediate temperature stages in the refrigeration system. Like the current leads, the structural supports are cooled to minimize heat conduction into the magnet. However, they are usually cooled at several discrete locations in a manner similar to the heat shields.

1.2.2 TYPICAL HELIUM REFRIGERATION SYSTEM

Figure 1.1 shows a possible refrigeration system for cooling a SMES magnet. The system produces low temperature helium to provide the cooling. At the top of the figure is the compressor, which is used to circulate helium throughout the refrigeration system. High pressure helium exits the compressor and is cooled through a series of regenerative heat exchangers until it is about 5 K at the exit of the last heat exchanger. The helium is then expanded to atmospheric pressure to produce liquid helium for cooling the magnet. Some of the liquid can be taken out to cool the current leads and then returned to the compressor. The remaining helium is sent back through the heat exchangers where it cools the high pressure stream.

This particular refrigerator is broken up into several modules. The three types of modules are the compressor, the expansion loop modules, and the lowest temperature module. The compressor provides the power to circulate the helium. In each expansion loop module, some of the high pressure stream is diverted to the low pressure side through an expander. This extra mass increases the heat capacity of the low pressure side to better balance the temperature changes in the heat exchangers. This diverted helium can also be used for cooling the secondary loads such as the heat shields and structural supports (heat loads Q'_1, Q'_2, and Q'_3). The lowest temperature module or "cold end" is where the refrigerator interacts with the magnet. This section of the refrigeration system is crucial for determining the overall performance of the system.

From a refrigeration standpoint, the lowest temperature areas of the system can be the most important. A refrigerator can be thought of as a device for removing entropy from a cold system and rejecting it to a warm environment. The inefficiencies of poorly designed systems will generate extra entropy which must also be removed. Any extra entropy generated at the cold end of the refrigerator must be processed through the entire refrigerator before being rejected to the environment [Smith]. A refrigerator with a poorly designed low temperature section will require larger components and more power consumption. Therefore, improving the cold end can reduce the size and operating cost of the entire refrigeration system.

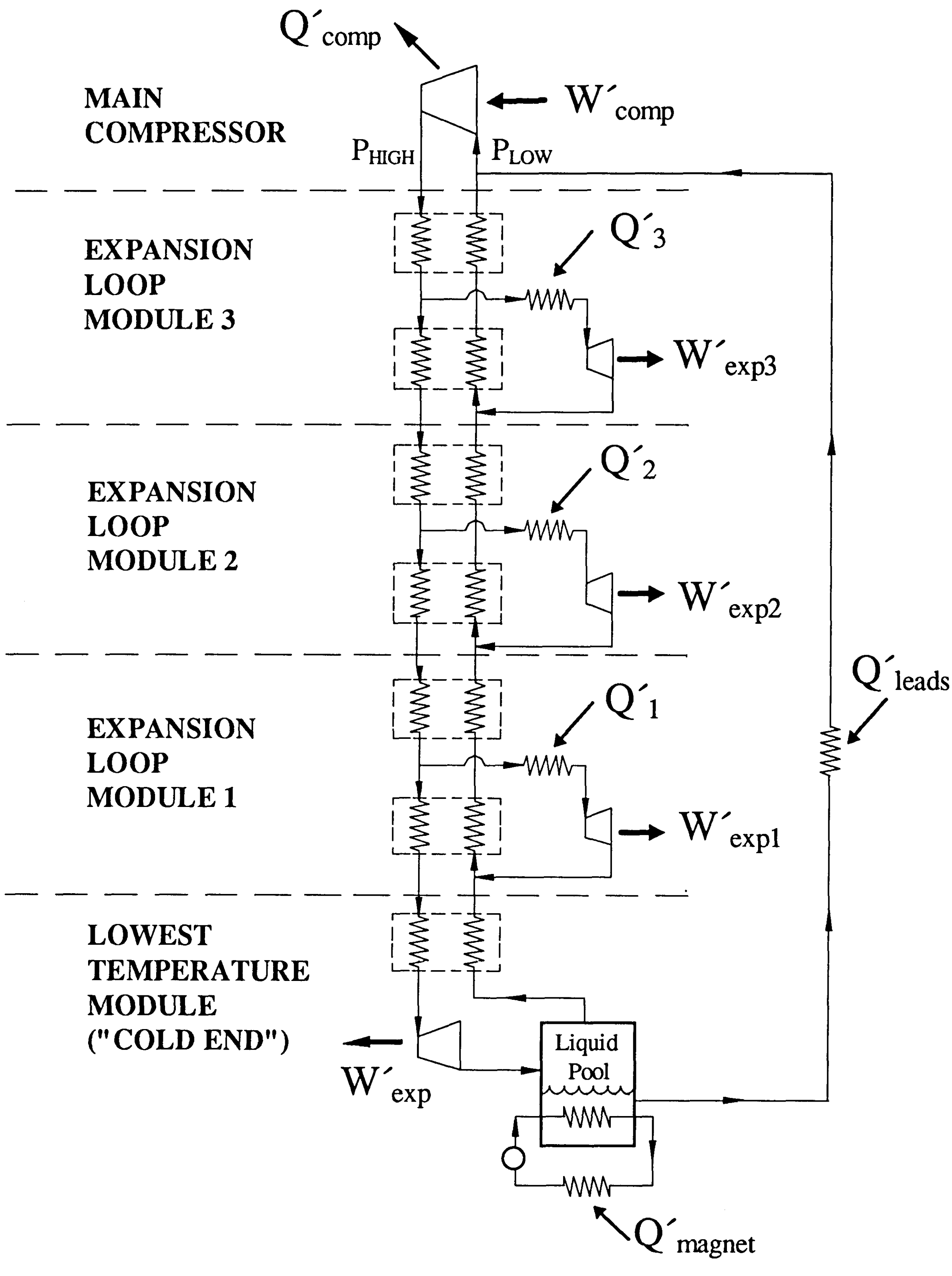


Figure 1.1: A Possible SMES Refrigeration System

1.3 FOCUS OF STUDY

The remaining chapters address issues related to the cold end of SMES refrigeration systems. Due to the interest in CICC magnets, the study focuses on systems using forced flow cooling. Chapter 2 models the steady state flow of cold helium through a CICC. Chapter 3 looks at ways to reduce the pumping power that is required for forced flow cooling. Chapter 4 compares various buffer systems, which are used to temporarily store the extra energy dissipated during magnet discharges. Chapter 5 describes a possible cold end cooling configuration and presents an analysis of its transient behavior. Chapter 6 lists the major conclusions and recommendations.

2. STEADY STATE MODELING OF HELIUM FLOW IN A CABLE-IN-CONDUIT CONDUCTOR

This chapter contains seven sections. Section 1 presents reasons for studying the steady state flow of helium through a cable-in-conduit conductor (CICC). Section 2 shows the derivation of the governing equations. Section 3 explains the finite difference modeling technique that was used to obtain results. Section 4 presents and discusses sample results from a particular CICC passage. Section 5 develops similarity relations to generalize the results and then plots the generalized results. Section 6 contains conclusions.

2.1 MOTIVATION BEHIND THE STEADY STATE MODELING

During the operation of a typical SMES system, a significant portion of the time is spent in a steady state mode. This mode occurs when the magnet is fully charged and is storing energy until needed. To reduce the operating cost, the refrigeration power required during these periods of steady state operation should be minimized. Therefore it is important to understand the parameters which affect the steady state cooling of the magnet.

In a CICC magnet, the flow passages of the magnet are located in the low temperature area of the refrigeration system. Since this is a critical area for determining the refrigerator performance, the flow passages themselves are important to study. Any extra entropy generated in the passages will increase the required size of the entire refrigeration system [Smith]. Therefore, this chapter looks at the steady state flow of cold helium through a long passage.

2.2 DEVELOPMENT OF GOVERNING EQUATIONS FOR FLUID FLOW THOUGH A PASSAGE

The flow of helium through cable in conduit conductors has been studied by numerous people [Long, var der Linde, Wang]. Sophisticated models can include the electrical properties of the conductor and effects that are specific to particular flow passage geometries [Long]. However, a model which focuses on the thermodynamic behavior of the helium is valuable for studying the behavior of the entire refrigeration system. This section develops the differential equations governing flow through a general passage of a given cross sectional area and hydraulic diameter.

Previous studies have also examined the optimum operating conditions of a general CICC flow passage [Katheder, Hale]. However, these studies were done using averaged fluid properties and pressure drop parameters. This study differs by using a differential model of the CICC and recalculating fluid properties along the length of the conductor. Since the properties of helium vary significantly around the range of interest, the differential model could be more accurate than a model with averaged properties.

The governing equations of flow through a long passage are easily found. The starting point is the first law of thermodynamics, which is written for an open system control volume as:

$$E' = Q' - W' + \sum m'_{in} h_{in} - \sum m'_{out} h_{out} \qquad (2.1)$$

where E' is the rate of change of the energy in the control volume, Q' is the rate of heat addition, W' is the rate of work removal, m' is the mass flow rate, and h is the specific enthalpy.

Figure 2.1 shows the steady state energy fluxes for a differential element of helium flowing in a long passage. The element has length Δx. It is in steady state, so there is no change in energy per time ($E'=0$). Therefore the energy balance equation is given by:

$$0 = q' \cdot \Delta x - w' \cdot \Delta x - m' \cdot \frac{d}{dx}\left(h + \frac{v^2}{2} + g \cdot z\right) \cdot \Delta x \qquad (2.2)$$

where q' is the rate of heat addition per unit length and w' is the rate of work removal per unit length. The mass flow rate (m') is constant due to the steady state assumption. The energy change of the fluid includes terms for enthalpy (h), velocity (v) and elevation (z). For this study, the heat load q' was assumed to be uniform and constant throughout the conductor.

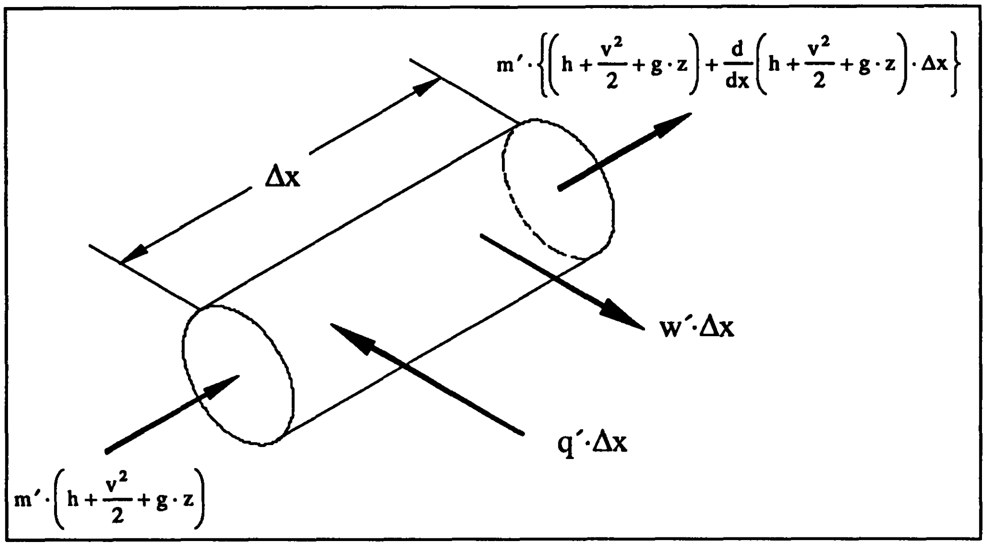


Figure 2.1: Energy Flux for a Differential Element of Helium Flow

Ignoring the gravitational potential energy and noting that there is no work transfer to the control volume, equation 2.2 reduces to:

$$0 = q' - m' \cdot \left[\frac{dh}{dx} + \frac{d}{dx} \left(\frac{v^2}{2} \right) \right] \tag{2.3}$$

The next step is to express the change in enthalpy per unit length in terms of the changes of pressure and temperature per unit length. A useful equation for this is the differential substitution rule [Cravalho, 289], which relates the change in a variable to the changes in other variables it depends on. Here c is considered to depend on the two variables a and b:

$$dc = \left. \frac{\partial c}{\partial a} \right|_b da + \left. \frac{\partial c}{\partial b} \right|_a db \tag{2.4a}$$

OR

$$\frac{dc}{dx} = \left. \frac{\partial c}{\partial a} \right|_b \frac{da}{dx} + \left. \frac{\partial c}{\partial b} \right|_a \frac{db}{dx} \tag{2.4b}$$

where the subscript on the vertical bar indicates that the subscripted variable is held constant while the partial derivative is taken.

Any thermodynamic property can be considered to be a function of two other properties. Since temperature and pressure are of the most interest, all property changes will be related to changes of temperature and pressure. Therefore, equation 2.4b is used to express the change in enthalpy per unit length as:

$$\frac{dh}{dx}=\left.\frac{\partial h}{\partial T}\right|_P \frac{dT}{dx}+\left.\frac{\partial h}{\partial P}\right|_T \frac{dP}{dx} \tag{2.5}$$

The first partial derivative in equation 2.5 is recognized to be the definition of specific heat at constant pressure (Cp).

$$Cp=\left.\frac{\partial h}{\partial T}\right|_P \tag{2.6}$$

Substituting equating 2.6 into equation 2.5 gives:

$$\frac{dh}{dx}=Cp\frac{dT}{dx}+\left.\frac{\partial h}{\partial P}\right|_T \frac{dP}{dx} \tag{2.7}$$

Referring back to equation 2.3 shows that the kinetic energy term is also needed. Although, this term will be negligible for most cases, it is included here for completeness. Once again, the goal is to relate the changes in this property to changes in temperature and pressure. First, the velocity is related to the mass flow rate (m'), density (ρ), and cross sectional area of the passage (A):

$$v=\frac{m'}{\rho \cdot A} \tag{2.8}$$

Since the mass flow rate and the cross sectional area are constant, the change in specific kinetic energy per unit length is:

$$\frac{d}{dx}\left(\frac{v^2}{2}\right)=\frac{1}{2}\frac{m'^2}{A^2}\frac{d}{dx}\left(\frac{1}{\rho^2}\right)=\frac{-m'^2}{\rho^3 \cdot A^2}\frac{d\rho}{dx} \tag{2.9}$$

Using equation 2.4b, the density derivative is:

$$\frac{d\rho}{dx}=\left.\frac{\partial \rho}{\partial T}\right|_P \frac{dT}{dx}+\left.\frac{\partial \rho}{\partial P}\right|_T \frac{dP}{dx} \tag{2.10}$$

Substituting equation 2.10 into equation 2.9, the change in specific kinetic energy per unit length of flow passage is:

$$\frac{d}{dx}\left(\frac{v^2}{2}\right)=\frac{-m'^2}{\rho^3 \cdot A^2}\left[\left.\frac{\partial \rho}{\partial T}\right|_P \frac{dT}{dx}+\left.\frac{\partial \rho}{\partial P}\right|_T \frac{dP}{dx}\right] \tag{2.11}$$

Substituting equations 2.7 and 2.11 in equation 2.3 and rearranging gives the change in temperature per unit length to be:

$$\frac{dT}{dx}=\frac{\frac{q'}{m'}+\left[\frac{m'^2}{\rho^3\cdot A^2}\frac{\partial\rho}{\partial P}\bigg|_T-\frac{\partial h}{\partial P}\bigg|_T\right]\cdot\frac{dP}{dx}}{Cp-\frac{m'^2}{\rho^3\cdot A^2}\frac{\partial\rho}{\partial T}\bigg|_P} \tag{2.12a}$$

In cases where the kinetic energy term in equation 2.3 can be neglected, the temperature change can be further simplified to be:

$$\frac{dT}{dx}=\frac{q'}{m'\cdot Cp}-\frac{1}{Cp}\frac{\partial h}{\partial P}\bigg|_T\cdot\frac{dP}{dx} \tag{2.12b}$$

Equations 2.12 show that the change in pressure per unit length is needed. Assuming the flow can be represented by flow through a pipe, the pressure change along the passage is given by:

$$\frac{dP}{dx}=\frac{-f\cdot\rho\cdot v^2}{2\cdot D_h}=\frac{-f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2} \tag{2.13}$$

where f is the friction factor, ρ is the fluid density, D_h is the hydraulic diameter, and A is the cross sectional area of the flow passage.

Substituting equation 2.13 into equations 2.12, the change in temperature per unit length is:

$$\frac{dT}{dx}=\frac{\frac{q'}{m'}+\left[\frac{\partial h}{\partial P}\bigg|_T-\frac{m'^2}{\rho^3\cdot A^2}\frac{\partial\rho}{\partial P}\bigg|_T\right]\cdot\frac{f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2}}{Cp-\frac{m'^2}{\rho^3\cdot A^2}\frac{\partial\rho}{\partial T}\bigg|_P} \tag{2.14a}$$

And if the change in kinetic energy is negligible, equation 2.12b becomes:

$$\frac{dT}{dx}=\frac{q'}{m'\cdot Cp}+\cdot\frac{f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2\cdot Cp}\cdot\frac{\partial h}{\partial P}\bigg|_T \tag{2.14b}$$

Therefore, the governing equations for steady state flow in a passage are:

$$\frac{dT}{dx}=\frac{\dfrac{q'}{m'}+\left[\left.\dfrac{\partial h}{\partial P}\right|_T-\dfrac{m'^2}{\rho^3\cdot A^2}\left.\dfrac{\partial \rho}{\partial P}\right|_T\right]\cdot\dfrac{f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2}}{Cp-\dfrac{m'^2}{\rho^3\cdot A^2}\left.\dfrac{\partial\rho}{\partial T}\right|_P}$$

$$\frac{dP}{dx}=\frac{-f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2} \tag{2.15}$$

where equation 2.14b can be used in place of this dT/dx equation if the change in specific kinetic energy per length is small compared to the change in enthalpy per unit length.

Equations 2.15 can be used to create a finite difference model for finding the conditions throughout a flow passage. Appendix A shows how the property derivatives can be expressed in terms of the derivatives in the available computer software [HEPROP]. These relations allow equations 2.15 to be rewritten as:

$$\frac{dT}{dx}=\frac{\dfrac{q'}{m'}+\left[\dfrac{1}{\rho}-\dfrac{T}{\rho^2}\cdot\dfrac{\left.\frac{\partial P}{\partial T}\right|_\rho}{\left.\frac{\partial P}{\partial \rho}\right|_T}-\dfrac{m'^2}{\rho^3\cdot A^2}\dfrac{1}{\left.\frac{\partial P}{\partial \rho}\right|_T}\right]\cdot\dfrac{f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2}}{\left[Cp+\dfrac{m'^2}{\rho^3\cdot A^2}\dfrac{\left.\frac{\partial P}{\partial T}\right|_\rho}{\left.\frac{\partial P}{\partial \rho}\right|_T}\right]}$$

$$\frac{dP}{dx}=\frac{-f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2} \tag{2.16}$$

And in cases where the change in kinetic energy per unit length is negligible, dT/dx can be written as:

$$\frac{dT}{dx}=\frac{q'}{m'\cdot Cp}+\frac{f\cdot m'^2}{2\cdot D_h\cdot\rho\cdot A^2\cdot Cp}\cdot\left[\frac{1}{\rho}-\frac{T}{\rho^2}\cdot\frac{\left.\frac{\partial P}{\partial T}\right|_\rho}{\left.\frac{\partial P}{\partial \rho}\right|_T}\right] \tag{2.17}$$

2.3 FINITE DIFFERENCE APPROXIMATION OF GOVERNING EQUATIONS FOR USE IN COMPUTER SIMULATION

Using the equations for dP/dx and dT/dx, a finite difference model can be created. This model looks at the conditions throughout the flow passage at many locations. Each location is considered to be a "node". As the nodes are placed closer and closer together, the derivatives of pressure and temperature can be approximated by the forward Euler finite difference equations [Strang, 562]:

$$\frac{dT}{dx} \cong \frac{T_{x+\Delta x} - T_x}{\Delta x}$$
$$\frac{dP}{dx} \cong \frac{P_{x+\Delta x} - P_x}{\Delta x} \qquad (2.18)$$

where the subscript x denotes the node at location x and the subscript x+Δx denotes the next node, which is a distance Δx away from the first node. Note that these equations can be formally obtained by expanding $T_{x+\Delta x}$ and $P_{x+\Delta x}$ into Taylor series and ignoring the terms of 2nd order or higher:

$$T_{x+\Delta x} = T_x + \left(\frac{dT}{dx}\right)_x \Delta x + \left(\frac{d^2T}{dx^2}\right)_x \frac{\Delta x^2}{2!} + \ldots\ldots + \left(\frac{d^nT}{dx^n}\right)_x \frac{\Delta x^n}{n!}$$
$$P_{x+\Delta x} = P_x + \left(\frac{dP}{dx}\right)_x \Delta x + \left(\frac{d^2P}{dx^2}\right)_x \frac{\Delta x^2}{2!} + \ldots\ldots + \left(\frac{d^nP}{dx^n}\right)_x \frac{\Delta x^n}{n!} \qquad (2.19)$$

For small values of Δx, higher order terms become negligible and equations 2.19 reduce to equations 2.18.

If the pressure and temperature at a given node are known, all properties at that node can be found. The derivatives dT/dx and dP/dx can then be found using equations 2.16. Rearranging equations 2.18 shows a way to find the temperature and pressure at the next node:

$$T_{x+\Delta x} = T_x + \frac{dT}{dx}\Delta x$$
$$P_{x+\Delta x} = P_x + \frac{dP}{dx}\Delta x \qquad (2.20)$$

Using equations 2.20, the temperature and pressure throughout a flow passage can be found for a given heat load and mass flow rate. The conductor is first divided up to place nodes from inlet to outlet. Starting with a known inlet temperature and pressure, equations 2.16 and 2.20 are used to find the temperature and pressure at the second node. All properties can then be found at the second node. Similarly, the conditions are found for the third node and all remaining nodes. Along the way, checks are made to ensure that the temperature never rises above the maximum allowable temperature and that the pressure remains above 3 atm. The value of the temperature constraint is chosen based on stability requirements. The 3 atm pressure constraint is chosen to ensure that there is no two-phase helium in the conductor. If either condition is violated, the mass flow rate or the heat load can be varied until a valid combination is found.

Using the procedure described above, a computer program was written to find the conditions throughout a flow passage. The program was written in FORTRAN and is named *CICC.EXE*. A listing of the program is included in Appendix B.

2.4 SAMPLE RESULTS OF A SPECIFIC HELIUM FLOW PASSAGE

Using the program *CICC.EXE*, the conditions in a particular flow passage were evaluated. The following assumptions were used:

- The heat addition to the helium (q′) is constant throughout the flow passage.
- A constant friction factor is assumed due to scatter of experimental data [Katheder]. This represents an average value over most flow conditions.
- No thermodynamic interaction between flow passages in adjacent turns of the magnet is considered.
- The heat transfer coefficient between the helium and the superconductor is assumed to be large due to the large contact area. Therefore, the conductor is assumed to be at the same temperature as the surrounding fluid.

In addition, the dimensions and constraints listed in Table 2.1 were used. These values were also used in a study that performed a similar analysis using averaged fluid properties [Hale]. The same values were used to allow a comparison and a check of the results.

A brief note on the friction factor is appropriate. Studies of flow in CICC flow passages have measured the friction factor for certain conductors [Katheder]. While correlations between the friction factor and the Reynolds number have been found, they are not particularly accurate. Due to the large fluctuation in the measured friction factor and the uncertainty of the correlations, an average value of 0.15 is suggested by Katheder for $6000 < Re < 15000$. Therefore, this study assumes a constant friction factor throughout the conductor flow passage.

Length (L)	Hydraulic Diameter (D_h)	Flow Area (A)	Friction Factor (f)	Inlet Temperature (T_{max})	Minimum Allowable Pressure (P_{min})
1140 m	5.40×10^{-4} m	1.16×10^{-4} m^2	0.15	4.5 K	3 atm

Table 2.1: CICC Flow Passage Dimensions and Constraints That Were Used

Using the assumptions stated above and the values of Table 2.1, the program *CICC.EXE* was used to study the helium flow through the passage. The program searches for the maximum heat load that can be removed for given inlet conditions. To start, relatively low values are chosen for the heat load (q′) and the mass flow rate (m′). The conditions are then found throughout the conductor. If the temperature at any location is too high, the mass flow rate is increased until the temperatures are acceptable. The heat load is then increased slightly and the new mass flow rate is found. This procedure is repeated until the heat load cannot be increased without violating either the maximum temperature or minimum pressure constraint. Figure 2.2 shows the results of this study by plotting the maximum allowable heat load for various inlet pressures and maximum allowable temperatures.

Figure 2.2 shows that the maximum heat removal occurs at a distinct inlet pressure for each maximum allowable temperature. In addition, each curve has a "kink" at higher inlet pressures. These kinks mark the point where the flow switches from being able to fully use both the temperature and pressure constraints to a point where only the temperature constraint can be fully used. However, this feature is irrelevant since it is undesirable to operate to the right side of the peak heat flux.

The lowest inlet pressure that can safely remove the heat load should be used. For example if 20 W must be removed with a maximum temperature of 6.0 K, it would be better to use an inlet pressure of 4.5 atm even though 11.5 atm would also work. The reason to use the lower inlet pressure is that the higher inlet pressure will have a higher pressure drop (assuming the same minimum pressure is occurring at the outlet). The higher pressure drop implies a larger mass flow rate and larger pumping power. It then makes sense to use the lower inlet pressure since it would have a lower pumping power requirement.

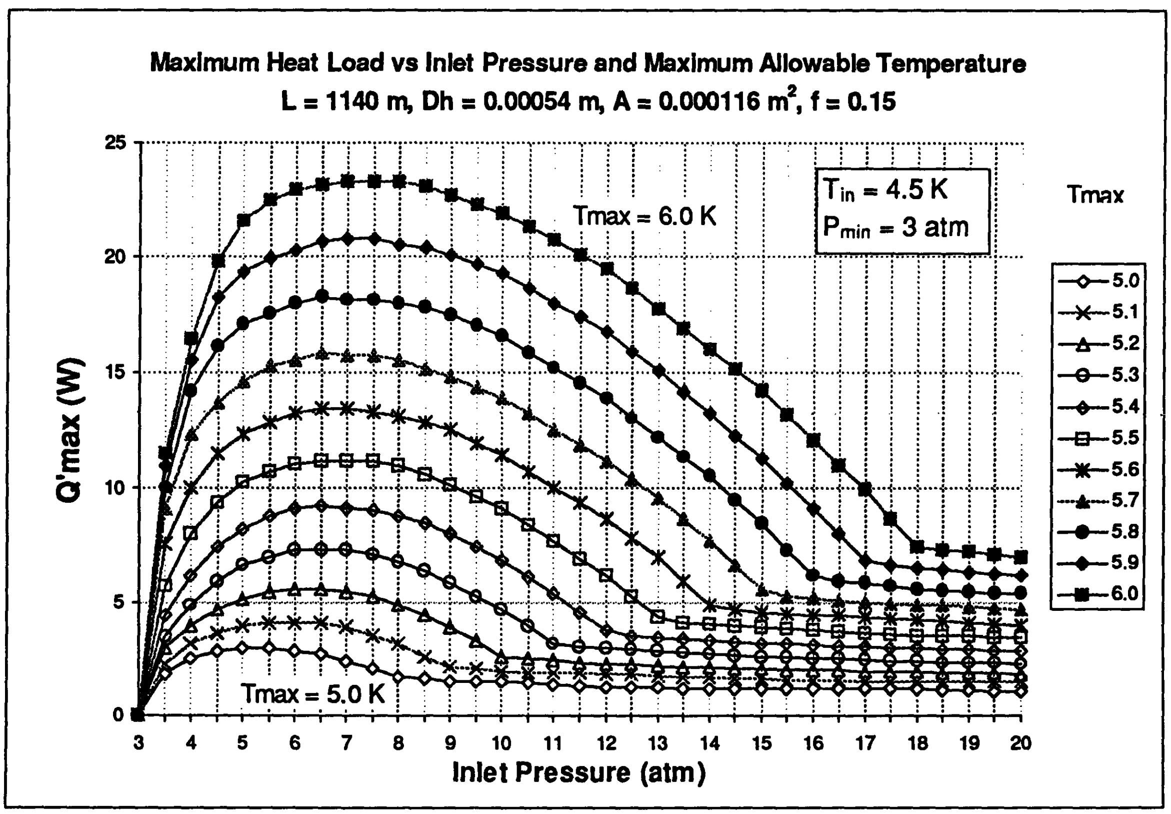


FIGURE 2.2: Maximum Heat Load that Can Be Removed versus Inlet Pressure

Figure 2.2 summarizes the most important results for the modeling of this particular conductor. However, a more detailed investigation can give further insight into the processes which control the amount of heat that the helium can remove. Therefore, a study was done to examine what happens as the mass flow rate through the conductor is varied. This study looked at the amount of heat that can be removed for different mass flow rates instead of only looking at the point of maximum heat removal.

Once again, the dimensions and constraints listed in Table 2.1 were used. The program *CICC.EXE* was used to find the heat load that can be removed for a range of mass flow rates. Figure 2.3 plots the results. Curves are provided for several inlet pressures. For all the curves shown, the maximum allowable temperature was set at 6.0 K. Note that the peak value for each of the curves is the one that was shown in Figure 2.2.

Figure 2.3 shows that Figure 2.2 is slightly misleading. Figure 2.2 suggests that you would always want to run at an inlet pressure of about 7 or 8 atmospheres for a maximum allowable temperature of 6.0 K. While it is true that this would allow the maximum possible removal of heat, it is not necessary if a smaller amount of heat is being removed. Figure 2.3 shows that the lowest inlet pressure that can safely remove the heat should be used. For example, consider a situation where 10 W must be removed from the conductor. By running at an inlet pressure of 4 atm, the heat can be removed with a mass flow rate of about 0.6 g/s. Using an inlet pressure of 8 atm would require a mass flow rate of 1.5 g/s, which is nearly 3 times as much. It would be inefficient to run at an inlet pressure of 8 atm due to the larger pumping power required to provide the higher mass flow rate.

Even though low inlet pressures should be used, a word of caution is required. The sharp drop-off of the curves in Figure 2.3 indicates a possible danger of running too close to the peak heat flux. For example, Figure 2.3 shows an inlet pressure of 4 atm could be used to remove 15 W from the conductor, However, if unforeseen circumstances cause the heat load to rise to 18 W, the 4 atm system could not remove the heat without violating the temperature or pressure constraints. Therefore, it is important to consider the maximum steady state flux that might have to be removed when choosing the steady state inlet pressure.

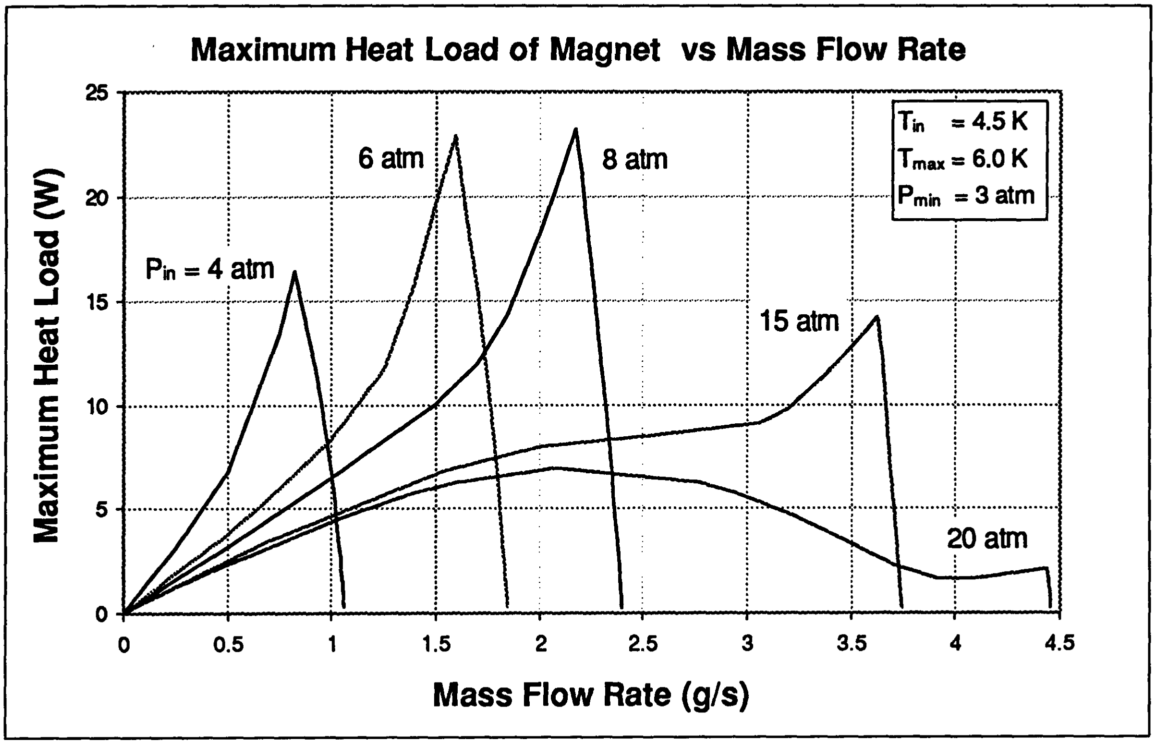


Figure 2.3: Maximum Heat Load that Can Be Removed versus Mass Flow Rate

The curves of Figure 2.3 all show a sudden drop. This drop can be explained by considering a pressure-enthalpy diagram. Figure 2.4 shows the conditions throughout the flow passage for several mass flow rates with inlet conditions of 4.5 K and 20 atm. Figure 2.5 shows similar data for an inlet pressure of 8 atm. For each case, the maximum allowable temperature is 6.0 K and the minimum allowable pressure is 3 atm. On these P-h diagrams, the lines of constant temperature bend due to the Joule-Thomson effect. The particular way that these lines bend is what limits the amount of heat that can be removed from the flow passage.

For example, consider the curves for the 20 atm inlet condition of Figure 2.4. At low mass flow rates (curve a), the pressure drop is relatively low. Therefore, the limiting constraint is the maximum temperature. For this case, the heat load can be adjusted until the outlet temperature is exactly on the 6.0 K line. As the mass flow rate is increased (curves b through d), the pressure drop becomes larger and the outlet condition moves down the 6.0 K line. However, the slope of the 6.0 K line causes the outlet enthalpy to become closer to the inlet enthalpy. Integration of equation 2.3 shows that the heat removed from the passage is:

$$
\begin{aligned}
Q' &= q' \cdot L = m' \cdot \left(h_{out} - h_{in}\right) \\
Q' &= m' \cdot \Delta h
\end{aligned}
\tag{2.21}
$$

where the kinetic energy term of equation 2.3 was found to be negligible and is omitted.

As the mass flow rate is increased, the Δh term decreases. So a tradeoff develops to find the maximum value of the product $m' \cdot \Delta h$. For this particular case, curve (a) has a large Δh but a low m'. Curve (b) has a smaller Δh but a larger m' that allows a larger heat removal. Curve (c) has such a small Δh that even its large m' cannot make the product $m' \cdot \Delta h$ larger than that of curve (b). Finally curve (d) has the largest mass flow rate but is incapable of removing any heat due a Δh of zero.

Figure 2.5 shows results for inlet conditions of 8 atm and 4.5 K. The important difference between this inlet condition and the 20 atm case is that the bend in the 6.0 K line is now used for an advantage. Like curve (a) of Figure 2.4, curve (e) has a relatively low mass flow rate. But this time the Δh is larger due to the divergence of the constant temperature lines. Increasing the mass flow rate slightly would allow the exit condition to move down the 6.0 K line. However, the sharp turn in the 6.0 K line allows curve (f) to barely touch the 6.0 K line and then continue on until it hits the 3 atm minimum pressure constraint. This allows a much larger Δh along with the increased mass flow rate. Attempting to further increase the mass flow rate without lowering the heat load would cause the pressure drop to be too large. Therefore, curve (g) has a large mass flow rate, but a reduced Δh since the pressure constraint is now the limiting constraint. This transition from simultaneously satisfying the temperature and pressure constraints (curve f) to only being able to use the pressure constraint (curves g and h) is the reason for the sharp drop-offs shown back in Figure 2.3.

The paths shown on Figures 2.3 and 2.5 are very similar to those shown by Katheder. In that study, the paths of maximum heat load were taken as straight lines. The lines were drawn from the inlet condition to a point on the minimum pressure line along a path that just barely touches the maximum temperature (similar to curve f on Figure 2.5). Figure 2.5 shows that the paths are not always straight lines, but that the straight-line-method would give similar results.

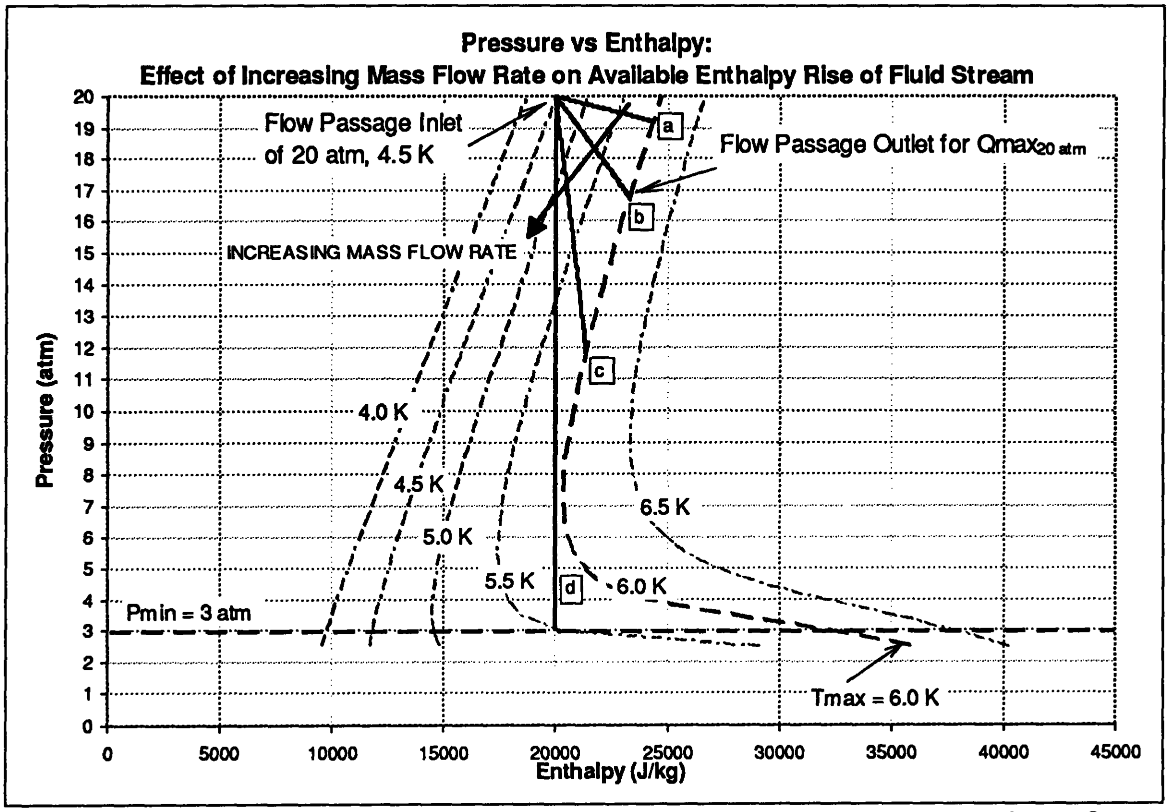


Figure 2.4: Pressure-Enthalpy Diagram for Steady State Helium Flow - Pin = 20 atm

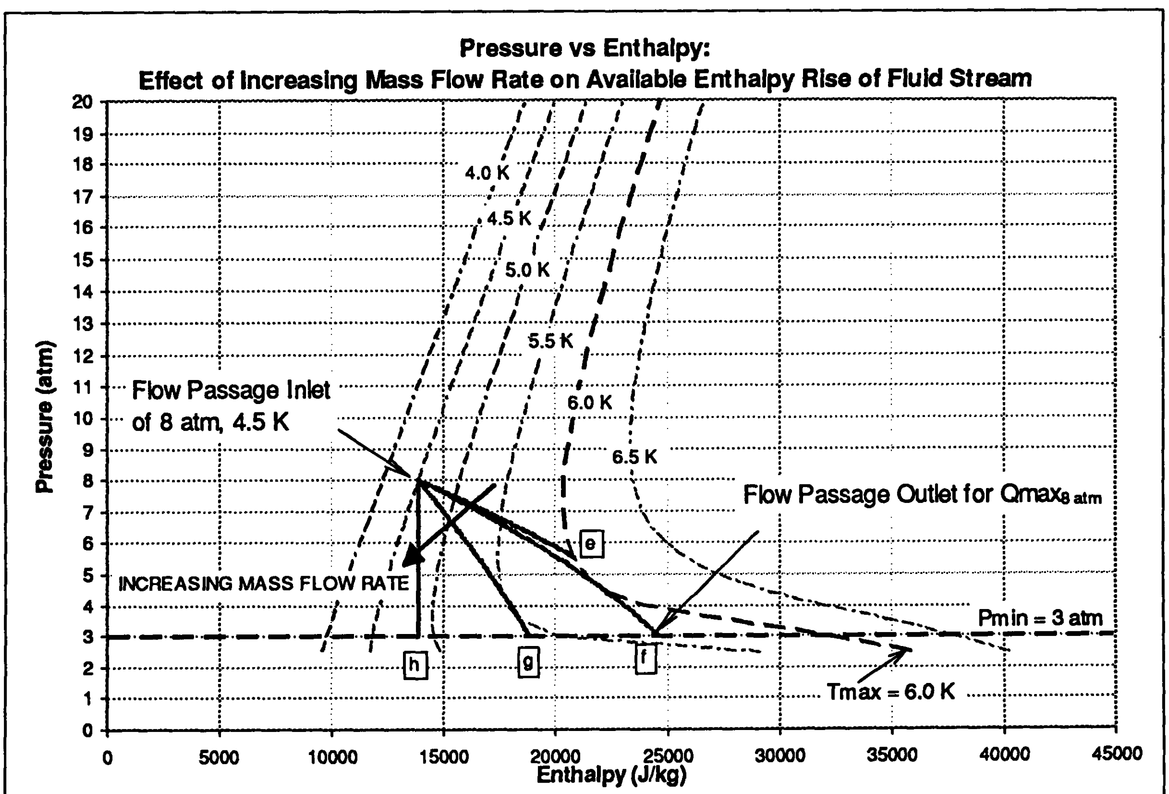


Figure 2.5: Pressure-Enthalpy Diagram for Steady State Helium Flow - Pin = 8 atm

2.5 SIMILARITY BETWEEN FLOW PASSAGES AND GENERALIZATION OF RESULTS

Although section 2.4 showed many useful results, they are limited to a specific flow passage. A more useful set of results would generalize the information for use with a flow passage of any size. Therefore, this section develops a method for determining what conditions make flow passages behave similarly. In this way, the results obtained in section 2.4 can be scaled for use with passages of different sizes.

The first step to finding a scaling method is to notice what makes two different flow passages behave in a similar way. The most obvious answer is that the thermodynamic conditions at the inlet, outlet, and all corresponding points in-between are same in each passage. In other words, the inlet temperature and pressure will be the same in each passage, as will the temperature and pressure half-way through each passage, three-quarters of the way through, at the exit, etc. Therefore, the governing equations are reformulated to use the fraction of the distance along the conductor rather than an absolute length. The dimensionless distance from the conductor inlet (X) is defined as:

$$X = \frac{x}{L} \tag{2.22}$$

where x is the distance from the CICC inlet and L is the total length of the conductor.

In addition, a dimensionless temperature (Θ) and dimensionless pressure (Π) can be defined by using a suitable reference temperature and pressure:

$$\Theta = \frac{T}{T_{ref}}, \qquad \Pi = \frac{P}{P_{ref}} \tag{2.23}$$

Next, equations 2.23 are substituted into equations 2.15 to find the dimensionless changes in pressure and temperature per unit length. Equation 2.12b is used to express dT/dx since the results of section 2.4 indicate that the kinetic energy terms are negligible. Therefore, the dimensionless changes in temperature and pressure per unit length at a given location (X) are:

$$\frac{d\Theta}{dX}(X) = \frac{q' \cdot L}{m' \cdot C_p(X) \cdot T_{ref}} + \frac{P_{ref}}{C_p(X) \cdot T_{ref}} \left[\frac{\Theta(X)}{\rho(X)^2} \cdot \frac{\partial \Pi}{\partial \Theta}(X)\bigg|_{\rho} \cdot \frac{\partial \rho}{\partial \Pi}(X)\bigg|_{\Theta} - \frac{1}{\rho(X)} \right] \frac{d\Pi}{dX}(X) \tag{2.24}$$

$$\frac{d\Pi}{dX}(X) = \frac{-f \cdot L \cdot m'^2}{2 \cdot \rho(X) \cdot D_h \cdot A^2 \cdot P_{ref}} \tag{2.25}$$

where f is the friction factor, D_h is the hydraulic diameter, A is the cross-sectional area, m′ is the mass flow rate, and q′ is the rate of heat transfer per unit length. Note that all fluid properties, including the specific heat (C_p), density (ρ), and property derivatives are functions of the position in the passage (X).

As mentioned, two flow passages will have similar behavior if at each point (X), the two passages have the same dimensionless temperature (Θ) and pressure (Π). To equate the temperature and pressure, equations 2.24 and 2.25 would have to be integrated for each passage. However, we can avoid the integration by noticing that the slopes must also be the same since the distance has been normalized. Thus, for two conductors to have the same temperature-pressure profile it must be true that at any location X:

$$\frac{d\Theta_1}{dX}(X) = \frac{d\Theta_2}{dX}(X) \quad \text{and} \quad \frac{d\Pi_1}{dX}(X) = \frac{d\Pi_2}{dX}(X) \tag{2.26}$$

where 1 denotes the first conductor flow passage and 2 denotes the second flow passage.

Combining equations 2.25 and 2.26 gives:

$$\frac{f_1 \cdot L_1 \cdot m_1'^2}{2 \cdot \rho(X) \cdot D_{h_1} \cdot A_1^2 \cdot P_{ref}} = \frac{f_2 \cdot L_2 \cdot m_2'^2}{2 \cdot \rho(X) \cdot D_{h_2} \cdot A_2^2 \cdot P_{ref}} \tag{2.27}$$

Similarly, equations 2.24 and 2.26 are combined to show:

$$\begin{aligned} &\frac{q_1' \cdot L_1}{m_1' \cdot C_p(X) \cdot T_{ref}} + \frac{P_{ref}}{C_p(X) \cdot T_{ref}} \left[\frac{\Theta(X)}{\rho(X)^2} \cdot \left.\frac{\partial \Pi}{\partial \Theta}(X)\right|_\rho \cdot \left.\frac{\partial \rho}{\partial \Pi}(X)\right|_\Theta - \frac{1}{\rho(X)} \right] \frac{d\Pi_1}{dX}(X) \\ &= \frac{q_2' \cdot L_2}{m_2' \cdot C_p(X) \cdot T_{ref}} + \frac{P_{ref}}{C_p(X) \cdot T_{ref}} \left[\frac{\Theta(X)}{\rho(X)^2} \cdot \left.\frac{\partial \Pi}{\partial \Theta}(X)\right|_\rho \cdot \left.\frac{\partial \rho}{\partial \Pi}(X)\right|_\Theta - \frac{1}{\rho(X)} \right] \frac{d\Pi_2}{dX}(X) \end{aligned} \tag{2.28}$$

It was assumed that the temperature and pressure of each passage are the same at the same value of X. Therefore, all other properties must also be the same. Since the bracketed term on each side of equation 2.28 involves fluid properties, it must be the same for each passage. In addition, equation 2.26 states that the dimensionless pressure drop is the same for each. Therefore, the second term on each side of equation 2.28 is the same and equation 2.28 reduces to:

$$\frac{q_1' \cdot L_1}{m_1' \cdot C_p(X) \cdot T_{ref}} = \frac{q_2' \cdot L_2}{m_2' \cdot C_p(X) \cdot T_{ref}} \tag{2.29}$$

To find an expression which does not include the mass flow rate, equation 2.29 is squared and multiplied by equation 2.27 to obtain:

$$\begin{aligned}&\frac{f_1}{2}\frac{L_1}{D_{h_1}}\left(\frac{q_1'\cdot L_1}{A_1}\right)^2\frac{1}{\rho(X)\cdot C_p(X)^2\cdot T_{ref}{}^2\cdot P_{ref}}\\&\qquad=\frac{f_2}{2}\frac{L_2}{D_{h_2}}\left(\frac{q_2'\cdot L_2}{A_2}\right)^2\frac{1}{\rho(X)\cdot C_p(X)^2\cdot T_{ref}{}^2\cdot P_{ref}}\end{aligned} \tag{2.30}$$

Since the density and specific heat will be the same in each conductor, those terms can be canceled. In addition the same reference conditions will be chosen for each conductor and can also be canceled. Finally, the equation that allows scaling between similar conductors is:

$$f_1\frac{L_1}{D_{h_1}}\left(\frac{q_1'\cdot L_1}{A_1}\right)^2=f_2\frac{L_2}{D_{h_2}}\left(\frac{q_2'\cdot L_2}{A_2}\right)^2 \tag{2.31}$$

Note that this equation is independent of fluid properties and position. Therefore, **the dimensional group of equation 2.31 must be a constant throughout the entire conductor**. Conductors with the same inlet conditions and the same temperature and pressure constraints will be related by equation 2.31. A more convenient form of this relationship is to note that this value is a constant (C) which depends on the inlet conditions and the constraints:

$$f\frac{L}{D_h}\left(\frac{q'\cdot L}{A}\right)^2=C(P_{in},T_{in},P_{min},T_{max}) \tag{2.32}$$

The most significant result of this relationship is that only one CICC geometry must be evaluated in detail. Once the value of the flow constant C is found for all inlet pressures and constraints of interest, any other geometry can be quickly evaluated. Of course, the value of C could be found for any valid heat load, but the value at the maximum heat load is of the most interest. Using the data from section 2.4, the value of the flow constant at the maximum heat load was found for an inlet temperature of 4.5 K and various inlet pressures. The results are shown graphically in Figures 2.6 and 2.7.

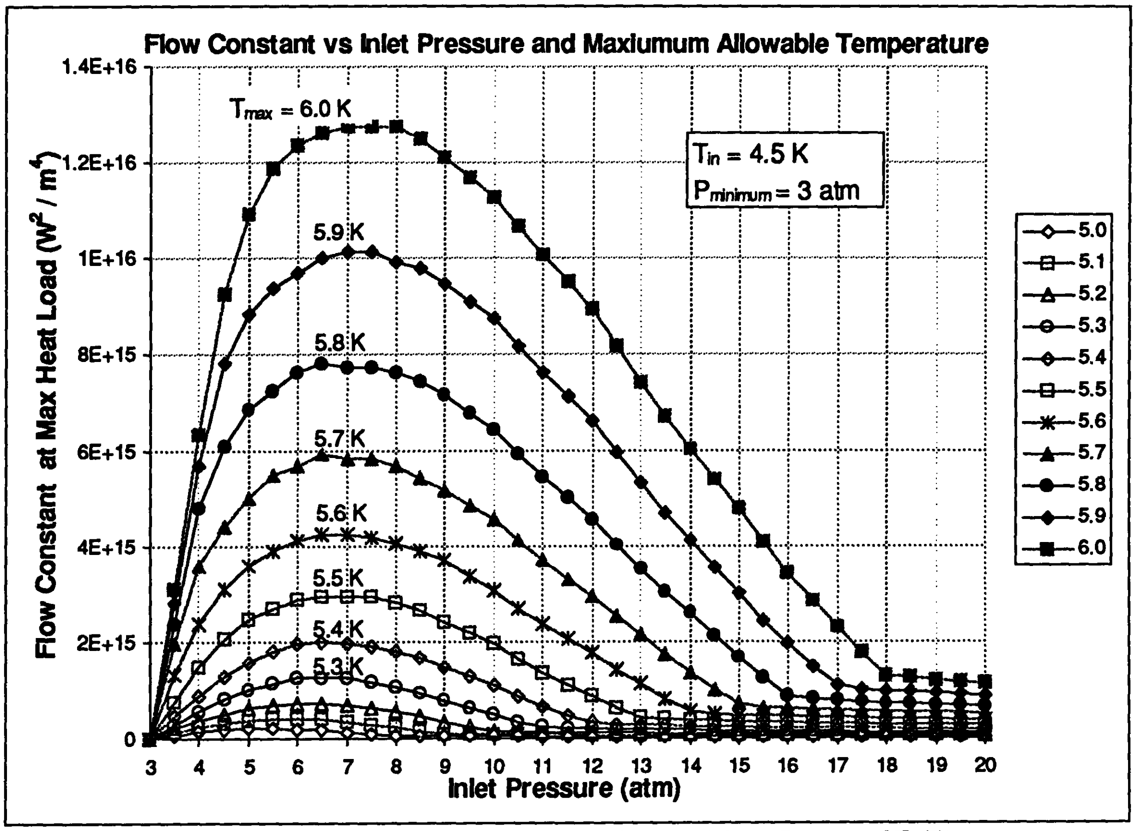


Figure 2.6: Flow Constant at Maximum Heat Load - 2D Plot

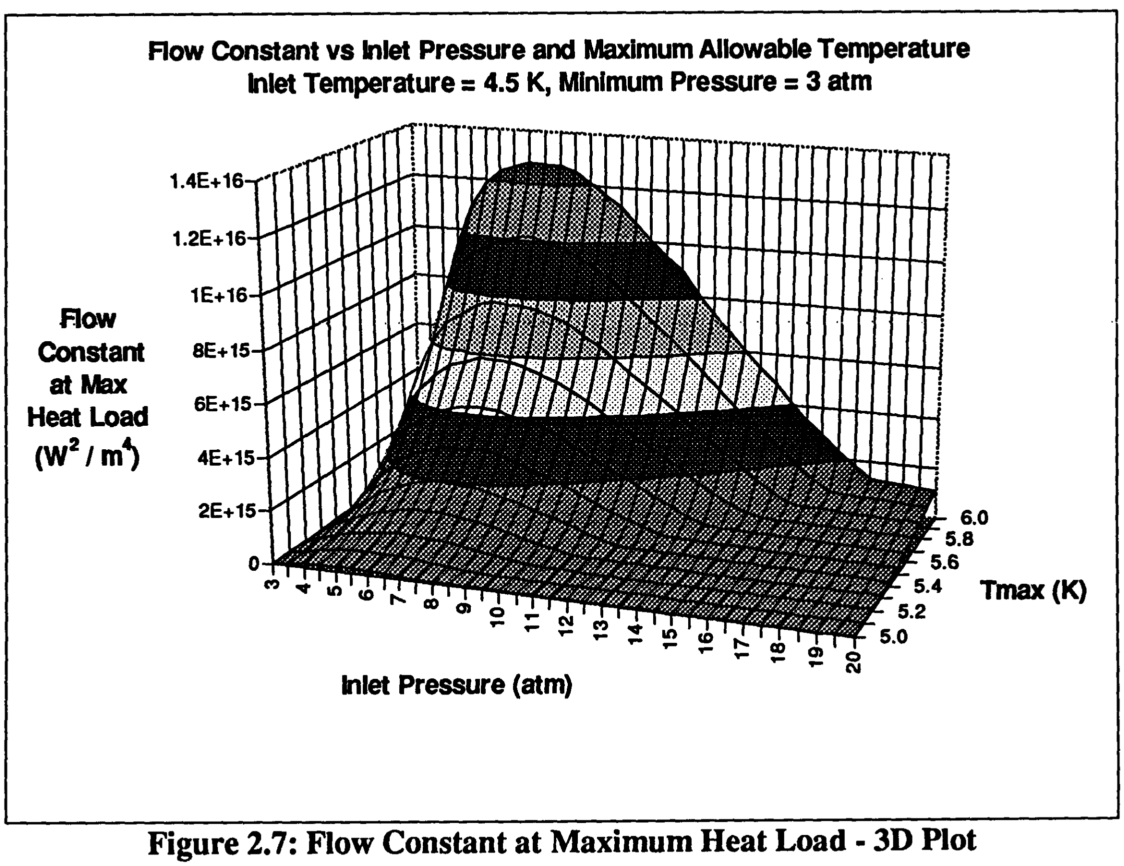


Figure 2.7: Flow Constant at Maximum Heat Load - 3D Plot

Equation 2.32 can be rearranged to make several important points. Equation 2.33 shows how the flow passage dimensions affect the amount of heat that the helium can remove per unit length. A larger heat load can be removed by increasing the flow area and hydraulic diameter, and by decreasing the friction factor and length of the conductor. The length has a greater influence than that area or hydraulic diameter since it is raised to the negative 1.5 power. For example, reducing the length by 50% will increase the possible heat load per unit length by a factor of 2.8.

$$\boxed{q' = \frac{A}{L}\sqrt{\frac{1}{f}\frac{D_h}{L}C(P_{in}, T_{in}, P_{min}, T_{max})}} \tag{2.33}$$

Each point on Figures 2.6 and 2.7 represent a passage with a specific dimensionless temperature and pressure profile. Passages with the same flow constant will have the same fluid properties at the same dimensionless distance through the passage. Therefore, the paths on a pressure-enthalpy diagram will be the same for conductors with the same flow constant. Figure 2.8 shows the paths which allow the maximum heat load for several inlet pressures. Once again, an inlet temperature of 4.5 K, a maximum allowable temperature of 6.0 K, and a minimum allowable pressure of 3 atm are used. Notice that most of the maximum heat load paths are able to fully use both the temperature and pressure constraint.

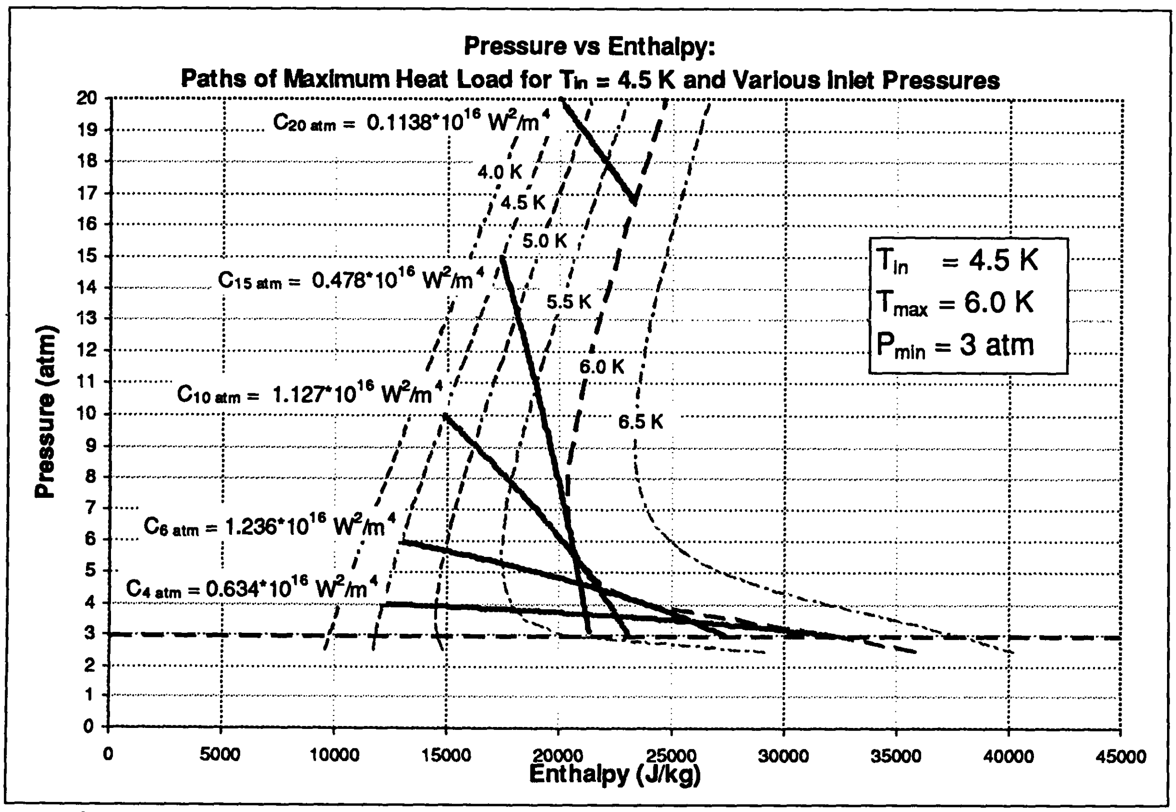


Figure 2.8: Pressure-Enthalpy Diagram with Paths for Maximum Heat Load

The results show that the maximum temperature may not occur at the exit. This phenomenon is shown by plotting the same data shown in Figure 2.8 on the temperature versus distance plot of Figure 2.9. This effect is related to the Joule-Thomson coefficient (defined as $dT/dP|_h$). For positive values of the Joule-Thomson coefficient, a decrease in pressure will cause the temperature to decrease. In pipe flow, friction causes the pressure to decrease in the direction of flow. So, if the Joule-Thomson coefficient becomes large enough, the temperature can show a net decrease in the flow direction even while heat is being added. Hale and Katheder show similar results. This effect should be considered when estimating the maximum steady state temperature.

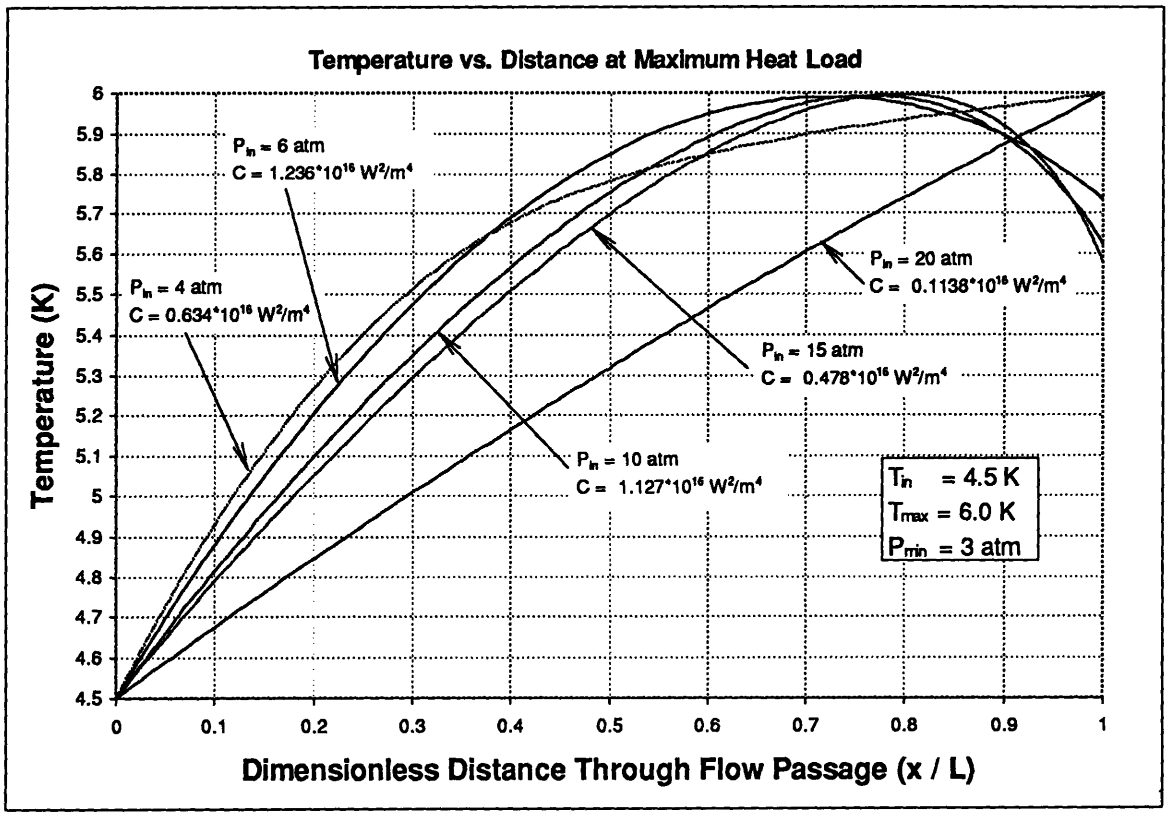


Figure 2.9: Temperature vs. Distance at Maximum Steady State Heat Load

The data can also be shown on the pressure versus distance plot of Figure 2.10 and the temperature versus pressure plot of Figure 2.11. The pressure shows a nearly linear drop through the passage with only slight curves due to the changing density. The temperature-pressure plot shows useful information, but it requires a brief explanation to be understood. The passage inlet is at the bottom right of each curve. As the flow proceeds through the conductor, the pressure drops and the temperature rises. At a certain point, most of the curves show a decrease in temperature from Joule-Thomson effects. Once again, these curves are applicable to any regular passage under the conditions of the maximum allowable heat load with the same inlet conditions and constraints as shown.

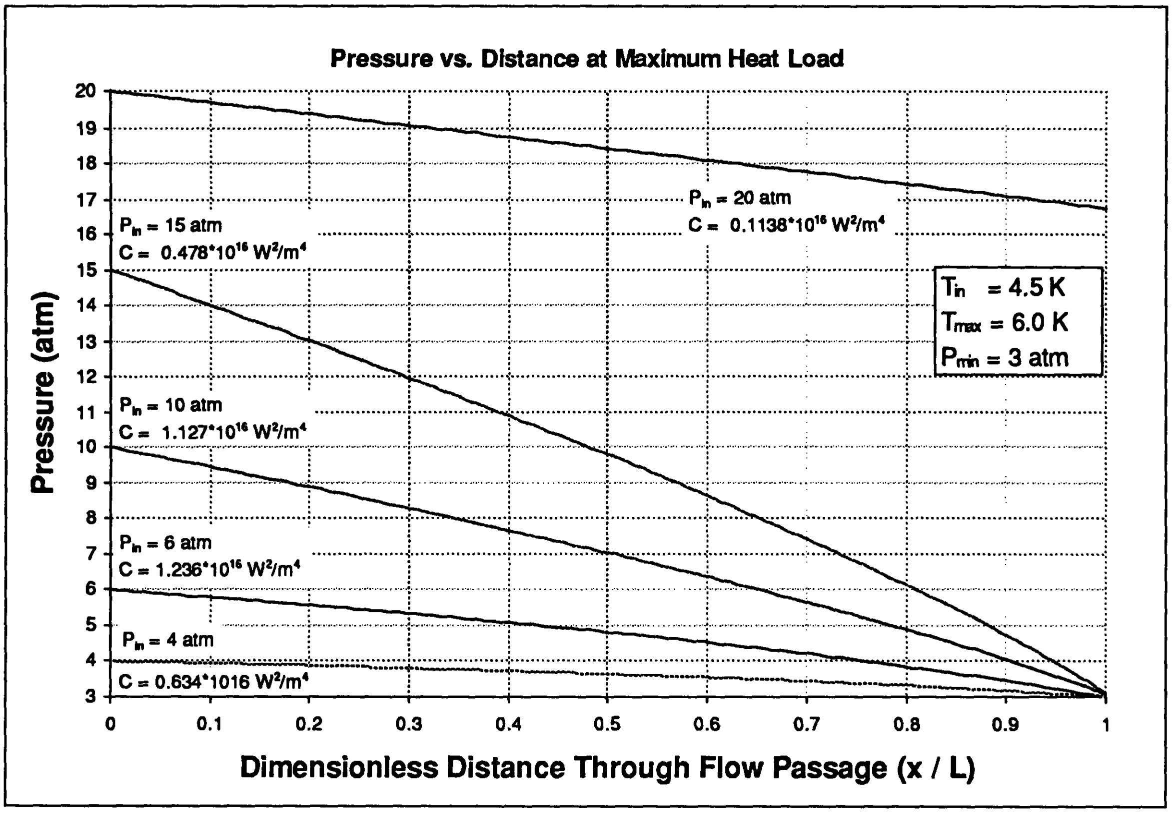


Figure 2.10: Pressure vs. Distance at Maximum Steady State Heat Load

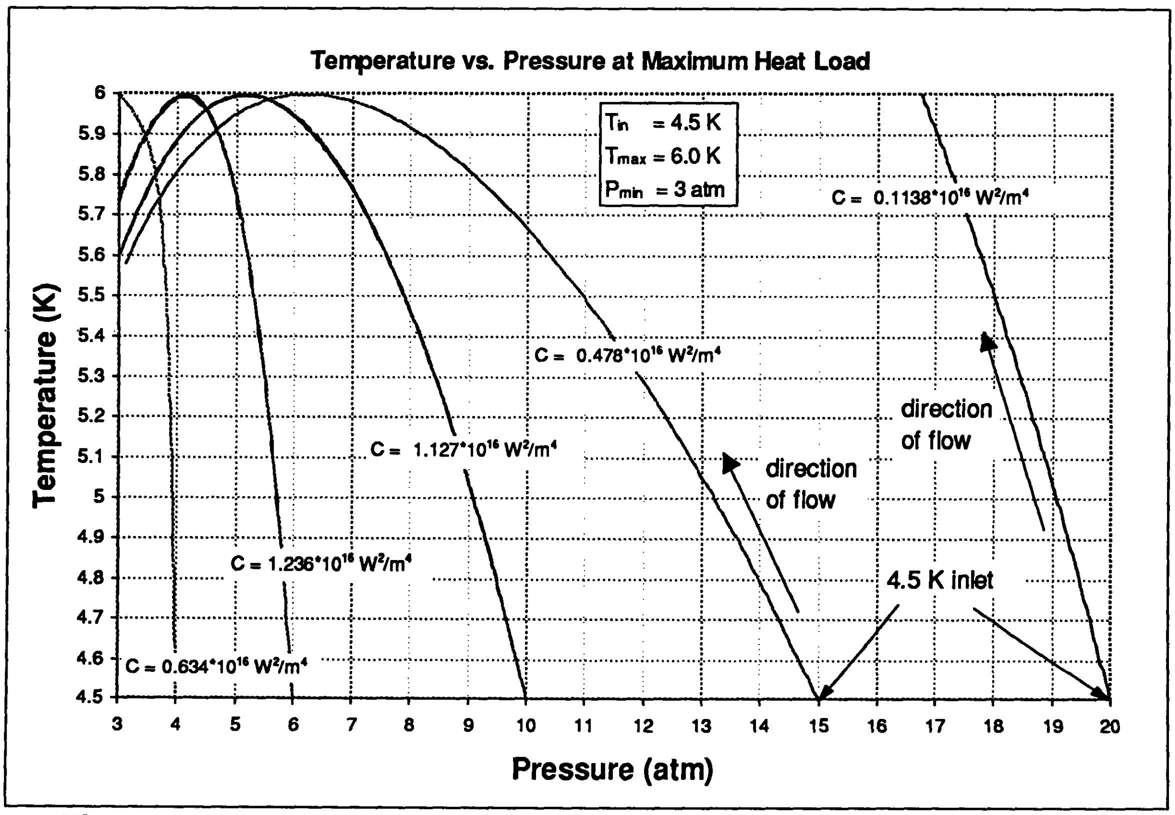


Figure 2.11: Temperature vs. Pressure at Maximum Steady State Heat Load

The generalized data can also show how efficiently the heat is removed from the passage. The entropy generated in the passage is used to measure this efficiency. A lower entropy generation signifies a higher heat removal efficiency. Figure 2.12 shows the ratio of the entropy generated in the flow passages to the entropy removed from the walls of the passage. Once again, the data at the maximum heat load was used for each curve. The results show that the entropy generation is lower at lower inlet pressures for a fixed allowable temperature. The results also show that allowing higher temperatures reduces the entropy generation for a given inlet pressure.

The curves are drawn to the point where the flow goes from being able to match both the temperature and pressure constraints to only matching the temperature constraint at the maximum heat load. Due to this sharp change, no results should be extrapolated beyond the curves shown.

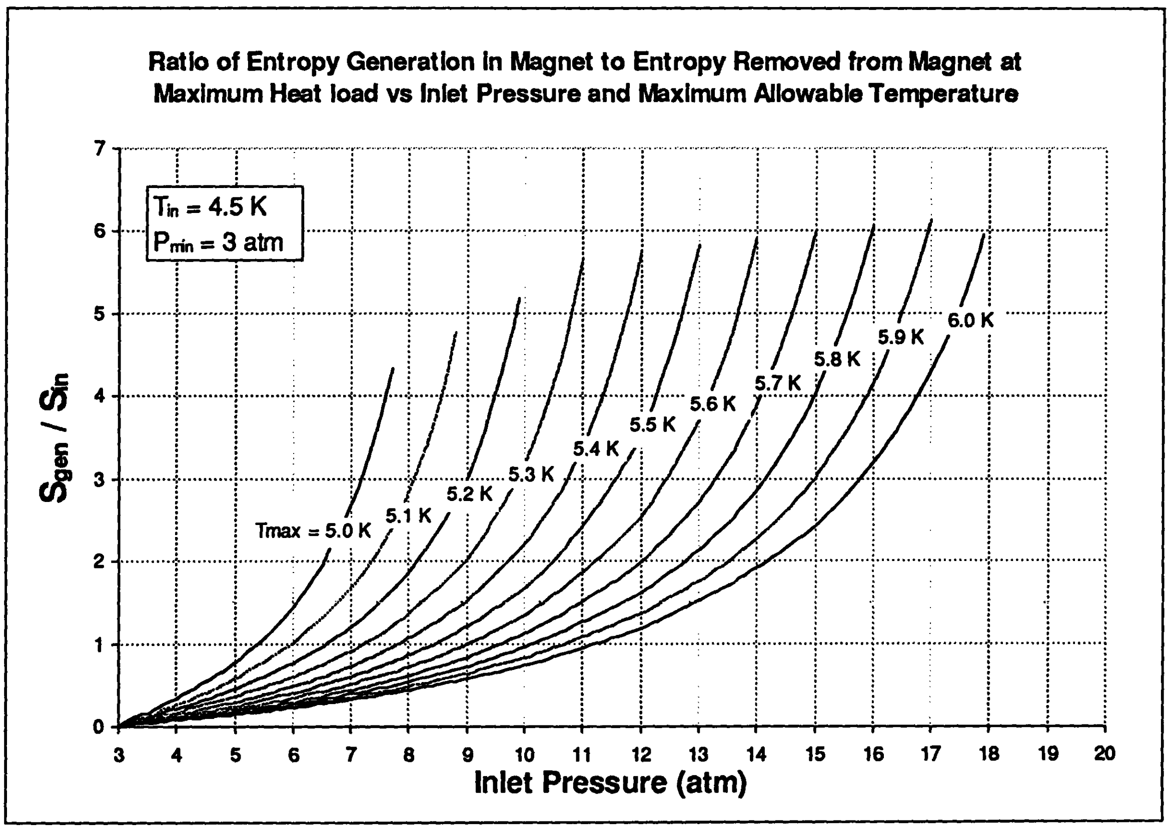


Figure 2.12: Steady State Entropy Generation in Magnet

2.6 CONCLUSIONS

The steady state modeling of helium flow through a passage resulted in the following conclusions:

- A finite difference model can be used to study the steady state heat removal capability of helium flowing through a passage.
- Computer programs using the finite difference technique found the maximum amount of heat that can be removed from a passage for given inlet conditions and constraints.
- The results can be generalized for use with passages of different sizes.
- Passages with the same temperature and pressure profile along the normalized distance through the passage (X) will have the same value of the flow constant (C) presented in equation 2.32.
- The flow constant is a function of the inlet conditions.
- The flow constant at maximum heat load is also a function of the temperature and pressure constraints.
- Equation 2.33 can be used along with Figure 2.6 to estimate the maximum amount of heat that can be removed from a given flow passage.
- In practice, passages should include a safety margin by being designed to operate below the maximum heat load as calculated by equation 2.33.
- To reduce steady state pumping power, cooling systems should use the lowest inlet pressure that will allow the heat load to be safely removed.
- The steady state entropy generation in the flow passage can be reduced by running at lower inlet pressures and allowing higher temperatures in the flow passage.
- When operating at high heat loads or in lower pressure regions, the maximum temperature in the flow passage may not occur at the passage outlet due to the temperature drop associated with Joule-Thomson effects.

3. PUMPING ISSUES

This chapter contains three sections. Section 1 describes the reasons for studying pumping issues related to the cooling of cable-in-conduit conductor SMES magnets. Section 2 examines the effect of the pump efficiency on the total refrigeration heat load. Section 3 looks at the effect of placing the pump at a location where it can pump colder, denser helium. Section 4 summarizes the conclusions.

3.1 MOTIVATION BEHIND THE STUDY OF PUMPING ISSUES

A cable-in-conduit conductor (CICC) is cooled by forcing cold helium through the conduit. The helium picks up heat energy as it flows in contact with the numerous superconducting strands. The pump (or compressor) that produces this flow of helium also adds energy to the helium. In a continuously operating magnet system, the energy from both the magnet conductor and the pump must be removed so that the helium can be reused to cool the conductor. Therefore, the refrigeration system must be designed to remove the combined heat load of the magnet and the pump.

Most studies concentrate on magnet issues and neglect the pump needed to cool the magnet. However, the pumping power can be the dominant load on the refrigeration system. A poorly designed refrigeration system could require a large pumping power to cool the magnet. The system would require a larger and more expensive refrigerator than if it were properly optimized to reduce the pumping power.

To minimize the size of the refrigeration system, it is important to consider the factors which affect the pumping power. This chapter looks at two of these factors. First, the isentropic efficiency of the pump is considered. Second, the location of the pump in the cooling loop is examined.

3.2 EFFECT OF PUMP EFFICIENCY ON THE TOTAL REFRIGERATION LOAD

The isentropic efficiency of a pump affects the power required for the pump to increase the pressure of a fluid. An inefficient pump requires more energy than an ideal pump. In a SMES system, any extra pumping power will increase the refrigeration load. Therefore, an inefficient pump can increase the size of the refrigerator needed. This section looks at how the pump adds to the refrigeration load and how the pump efficiency determines the amount the load is increased.

3.2.1 EFFECT OF PUMPING POWER ON REFRIGERATOR LOAD

Figure 3.1 shows a possible configuration for cooling a SMES magnet. High pressure helium circulates in a closed loop (ABCD). Helium exits the pump at location A and is then cooled in a heat exchanger that is inside a pool of liquid helium. The helium enters the magnet at B and then absorbs heat from the magnet. After leaving the magnet at C, the helium returns to the pump to be recirculated. This configuration is called a recirculator loop.

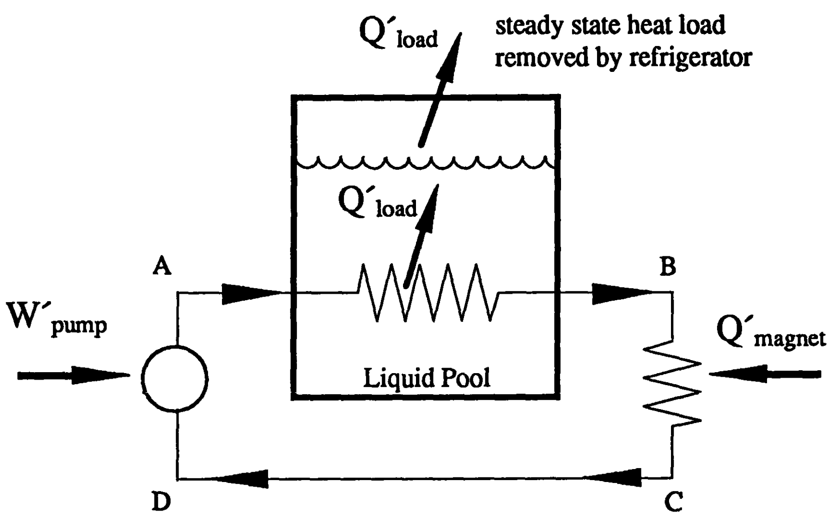


Figure 3.1: A Possible SMES Cooling Configuration

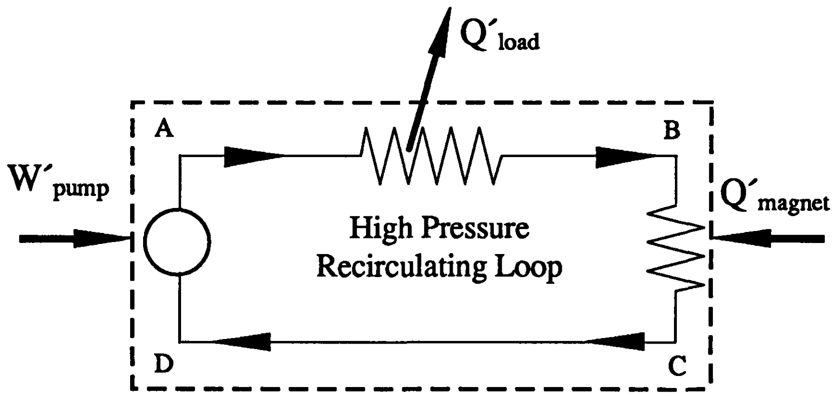


Figure 3.2: Energy Balance on Recirculator Loop

Figure 3.2 shows the recirculator loop with a control volume drawn around it. The steady state energy balance is given by:

$$Q'_{load} = Q'_{magnet} + W'_{pump} \tag{3.1}$$

It is apparent that the refrigerator must remove the heat load of both the magnet and the recirculating pump. A low efficiency pump will require a large power input and will increase the heat load on the refrigerator. To study this effect, steady state models of the recirculator loop were made.

3.2.2 SIMPLE MODEL OF STEADY STATE RECIRCULATOR LOOP

To illustrate the effect of the pump efficiency on the refrigerator load, a simple model can be used. The three equations which determine the heat flows in the recirculator loop are:

$$Q'_{magnet} = (h_C - h_B) \cdot m' \tag{3.2}$$

$$W'_{pump} = (h_A - h_D) \cdot m' \tag{3.3}$$

$$Q'_{load} = (h_A - h_B) \cdot m' \tag{3.4}$$

where m' is the mass flow rate and h is the specific enthalpy.

The equation which relates the enthalpy at the pump inlet and outlet is:

$$h_A = h_D + \frac{1}{\eta} \cdot (h_{A_s} - h_D) \tag{3.5}$$

where η is the pump's isentropic efficiency and h_{A_s} is the ideal exit enthalpy of the pump.

To analyze the system performance, the conditions in the loop are needed. First, the conditions at locations B and C are chosen. Assuming there are no losses between the magnet exit and the pump, the conditions at D are the same as at C. Assuming that the pressure drop across the pool heat exchanger is small, the pressure at A is nearly the same as at B ($P_A = P_B$). Using the pressure at A and the conditions at D, the ideal (isentropic) enthalpy at A is found. Using equation 3.5, the actual conditions at A can then found. Once, all conditions are known, equations 3.2 through 3.4 are used to evaluate the heat loads.

For this study, the magnet inlet temperature (T_B) was taken to be 4.3 K, the magnet exit temperature (T_C) was 5.0 K, and the pressure at the pump exit (P_A) was 10 atm.

The pressure drop across the magnet was varied to simulate different frictional characteristics of the magnet. Figure 3.3 shows the ratio of the refrigerator heat load to the magnet heat load. As expected, the pump's isentropic efficiency affects the refrigerator's heat load. For example, a system with a 50% efficient pump has a refrigerator load that is about 30% larger than a system with a 70% efficient pump.

Figure 3.3 also shows how the refrigerator heat load increases as the pressure drop increases. For example, if the pressure drop across the magnet is 0.5 atm, the refrigerator heat load is about 30 to 40 % larger than the heat load removed from the magnet. But for a 2 atm drop, the refrigerator heat load is 200 to 300 % larger than the cooling supplied to the magnet. Therefore, the pressure drop across the magnet should be kept to a minimum to reduce the size of the refrigerator required.

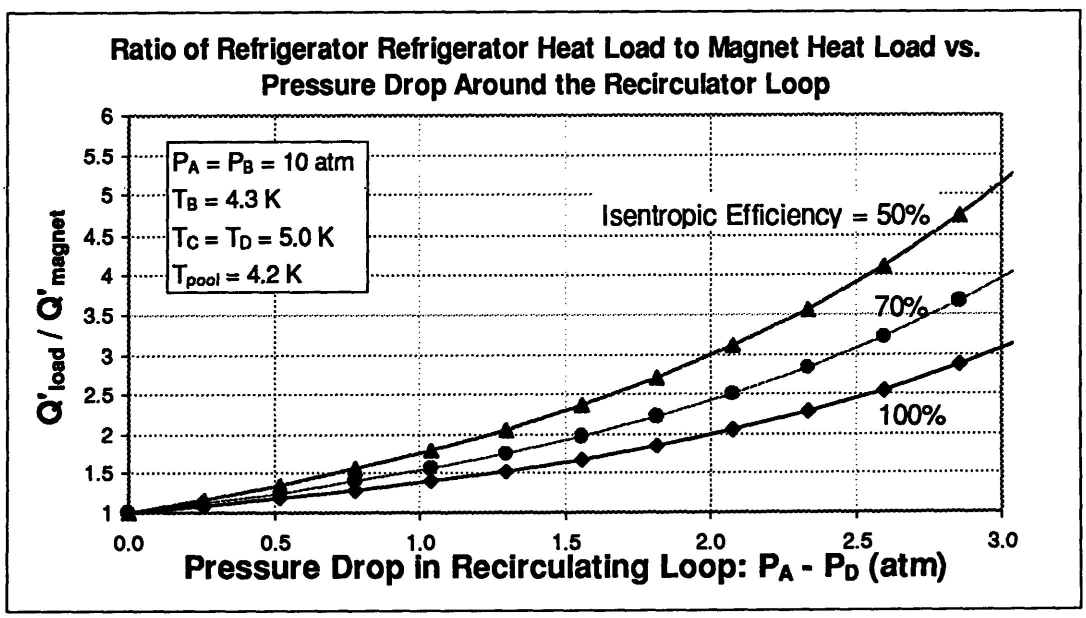


Figure 3.3: Ratio of Refrigerator Heat Load to Magnet Heat Load versus Pressure Drop in Recirculator Loop

Another way to measure the impact of the recirculator loop is to look at the entropy generated in the loop. Entropy generated in the magnet, in the pump, and in the pool heat exchanger will increase the entropy load on the refrigerator. Figure 3.4 shows an entropy flow diagram. Entropy is removed from the magnet (S_{magnet}) and put into the recirculator loop where it passes through the pump and finally into the helium pool of the refrigerator. The entropy generated along the way must also be removed. Therefore, the entropy load of the refrigerator pool (S_{load}) is larger than the entropy removed from the magnet (S_{magnet}):

$$S_{load} = S_{magnet} + Sgen \tag{3.6}$$

Where the total entropy generation (Sgen) is the sum of the entropy generated in the magnet, pump, and heat exchanger:

$$Sgen = Sgen_{magnet} + Sgen_{pump} + Sgen_{HX} \tag{3.7}$$

For this model, no entropy generation was considered inside the magnet (i.e. $Sgen_{magnet} = 0$, $S_{in} = S_{magnet}$). Only the entropy generated in the pump and in the heat transfer of the pool heat exchanger were included. However, these two components alone can add a significant amount of entropy to the system. Figure 3.5 shows the influence of the pump's isentropic efficiency. A system with a perfect pump would increase the entropy load by about 10% while a 50% efficient pump could create 80% more entropy load on the refrigerator. It should be noted that most magnets are designed for a pressure drop under 1 or 2 atm. However, Figure 3.5 shows the dramatic influence of larger pressure drops on a low efficiency pump.

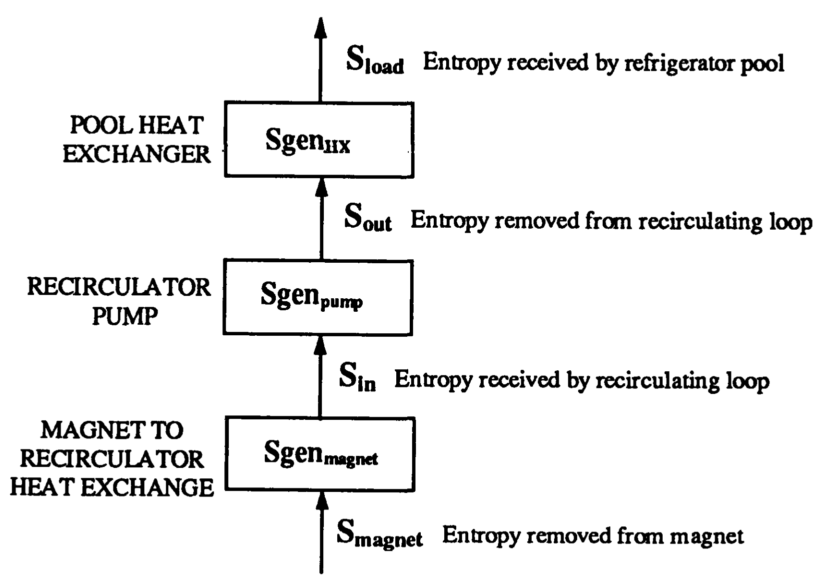


Figure 3.4: Entropy Flow Diagram of Recirculator Loop

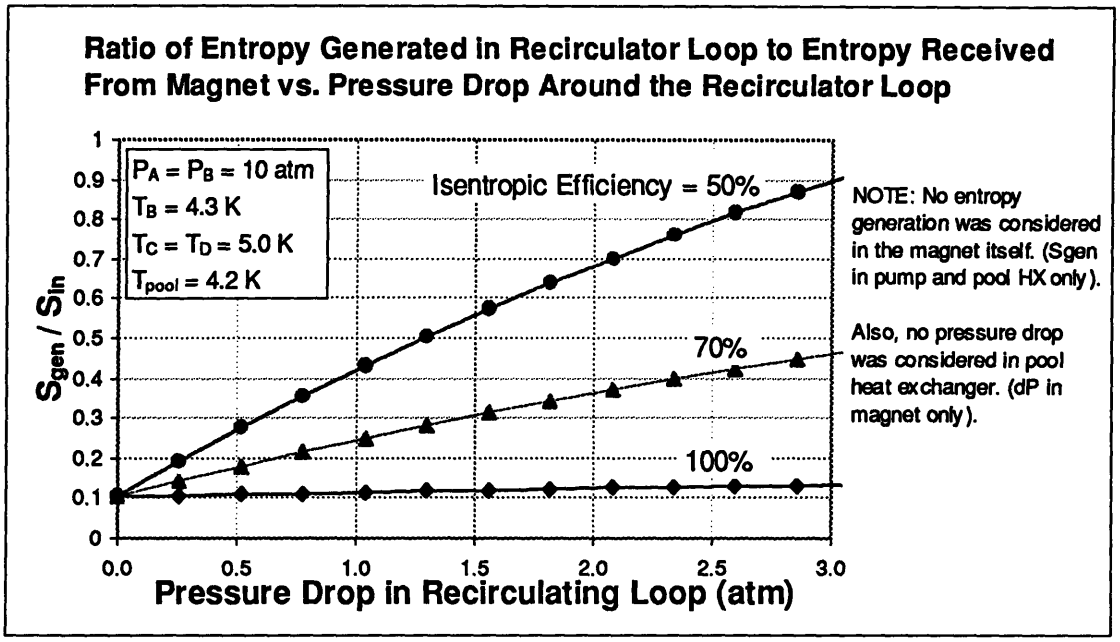


Figure 3.5: Entropy Generation in Recirculator Loop

3.2.3 STEADY STATE RECIRCULATOR LOOP MODEL USING CICC CHARACTERISTICS

The simple model illustrated how the recirculator pump increases the load on the refrigerator. That model assumed that the conditions at the magnet inlet and outlet can be chosen arbitrarily. A more detailed model would find these conditions by including the characteristics of the conductor in the magnet. This section illustrates how the conductor model of Chapter 2 can be used as a part of a larger model of the entire recirculator loop. In Chapter 5, the model will be further enhanced to include more detailed characteristics of the pump and heat exchangers.

The magnet was modeled using the finite-difference technique of Section 2.3. The magnet inlet and outlet conditions were taken from the example in Section 2.4.[1] Equations 3.2 to 3.5 were then used to evaluate the recirculator loop performance. A program[2] was written to solve the equations.

Figure 3.6 shows the ratio of the refrigerator heat load to the heat removed from the magnet. This ratio is the multiplying factor by which the recirculator loop increases the load on the refrigerator pool. In a perfect system, no pump power would be required and the ratio would be 1.0. However, a real system requires pumping power to force the

[1] This example found the maximum amount of heat that could be removed from the conductor for various mass flow rates. Data from this example is also plotted in Figure 2.3.

[2] program named *CICC4.EXE*. see appendix for listing

helium through the magnet. The pump power adds to the refrigerator load, making the Q'_{load} / Q'_{magnet} ratio greater than 1.0.

Figure 3.6 shows three trends. First, as the heat load on the magnet is increased, the recirculator loop has a larger multiplying effect on the refrigerator load. This is caused by the additional pumping power required to supply more mass flow at higher heat loads. Second, a low efficiency pump requires more pumping power and further increases the load on the refrigerator. Finally, a lower magnet inlet pressure can reduce the refrigerator load due a lower pumping power requirement. However, this assumes that a pump can be found to match the pressure drop and mass flow rate calculated by the model.

This example shows that cooling system models should include the effect of the pump on the refrigeration load. It also shows how the model for any conductor can be used as a part of the overall system model.

The curves of Figure 3.6 are drawn to the point where the flow goes from being able to match both the temperature and pressure constraints to only matching the temperature constraint at the maximum heat load (see section 2.4). Due to this sharp change, no results should be extrapolated beyond the curves shown.

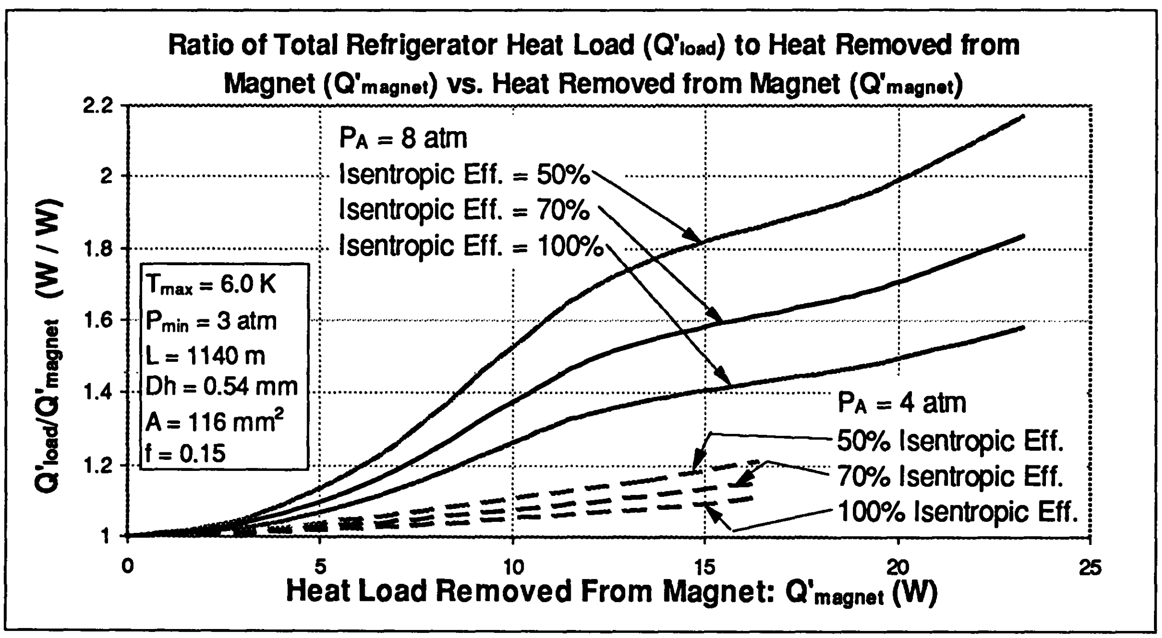


Figure 3.6: Ratio of Refrigerator Load to Heat Removed from Magnet

3.3 COLD PUMPING

Another way to reduce the refrigeration load is to place the recirculator pump at the coldest part of the recirculator loop. Pumping colder, denser helium will reduce the pump power. This reduction can be explained by looking at the equation which determines pump work for an ideal, steady state pump [Jones, 33]:

$$w = \int v \cdot dP \tag{3.8}$$

where w is the pump work per unit mass, P is the pressure, and v is the specific volume.

Equation 3.8 shows that the ideal pump work is proportional to the specific volume of the fluid flowing through the pump. The work required to produce a given pressure rise can be reduced by pumping a cold, high density (low v) fluid. This concept can be illustrated by comparing two ideal pumps.

Figure 3.7 compares ideal pumps operating between 3 atm and 5 atm. One pump has a fluid inlet temperature of 4.5 K while the other is one degree warmer at 5.5 K The paths for ideal pumps are shown as a dark line. The area under this line is equal to the required pump work (equation 3.8). The area under the 4.5 K curve (diagonal pattern) is 29% less than the area under the 5.5 K curve (horizontal pattern). Thus, a 1.0 K reduction in inlet temperature can significantly decrease the required pump work.

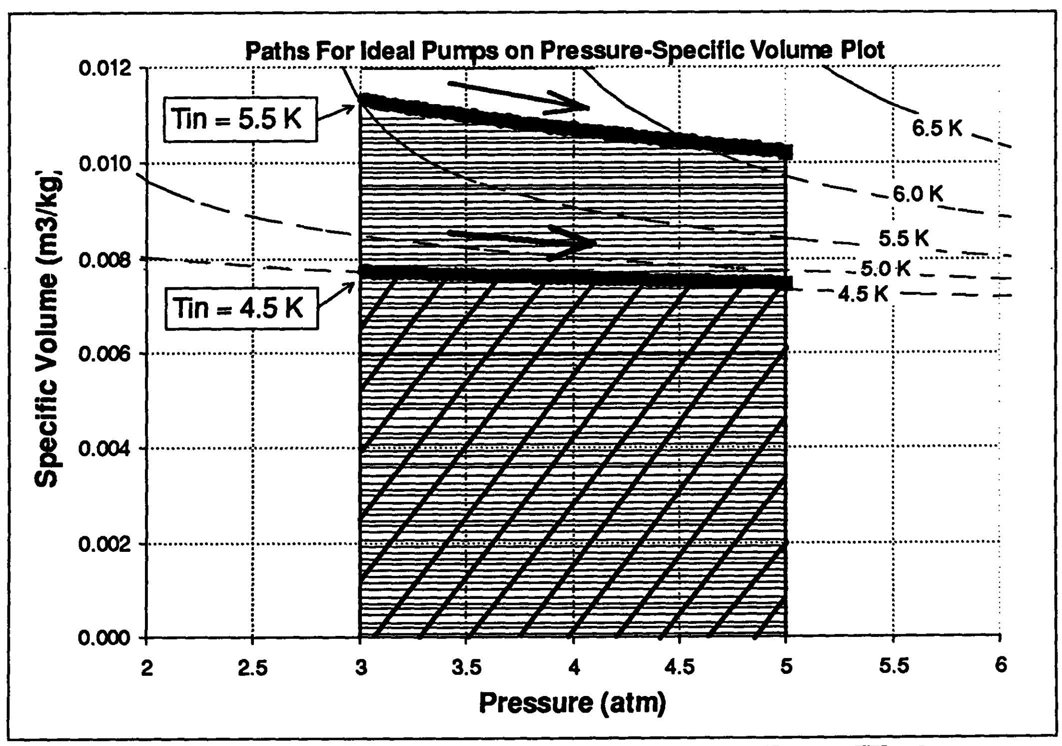
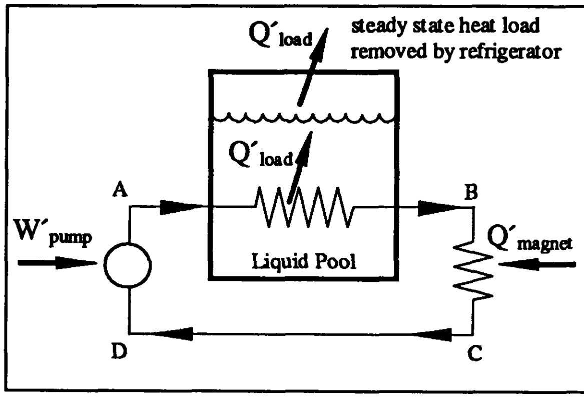


Figure 3.7: Effect Fluid Temperature on Ideal Pump Work

Figure 3.8a shows a recirculator loop that places the pump immediately after the magnet. The helium entering the pump is relatively warm from the energy it gained in the magnet. A better design would recool the helium before it enters the pump. Figure 3.8b shows a cold-pumping configuration. An additional heat exchanger is added to the loop so that the pump receives cold helium at D. In this system, the temperatures at B and D should be as cold as possible and approximately equal. Therefore, heat exchanger 2 (HX2) is designed to remove the magnet load and heat exchanger 1 (HX1) is designed to remove the pump power.



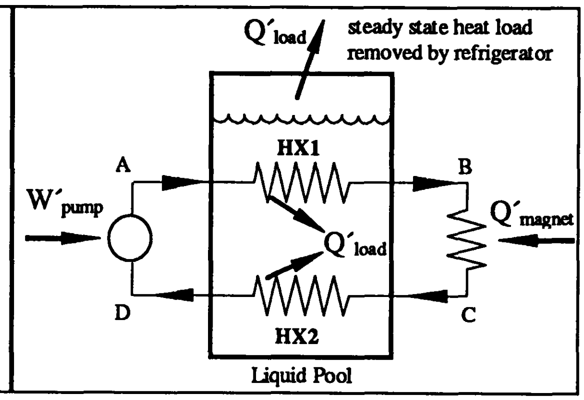


Figure 3.8a: Warm Pumping **Figure 3.8b: Cold Pumping with Re-Cool**

The two cooling configurations can be compared by using the generalized conductor data of section 2.5 to model the magnet. This data can be used to obtain the conditions at B and C. The conditions at D are found by either assuming that they are the same as at C for the warm pumping system, or by assuming HX2 is sized to re-cool the helium to the same temperature as at B. Equations 3.2 to 3.5 are then used to evaluate the system performance.

Once the data from section 2.5 is used to find the conditions at B and C, the conditions at A and D are fixed by the assumptions stated above. Since all conditions are fixed, the ratio of pumping power to rate of heat removal from the magnet will be constant for a given magnet operating condition:

$$\frac{W'_{pump}}{Q'_{magnet}} = \frac{(h_A - h_D)\cdot m'}{(h_C - h_B)\cdot m'} = \frac{(h_A - h_D)}{(h_C - h_B)} \tag{3.9}$$

The data that was used to find the properties at B and C is for the maximum allowable heat removal in the magnet. Therefore, the W'_{pump} / Q'_{magnet} ratio will be a function of the magnet inlet conditions and the constraints. This ratio also depend on the cooling configuration, which sets the conditions at A and D. Therefore, the ratio is a constant for a given cooling configuration and a given set of magnet inlet conditions and constraints:

$$\frac{W'_{pump}}{Q'_{magnet}} = \Im(P_{in}, T_{in}, P_{min}, T_{max}, \text{configuration}) \tag{3.10}$$

For a given set of conditions at B and C, the warm pumping power and cold pumping power can be compared by using equation 3.10:

$$\frac{\left(W'_{pump}\right)_{warm}}{\left(W'_{pump}\right)_{cold}} = \frac{\dfrac{(h_A - h_d)_{warm}}{(h_C - h_B)}}{\dfrac{(h_A - h_d)_{cold}}{(h_C - h_B)}} = \frac{\dfrac{\left(W'_{pump}\right)_{warm}}{Q'_{magnet}}}{\dfrac{\left(W'_{pump}\right)_{cold}}{Q'_{magnet}}} = \frac{\Im(P_{in}, T_{in}, P_{min}, T_{max}, \text{warm})}{\Im(P_{in}, T_{in}, P_{min}, T_{max}, \text{cold})} \tag{3.11}$$

Figure 3.9 shows the ratio of warm pumping to cold pumping. Figure 3.10 shows the ratio of pumping power to the rate of heat removal from the magnet for both the cold and warm pumping configurations. For each case, the magnet inlet temperature is 4.5 K and the minimum allowable pressure is 3 atm.

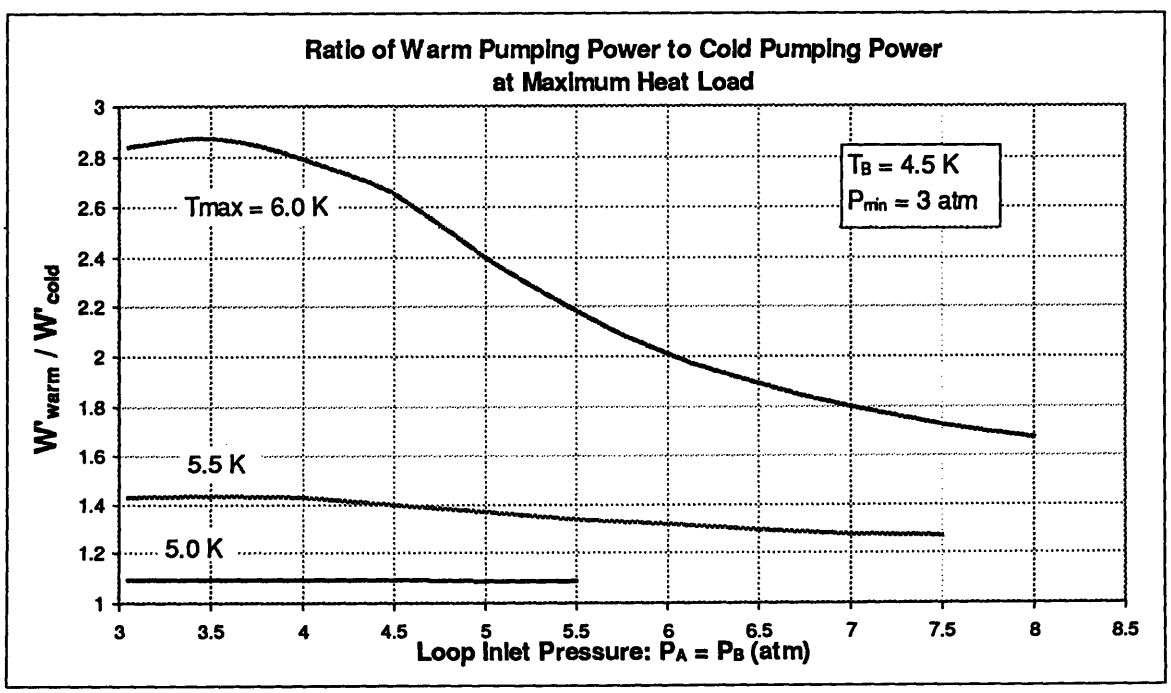


Figure 3.9: Ratio of Warm to Cold Pumping Power at Maximum Heat Load

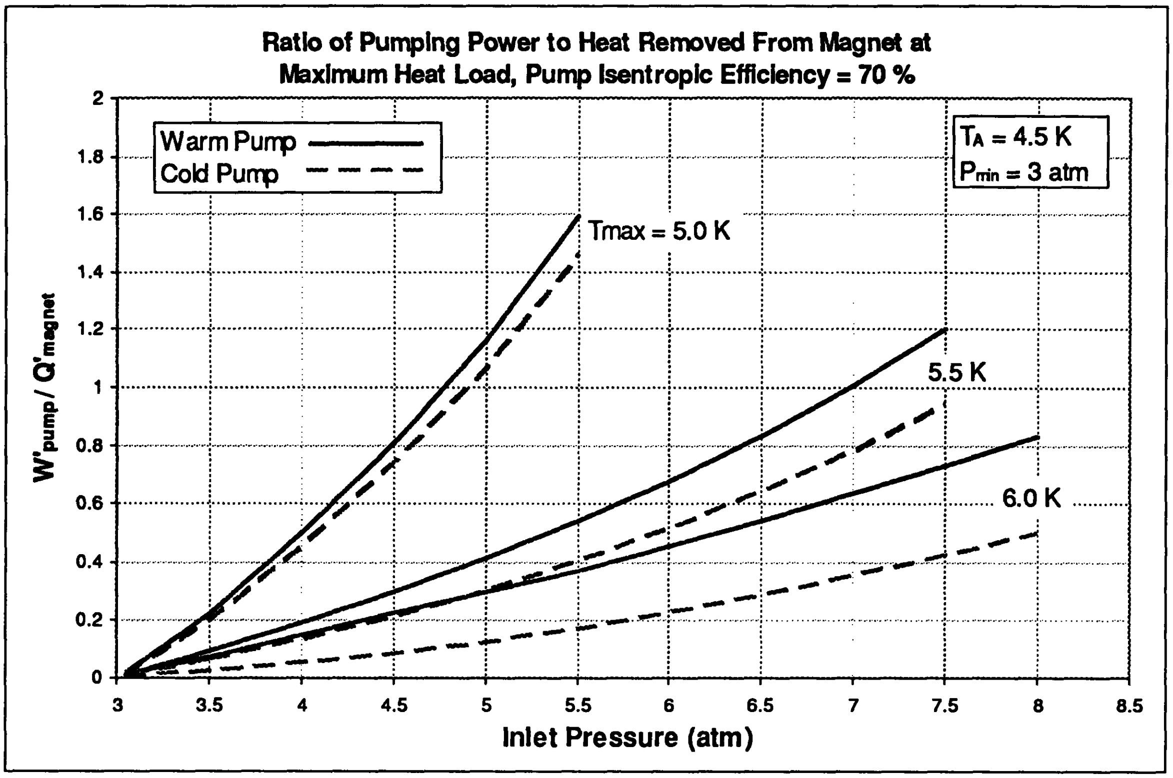


Figure 3.10 Ratio of Pumping Power to Heat Removed from the Magnet at Maximum Heat Load

Figure 3.9 shows that pumping the helium at a colder temperature will reduce the amount of pump work. The difference between warm and cold pumping becomes greater as the maximum helium temperature in the magnet is increased. This occurs since the warm pumping is done at approximately T_{max}, while the cold pumping is always done at 4.5 K. Therefore, the difference between warm and cold pumping becomes greater as T_{max} is raised.

Figure 3.10 shows that the refrigerator load decreases when cold pumping is used. Therefore, cold pumping can reduce the size of the refrigerator required.

The plot also shows that allowing higher temperatures can significantly reduce the pumping power due to the lower mass flow requirement.

Section 2.4 showed that, when the temperature is constrained, there is a practical limit to the conductor inlet pressure. Above this limiting inlet pressure, no more heat can be removed. Therefore the curves of Figures 3.9 and 3.10 are only drawn up to the limiting inlet pressure. Results should not be interpolated beyond the curves shown.

3.4 CONCLUSIONS

The study of SMES pumping issues resulted in the following conclusions:

- In a recirculator loop cooling configuration, the refrigerator must remove the heat load of both the magnet and pump.
- The recirculator loop acts as a heat load multiplier by rejecting more heat to the refrigerator than it removes from the magnet.
- Low efficiency pumps will increase the load on the refrigerator.
- The pump power can be reduced by re-cooling the helium before entering the pump.
- The pressure drop across the magnet should be minimized to reduce the pumping power required.
- The CICC model of Chapter 2 can be used as a part of a larger recirculator loop model which includes the pump and heat exchanger(s).
- Lower magnet heat loads will require less helium flow and less pump power.
- To reduce pumping power, systems should be designed to run at much less than the maximum allowable heat load.

4. BUFFER SYSTEMS

This chapter contains 4 sections. Section 1 describes reasons for using a buffer in a SMES refrigeration system. Section 2 lists the results of the buffer system models. Section 3 discusses the buffering schemes that were analyzed and the thermodynamic models that were used. Section 4 summarizes the conclusions.

4.1 MOTIVATION BEHIND THE STUDY OF BUFFER SYSTEMS

When a SMES magnet is charged or discharged, AC losses deposit heat energy in the conductor. The refrigeration system must remove this energy to keep the conductor cold. However, this heat is deposited at a much higher rate than the steady state refrigerator load. Only a high power refrigerator could remove this energy "pulse" at the same rate as it is deposited. This refrigerator would be expensive and oversized for most times of operation.

A buffer system temporarily stores the pulse energy in some material or fluid (usually helium). After the pulse, the refrigerator can slowly remove the stored energy. Therefore, the refrigerator can be smaller than if it had to remove the energy at the full power level of the pulse. In effect, a buffer system "averages" the transient loads over a long period of time so that a smaller refrigerator can be used.

While there are many possible buffer systems, some must be much larger than others to store a given amount of energy. In most applications, the space for a buffer is limited. Therefore, this chapter looks at which systems can store the most energy per volume.

4.2 RESULTS OF BUFFER SYSTEM MODELING

Several buffer systems were considered. The results are summarized in Table 4.1. The evacuated tank concept can store the most energy per volume, but could be difficult to carry out in a real system. The divided tank allows the second most energy storage, but an actual system may not perform as well as the ideal system that was modeled. The closed container with two-phase helium is probably the most practical system. Each of the systems are described in the next section.

METHOD	SPECS	ENERGY STORAGE PER VOLUME (J/m^3)	PUMP WORK	PROS	CONS
1. Boiling At 1 atm With An Expandable External Tank To Hold Vapor	Start with saturated liquid, end with saturated vapor. (No superheat allowed to keep at 4.2 K)	344,000	none	Simple. Constant Temp of 4.2 K	Poor Q/V ratio due to large vapor volume.
2. Heat Compressed Liquid In Closed Container	Start with compressed liquid at 4.2 K. Optimum P_{start} varies with T_{max}	285,000 (T_{max} = 5 K) (22 to 27 atm) 659,000 (T_{max} = 6 K) (28 to 39 atm)	pressurize once when filling tank	Simple	Pressure vessel required. Requires temp rise.
3. Heat 2-Phase Helium In Closed Container	Start at 4.2 K, two phase liquid	424,000 (T_{max} = 5 K) ($x1_{init}$ = 0.37) 720,000 (T_{max} = 6 K) ($x1_{init}$ = 0.07)	none	Simple.	Requires temp rise.
4. Heating Two-Phase Helium With A Rigid External Tank To Hold Vapor	Start at 4.2 K. Two phase in main tank, saturated vapor in external tank.	727,000 (T_{max} = 6K) (V2/V1=0.36)	none	---	More complex than 3, but no gain.
5. Heat Liquid Helium in Evacuated Closed Container	Start with saturated liquid at 0.1 atm (T=2.5 K). Close and add heat at constant density.	806,000 (T_{max} = 5 K) 1,158,000 (T_{max} = 6 K)	Pump enough to remove boil-off vapor	Very Good Q/V ratio	Hard to maintain 2.5 K. Requires vacuum pump.
6. Divided Tank With Energy Transfer To Environment	Insulated float divides tank into a section at 4.2 K and a section at 300 K. P_1= 1 atm	795,400 (T_{max} = 6 K) ρ_{A1} = 125 kg/m^3, x_{A1} = 0.0 P_2 = 4atm, Γ = 0.66, ϑ = 0.5 (assumed no conduction losses)	none	Good Q/V ratio	Requires temp rise.

Table 4.1: Comparison of Energy Storage Methods

4.3 COMPARISON OF POSSIBLE BUFFERING SYSTEMS

A buffer system stores thermal energy by either increasing the temperature or changing the phase of a substance. Helium is the best substance for storing energy in the temperature range of interest (~ 4.2 to 6 K) since it changes phase and has a high specific heat.

Several options for storing energy in helium were looked at. They may be classified as either boiling or heating in a closed container. Boiling maintains a constant temperature, but requires a large volume to hold the vapor that is produced. The closed container uses a smaller volume, but requires a temperature rise. Table 4.1 summarizes the results of the study. The remainder of this section describes the buffer systems that were studied and the thermodynamic models that were used.

4.3.1 BOILING AT 1 atm WITH AN EXPANDABLE EXTERNAL TANK TO HOLD VAPOR

Figure 4.1 shows a buffer system that absorbs heat by boiling helium at 1 atm. The advantage of this system is that the temperature remains constant at 4.2 K. Therefore, there is little danger of the magnet temperature rising. However, the disadvantage is that the vapor takes up much more volume than the liquid. Therefore, a large external tank is required to hold the vapor being produced. The model for this system assumes that the external tank can expand to always keep the pressure at 1 atm. This leads to a large final volume $V2_{final}$ and a relatively small energy storage per total tank volume.

The model also assumes that the rigid tank starts with saturated liquid and the expandable tank starts at zero volume. At the final state, saturated vapor exists in both tanks. Therefore, the energy storage per volume is the product of the vapor density and the heat of vaporization:

$$\frac{Q}{V1 + V2_{final}} = \rho_v \cdot h_{fg} \tag{4.1}$$

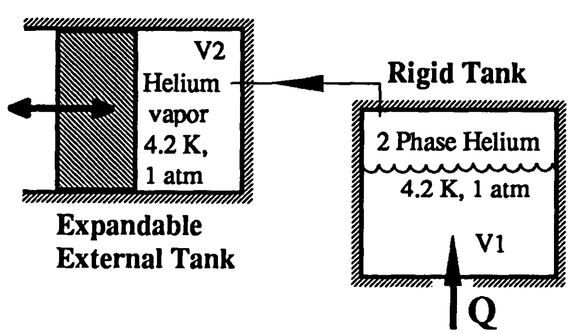


Figure 4.1: Boiling at 1 atm With an Expandable External Tank to Hold Vapor

4.3.2 HEATING COMPRESSED LIQUID IN A CLOSED CONTAINER

The large tank to hold vapor can be eliminated if the helium container is not vented and the temperature is allowed to rise slightly. The tank can be initially pressurized to fit more mass in the container. As heat is added, the pressure and temperature will rise. Since the tank volume and mass are fixed, this is a constant density process and the final conditions are easily found. The energy storage per volume is given by the overall density times the change in specific internal energy:

$$\frac{Q}{V} = \rho \cdot (u_{T_{final},\rho} - u_{P_{initial},T_{initial}}) \tag{4.2}$$

A program[1] was written to find the optimum starting pressure for a starting temperature of 4.2 K and different final temperatures. These values are listed in Table 4.1 along with the heat stored per volume. This process could allow more heat to be stored per unit volume than the boiling of method 1. However the temperature rise could create problems for the magnet stability if the tank is in direct contact with the fluid flowing through the magnet.

4.3.3 HEATING TWO-PHASE HELIUM IN A CLOSED CONTAINER

Another closed container concept uses two-phase helium that starts at 4.2 K and 1 atm. This is a constant density process, which can be evaluated using Equation 4.2. A program[2] was written to search for the optimum initial density (which fixes the initial quality). The program showed that this method can store more heat per unit volume than the compressed liquid of method 2. Although the 1 atm helium is less dense than the compressed liquid, the latent heat allows more energy to be stored per volume of container. However, as in method 2, a temperature rise is required.

4.3.4 HEATING TWO-PHASE HELIUM WITH A RIGID EXTERNAL TANK TO HOLD VAPOR

Instead of confining the helium to a closed tank, a rigid external tank can be used as an "overflow" for some of the vapor produced. This system would be easier to design and require less volume than the expandable tank system of method 1.

Figure 4.2 shows the setup of the tanks. Heat is added to tank 1, which starts with two-phase helium at 4.2 K. As heat is added, the liquid and vapor change density. However, the densities change at different rates, causing vapor to move from tank 1 to tank 2. This vapor removes energy from tank 1 since it has a higher enthalpy than the liquid. Removing this energy could allow more heat to be added for a given temperature

[1] The program is named *rigid2.for*. See appendix for listing.
[2] The program is named *rigid4.for*. See appendix for listing.

change in tank 1. A program[3] was written to evaluate this system for various volume ratios. The program uses a differential model to tabulate the amount of heat absorbed as the temperature in tank 1 increases by tiny amounts.

Figure 4.3 shows that, if the ratio of the tank volumes is 0.36, this system can store about 1% more energy than a single closed tank (V2/V1 = 0). However, this analysis neglected heat leaks. In a real system, the extra surface area of the second tank would add to the heat leak and probably eliminate this small advantage. Therefore, this method does not seem to be any better than using one closed tank (method 3).

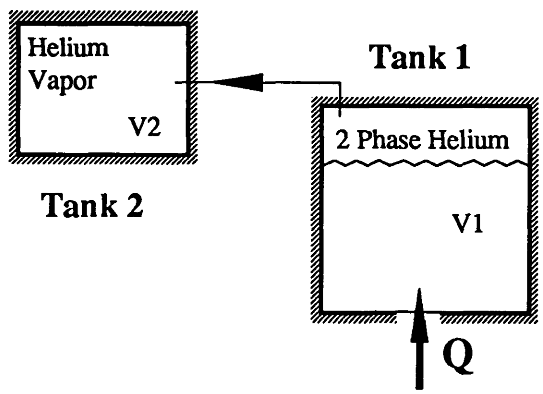


Figure 4.2: Heating Two-Phase Helium with a Rigid External Tank

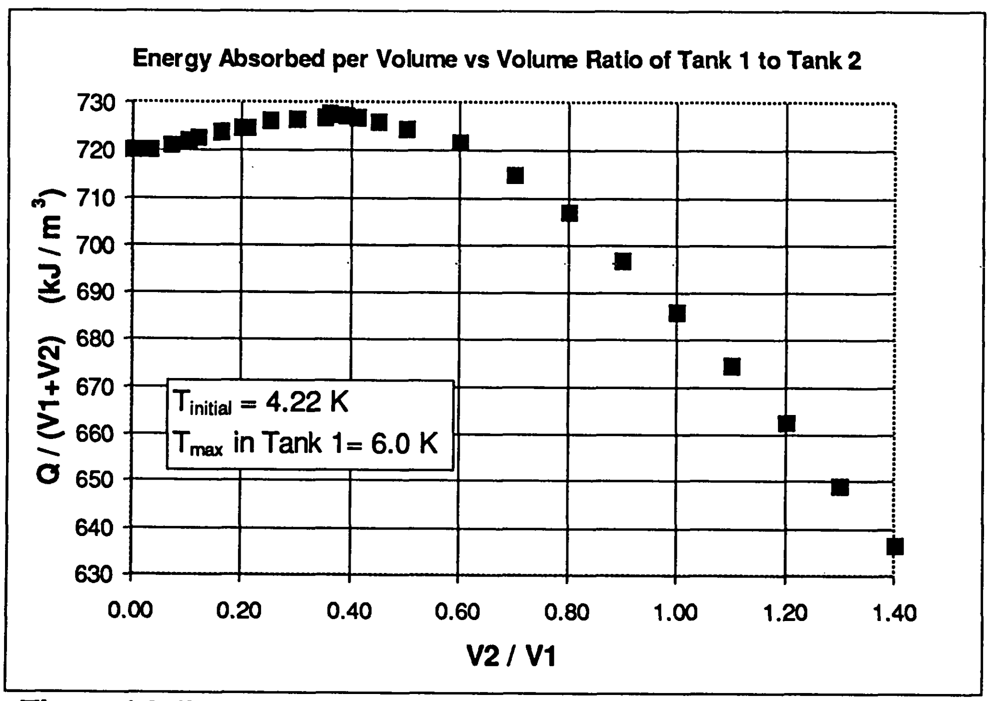


Figure 4.3: Energy Storage per Volume for Two-Tank Buffer System

[3] The program is named *double3.for*. See appendix for listing.

4.3.5 EVACUATED CLOSED CONTAINER

Another way of storing energy in a closed container is to start with liquid helium at a pressure much lower than 1 atm. This would decrease the initial temperature in the tank and allow more energy to be absorbed than if the tank started at 1 atm. Using a vacuum pump, a container with saturated liquid helium could be kept at a low pressure. Just before the pulse load, the container would be closed. Heat could then be added to the container in a constant density process. As shown in Table 1, this method allows a rather large amount of heat to be absorbed per unit volume. However, cooling and maintaining saturated liquid at low pressures (and temperatures) could be difficult. And once again, the temperature rise could create stability problems for the magnet.

4.3.6 DIVIDED TANK WITH ENERGY TRANSFER TO ENVIRONMENT

The size of the storage system could be reduced if the ambient environment is used to store some of the energy. A direct heat transfer to the environment would warm the system undesirably. However, the system can be kept cold if the heat is transferred to the environment via a work transfer. A theoretical sketch of such a system is shown in Figure 4.4.

The system uses a rigid container that is divided by an insulated float into 2 compartments. Compartment A holds cold helium and is where the magnet energy is rejected to. Compartment B holds warm helium that is in thermal contact with the ambient environment. As heat is added to A, the helium in A expands and increases in pressure. This expansion causes the float to rise upward and compress the warm helium in B. Since B is held at a constant temperature via contact with the ambient, the compressive work done on B will be rejected to the atmosphere through the ideal gas relation:

$$Q_{out} = W_{AB} = m_B R_{He} T_{ambient} \ln\left(\frac{V_{B_{inital}}}{V_{B_{final}}}\right) \tag{4.3}$$

As compartment A is re-cooled, the float will move back down. Heat will be transferred back from the ambient to compartment B and then to A through a work transfer. In effect, the system uses the ambient environment to temporarily store some of the energy that the refrigerator cannot immediately absorb.

A program[4] was written to evaluate the performance of this system. The program searches for the optimum initial and final volume ratios of the two compartments. The results are shown in Table 4.1. The model showed that this system can absorb 10% more heat per total volume (32% more per cold mass) than an undivided closed container. However, the analysis neglected conduction losses, which would decrease the performance. A more detailed model could determine if the advantage can be realized.

[4] The program is named *float.for*. See appendix for listing.

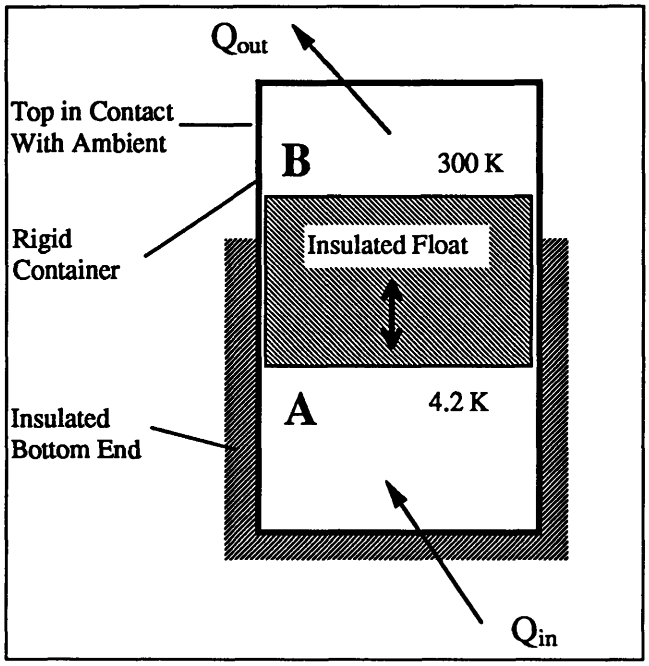


Figure 4.4: Divided Tank With Energy Transfer to Environment

Since this is a novel storage concept, the governing equations are developed below.

The energy balance for the entire system is:

$$\Delta E = m_A(u_{A_2} - u_{A_1}) + m_B(u_{B_2} - u_{B_1}) = Q_{in} - Q_{out} \tag{4.4}$$

where m is mass, u is specific internal energy, 1 denotes the initial state, and 2 denotes the final state.

Modeling B as an ideal gas, its internal energy depends only on temperature. Since it is always at 300 K, its internal energy does not change and equation 4.4 becomes:

$$m_A(u_{A_2} - u_{A_1}) = Q_{in} - Q_{out} \tag{4.5}$$

The energy balance for compartment B is

$$\Delta E = 0 = Q_{out} + W_{AB} = Q_{out} + m_B R_{He} T_{ambient} \ln\left(\frac{V_{B_1}}{V_{B_2}}\right) \tag{4.6}$$

where W_{AB} is the work done by compartment A on compartment B.

Combining equations 4.5 and 4.6, the heat input to the system is:

$$Q_{in} = m_A(u_{A_2} - u_{A_1}) + m_B R_{He} T_{ambient} \ln\left(\frac{V_{B_1}}{V_{B_2}}\right) \tag{4.7}$$

The total volume of compartments A and B is constant:

$$V_{tot} = V_{A_1} + V_{B_1} = V_{A_2} + V_{B_2} \tag{4.8}$$

Defining the increase in volume of compartment A as ΔV_A, using the ideal gas law, and rearranging equations 4.7 and 4.8 , the energy received per total volume of helium is:

$$\frac{Q_{in}}{V_{tot}} = \frac{\rho_{A_1}}{1+\frac{V_{B_1}}{V_{A_1}}}(u_{A_2} - u_{A_1}) + \frac{P_1}{1+\frac{V_{A_1}}{V_{B_1}}}\ln\left(\frac{1}{1-\frac{\Delta V_A}{V_{B_1}}}\right) \tag{4.9}$$

where P_1 is the initial pressure in both compartments.

To simplify the equations, Γ is defined as the initial ratio of volume B to volume A, and ϑ is defined as the ratio of the change in volume A to the initial volume of A:

$$\Gamma = \frac{V_{B_1}}{V_{A_1}}, \qquad \vartheta = \frac{\Delta V_A}{V_{A_1}} \tag{4.10}$$

Finally, the energy stored per total volume is

$$\boxed{\frac{Q_{in}}{V_{tot}} = \frac{\rho_{A_1}}{1+\Gamma}(u_{A_2} - u_{A_1}) + \frac{P_1}{1+\frac{1}{\Gamma}}\ln\left(\frac{1}{1-\frac{\vartheta}{\Gamma}}\right)} \tag{4.11}$$

To evaluate equation 4.11, the initial conditions in compartment A are chosen along with the volume ratios Γ and ϑ. The final conditions in A are found by using the ideal gas equation and mass conservation to obtain:

$$P_2 = \frac{P_1}{1-\frac{\vartheta}{\Gamma}}, \qquad \rho_{A2} = \frac{\rho_{A1}}{1+\vartheta}, \qquad u_{A_2} = u_{P_2,\rho A_2} \tag{4.12}$$

The computer program[4] sweeps through a wide range of Γ, ϑ, and ρ_{A_1} for an initial pressure of 1 atm and a given maximum temperature in A. The optimum values found for T_{max} = 6.0 K are Γ = 0.66 and ϑ = 0.5 with ρ_{A1} = 135 kg/m^3 (x_{A1} = 0.0).

4.4 CONCLUSIONS

The modeling of buffer systems resulted in the following conclusions:

- A buffer system can temporarily store the energy deposited in a SMES magnet during charging and discharging.
- The buffer system allows a smaller refrigerator to be used by "averaging" the transient heat loads over a long period of time.
- Helium is the best buffer material since it changes phase and has a high specific heat in the temp range of interest (~ 4.2 to 6 K).
- Boiling helium at 1 atm retains a constant temperature but requires a large volume to store the vapor that is produced.
- Heating two-phase helium in a closed container requires a temperature rise, but is a simple, practical buffer system.
- An evacuated tank of low pressure helium was found to absorb the most heat, but may be difficult to implement.
- A novel buffer concept was proposed. The system uses the ambient environment to store some of the energy.
- An idealized model of the proposed buffer showed a 10% increase in the energy stored per total volume (32% per cold mass) over a closed container, but the concept should be further studied to see if a real system would be practical.

5. TRANSIENT MODELING OF A SMES REFRIGERATOR COLD END

This chapter contains ten sections. Section 1 gives the reasons for modeling the transient behavior of a SMES refrigeration system. Section 2 describes the system that was modeled. Section 3 develops the governing equations. Section 4 gives the finite difference approximation to the governing equations. Section 5 shows that iteration is necessary for the solution to be consistent with the pump characteristics. Section 6 describes the computer program. Section 7 lists the input parameters that were used. Section 8 presents results. Section 9 discusses the results. Section 10 summarizes the conclusions.

5.1 MOTIVATION BEHIND THE TRANSIENT MODELING

When a SMES magnet is charged or discharged, AC losses deposit heat energy in the conductor. The energy is deposited much faster than the refrigeration system can react. Therefore, the refrigerator will experience a transient period while it adjusts to the extra heat load.

During the transient period, the temperature of the helium in the magnet will rise. If the system is not properly designed, the rising temperature could jeopardize the magnet stability or push the refrigerator off its design point and reduce the cooling power at the time it is most needed. If the transient load is large, liquid helium in the system could vaporize so quickly that it must be vented to prevent over-pressurization. Losing this helium is undesirable, especially on shipboard applications where it is difficult to replenish.

The transient cooling of SMES magnets is not completely understood. The complexity of the refrigerator makes it difficult to predict the transient performance. Systems which require multiple discharges in a short time are even less understood since the system may not have fully recovered between discharges.

Although transient models of SMES magnets have been made [van der Linden, Wang], the models do not usually include other components in the refrigeration system. A model which includes characteristics of the pump and buffer could help to better understand the factors that are important to cooling.

This chapter discusses a transient model that includes characteristics of the pump, heat exchangers, and buffer. The model simulates a specific SMES cooling configuration but also yields results that apply to general cooling systems.

5.2 DESCRIPTION OF A POSSIBLE COOLING CONFIGURATION

Figure 5.1 shows the cooling configuration that was modeled. The system contains helium in three separate sub-systems: the recirculator loop, the buffer tank, and the refrigerator pool.

The recirculator loop (ABCD) contains pressurized helium that circulates through the magnet. The helium removes heat from the magnet (Q'_{magnet}) and rejects heat to the buffer tank through two heat exchangers (Q'_{load}).

The buffer tank is a closed tank containing two-phase helium. It accepts the heat from the recirculator loop (Q'_{load}) while allowing the refrigerator to remove heat (Q'_{ss}). During steady state operation, these heat loads are equal. However, during transient operation, Q'_{load} is greater than Q'_{ss} and the buffer absorbs the excess heat that the refrigerator cannot immediately absorb.

The refrigerator pool contains two-phase helium that is produced by the refrigerator. Liquid from this tank runs through a heat exchanger inside the buffer tank. The liquid absorbs heat from the buffer (Q'_{ss}) and then returns to the refrigerator to be re-cooled. This model assumes that the refrigerator is sized to remove the steady state load. Therefore, a larger Q'_{ss} will require, a larger refrigerator.

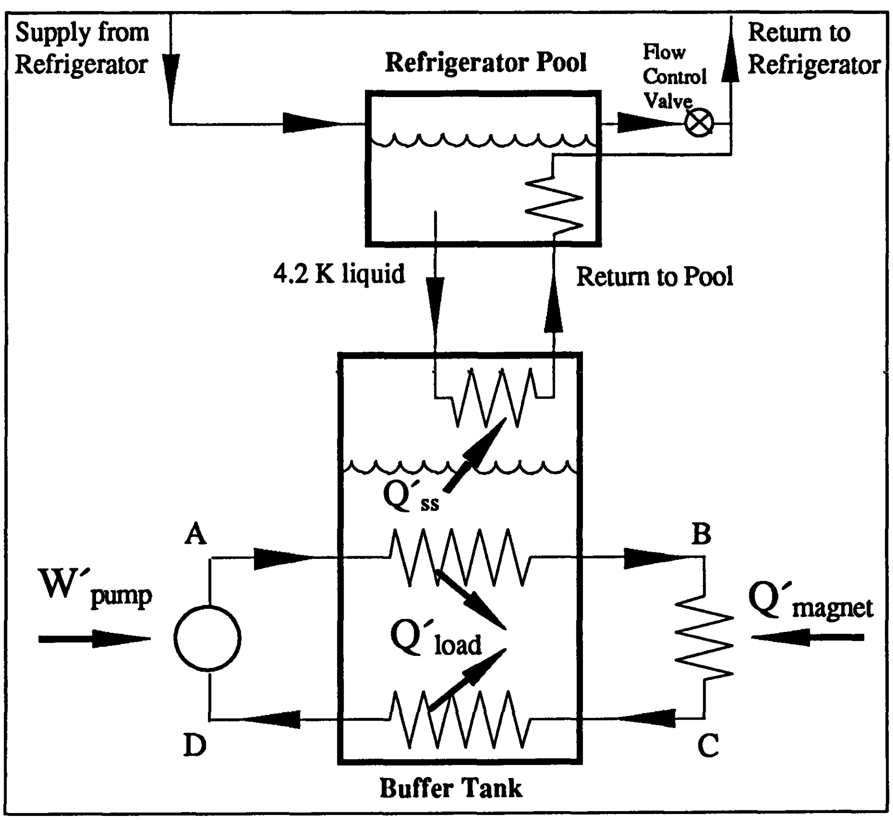


Figure 5.1: A Possible SMES Cooling Configuration

This cooling configuration has several desirable features:

- First, the buffer tank decouples the magnet and refrigerator. The refrigerator can operate at a constant cooling power while the magnet heat load fluctuates.

- Second, a large transient heat load can be absorbed in the buffer tank without heating the refrigerator pool. Heating the pool would push the refrigerator off its design point and decrease the amount of cooling that could be provided.

- Third, the fixed-volume buffer tank can absorb heat without vaporizing helium and requiring large storage tanks to hold the vapor.

- Finally, the separate volumes of helium can be isolated from each other if the magnet quenches. Normally, a quench will add so much heat to the system that helium must be vented to prevent over-pressurization. In this system, the recirculator loop could be vented while the helium in the buffer and the refrigerator is preserved. In a case where the buffer tank also overheats, the helium in the refrigerator could still be saved by stopping the flow from the pool to the buffer tank.

5.3 DEVELOPMENT OF GOVERNING EQUATIONS

5.3.1 MAGNET AND RECIRCULATOR LOOP HEAT EXCHANGERS

This section develops the equations for transient conditions in both the CICC magnet and the recirculator loop heat exchangers. Like the steady state equations, the transient equations are developed using a differential element of the recirculator loop.

Figure 5.2 shows a generalized segment of the loop between nodes at a point x and the next node at point x + Δx. At each node, the temperature and pressure are required to fix the thermodynamic state. In addition, the mass flow rate is required since the segment could gain or lose mass (unlike the steady state case). Therefore, each node has three variables associated with it: T , P and m′. For each segment, three equations are needed to relate these three unknown variables. For a small segment size (Δx), the entire segment can be assumed to have the same fluid properties as those at node x. Neglecting the heat capacity of the conductor material, the energy balance for the helium in the segment is:

$$\frac{dE}{dt}\cdot\frac{1}{\Delta x}=q'-w'-\frac{d}{dx}\left\{m'\left(h+\frac{v^2}{2}+g\cdot z\right)\right\} \tag{5.1}$$

where E is the total energy contained in the control volume, q′ is the heat load per unit length, w′ is the work transfer per unit length, m′ is the mass flow rate, h is the specific enthalpy, v is the velocity, and z is the elevation.

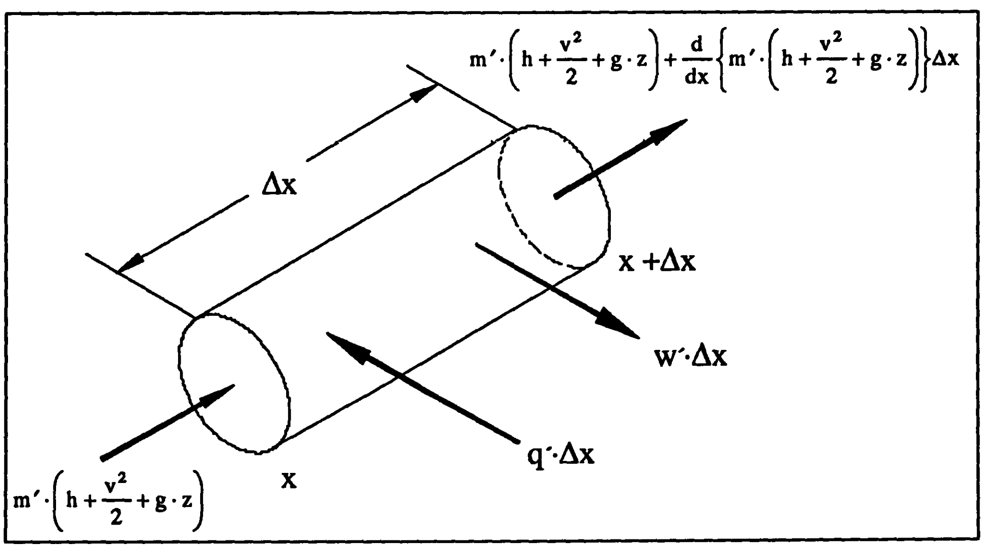


Figure 5.2: Energy Flow Through Recirculator Loop Segment

Noting there is no work transfer to the control volume, neglecting the changes in kinetic and potential energy, and substituting for the energy stored in the conductor, equation 5.1 becomes:

$$\frac{1}{\Delta x}\frac{\partial}{\partial t}\{M_{CV}\cdot u\}=q'-\frac{\partial}{\partial x}\{m'\cdot h\} \tag{5.2}$$

where M_{CV} is the total mass inside the control volume and u is it's specific internal energy. Noting that M_{CV} = (density)·(cross sectional area)·(Δx) = $\rho\cdot A\cdot\Delta x$ and taking derivatives, equation 5.2 becomes:

$$u\cdot\frac{\partial M_{CV}}{\partial t}\frac{1}{\Delta x}+A\cdot\rho\cdot\frac{\partial u}{\partial t}=q'-h\cdot\frac{\partial m'}{\partial x}-m'\cdot Cp\cdot\frac{\partial T}{\partial x}-m'\cdot\left.\frac{\partial h}{\partial P}\right|_T\cdot\frac{\partial P}{\partial x} \tag{5.3}$$

where the cross-sectional area A is assumed to be constant for the segment.

The mass conservation equation for the segment is:

$$\frac{\partial M_{CV}}{\partial t}=(-)\frac{\partial m'}{\partial x}\Delta x \tag{5.4}$$

Substituting for $M_{CV} = \rho\cdot A\cdot\Delta x$:

$$A\frac{\partial\rho}{\partial t}+\frac{\partial m'}{\partial x}=0 \tag{5.5}$$

$$A\left.\frac{\partial\rho}{\partial T}\right|_P\frac{\partial T}{\partial t}+A\left.\frac{\partial\rho}{\partial P}\right|_T\frac{\partial P}{\partial t}+\frac{\partial m'}{\partial x}=0 \tag{5.6}$$

Once again, the pressure drop in the control volume is assumed to follow the relationship:

$$\frac{\partial P}{\partial x}=\frac{-f\cdot m'^2}{2\cdot\rho\cdot D_h\cdot A^2} \tag{5.7}$$

Finally, after substituting equation 5.4 into equation 5.3, and rewriting equations 5.6 and 5.7, the three equations in T, P, and m′ are:

$$\boxed{\begin{aligned}
&\frac{P}{\rho}\cdot\frac{\partial m'}{\partial x}+A\cdot\rho\left.\frac{\partial u}{\partial T}\right|_P\frac{\partial T}{\partial t}+A\cdot\rho\left.\frac{\partial u}{\partial P}\right|_T\frac{\partial P}{\partial t}+m'\cdot Cp\frac{\partial T}{\partial x}+m'\left.\frac{\partial h}{\partial P}\right|_T\frac{\partial P}{\partial x}-q'=0\\
&A\left.\frac{\partial \rho}{\partial T}\right|_P\frac{\partial T}{\partial t}+A\left.\frac{\partial \rho}{\partial P}\right|_T\frac{\partial P}{\partial t}+\frac{\partial m'}{\partial x}=0\\
&\frac{\partial P}{\partial x}+\frac{f\cdot m'^2}{2\cdot\rho\cdot D_h\cdot A^2}=0
\end{aligned}} \tag{5.8}$$

Equations 5.8 apply to both the conductor passage and the heat exchangers. The heat load per unit length (q') is assumed to be known for the conductor. However, the heat load in the heat exchangers can be expressed as

$$q'_{HX}=ua\cdot(T_{Buffer}-T) \tag{5.9}$$

where ua is the overall heat transfer coefficient per unit length of heat exchanger and T_{Buffer} is the temperature of the buffer tank.

Therefore, the equations that govern the helium flow in the heat exchangers are:

$$\boxed{\begin{aligned}
&\frac{P}{\rho}\cdot\frac{\partial m'}{\partial x}+A\cdot\rho\left.\frac{\partial u}{\partial T}\right|_P\frac{\partial T}{\partial t}+A\cdot\rho\left.\frac{\partial u}{\partial P}\right|_T\frac{\partial P}{\partial t}+m'\cdot Cp\frac{\partial T}{\partial x}+m'\left.\frac{\partial h}{\partial P}\right|_T\frac{\partial P}{\partial x}+ua(T-T_{Buffer})=0\\
&A\left.\frac{\partial \rho}{\partial T}\right|_P\frac{\partial T}{\partial t}+A\left.\frac{\partial \rho}{\partial P}\right|_T\frac{\partial P}{\partial t}+\frac{\partial m'}{\partial x}=0\\
&\frac{\partial P}{\partial x}+\frac{f\cdot m'^2}{2\cdot\rho\cdot D_h\cdot A^2}=0
\end{aligned}} \tag{5.10}$$

5.3.2 BUFFER TANK

The buffer tank is modeled as an insulated closed container with helium inside. The helium temperature will rise if the tank receives heat from the recirculator loop faster than the refrigerator removes heat. It is important to track the temperature rise since the heat exchanger performance depends on the tank temperature (equation 5.8).

The temperature of the tank can be found by first relating the net energy added to the change of internal energy.

$$dQ_{net} = m_{Buffer} \cdot du \tag{5.11}$$

where m_{Buffer} is the mass of helium in the buffer tank.

If the initial condition is known, the density and the initial specific internal energy can be found. The final specific internal energy can then be found:

$$u_{final} = u_{initial} + \frac{1}{m_{Buffer}} \cdot \int dQ_{net} \tag{5.12}$$

Since the tank volume and the mass in the tank are constant, the overall density is constant. Using the density and the final internal energy, the new temperature in the tank can be found: $T_{final} = T(u_{final}, \rho)$.

5.4 FINITE DIFFERENCE APPROXIMATION OF GOVERNING EQUATIONS

The transient equations contain both space and time derivatives. As in the steady state model of Chapter 2, the derivatives are approximated using a finite difference method. However, the transient model uses the more stable backward Euler approximation instead of a forward Euler approximation [Strang, 563]. These derivatives are estimated as:

$$
\begin{aligned}
\frac{\partial T}{\partial x} &= \frac{T_{x,t} - T_{x-\Delta x,t}}{\Delta x}, & \frac{\partial T}{\partial t} &= \frac{T_{x,t} - T_{x,t-\Delta t}}{\Delta t} \\
\frac{\partial P}{\partial x} &= \frac{P_{x,t} - P_{x-\Delta x,t}}{\Delta x}, & \frac{\partial P}{\partial t} &= \frac{P_{x,t} - P_{x,t-\Delta t}}{\Delta t} \\
\frac{\partial m'}{\partial x} &= \frac{m'_{x,t} - m'_{x-\Delta x,t}}{\Delta x}, & \frac{\partial m'}{\partial t} &= \frac{m'_{x,t} - m'_{x,t-\Delta t}}{\Delta t}
\end{aligned}
\qquad (5.13)
$$

where x denotes the node location, t denotes the time, Δx is the distance between nodes and Δt is the time step. The subscript $x - \Delta x$ denotes the node immediately before x, and $t - \Delta t$ denotes the previous time step.

Using equations 5.13 and 5.8, the finite difference approximations of the governing equations for the CICC magnet are:

$$
\begin{aligned}
&\left[\frac{m' \cdot Cp}{\Delta x} + \frac{\rho \cdot A}{\Delta t} \cdot \left.\frac{\partial u}{\partial T}\right|_P\right] \cdot T_{x,t} + \left[\frac{m'}{\Delta x} \cdot \left.\frac{\partial h}{\partial P}\right|_T + \frac{\rho \cdot A}{\Delta t} \cdot \left.\frac{\partial u}{\partial P}\right|_T\right] \cdot P_{x,t} + \left[\frac{P}{\rho \cdot \Delta x}\right] \cdot m'_{x,t} \\
&\quad + \left[\frac{(-)m' \cdot Cp}{\Delta x}\right] \cdot T_{x-\Delta x,t} + \left[\frac{(-)m'}{\Delta x} \cdot \left.\frac{\partial h}{\partial P}\right|_T\right] \cdot P_{x-\Delta x,t} + \left[\frac{(-)P}{\rho \cdot \Delta x}\right] \cdot m'_{x-\Delta x,t} \\
&\quad = \left[\frac{\rho \cdot A}{\Delta t} \cdot \left.\frac{\partial u}{\partial T}\right|_P\right] \cdot T_{x,t-\Delta t} + \left[\frac{\rho \cdot A}{\Delta t} \cdot \left.\frac{\partial u}{\partial P}\right|_T\right] \cdot P_{x,t-\Delta t} + q' \\[2ex]
&\left[\frac{A}{\Delta t} \left.\frac{\partial \rho}{\partial T}\right|_P\right] \cdot T_{x,t} + \left[\frac{A}{\Delta t} \left.\frac{\partial \rho}{\partial P}\right|_T\right] \cdot P_{x,t} + \left[\frac{1}{\Delta x}\right] \cdot m'_{x,t} + \left[\frac{(-)1}{\Delta x}\right] \cdot m'_{x-\Delta x,t} \\
&\quad = \left[\frac{A}{\Delta t} \left.\frac{\partial \rho}{\partial T}\right|_P\right] \cdot T_{x,t-\Delta t} + \left[\frac{A}{\Delta t} \left.\frac{\partial \rho}{\partial P}\right|_T\right] \cdot P_{x,t-\Delta t} \\[2ex]
&\left[\frac{1}{\Delta x}\right] \cdot P_{x,t} + \left[\frac{-1}{\Delta x}\right] \cdot P_{x-\Delta x,t} = \frac{-f \cdot m' \cdot |m'|}{2 \cdot \rho \cdot D_h \cdot A^2}
\end{aligned}
\qquad (5.14)
$$

The equations for the recirculator loop heat exchangers are found by substituting for the heat load (q′) using Equation 5.9:

$$\begin{aligned}
&\left[\frac{m'\cdot Cp}{\Delta x}+\frac{\rho\cdot A}{\Delta t}\cdot\left.\frac{\partial u}{\partial T}\right|_P+ua\right]\cdot T_{x,t}+\left[\frac{m'}{\Delta x}\cdot\left.\frac{\partial h}{\partial P}\right|_T+\frac{\rho\cdot A}{\Delta t}\cdot\left.\frac{\partial u}{\partial P}\right|_T\right]\cdot P_{x,t}+\left[\frac{P}{\rho\cdot\Delta x}\right]\cdot m'_{x,t}\\
&\quad+\left[\frac{(-)m'\cdot Cp}{\Delta x}\right]\cdot T_{x-\Delta x,t}+\left[\frac{(-)m'}{\Delta x}\cdot\left.\frac{\partial h}{\partial P}\right|_T\right]\cdot P_{x-\Delta x,t}+\left[\frac{(-)P}{\rho\cdot\Delta x}\right]\cdot m'_{x-\Delta x,t}\\
&\qquad=\left[\frac{\rho\cdot A}{\Delta t}\cdot\left.\frac{\partial u}{\partial T}\right|_P\right]\cdot T_{x,t-\Delta t}+\left[\frac{\rho\cdot A}{\Delta t}\cdot\left.\frac{\partial u}{\partial P}\right|_T\right]\cdot P_{x,t-\Delta t}+ua\cdot T_{Buffer}\\
&\left[\frac{A}{\Delta t}\left.\frac{\partial\rho}{\partial T}\right|_P\right]\cdot T_{x,t}+\left[\frac{A}{\Delta t}\left.\frac{\partial\rho}{\partial P}\right|_T\right]\cdot P_{x,t}+\left[\frac{1}{\Delta x}\right]\cdot m'_{x,t}+\left[\frac{(-)1}{\Delta x}\right]\cdot m'_{x-\Delta x,t}\\
&\qquad=\left[\frac{A}{\Delta t}\left.\frac{\partial\rho}{\partial T}\right|_P\right]\cdot T_{x,t-\Delta t}+\left[\frac{A}{\Delta t}\left.\frac{\partial\rho}{\partial P}\right|_T\right]\cdot P_{x,t-\Delta t}\\
&\left[\frac{1}{\Delta x}\right]\cdot P_{x,t}+\left[\frac{-1}{\Delta x}\right]\cdot P_{x-\Delta x,t}=\frac{-f\cdot m'\cdot|m'|}{2\cdot\rho\cdot D_h\cdot A^2}
\end{aligned}\tag{5.15}$$

If the conditions in the heat exchangers are known, they can be used to find the amount of heat being added to the buffer tank. Each segment of the heat exchanger adds heat to the buffer at a different rate:

$$\Delta q'_x = ua\cdot(T_x - T_{Buffer})\cdot\Delta x \tag{5.16}$$

The total rate of heat rejection to the buffer is the sum over all segments of both heat exchangers:

$$Q'_{load} = \sum_{HX1} ua_{HX1}(T_x - T_{Buffer})\Delta x_{HX1} + \sum_{HX2} ua_{HX1}(T_x - T_{Buffer})\Delta x_{HX2} \tag{5.17}$$

The net heat addition to the buffer tank during the time step Δt is:

$$Q_{net} = (Q'_{load} - Q'_{ss})\cdot\Delta t \tag{5.18}$$

The new buffer tank temperature can be found by using the density and the new specific internal energy: $T_t = T(\rho, u_t)$. Using equation 5.12, the new internal energy is:

$$u_t = u_{t-\Delta t} + \frac{Q_{net}}{m_{Buffer}} \tag{5.19}$$

5.5 PUMP CHARACTERISTICS AND OPERATING POINT

The model includes characteristics of the recirculator pump. The pump not only influences the refrigerator heat load (Chapter 3), it also determines the boundary conditions of the recirculator loop. The model must find the mass flow rate that matches the pressure drop around the loop with the pressure rise supplied by the pump. It must also find pump inlet and outlet temperatures that match the isentropic efficiency of the pump.

Figure 5.3 shows the pressure drop of the system and the pressure rise of the pump versus the mass flow rate. As the mass flow rate increases, the pressure drop of the system increases while the pressure rise of the pump decreases. The point where these curves intersect is where the system will operate.

While the pump curve is usually well known, the system curve can depend on several factors. Since the properties of helium vary widely at low temperatures, the system curve is dependent on the conditions in the system. Changing the heat input from the magnet can alter the ΔP versus m' behavior of the system. Therefore, an iteration is required to find the operating point each time the conditions change.

The computer model iterates to find the operating point. A mass flow rate is chosen and the pressure drop around the loop is calculated (location 1). This pressure drop is used to find the mass flow rate that is required for the pump to provide the same pressure rise (1a). A weighted average of these two mass flow rates (2) is used as the next guess. Once again, the pressure drop around the loop is calculated and the corresponding pump mass flow rate is found (2a). The weighted average of these mass flow rates (3) becomes the next guess and the iteration continues until the operating point is found.

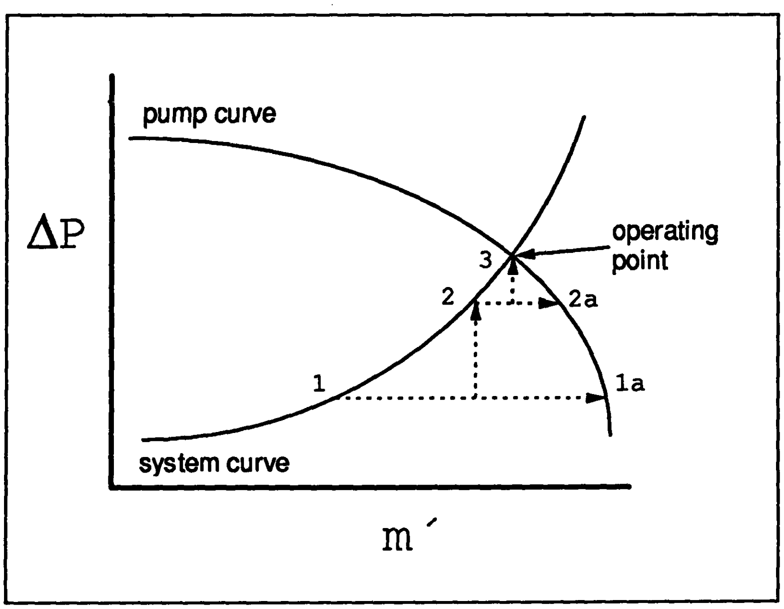


Figure 5.3: Location of Operating Point on Pressure Change versus Mass Flow Plot

5.6 DESCRIPTION OF COMPUTER PROGRAM

Figure 5.4 shows a simplified diagram of the system that the computer program simulates. The recirculator loop (ABCD) contains the high pressure helium that cools the magnet and rejects heat to the buffer tank. The refrigerator removes heat from the tank at the steady state rate. The model breaks the loop into segments by placing nodes throughout the magnet and the two heat exchangers. Equations 5.14 and 5.15 are used, along with an iteration routine, to find the conditions in the loop as the magnet heat load changes.

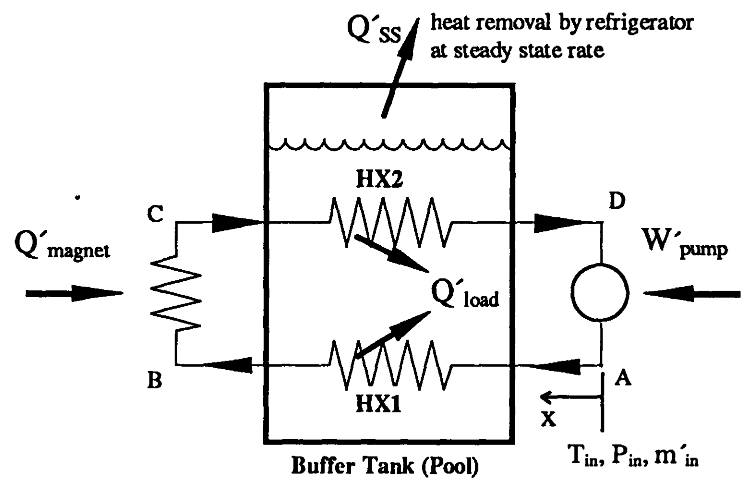


Figure 5.4: Recirculator Loop Cooling Configuration

The core of the computer program is a routine that solves equations 5.14 and 5.15 for given loop inlet conditions (T_{in}, P_{in}, m'_{in}). This routine puts the equations into matrix form: A x = b. The A matrix contains the terms in the square brackets on the left side of equations 5.14 and 5.15. The b vector contains all terms on the right side of the equals signs. The fluid properties in A and b are found using guesses for the conditions in the loop. The program then solves for the x vector, which contains the temperature, pressure, and mass flow rate at all nodes. Once x is found, it is used to update the fluid properties in A and b. A new x vector is then found. These corrections continue until the solution converges.

The routine can be used to find the conditions throughout the loop for a given set of inlet conditions. The "trick" is to vary the inlet conditions until the solution is consistent with the pump characteristics. The iteration is described below.

The program assumes that the system starts in steady state before any transient loads are applied. For steady operation, one of the three variables at location A (T_{in}, P_{in}, m'_{in}) can be chosen while the other two are found by iteration. A value is chosen for the pressure since it can be independently set by the amount of helium that is put in the fixed-volume, closed loop.

To start the steady state iteration, P_{in} is chosen and guesses are made for T_{in} and m'_{in}. The conditions in the loop are then found. Using the iteration scheme described in section 5.5, m'_{in} is varied until the pressure drop around the loop is the same as the pressure rise provided by the pump. At the same time, the conditions at D are used to find the temperature at A that matches the isentropic efficiency of the pump. This new temperature is used for the next guess of T_{in} in successive iterations. Since the m'_{in} and T_{in} iterations are relatively stable, they can be done at the same time to speed the convergence.

For transient operation, P_{in} must also be found by iteration. The same iteration as in the steady state case is used to find m'_{in} and T_{in}. An outer loop is added to vary P_{in} until the mass flow rates at A and D are equal. These mass flow rates should be equal since the pump is assumed to have such a small internal volume that it cannot store mass.

Table 5.1 summarizes the iteration methods used to find the T_{in}, P_{in}, and m'_{in} that give a solution consistent with the pump characteristics. A flow chart and description of the computer subroutines are included in Appendix E along with a listing of the program (*TRANS6.EXE*).

Parameter	Criteria for Iteration
m'_{in}	vary until $\Delta P_{system} = \Delta P_{pump}$
T_{in}	consistent with isentropic efficiency and conditions at location D
P_{in}	vary until $m'_{in} = m'_D$

Table 5.1: Iteration Criteria

5.7 INPUT PARAMETERS

Computer simulations were run to investigate two effects. The first effect is the influence of parallel or series flow through the magnet. The second effect is the influence of the initial pressure in the recirculator loop.

To study the effect of parallel or series flow, two flow configurations were simulated. The first simulation was taken to be the "series flow" case by considering it to consist of one pass through a magnet in which all conductors are connected end-to-end. A "parallel flow" case was simulated by considering the same conductors to be split into two parallel paths, each half of the total length. For the computer to simulate the parallel flow, three parameters were changed. The passage length was cut in half, the effective cross sectional area was doubled, and the heat load per unit length was doubled. Table 5.2 summarizes the component characteristics that were used. Note that the same heat exchangers and buffer tank were used in each simulation.

To study the effect of the initial pressure in the recirculator loop, the parallel flow configuration was re-run using a lower initial loop pressure. The two pressures used were 8 atm and 5 atm.

Table 5.3 shows the pump characteristics used for all three simulations. Table 5.4 lists the initial conditions. Figure 5.6 shows how the total heat load in the magnet was varied to simulate a series of five magnet discharges. This heat load was assumed to be uniformly distributed over the passage length. A recovery period was included after the five discharges.

	Magnet (mag)		Heat Exchanger 1 (HX1)	Heat Exchanger 2 (HX2)	Buffer Tank
	SERIES	PARALLEL			
L	20 m	10 m	5 m	10 m	---
A	$0.1 \times 10^{-3}\,m^2$	$0.2 \times 10^{-3}\,m^2$	$3.9 \times 10^{-3}\,m^2$	$3.9 \times 10^{-3}\,m^2$	---
Dh	0.4×10^{-3} m	0.4×10^{-3} m	10×10^{-3} m	10×10^{-3} m	---
f	0.15	0.15	0.15	0.15	---
ua	---	---	5 W/m	5 W/ m	---
n	31 nodes	31 nodes	11 nodes	11 nodes	---
V_{tank}	---	---	---	---	$0.01\,m^3$
x_{init}	---	---	---	---	0.09

where: L = Length of Each Passage
A = Total Cross Sectional Area of the Flow
Dh = Hydraulic Diameter
f = Friction Factor
ua = Overall Heat Transfer Coefficient Per Unit Length
n = Number of Nodes Used in Simulation
V_{tank} = Volume of Buffer Tank
x_{init} = Initial Quality of Buffer Tank

Table 5.2: Component Characteristics Used in the Simulation

isentropic efficiency *	mass flow rate (g/s) vs. pressure rise (atm) **
60 %	$m = -1.29\Delta P^2 + 0.61\Delta P + 6$

* actual system would have efficiency that varies with the operating condition

** rough fit of data from a helium pump at 100 rpm. [Lue]

Table 5.3: Pump Characteristics Used in the Simulation

	SERIES FLOW	PARALLEL FLOW
Initial Magnet Heat Load (q')	0.03 W/m	0.06 W/m
Initial Pressure at Pump Exit ($P_{initial}$)	8 atm	8 atm and 5 atm
Initial Temperature of Buffer Tank ($T_{Buffer\_initial}$)	4.3 K	4.3 K
Initial Quality of Buffer Tank (x_{init})	0.09	0.09

Table 5.4: Initial (Steady State) Conditions in the Recirculator Loop

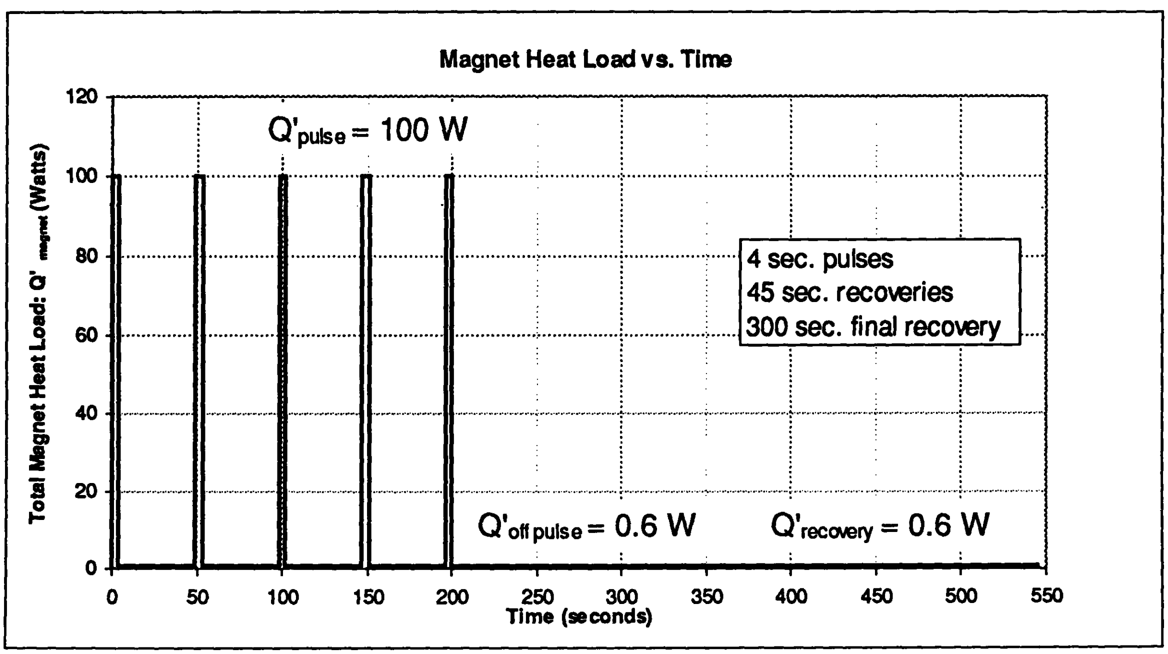


Figure 5.5: Assumed Magnet Heat Load Profile

5.8 RESULTS

Table 5.5 shows the steady state values of the system, which existed immediately before the transient heat load was applied. Figure 5.6 shows the heat load on the buffer tank and the pump power for the three simulations. Figure 5.7 plots the temperature at the magnet exit. Figure 5.8 plots the buffer tank temperature. Figure 5.9 shows the pressure at the pump inlet and outlet. Figures 5.10 and 5.11 show the mass flow rate at the magnet inlet and exit.

	SERIES FLOW	PARALLEL FLOW	PARALLEL FLOW
Initial Pressure at Pump Exit ($P_{in\ initial}$)	8 atm	8 atm	5 atm
Steady State Mass Flow Rate (m')	5.64 g/s	6.05 g/s	6.06 g/s
Steady State Magnet Heat Load (Q'_{magnet})	0.6 W	0.6 W	0.6 W
Steady State Pumping Power (W'_{pump})	5.34 W	0.89 W	0.81 W
Steady State Refrigerator Heat Load (Q'_{ss})	5.93 W	1.49 W	1.41 W
Steady State Refrigerator Heat Load as a Percentage of the Heat Removed From the Magnet	983 %	248 %	235 %

Table 5.5: Steady State Values Found by Computer Iteration

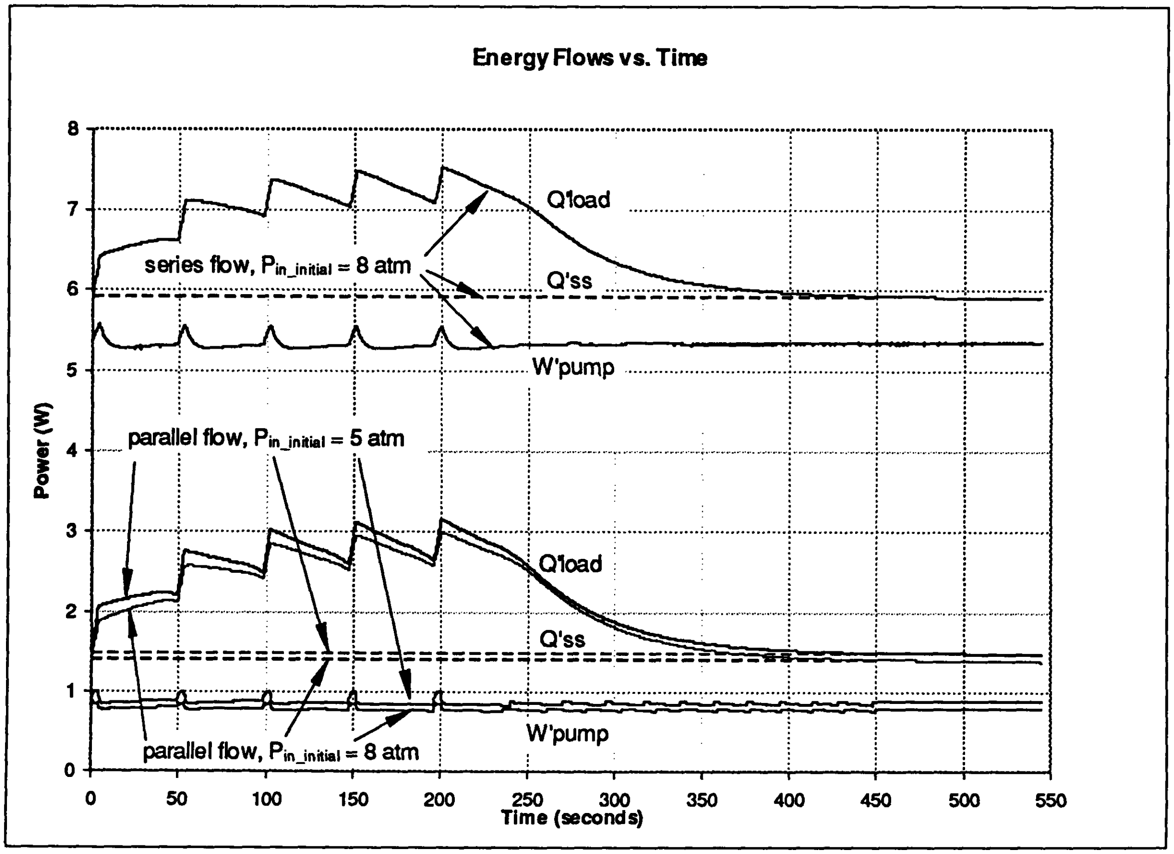


Figure 5.6: Buffer Tank Heat Loads and Pumping Power versus Time

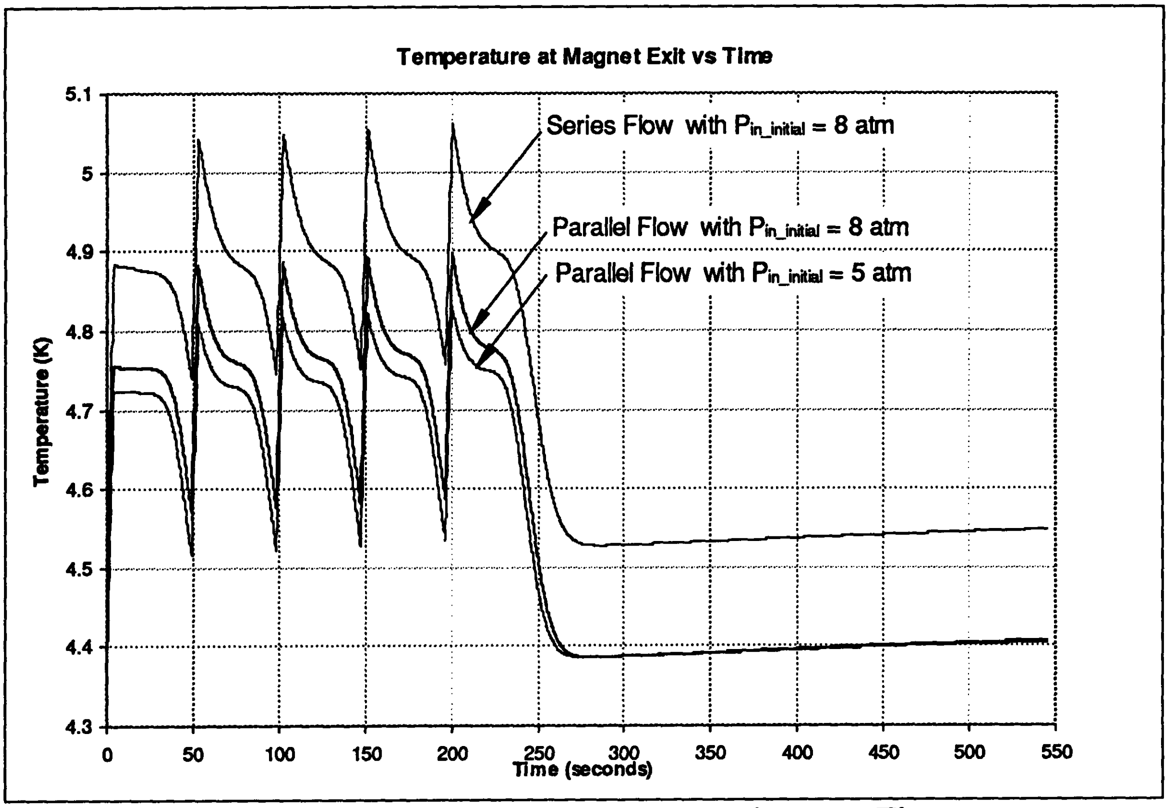


Figure 5.7: Temperature at Magnet Exit versus Time

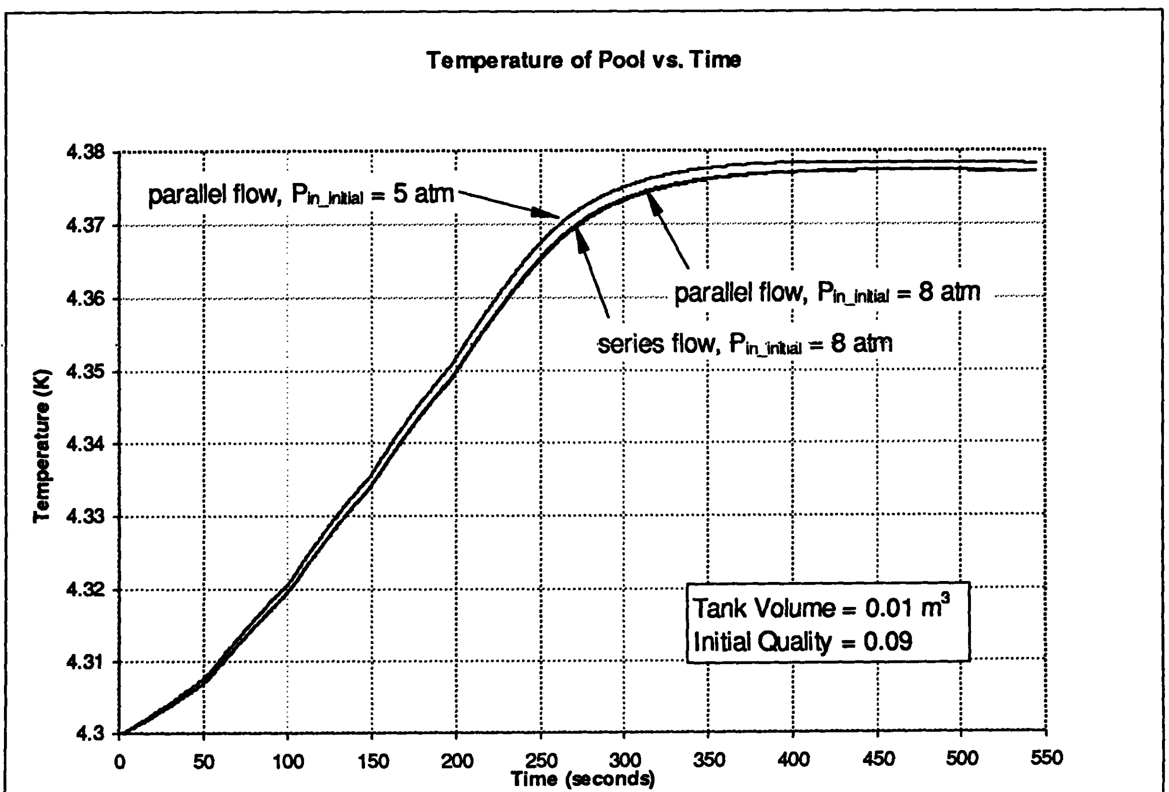


Figure 5.8: Buffer Tank Temperature versus Time

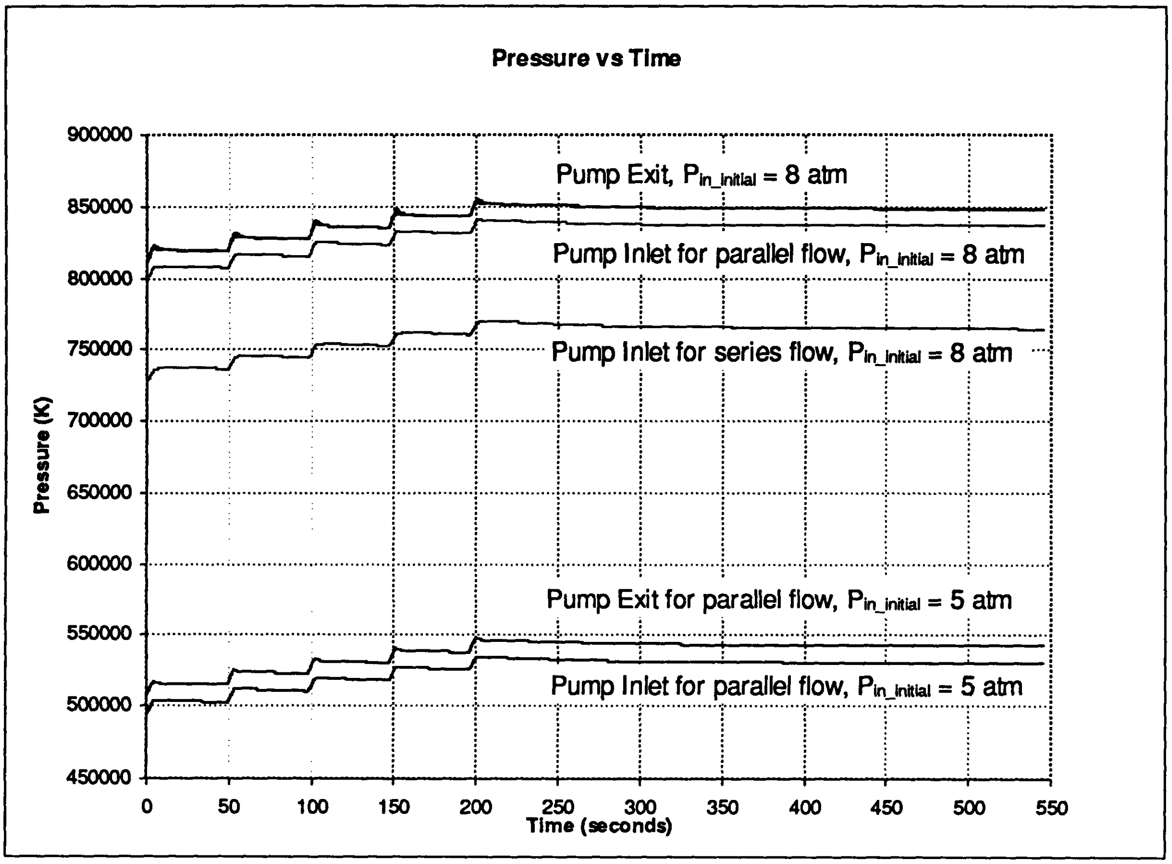


Figure 5.9: Pressures at Pump Inlet and Exit versus Time

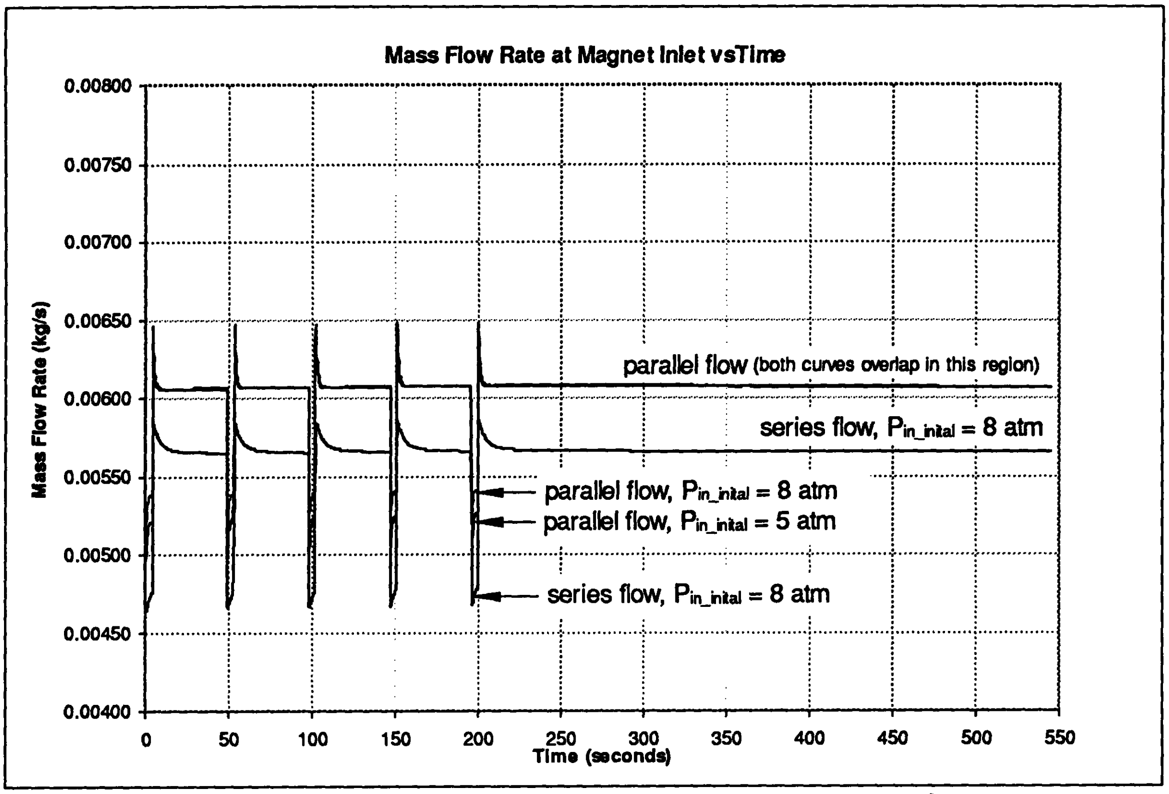


Figure 5.10: Mass Flow Rate at Magnet Inlet versus Time

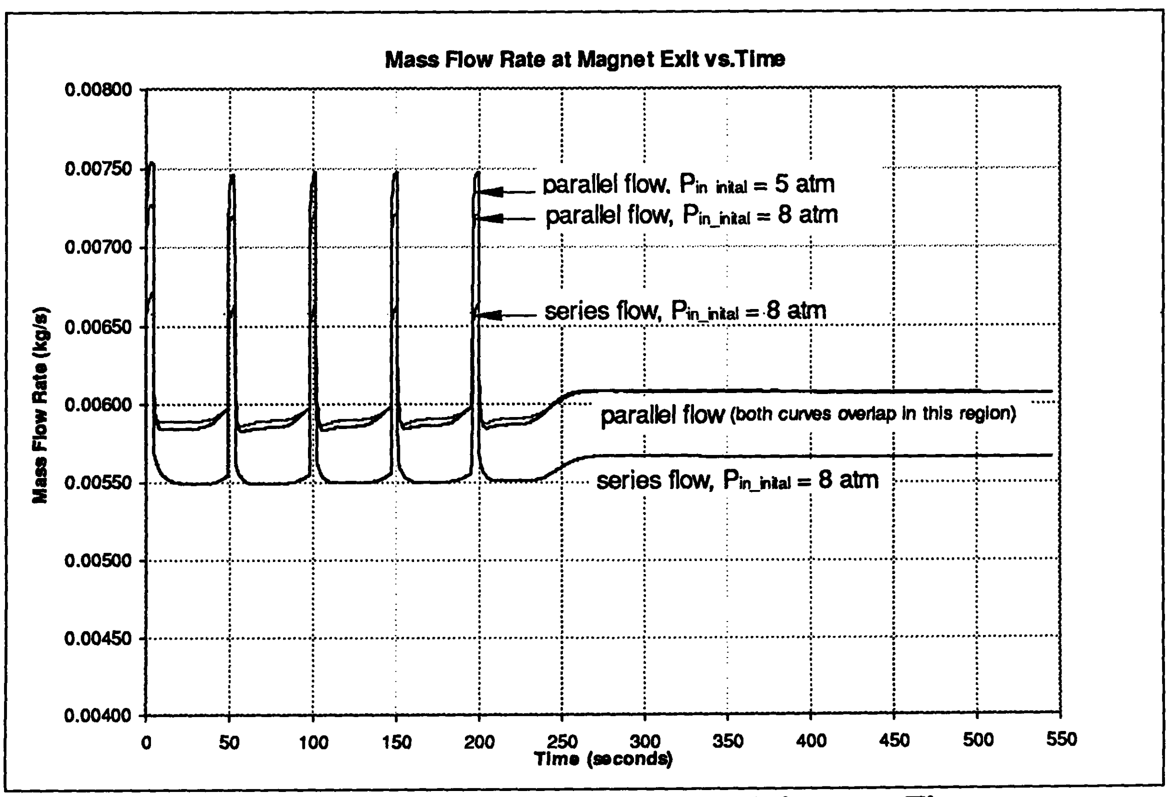


Figure 5.11: Mass Flow Rate at Magnet Exit versus Time

5.9 DISCUSSION OF RESULTS

The input parameters that were used are not necessarily values that would be found in an actual SMES system. The simulations were run only to demonstrate a method for modeling a cooling system that includes the characteristics of the pump, heat exchangers, and buffer. The results can be used to make generalizations about forced flow cooling of SMES magnets.

Table 5.5 shows that the flow configuration strongly influences the steady state refrigerator load. The series flow configuration requires a refrigerator that is 4 times larger than for the parallel flow configurations. The pumping power accounts for this difference. A larger pumping power is required to force the helium through the longer, narrower passage of the series system. Since the refrigerator is assumed to be sized for the steady state load, the series configuration will require a larger, more expensive refrigerator.

Figure 5.6 shows that the transient heat load on the buffer tank (Q'_{load}) is related to the steady state heat load (Q'_{ss}). The shape of the heat load curve is similar for the parallel flow and series flow systems. The curves are just offset by the difference in their steady state heat load, which is caused by a difference in steady state pumping power.

Figure 5.6 also shows that the initial pressure in the recirculator loop can influence the buffer tank heat load (Q'_{load}). For the same parallel flow configuration, an 8 atm initial pressure results in a slightly lower heat load than a 5 atm initial pressure. Once again, this difference is due to a difference in pumping power. The 8 atm helium requires slightly less pumping power since it has a higher density than the 5 atm helium.

Figure 5.7 shows that the series configuration has a higher magnet-exit temperature than either of the parallel configurations The higher frictional losses in the series configuration can account for this difference. These frictional losses act as an extra heat addition to the passage and can therefore increase the temperature.

The initial loop pressure also affects the temperature. For the same parallel flow system, the 5 atm system shows slightly lower temperatures than the 8 atm system. This difference occurs because helium has a higher specific heat at 5 atm than at 8 atm. The higher specific heat means that the temperature will rise less for the same heat input. Therefore, lower pressure systems may allow a greater temperature margin in the magnet.

Figure 5.8 shows that the pool temperature is nearly the same for all three configurations. The values differ by less than 0.005 K, which is probably beyond the accuracy of the model. It may seem that the series configuration should have a higher pool temperature since it's recirculator loop rejects the most heat to the pool. However, it's refrigerator is sized to remove a larger steady state heat load. Therefore, the net heat load on the pool (Q'_{load} - Q'_{ss}) and the pool temperature are nearly the same for each system.

For this system, the buffer tank was probably oversized. The temperature increased by less than 0.1 K after 5 discharges. However, a system requiring 20 magnet discharges would see an increase of about 0.5 K, which could affect the magnet stability.

Figure 5.9 shows that the pressure in the system rises during the transient period. This pressure rise means that energy is stored in the recirculator loop. For this example, there is actually more energy stored in the loop than in the buffer tank:

The extra energy that the magnet adds to the system over the steady state heat load is:

$$(100 \text{ W} - 0.6 \text{ W})(4 \text{ sec/pulse})(5 \text{ pulses}) = 1998 \text{ J}.$$

From Figure 5.6, the energy added to the buffer tank is approximately:

$$(Q'_{load} - Q'_{ss})\Delta t = (1 \text{ W})(300 \text{ sec}) = 300 \text{ J}.$$

Therefore, about 1698 J are stored in the pressure change in the loop. This shows that the loop can store a significant amount of energy in addition to the energy stored in the buffer tank. Part of the buffer mass could actually be placed inside the loop instead of in the buffer tank. This example uses large heat exchangers, which add extra volume to the loop and allow the energy storage. This energy storage technique should be further studied to look at optimum ratios of loop and buffer volumes.

This simulation is unique in that the system was not allowed to fully recover after the last discharge. The buffer tank temperature and the loop pressure never return to the original steady state values. The reason that the values don't return is that the refrigerator is assumed to only remove heat at the non-discharge, steady-state rate. When the magnet heat load is increased, more energy is being added to the system than is being removed. However, the system never removes more heat than is being added. Therefore, the extra energy from the magnet discharge is never removed. It is eventually stored in the buffer tank and recirculator loop as a new equilibrium is established. A real system would either increase the refrigeration power during the discharge or set the steady refrigeration power to be the average of all heat loads, not just the lowest load.

Figures 5.10 and 5.11 show that the mass flow rates in the magnet change when the heat load changes. When the heat load is suddenly increased, the pressure in the magnet increases and mass is driven from the magnet. As the mass leaves, the mass flow rate at the exit becomes higher than at the inlet. The exit mass flow increases while the inlet mass flow rate decreases. This effect explains the sudden dip in the inlet mass flow rate and the sudden rise in the exit mass flow rate during each discharge. After the discharge, mass re-accumulates in the magnet as the mass flow rate becomes higher at the inlet than at the exit.

5.10 CONCLUSIONS

The transient modeling of a recirculator loop SMES cooling configuration resulted in the following conclusions:

- A finite difference technique can be used to model the forced flow cooling of a SMES magnet.
- A cooling scheme that uses a recirculator loop and a buffer tank can decouple the refrigerator cooling load from the transient variations in the magnet heat load.
- Models of the cooling system should include characteristics of the pump, heat exchangers, and buffer tank.
- An iteration technique can be used to find solutions to the governing equations that are consistent with the characteristics of the pump, heat exchangers, and buffer tank.
- The pumping power required to cool the magnet can be reduced by using flow configurations with short passages and large cross-sectional areas. (i.e., parallel flow).
- The reduced pumping power means that a smaller refrigerator can be used with a parallel flow configuration than with a series flow configuration.
- The transient heat load on the buffer tank has a similar shape for each of the configurations simulated. The curves are offset by the difference in steady state pumping power.
- The pumping power during transient conditions shows only a small increase over the steady state pumping power.
- The steady state pumping power could be used as means of determining the average cooling power needed for different cooling schemes without performing a transient simulation.
- The maximum temperature in the magnet during transient conditions can be reduced by using a parallel flow configuration and a lower initial pressure in the recirculator loop.
- A higher pressure in the recirculator loop can reduce the pump power slightly but it will allow a larger temperature increase in the magnet during discharges.
- The helium in the recirculator loop increases in pressure to store some of the transient heat load energy. This effect should be studied further.

6. CONCLUSIONS AND RECOMMENDATIONS

This chapter summarizes the overall conclusions of the project and makes recommendations for areas of further study.

6.1 PROJECT CONCLUSIONS

The project conclusions can be summarized by topic:

Steady State Modeling Of Helium Flow Passages

- A finite difference model can be used to estimate the amount of heat that helium can remove from the flow passage of a cable-in-conduit conductor.
- The results can be generalized for use with passages of different sizes.
- Equation 2.33 can be used along with Figure 2.6 to estimate the maximum amount of heat that can be removed from a given flow passage.
- The steady state entropy generation in the passage can be reduced by using lower inlet pressures and allowing higher temperatures.

Pumping Concerns

- In a recirculator loop cooling configuration, the refrigerator must remove the heat load of both the magnet and pump.
- The pump power can be reduced by using higher efficiency pumps and by re-cooling the helium before it enters the pump.
- The pressure drop across the magnet should be minimized to reduce the pumping power required.
- Lower magnet heat loads will require less helium flow and less pump power.
- To reduce pumping power, systems should be designed to run at much less than the maximum allowable heat load.

Buffer Systems

- A buffer system can temporarily store the energy deposited in a SMES magnet during charging and discharging.

- The buffer system allows a smaller refrigerator to be used by "averaging" the transient heat loads over a long period of time.

- The most practical buffer is probably a closed container containing two-phase helium.

- A novel buffer concept, which uses the ambient environment to store some of the energy, was proposed. An idealized model of this buffer showed that it could store more energy per volume than a closed container, but the concept should be further studied to see if a real system would be practical.

Transient Modeling Of Entire Refrigerator Cold End

- Models of SMES cooling systems should include techniques to make the solution consistent with the characteristics of the pump, heat exchangers, and buffer tank.

- A cooling scheme that uses a recirculator loop and a buffer tank can decouple the refrigerator cooling load from the transient variations in the magnet heat load.

- Due to the reduced pumping power, a smaller refrigerator can be used with parallel flow cooling configuration than a series flow configuration.

- The maximum temperature in the magnet during transient conditions can be reduced by using a parallel flow configuration and a lower initial pressure in the recirculator loop.

- A higher pressure in the recirculator loop can reduce the pump power slightly but it will allow a larger temperature increase in the magnet during discharges.

- The helium in the recirculator loop increases in pressure to store some of the transient heat load energy.

- The steady state pumping power could be used as means of determining the average cooling power needed for different cooling schemes without performing a transient simulation.

6.2 RECOMMENDATIONS

- Since the pumping power can dominate the refrigeration load, future studies should look at ways to cool SMES magnets using low pumping power configurations such as parallel flow or externally forced flow.

- Sophisticated models of SMES magnets currently include electrical properties, material properties, and geometry effects. These models may be improved by adding refrigerator components such as the pump, heat exchangers, and buffer.

- The ability of recirculator loop cooling configurations to store transient loads should be further studied to determine the optimum ratio of loop and buffer tank volumes.

- A more detailed model of the divided-tank buffering system should be made to determine if the advantages shown in the ideal model can be realized.

7. REFERENCES

Cravalho, Ernest G., and Smith, Joseph L., Jr., Engineering Thermodynamics. Reprinted by Authors, 1992.

Hale, J. R., *Cooling Capacity of Flowing Supercritical Helium in GEM CICC Conductor*. Internal Memorandum of the MIT Plasma Fusion Center - June 22, 1992, MIT-GEM-CR-001.

HEPROP by Cryodata. Fortran subroutine version of HEPAK, a computer program for calculating the thermophysical properties of helium. Version 3.30, October 1994.

Jones, J. B, and Hawkins, G. A., Engineering Thermodynamics. 2nd Ed. John Wiley & Sons, Inc., New York, 1986.

Katheder, H., *Thermodynamic considerations of helium flow in cable and conduit superconductors*, Memorandum of The NET team at Garching - February 20, 1991, N/R/3511/1/A.

Long, Alexander E., "Transverse Heat Transfer in a Cable-in-Conduit Conductor with Central Cooling Channel". Masters thesis, Massachusetts Institute of Technology, June 1995.

Lue, J.W., et al *Test of a Cryogenic Helium Pump*. Advances in Cryogenic Engineering, Vol. 27, Plenum Press, New York, 1982.

Press, et al. Numerical Recipes: The Art of Scientific Computing. 2nd Ed. Fortran Version. Cambridge University Press. Cambridge, 1989.

Schoettler, R. M., *Cooling of a small SMES system with forced flow supercritical helium.* Cryogenics, Vol 34. No. 10. 1994

Smith, Joseph L., Jr., *Entropy Flow and Generation in Energy Conversion Systems.* AES-Vol.30/HTD-Vol. 266, Thermodynamics and the Design, Analysis and Improvement of Energy Systems. ASME 1993.

Strang, Gilbert, Introduction to Applied Mathematics. Wellesley-Cambridge Press, Wellesley, MA, 1986.

var der Linden, R.J. and Hoogendoorn, C.J., *Transient cooling of an internally cooled superconducting magnet.* Cryogenics, Vol. 29. March 1989.

Wang, P. W., *Steady-state heat removal of the SMES 0.1 MWhr conductor*. Memorandum of the MIT Plasma Fusion Center, May 13, 1996, NAVSEASMES-MIT-PWang-96-0801-16.

Wang, P. W., *Transient heat removal of the SMES 0.1 MWhr conductor*. Memorandum of the MIT Plasma Fusion Center, August 1, 1996, NAVSEASMES-MIT-PWang-96-0801-16.

APPENDIX A:

EXPRESSING PROPERTY DERIVATIVES IN TERMS OF THOSE IN THE AVAILABLE PROPERTY ROUTINE

To create a finite difference model of helium flow through a long passage, equations 2.15 are used. A computer program by Cryodata [HEPROP] is used to find the fluid properties of helium. However, not all of the property derivatives that are in equations 2.15 are directly available in the program. The derivatives that are available in this software are $\delta P/\delta T|_\rho$ and $\delta P/\delta\rho|_T$. Therefore, this section rewrites the needed property derivatives in terms of the two that are available in HEPROP.

Three property derivatives are needed for equations 2.15. The easiest one to rewrite in terms of the available derivatives is $\delta P/\delta\rho|_T$. Using the reciprocal relation for partial derivatives [Cravalho, 289], this derivative can rewritten as:

$$\left.\frac{\partial \rho}{\partial P}\right|_T = \frac{1}{\left.\frac{\partial P}{\partial \rho}\right|_T} \tag{A.1}$$

The second property derivative that is required is $\delta h/\delta P|_T$. Maxwell relations are used to express this derivative in terms of derivatives that are included in the available software. First, the equation which defines enthalpy is written:

$$dh = T \cdot ds + \frac{1}{\rho} \cdot dP \tag{A.2}$$

Next, the change in entropy is rewritten in terms of temperature and pressure using equation 2.4a:

$$ds = \left.\frac{\partial s}{\partial T}\right|_P dT + \left.\frac{\partial s}{\partial P}\right|_T dP \tag{A.3}$$

Combining equations A.2 and A.3 gives:

$$dh = \left(T \cdot \left.\frac{\partial s}{\partial T}\right|_P\right) \cdot dT + \left(\frac{1}{\rho} + T \cdot \left.\frac{\partial s}{\partial P}\right|_T\right) dP \tag{A.4}$$

which is of the form:

$$dh = \left.\frac{\partial h}{\partial T}\right|_P dT + \cdot \left.\frac{\partial h}{\partial P}\right|_T dP \tag{A.5}$$

By matching terms in equations A.4 and A.5, the partial derivative of enthalpy with respect to pressure at constant temperature can be written as:

$$\left.\frac{\partial h}{\partial P}\right|_T = \frac{1}{\rho} + T \cdot \left.\frac{\partial s}{\partial P}\right|_T \tag{A.6}$$

Next, the following Maxwell relation is needed [Cravalho, 282]:

$$\left.\frac{\partial s}{\partial P}\right|_T = (-)\left.\frac{\partial \upsilon}{\partial T}\right|_P = \frac{1}{\rho^2} \cdot \left.\frac{\partial \rho}{\partial T}\right|_P \tag{A.7}$$

Applying the cyclical relation for partial derivatives [Cravalho, 289]:

$$\left.\frac{\partial \rho}{\partial T}\right|_P = (-)\left.\frac{\partial P}{\partial T}\right|_\rho \cdot \left.\frac{\partial \rho}{\partial P}\right|_T \tag{A.8}$$

Combining equations A.6, A.7, and A.8 gives:

$$\left.\frac{\partial h}{\partial P}\right|_T = \frac{1}{\rho} - \frac{T}{\rho^2} \cdot \left.\frac{\partial P}{\partial T}\right|_\rho \cdot \left.\frac{\partial \rho}{\partial P}\right|_T \tag{A.9}$$

Substituting equation A.1 into equation A.9, the partial derivative of enthalpy with respect to pressure at constant temperature is written in terms of the derivatives available in the computer software:

$$\left.\frac{\partial h}{\partial P}\right|_T = \frac{1}{\rho} - \frac{T}{\rho^2} \cdot \frac{\left.\frac{\partial P}{\partial T}\right|_\rho}{\left.\frac{\partial P}{\partial \rho}\right|_T} \tag{A.10}$$

The last derivative needed for equations 2.15 is $\delta\rho/\delta T|_P$. Once again we are trying to express it in terms of the derivatives in the available computer software ($\delta P/\delta T|_\rho$ and $\delta P/\delta\rho|_T$). Therefore the cyclical relation [Cravahlo, 289] is used to express this derivative as:

$$\left.\frac{\partial \rho}{\partial T}\right|_P = \frac{(-)\left.\frac{\partial P}{\partial T}\right|_\rho}{\left.\frac{\partial P}{\partial \rho}\right|_T} \tag{A.11}$$

APPENDIX B

LISTING OF STEADY-STATE CICC COMPUTER PROGRAM NAMED *CICC.EXE*

```
**********************************************************************
*   CICC.FOR
*
*   Program to evaluate the temperature profile in a Cable in Conduit
*   Conductor.
*
*   The program asks for the inlet conditions, the maximum allowable
*   temperature, and the minimum allowable pressure in the passage.
*   It also asks for starting values for the heat load per unit length (q)
*   and the mass flow rate (m).
*
*   The core of the program uses a finite difference method.  It takes the
*   conditions at one location, calculates dT/dx and dP/dx, and uses these
*   to get the conditions at the next node:
*         T(n+1) = T(n) + (dT/dx) * (dx)
*         P(n+1) = P(n) + (dP/dx) * (dx)
*
*   Once the inlet conditions are specified the conditions
*   at the rest of the nodes can be found.
*
*   The program starts with a low mass flow rate (m) and heat flux (q) and
*   calculates the conditions throughout the passage.  If at any point the
*   temperature is higher than the maximum allowable, the mass
*   flow rate is increased and the program starts over.  When a mass flow
*   rate is found that gives acceptable temperatures, the values of q and m
*   are saved.  Then q is increased and the process starts over.  Eventually
*   a point is reached where the mass flow rate (m) becomes so high that the
*   exit pressure is too low due to the frictional pressure drop  At this point,
*   the conditions for the highest valid heat load (q) are written to the
*   output file named CABLE.OUT
*
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   6-14-96
*
*   last modified 6-14-96
**********************************************************************
*
*   VARIABLES USED
*
*     ANS      = Character variable for a Y/N answer from the user
*     A        = Area of flow passage (m2)
*     Cp       = Specific heat at current location (J/kg-K)
*     Dh       = Hydraulic diameter of flow passage (m)
*     dPdR     = dP/dRho at constant temperature at current location (Pa-m3/kg)
*     dPdT     = dP/dT at constant density at current location (Pa/K)
*     dPdx     = dP/dx - Pressure change due to friction at current location (Pa/m)
*     dx       = Distance between nodes (m)
*     dTdx     = dT/dx Temperature change per length at current location (K/m)
*     f        = Fricton factor
*     h        = Specific enthalpy of helium at current location (J/kg)
*     IERR     = Flag variable used to determine whether you are trying to
*                write to an output file that already exits. (used to prevent
*                accidental overwriting)
*     L        = Length of flow passage (m)
*     m        = Mass flow rate (kg/s)
*     mlast    = Last sucessful value of m
*     NumSteps= Number of divisions in the mesh
*     OUTFILE = Name of output file
*     OUTPUT  = Flag that determines whether to write an output file
*     P        = Pressure at current location (Pa)
*     Pin      = Inlet pressure (Pa)
*     Pout     = Minimum Allowable Outlet pressure (Pa)
*     q        = Heat flow rate into passage per unit length (W/m)
*     qlast    = Last successful value of q
*     Rho      = Density at current location (kg/m3)
*     s        = Specific entropy at current location (J/kg-K)
*     T        = Temperature at current location (K)
*     Tin      = Inlet temperature (K)
*     Tmax     = Maximum allowable temperature at any location(K)
*     Ttop     = Highest acutal temperature in the conductor
*     x        = Current location
**********************************************************************
*
*  My Variable assignments
       real A, Dh, f, L, NumSteps
       real m, q
       real Pin, Pout, Tin, Tmax
       real Cp, dPdR, dPdT, h, Rho, s
       real P, T
       real mlast, qlast
       real RhoPrev, sPrev, hPrev, Tprev
       real Sin, Sgen, SgenTot
       real dPdx, dTdx
       real x, dx
       character*10 OUTFILE
       character*1 ANS
       integer IERR, OUTPUT
*
```

```
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
**********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=2
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state varriables and derivatives)
               JOUT=11000
*
************************************************************************
************************************************************************
************************************************************************
*
*    Given conditions
*
       print *,'Inlet pressure (atm) = ?'
       read(*,*) Pin
       Pin = Pin *101325

       print *,'Inlet temperature (K) = ?'
       read(*,*) Tin

       print *,'Minimum allowable pressure (atm) = ?'
       read(*,*) Pout
       Pout = Pout *101325

       print *,'Maximum allowable temperature (K) = ?'
       read(*,*) Tmax

       PRINT *,'Starting q ?'
       read(*,*) q

       PRINT *,'Starting m ?'
       read(*,*) m

* flow passage dimensions and number of nodes
       A = 0.000116
       Dh =0.00054
       f = 0.15
       L = 1140
       NumSteps= 1140

*      Length Step
       dx = L/NumSteps

* name of output file
       OUTFILE = 'CABLE.OUT'

* set variable OUTPUT equal to 0 so that the program knows
* that the maximum heat load has not been found yet
* (OUTPUT is set to 1 once the maximum q has been found)
       OUTPUT= 0


* control is sent back here when the program has found qmax
400    CONTINUE

***********************************************************************
*
*     Opening an output file if the maximum heat load has been
*     found (OUTPUT=1)

      IF (OUTPUT.EQ.1) THEN
       OPEN (Unit=10,FILE=OUTFILE,status='new',IOSTAT=IERR)

       IF (IERR.NE.0) THEN
          PRINT *,' The output file ', OUTFILE,' already exists.'
          PRINT *,' Do you wish to overwrite it?'
          PRINT *,' (Y to overwrite, N or <RETURN> to not overwrite)'
          READ (*,1) ANS
1         FORMAT (A1)
          IF (ANS.NE.'Y'.and.ANS.NE.'y') STOP
       ENDIF
      ENDIF

      IF (OUTPUT.EQ.1) THEN
        OPEN (Unit=10,FILE=OUTFILE,status='old')
        write(10,50)'q = ',q
        write(10,60)'m = ',m
        write(10,70)'Ttop = ',TtopLast
        write(10,80)'Pexit = ',Poutlast
        write(10,90)'SinTot = ',SinLast
        write(10,100)'SgenTot = ',SgenLast
        write(10,110)'Sgen/Sin = ',SgenLast/SinLast
        WRITE(10,*)' '
        WRITE(10,10)'   x       T      dtDx       P      dPdx       s      ',
     ;        '    h       Cp       Rho      dPdT     dPdRho     ',
     ;        '  q       Sin     Sgen    SgenTot'
        WRITE(10,10)'  (m)      (K)    (K/m)     (Pa)    (Pa/m)  (J-kg-K)',
     ;        '  (J/kg) (J/kg-K) (kg/m3)   (Pa/K)   (Pa-m3/kg) ',
     ;        ' (W/m)   (J/K)   (J/K)     (J/K) '

      ENDIF

10     FORMAT(A48,A47,A30)
```

```
500    CONTINUE
*************************************************************************
*
*      Assigning the values at the first location
*
       T = Tin
       P = Pin
       x = 0.0000000000000000000

*************************************************************************
*
*      Starting the loop to march through the length of the conductor

1000   CONTINUE
*       print *, x,T,P,m

*************************************************************************
*
*      Save the conditions of the previous location
          RhoPrev = Rho
          sPrev   = s
          hPrev   = h
          TPrev   = T

*************************************************************************
*
*       Finding the conditions at the current location
        JIN1=1
        VALU1=P
        JIN2=2
        VALU2=T

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           Rho = PROP(3,0)
           s   = PROP(8,0)
           h   = PROP(9,0)
           Cp  = PROP(14,0)
           dPdR = PROP(22,0)
           dPdT = PROP(23,0)

           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF
*************************************************************************
*
*     Heat Input, Entropy Input, and Entropy Generation
*
      IF (x.ne.0) THEN
         qin =  m*( h-hPrev  +0.5*(m/A)**2*(1/Rho**2 - 1/RhoPrev**2))/dx
         Sin = qin/( (T+TPrev)/2)*dx
         SinTot  = SinTot + Sin
         Sgen = m*(s-sPrev) - Sin
         SgenTot = SgenTot + Sgen
      ELSE
         qin = 0.0
         Sin = 0.0
         Sgen = 0.0
         SgenTot = 0.0
         SinTot = 0.0
      ENDIF


*************************************************************************
*
*      Slopes of temerature and pressure change per length

       dTdx = ( q/m + ( 1/Rho - T*dPdT/(Rho**2 * dPdR) -
     ;    m**2/(Rho**3 * A**2 * dPdR) ) * f*m**2/(2*Rho* A**2 *Dh) ) /
     ;    ( Cp + m**2*dPDT/(Rho**3 * A**2 * dPdR) )

       dPdx =  -f* m**2 /(2* Rho * A**2 * Dh)


*************************************************************************
*
*       OUTPUT
*
      IF(OUTPUT.EQ.1) THEN
          WRITE(10,20) x, T, dTdx, P, dPdx, s, h, Cp, Rho, dPdT, dPdR,
     ;            qin, Sin, Sgen, SgenTot

20        FORMAT (F8.3,2x,  F5.3,1x,  F7.5,2x,  F8.0,1x,  F6.1,2x, F7.1,
     ;          3x,  F7.1,1x, F7.1,2x,  F6.1,1x,  F9.1,1x,  F8.1,3x,
     ;          F6.5,1x, F6.5,1x,  F7.6,2x, F6.5)
      ENDIF

*************************************************************************
*
*      Finding the conditions at the NEXT location

      IF (x.LT.L) THEN
         T = T + dTdx*dx
         P = P + dPdx*dx
      ENDIF
      x = x + dx

*************************************************************************
```

```
*
*        CHECK TO MAKE SURE THAT THE TEMPERATURE DOES NOT EXCEED THE
*        MAXIMUM ALLOWABLE TEMPERATUE. IF IT DOES, INCREASE THE MASS FLOW
*
       IF (T.GT.Tmax) THEN
*           CLOSE(10)
          Ttop = T
          M = M + .00001
          PRINT *,'******MAX TEMP EXCEEDED -- MASS FLOW INCREASED'
          WRITE(*,40)' Pexit = ',P,' Ttop = ',Ttop,' q = ',q,' m = ',m
          Ttop=0.0
          GOTO 500
       ENDIF
**************************************************************************
*
*       CHECK TO SEE IF THE PRESSURE GETS BELOW THE MINIMUM ALLOWABLE
*       TEMPERATURE.  IF IT DOES, THEN THE MASS FLOW RATE IS TOO LARGE.
*       WE ASSUME THIS MEANS THAT THE LAST CONDITION EVALUATED IS THE
*       MAXIUMUM HEAT LOAD CONDTION.  THEREFORE RESET THE CONDITIONS TO
*       THE LAST VALID HEAT LOAD AND SET OUTPUT = 1 SO THAT THE VALUES
*       ARE SAVED TO THE OUTPUT FILE
*
*
       IF (P.LT.Pout) THEN
*          CLOSE(10)
         PRINT *,' '
         PRINT *,'   **************************************************'
         PRINT *,'   *   PRESSURE BELOW MINIMUM ALLOWABLE PRESSURE   *'
         PRINT *,'   *         THE CABLE CANNOT REMOVE THIS          *'
         PRINT *,'   *        LARGE OF OF A HEAT LOAD WITH THE       *'
         PRINT *,'   *        GIVEN MAXIMUM ALLOWED TEMPERATURE      *'
         PRINT *,'   **************************************************'
         PRINT *,' '
         OUTPUT = 1
         q = qlast
         m = mlast
         Ttop = 0.0
         GOTO 400
*         STOP
       ENDIF
**************************************************************************
*
*      RECORD THE MAXIMUM TEMPERATURE IN THE PASSAGE
*
       IF (T.GT.Ttop.and.x.LE.L) THEN
         Ttop = T
       ENDIF
**************************************************************************
*
*        IF NO ERRORS WERE FOUND, THEN FIND CONDITIONS AT NEXT POINT
*        (UNLESS WE ARE AT THE EXIT -- THEN CHECK TO SEE IF THE
*         EXIT PRESSURE IS GREATER THAN NEEDED---IF IT IS INCREASE MASS
*         FLOW RATE OR THE HEAT LOAD)
*

      IF (x.LE.L) Then
         GO TO 1000
      ELSE
         CLOSE(10)
        IF (P.GT.Pout.and.OUTPUT.EQ.0) THEN
          print *,'EXIT PRESSURE HIGHER THAN REQUIRED - INCREASING q'
          WRITE(*,40)' Pexit = ',P,' Ttop = ',Ttop,' q = ',q,' m = ',m
40        FORMAT(A8,F8.0,A8,F6.4,A5,F6.5,A5,F6.5)
*            m = 1.01 *m
          qlast = q
          mlast = m
          TtopLast = Ttop
          PoutLast = P
          SinLast = SinTot
          SgenLast = SgenTot
          q = q + 0.0001
          Ttop=0.0
          GOTO 500
        ENDIF
      ENDIF

        write(*,*) 'CONDITIONS AT MAXIMUM HEAT LOAD'
        write(*,50)'q = ',q
        write(*,60)'m = ',m
        write(*,70)'Ttop = ',Ttop
        write(*,80)'Pexit = ',P
        write(*,90)'SinTot = ',SinTot
        write(*,100)'SgenTot = ',SgenTot
        write(*,110)'Sgen/Sin = ',SgenTOT/SinTOT

50      format(1x,a4,f6.5)
60      format(1x,a4,f6.5)
70      format(1x,a7,f5.3)
80      format(1x,a8,f8.0)
90      format(1x,a9,f6.4)
100     format(1x,a10,f5.4)
110     format(1x,a11,f6.3)

        END
```

APPENDIX C

LISTING OF PUMPING ANALYSIS COMPUTER PROGRAM (*CICC4.EXE*)

```
**********************************************************************
*    CICC4.FOR
*
*    Program to evaluate the temperature profile in a Cable in Conduit
*    Conductor.  This version is very similar to CICC.EXE but it also
*    calculates the required pump work for cold pumping and warm pumping
*    for 3 pump isentropic efficiencies.(see chapter 2 of the thesis for
*    a sketch of the system)
*
*
*    The core of the program uses a finite difference method.  It takes the
*    conditions at one location, calculates dT/dx and dP/dx, and uses these
*    to get the conditions at the next node:
*          T(n+1) = T(n) + (dT/dx) * (dx)
*          P(n+1) = P(n) + (dP/dx) * (dx)
*
*    The inlet conditions are specified and then the conditions
*    at the rest of the nodes are found.
*
*    The program starts with a low mass flow rate (m) and heat flux (q) and
*    calculates the conditions throughout the passage.  If at any point the
*    temperature is higher than the maximum allowable, the mass
*    flow rate is increased and the program starts over.  When a mass flow
*    rate is found that gives acceptable temperatures, the values of q and m
*    are saved.  Then q is increased and the process starts over.  Eventually
*    a point is reached where the mass flow rate (m) becomes so high that the
*    exit pressure is too low due to the frictional pressure drop  At this point,
*    the conditions for the highest q that was ok are used to find the pump
*    power required for a simple loop model (section 3.2.2).
*
*    Since it can take quite a while to search for the highest q, the program
*    asks for initial guesses for q and m. (which are good if you have already
*    run the program once and know approximately what the maximum vales will be.)
*    You can save a lot of time by guessing close, but slightly less than
*    the maximum values.  I did this so many times that I made an input
*    file of guesses for each maximum temperature. The file has values
*    for inlet pressure Pin, and guesses for q and m that are close to the maximum
*    values. The files were named for the maximum allowable temperature: ex 52.dat
*    for 5.2 K, 58.dat for 5.8 K, etc.  I then made a file called infile.dat that contains
*    the names of each of these files so that the program could call them
*    one after the other.  If you are running the program for the first time
*    you can remove the lines that read Pin, m and q from the input file and
*    uncomment the lines that prompt the user to enter the values.  The input files
*    are listed at the end of the program.
*
*
*    I realize that the looping structure I used is not the best since
*    I used GOTO statements, but the program is not too complicated and
*    it works.
*
*    This program must be linked with HEPROP.OBJ a property routine
*    program by CRYODATA
*
*
*    Brian Bowers
*    original version started on
*    6-14-96
*
*    last modified 8-7-96
**********************************************************************
*
*    VARIABLES USED
*
*      ANS      = Character variable for a Y/N answer from the user
*      A        = Area of flow passage (m2)
*      Cp       = Specific heat at current location (J/kg-K)
*      CpIN     = Specific heat at inlet (J/kg-K)
*      Dh       = Hydraulic diameter of flow passage (m)
*      dPdR     = dP/dRho at constant temperature at current location (Pa-m3/kg)
*      dPdT     = dP/dT at constant density at current location (Pa/K)
*      dPdx     = dP/dx - Pressure drop due to friction at current location (Pa/m)
*      dx       = Increment of length (m)
*      dTdx     = dT/dx Temperature change per length at current location (K/m)
*      f        = Friction factor
*      h        = Specific enthalpy of helium at current location (J/kg)
*      Hinlet   = Specific enthalpy of helium at CICC inlet (J/kg-K)
*      IERR     = Flag variable used to determine whether you are trying to
*                 write to an output file that already exits. (used to prevent
*                 accidental overwriting)
*      L        = Length of flow passage (m)
*      m        = Mass flow rate (kg/s)
*      mlast    = Last successful value of m
*      NumSteps= Number of divisions in the mesh
*      OUTFILE  = Name of output file
*      OUTPUT   = Flag the determines whether to write and output file
*      P        = Pressure at current location (Pa)
*      Pin      = Inlet pressure (Pa)
*      Pout     = Outlet pressure (Pa)
*      q        = Heat flow rate per unit length (W/m)
*      qlast    = Last successful value of q
*      Rho      = Density at current location (kg/m3)
*      RhoIN    = Density at inlet (kg/m3)
*      s        = Specific entropy at current location (J/kg-K)
```

```
*     Sgen    = Total entropy generation in the current segment of CICC
*     SgenTOT = Cumulative total entropy generation from CICC inlet to outlet
*     Sin     = Entropy gained by helium in the CICC (J/K)
*     Sinlet  = Specific entropy of helium at CICC inlet (J/kg-K)
*     SoutLast= Specific entropy at CICC exit for the last valid solution (J/kg-K)
*     Sout    = Total entropy rejected from recirculator loop to pool (J/K)
*     Spool   = Total entropy received by the 4.2 K pool (J/K)
*     T       = Temperature at current location (K)
*     Tin     = Inlet temperature (K)
*     Tmax    = Maximum allowable temperature at any location(K)
*     Ttop    = Highest actual temperature in the conductor
*     W       = Pumping power:  W1: warm pump - 100% efficient
*               W2: warm, 70%, W3: warm, 50%,
*               W4: cold, 100%,  W5: cold 70%, W6: warm 50%
*     x       = Current location
*********************************************************************
*
*   My Variable assignments
        real A, Dh, f, L, NumSteps
        real m, q
        real Pin, Pout, Tin, Tmax
        real Cp, dPdR, dPdT, h, hs, Rho, s
        real P, T
        real mlast, qlast, TtopLast, PoutLast
        real Sinlast,SgenLast,SoutLast,houtLast
        real h1,h2,h3,h4,h5,h6
        real RhoIN, CpIN
        real RhoPrev, sPrev, hPrev, Tprev
        real Sin, Sgen, SgenTot
        real dPdx, dTdx
        real x, dx
        real Sout1,Sout2,Sout3,Sout4,Sout5,Sout6
        real Spool1,Spool2,Spool3,Spool4,Spool5,Spool6
        real SgenPmp1,SgenPmp2,SgenPmp3,SgenPmp4,SgenPmp5,SgenPmp6
        real s2, s3, s5, s6
        real w1, w2, w3, w4, w5, w6
        character*10 OUTFILE, INFILE
        character*1 ANS
        integer IERR, OUTPUT
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
**********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=2
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
               JOUT=11000
*
***********************************************************************
***********************************************************************
*
*     OPENING AN OUTPUT FILE
       OUTFILE = 'CABLE.OUT'
       OPEN (Unit=10,FILE=OUTFILE,status='new',IOSTAT=IERR)

       IF (IERR.NE.0) THEN
          PRINT *,' The output file ', OUTFILE,' already exists.'
          PRINT *,' Do you wish to overwrite it?'
          PRINT *,' (Y to overwrite, N or <RETURN> to not overwrite)'
          READ (*,1) ANS
1         FORMAT (A1)
          IF (ANS.NE.'Y'.and.ANS.NE.'y') STOP
          OPEN (Unit=10,FILE=OUTFILE,status='old',IOSTAT=IERR)
       ENDIF


***********************************************************************
*     OPENING INPUT FILE
*
*  read the name of the file that has the guess values for m and q
      open(unit=99,file="infile.dat",status='old')

200   close(unit=5)
      read(99,55) infile
55    format(A10)

      OPEN(UNIT=5,FILE=infile,STATUS='OLD')

*  FIRST LINE OF INPUT  FILE HAS THE MAXIMUM ALLOWABLE TEMPERATURE
      Read(5,*)Tmax
      write(10,*)' '
      write(10,44)' Tmax = ',Tmax

*  WRITE HEADING TO THE OUTPUT FILE
44    format(a8,f5.2)
      WRITE(10,2)'                              ',
      ;        '                                          ',
      ;        '                                        n=1.0  ',
     ;        'n=0.7    n=0.5     n=1.0    n=0.7    n=0.5     n=1.0',
     ;        ' n=0.7    n=0.5     n=1.0    n=0.7    n=0.5',
     ;        '  n=0.7    n=0.5    n=0.7     n=0.5     n=1.0  ',
     ;        '  n=0.7     n=0.5     n=1.0     n=0.7     n=0.5 ',
     ;        'n=1.0     n=0.7     n=0.5    n=1.0     n=0.7      n=0.5'
       WRITE(10,2)'  Tin     RhoIN      CpIN ',
     ;        '     Pin     Pout     Pin   Pout    qmax   Qmax  ',
     ;        '  m      m        Sin       Sgen   Sgen/Sin       Wp1   ',
     ;        '  Wp2       Wp3     Qload1   Qload2   Qload3      Wp4 ',
```

```
     ;         '  Wp5       Wp6      Qload4   Qload5   Qload6',
     ;         ' SgenPmp2 SgenPmp3 SgenPmp5  SgenPmp6   Sout1  ',
     ;         ' Sout2     Sout3     Sout4     Sout5     Sout6 ',
     ;         'Spool1    Spool2    Spool3    Spool4    Spool5    Spool6'
      Write(10,2)'  (K)  (kg/m3)  (J/kg-K)',
     ;         '    (Pa)      (Pa)    (atm) (atm)  (W/m)     (W)  ',
     ;         '(kg/s) (g/s)    (J/K)      (J/K)                  (W)   ',
     ;         '  (W)       (W)       (W)       (W)       (W)       (W) ',
     ;         '  (W)       (W)        (W)       (W)       (W) ',
     ;         '  (J/K)    (J/K)      (J/K)     (J/K)     (J/K)  ',
     ;         ' (J/K)      (J/K)      (J/K)     (J/K)     (J/K) ',
     ;         ' (J/K)      (J/K)      (J/K)     (J/K)     (J/K)     (J/K)'
2     format(A24,A46,A53,A53,A45,A46,A50,A58)

300   CONTINUE

*  READ THE INLET PRESSURE, HEAT FLUX PER LENGTH AND MASS FLOW RATE
      READ(5,fmt=*,Err=200) Pin, q, m
      mLast= m

************************************************************************
*
*    Given conditions
*
*  these line are commented out - they are used if the user enters Pin,
*  m, and q instead of reading from a file.
*       print *,'Inlet pressure (atm) = ?'
*       read(*,*) Pin
*       Pin = 1519875
*       q = 0.00
*       PRINT *,'Starting q ?'
*       read(*,*) q
*       m = 0.0001
*       PRINT *,'Starting m ?'
*       read(*,*) m

       Tin = 4.5
       Pin = Pin *101325

       A = 0.000116
       Dh =0.00054
       f = 0.15
       L = 1140
*       Tmax = 6.0
       Pout = 303975
       NumSteps= 1140
*       OUTFILE = 'CABLE.OUT'
       OUTPUT= 0

*      Length Step
       dx = L/NumSteps
*
***********************************************************************
*
*       Finding the conditions at the inlet
        JIN1=1
        VALU1=Pin
        JIN2=2
        VALU2=Tin

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           RhoIN = PROP(3,0)
           CpIN  = PROP(14,0)
*           dPdR = PROP(22,0)
*           dPdT = PROP(23,0)

           IF (IDID .EQ. 3) THEN
*                  This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF



***********************************************************************
*
*     Opening an output file
*     THIS SECTION IS USED IF YOU WANT TO SAVE THE TEMPERATURE AND
*     PRESSURE VALUES ALL ALONG THE CONDUCTOR INSTEAD OF JUST THE OVERALL
*     HEAT LOAD, PUMP WORK, ETC.  THESE LINES ARE COMMENTED OUT, BUT I
*     DIDN'T DELETE THEM IN CASE SOMEONE WANTS TO GET THIS TYPE OF DATA LATER.
*     (THERE ARE ALSO A FEW LINES BELOW -in an "IF (OUTPUT.EQ.1)" section -
*     THAT SHOULD BE UNCOMMENTED TO GET THIS OUTPUT)

400   CONTINUE

*      IF (OUTPUT.EQ.1) THEN
*       OPEN (Unit=10,FILE=OUTFILE,status='new',IOSTAT=IERR)

*       IF (IERR.NE.0) THEN
*          PRINT *,' The output file ', OUTFILE,' already exists.'
*          PRINT *,' Do you wish to overwrite it?'
*          PRINT *,' (Y to overwrite, N or <RETURN> to not overwrite)'
*          READ (*,1) ANS
*          IF (ANS.NE.'Y'.and.ANS.NE.'y') STOP
*       ENDIF
*      ENDIF

500    CONTINUE
```

```
*       IF (OUTPUT.EQ.1) THEN
*         OPEN (Unit=10,FILE=OUTFILE,status='old')
*         write(10,50)'q = ',q
*         write(10,60)'m = ',m
*         write(10,70)'Ttop = ',TtopLast
*         write(10,80)'Pexit = ',Poutlast
*         write(10,90)'SinTot = ',SinLast
*         write(10,100)'SgenTot = ',SgenLast
*         write(10,110)'Sgen/Sin = ',SgenLast/SinLast
*         WRITE(10,*)' '
*         WRITE(10,10)'    x       T      dtDx       P       dPdx       s     ',
*      ;       '    h       Cp       Rho      dPdT      dPdRho    ',
*      ;       '   q     Sin     Sgen    SgenTot'
*         WRITE(10,10)'  (m)     (K)     (K/m)     (Pa)     (Pa/m)   (J-kg-K)',
*      ;       '    (J/kg)  (J/kg-K) (kg/m3)   (Pa/K)   (Pa-m3/kg) ',
*      ;       ' (W/m)   (J/K)   (J/K)     (J/K) '

*       ENDIF

*10     FORMAT(A48,A47,A30)

*************************************************************************
*
*       Assigning the values at the first location
*
        T = Tin
        P = Pin
        x = 0.000

*************************************************************************
*
*       Starting the loop to march through the length of the conductor

1000    CONTINUE
*        print *, x,T,P,m

*************************************************************************
*
*       Save the conditions of the previous location
           RhoPrev = Rho
           sPrev   = s
           hPrev   = h
           TPrev   = T

*************************************************************************
*
*        Finding the conditions at the current location
         JIN1=1
         VALU1=P
         JIN2=2
         VALU2=T

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +      NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            Rho = PROP(3,0)
            s   = PROP(8,0)
            h   = PROP(9,0)
            Cp  = PROP(14,0)
            dPdR = PROP(22,0)
            dPdT = PROP(23,0)

            IF (IDID .EQ. 3) THEN
*                  This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

*************************************************************************
*
*      Heat Input, Entropy Input, and Entropy Generation
*
       IF (x.ne.0) THEN
          qin =  m*( h-hPrev  +0.5*(m/A)**2*(1/Rho**2 - 1/RhoPrev**2))/dx
          Sin = qin/( (T+TPrev)/2)*dx
          SinTot  = SinTot + Sin
          Sgen = m*(s-sPrev) - Sin
          SgenTot = SgenTot + Sgen
       ELSE
          qin = 0.0
          Sin = 0.0
          Sgen = 0.0
          SgenTot = 0.0
          SinTot = 0.0
       ENDIF

*************************************************************************
*
*       Slopes of temerature and pressure change per length

        dTdx = ( q/m + ( 1/Rho - T*dPdT/(Rho**2 * dPdR) -
     ;     m**2/(Rho**3 * A**2 * dPdR) ) * f*m**2/(2*Rho* A**2 *Dh) ) /
     ;     ( Cp + m**2*dPDT/(Rho**3 * A**2 * dPdR) )

        dPdx =  -f* m**2 /(2* Rho * A**2 * Dh)

*************************************************************************
*
*        OUTPUT
*
*       IF(OUTPUT.EQ.1) THEN
*           WRITE(10,20) x, T, dTdx, P, dPdx, s, h, Cp, Rho, dPdT, dPdR,
*      ;            qin, Sin, Sgen, SgenTot

*20         FORMAT (F8.3,2x,  F5.3,1x,  F7.5,2x,  F8.0,1x,  F6.1,2x, F7.1,
```

```
*    ;          3x,  F7.1,1x, F7.1,2x,  F6.1,1x,  F9.1,1x,  F8.1,3x,
*    ;          F6.5,1x, F6.5,1x,  F7.6,2x, F6.5)
*     ENDIF

************************************************************************
*
*     Finding the conditions at the NEXT location

      IF (x.LT.L) THEN
         T = T + dTdx*dx
         P = P + dPdx*dx
      ENDIF
      x = x + dx

************************************************************************
*
*      CHECK TO MAKE SURE THAT THE TEMPERATURE DOES NOT EXCEED THE
*      MAXIMUM ALLOWABLE TEMPERATUE. IF IT IS, INCREASE THE MASS FLOW
*
      IF (T.GT.Tmax) THEN
*          CLOSE(10)
          Ttop=0.0
          M = M + .00001
*          PRINT *,'MAX TEMP EXCEEDED -- MASS FLOW INCREASED'
          GOTO 500
      ENDIF

************************************************************************
*
*     CHECK TO SEE IF THE PRESSURE GETS BELOW THE MINIMUM ALLOWABLE
*     TEMPERATURE.  IF IT DOES, THEN THE HEAT LOAD IS TOO LARGE.
*     USE VALUES FROM THE LAST VALID HEAT LOAD TO FIND PUMPING
*     POWER AND THEN WRITE DATA TO OUTPUT FILE
*
*      IF (P.LT.Pout) THEN
      IF (P.LT.Pout.or.m.GT.mLast+.0002) THEN

*          CLOSE(10)
         PRINT *,' '
         PRINT *,'   *************************************************'
         PRINT *,'   *   PRESSURE BELOW MINIMUM ALLOWABLE PRESSURE   *'
         PRINT *,'   *              HEAT LOAD TOO LARGE              *'
         PRINT *,'   *   WRITING LAST VALID HEAT LOAD DATA TO FILE   *'
         PRINT *,'   *************************************************'
         PRINT *,' '
*         PRINT *,'q = ',q,' W/m'
*         PRINT *,'Ttop = ', Ttop

*        CONDTIONS AT INLET OF CICC
         JIN1=1
         VALU1=Pin
         JIN2=2
         VALU2=Tin

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            Sinlet   = PROP(8,0)
            Hinlet   = PROP(9,0)
            IF (IDID .EQ. 3) THEN
*                This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF


*   ********** WARM SIDE PUMPING **********

*       Isentropic pump exit conditions
         JIN1=1
         VALU1=Pin
         JIN2=5
         VALU2=SoutLast

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           hs = PROP(9,0)
           IF (IDID .EQ. 3) THEN
*                This code not used in this case
           ENDIF
         ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
         ENDIF
*
*        Actual exit enthalpy of compressor if it was adiabatic
*
         h1 = hs
         h2 = ( hs - houtLast )/0.7  +  houtLast
         h3 = ( hs - houtLast )/0.5  +  houtLast


*
*       Warm Pump power

         W1 = mlast*( h1 - houtLast)
         W2 = mlast*( h2 - houtLast)
         W3 = mlast*( h3 - houtLast)


*       Finding the exit entropy of the pump - Eta = 70%
        JIN1=1
```

```
         VALU1=Pin
         JIN2=6
         VALU2=h2

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +      NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
             s2=PROP(8,0)
             IF (IDID .EQ. 3) THEN
*                This code not used in this case
             ENDIF
         ELSE
             CALL ERRMSG (MESSAG, IDID)
             WRITE (*, '(1X,A60)') MESSAG
             STOP
         ENDIF

*      Finding the exit entropy of the pump - Eta = 50%
         JIN1=1
         VALU1=Pin
         JIN2=6
         VALU2=h3

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +      NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
             s3=PROP(8,0)
             IF (IDID .EQ. 3) THEN
*                This code not used in this case
             ENDIF
         ELSE
             CALL ERRMSG (MESSAG, IDID)
             WRITE (*, '(1X,A60)') MESSAG
             STOP
         ENDIF


*        Entropy Generated in Pump
         SgenPmp1 = 0
         SgenPmp2 = mlast*(s2 - SoutLast)
         Sgenpmp3 = mlast*(s3 - SoutLast)


*        Entropy rejected by recirculator loop to pool
         Sout1 = mlast*(SoutLast - Sinlet)
         Sout2 = mlast*(s2 - Sinlet)
         Sout3 = mlast*(s3 - Sinlet)

*        Entropy received by Pool
         Spool1 =  (qlast * L + W1)/4.2
         Spool2 =  (qlast * L + W2)/4.2
         Spool3 =  (qlast * L + W3)/4.2

*
*   *********   COLD SIDE PUMPING - PpumpIN = Pexit,  TpumpIN = 4.5 K  *******

*      Conditions at pump inlet
         JIN1=1
         VALU1=PoutLast
         JIN2=2
         VALU2=4.5

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +      NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            s   = PROP(8,0)
            h   = PROP(9,0)
            IF (IDID .EQ. 3) THEN
*                This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

*       Isentropic pump exit conditions
         JIN1=1
         VALU1=Pin
         JIN2=5
         VALU2=s

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            hs = PROP(9,0)
            IF (IDID .EQ. 3) THEN
*                This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

*      Actual exit enthalpy of compressor if it was adiabatic
*
         h4 = hs
         h5 = ( hs - h )/0.7  +  h
         h6 = ( hs - h )/0.5  +  h

*      Cold Pump power

         W4 = mlast*( h4 - h)
         W5 = mlast*( h5 - h)
         W6 = mlast*( h6 - h)

*      Finding the exit entropy of the pump - Eta = 70%
```

```
        JIN1=1
        VALU1=Pin
        JIN2=6
        VALU2=h5

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            s5=PROP(8,0)
            IF (IDID .EQ. 3) THEN
*               This code not used in this case
            ENDIF
        ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
        ENDIF

*      Finding the exit entropy of the pump - Eta = 50%
        JIN1=1
        VALU1=Pin
        JIN2=6
        VALU2=h6

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            s6=PROP(8,0)
            IF (IDID .EQ. 3) THEN
*               This code not used in this case
            ENDIF
        ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
        ENDIF

*       Entropy Generated in Pump
        SgenPmp4 = 0
        SgenPmp5 = mlast*(s5 - s)
        Sgenpmp6 = mlast*(s6 - s)

*       Entropy rejected by recirculator loop to the pool
        Sout4 = mlast*( (SoutLast - s) + (s - sinlet) )
        Sout5 = mlast*( (SoutLast - s) + (s5 - sinlet) )
        Sout6 = mlast*( (SoutLast - s) + (s6 - sinlet) )

*       Entropy received by Pool
        Spool4 =  (qlast * L + W4)/4.2
        Spool5 =  (qlast * L + W5)/4.2
        Spool6 =  (qlast * L + W6)/4.2

        WRITE(10,30) Tin, RhoIn, CpIN,
     ;          Pin, PoutLast,Pin/101325,PoutLast/101325,
     ;          qlast,qlast*L,mlast,mlast*1000,
     ;          Sinlast,SgenLast,SgenLast/Sinlast,W1,W2,W3,
     ;          W1+qlast*L,W2+qlast*L,W3+qlast*L,W4,W5,W6,
     ;          W4+qlast*L,W5+qlast*L,W6+qlast*L,
     ;          SgenPmp2,SgenPmp3,SgenPmp5,SgenPmp6,
     ;          Sout1,Sout2,Sout3,Sout4,Sout5,Sout6,
     ;          Spool1,Spool2,Spool3,Spool4,Spool5,Spool6
30      format(1x,F5.2,2x,F6.1,2x,F7.1,2x,
     ;         f8.0,1x,f8.0,1x,F5.2,1x,f5.2,1x,
     ;         f6.5,2x,f7.3,1x,f6.5,2x,f4.2,2x,
     ;         F7.4,2x,f7.4,3x,f6.4,3x,f8.4,1x,f8.4,1x,f8.4,2x,
     ;         f7.3,2x,f7.3,2x,f7.3,2x,f8.4,1x,f8.4,1x,f8.4,2x,
     ;         f7.3,2x,f7.3,2x,f7.3,1x,
     ;         f7.5,2x,f7.5,2x,f7.5,4x,f7.5,1x,
     ;         f8.4,2x,f8.4,2x,f8.4,2x,f8.4,2x,f8.4,2x,f8.4,2x,
     ;         f8.4,2x,f8.4,2x,f8.4,2x,f8.4,2x,f8.4,2x,f8.4)

        goto 300

*        OUTPUT = 1
*        q = qlast
*        m = mlast
*        Ttop = 0.0
*        GOTO 400
*        STOP
     ENDIF

************************************************************************
*
*    RECORD THE MAXIMUM TEMPERATURE IN THE PIPE
*
     IF (T.GT.Ttop.and.x.LE.L) THEN
        Ttop = T
     ENDIF

************************************************************************
*
*     IF NO ERRORS WERE FOUND, THEN FIND CONDITIONS AT NEXT POINT
*     (UNLESS WE ARE AT THE EXIT -- THEN CHECK TO SEE IF THE
*      EXIT PRESSURE IS GREATER THAN NEEDED---IF IT IS INCREASE MASS
*      FLOW RATE OR THE HEAT LOAD AND SAVE THESE CONDITIONS AS THE
*      LAST VALID SOLUTION FOUND.)
*

    IF (x.LE.L) Then
*    if (x-L.LT.dx) then
       GO TO 1000
    ELSE
*      CLOSE(10)
*     IF ((P-Pout)/Pout.GT.0.01) THEN
      IF (P.GT.Pout.and.OUTPUT.EQ.0) THEN
       print *,'EXIT PRESSURE HIGHER THAN REQUIRED - INCREASING q'
       WRITE(*,40)'Pexit = ',P,' Ttop = ',Ttop,' q = ',q,' m = ',m
40     FORMAT(1x,A8,F8.0,A8,F7.4,A5,F6.5,A5,F6.5)
```

```
*           m = 1.01 *m
          qlast = q
          mlast = m
          TtopLast = Ttop
          PoutLast = P
          SinLast = SinTot
          SgenLast = SgenTot
          SoutLast = S
          houtLAst = h
          q = q + 0.0001
          Ttop=0.0
          GOTO 500
        ENDIF
      ENDIF

*         write(*,50)'q = ',q
*         write(*,60)'m = ',m
*         write(*,70)'Ttop = ',Ttop
*         write(*,80)'Pexit = ',P
*         write(*,90)'SinTot = ',SinTot
*         write(*,100)'SgenTot = ',SgenTot
*         write(*,110)'Sgen/Sin = ',SgenTOT/SinTOT

*50       format(1x,a4,f6.5)
*60       format(1x,a4,f6.5)
*70       format(1x,a7,f5.3)
*80       format(1x,a8,f8.0)
*90       format(1x,a9,f6.4)
*100      format(1x,a10,f5.4)
*110      format(1x,a11,f6.3)

        END


***********************************************************************
**********************************************************************
 INPUT FILES


INFILE.DAT:
5.in
51.in
52.in
525.in
53.in
54.in
55.in
56.in
57.in
575.in
58.in
59.in
6.in
7.in
8.in
10.in
15.in
20.in


5.IN:
5
3.05        0.0005      0.00021
3.5         0.0016      0.00071
4           0.0022      0.00104
4.5         0.0025      0.00128
5           0.0026      0.00147
5.5         0.0026      0.00164
6           0.0025      0.0018
6.5         0.0024      0.00198
7           0.0021      0.00211
7.5         0.0018      0.00225
7.7         0.0016      0.00228
7.8         0.0015      0.00122
8           0.0015      0.00129
8.5         0.0014      0.00116
9           0.0013      0.00104
10          0.0013      0.00119
12          0.0011      0.00094
15          0.001       0.00091
18          0.001       0.00105
20          0.0009      0.00088


51.IN:
5.1
3.05        0.0006      0.0002
3.5         0.002       0.0007
4           0.0028      0.00102
4.5         0.0032      0.00125
5           0.0035      0.00146
5.5         0.0036      0.00164
6           0.0036      0.0018
6.5         0.0036      0.00197
7           0.0034      0.0021
7.5         0.0031      0.00223
8           0.0028      0.00237
8.5         0.0023      0.00247
8.8         0.002       0.00254
8.9         0.0019      0.00136
9           0.0019      0.0014
10          0.0017      0.00119
12          0.0016      0.00127
15          0.0014      0.0011
18          0.0013      0.00107
20          0.0013      0.0012


52.IN:
5.2
3.05        0.0008      0.00022
3.5         0.0026      0.00072
4           0.0035      0.00102
4.5         0.0041      0.00125
5           0.0045      0.00145
5.5         0.0048      0.00164
6           0.0049      0.0018
6.5         0.0049      0.00195
7           0.0048      0.00209
7.5         0.0046      0.00223
8           0.0043      0.00236
8.5         0.0039      0.00248
9           0.0034      0.00258
9.5         0.0029      0.00271
9.7         0.0027      0.00276
9.8         0.0025      0.00275
9.9         0.0024      0.00278
10          0.0023      0.00144
12          0.002       0.00125
15          0.0018      0.00119
18          0.0017      0.00122
20          0.0016      0.00114


53.IN:
5.3
3.05        0.001       0.00022
3.5         0.0031      0.0007
4           0.0043      0.001
4.5         0.0052      0.00125
5           0.0058      0.00145
5.5         0.0061      0.00162
6           0.0064      0.00179
6.5         0.0064      0.00194
7           0.0064      0.00208
7.5         0.0062      0.00221
8           0.0059      0.00234
8.5         0.0056      0.00247
9           0.0051      0.00257
10          0.0041      0.0028
11          0.0028      0.00301
11.5        0.0027      0.0016
12          0.0026      0.00149
15          0.0023      0.00137
18          0.0021      0.00129
20          0.002       0.00125
```

54.IN
5.4

3.05	0.0012	0.00021
3.5	0.0039	0.0007
4	0.0054	0.001
4.5	0.0065	0.00123
5	0.0072	0.00143
5.5	0.0077	0.00161
6	0.008	0.00177
6.5	0.0081	0.00192
7	0.008	0.00206
7.5	0.0079	0.0022
8	0.0077	0.00233
8.5	0.0074	0.00245
9	0.007	0.00257
10	0.006	0.00279
11	0.0047	0.00299
12	0.0033	0.0032
12.5	0.0031	0.00155
15	0.0028	0.00146
18	0.0026	0.00145
20	0.0025	0.00145

55.IN:
5.5

3.05	0.0015	0.00021
3.5	0.005	0.00069
4	0.007	0.00098
4.5	0.0082	0.00121
5	0.009	0.00141
5.5	0.0094	0.00159
6	0.0097	0.00175
6.5	0.0098	0.0019
7	0.0098	0.00204
7.5	0.0098	0.00218
8	0.0096	0.00231
8.5	0.0093	0.00243
9	0.0089	0.00255
10	0.008	0.00277
12	0.0054	0.00318
13	0.0038	0.00337
13.5	0.0036	0.00167
15	0.0034	0.00163
17	0.0032	0.00159
18	0.0031	0.00153
20	0.003	0.00158

56.IN:
5.6

3.05	0.002	0.0002
3.5	0.0066	0.00066
4	0.0088	0.00095
4.5	0.0101	0.00118
5	0.0108	0.00138
5.5	0.0113	0.00156
6	0.0116	0.00172
6.5	0.0118	0.00188
7	0.0118	0.00202
7.5	0.0117	0.00216
8	0.0115	0.00228
8.5	0.0113	0.00241
9	0.011	0.00253
10	0.01	0.00275
12	0.0076	0.00316
13	0.0061	0.00336
14	0.0043	0.00353
14.5	0.0041	0.00174
15	0.004	0.00169
17	0.0038	0.00174
18	0.0037	0.00172
20	0.0035	0.00162

57.IN:
5.7

3.05	0.0024	0.00019
3.5	0.008	0.00063
4	0.0108	0.00091
4.5	0.012	0.00114
5	0.0128	0.00135
5.5	0.0134	0.00153
6	0.0136	0.00169
6.5	0.0139	0.00185
7	0.0138	0.00199
7.5	0.0138	0.00213
8	0.0136	0.00226
8.5	0.0133	0.00238
9	0.013	0.0025
10	0.0122	0.00273
12	0.0098	0.00314
13	0.0084	0.00334
14	0.0067	0.00352
15	0.0049	0.0037
15.5	0.0046	0.00179
17	0.0044	0.00178
18	0.0043	0.0018
20	0.0041	0.00178

58.IN:
5.8

3.05	0.0028	0.00019
3.5	0.0088	0.00061
4	0.0125	0.00087
4.5	0.0141	0.0011
5	0.015	0.00131
5.5	0.0154	0.00149
6	0.0158	0.00166
6.5	0.016	0.00182
7	0.0159	0.00196
7.5	0.0159	0.0021
8	0.0158	0.00223
8.5	0.0156	0.00236
9	0.0153	0.00248
10	0.0145	0.00271
12	0.0122	0.00312
13	0.0107	0.00331
14	0.0092	0.0035
15	0.0074	0.00368
16	0.0054	0.00384
16.5	0.0052	0.00196
17	0.0051	0.00191
18	0.0049	0.00182
20	0.0047	0.00184

59.IN:
5.9

3.05	0.0029	0.00018
3.5	0.0096	0.00059
4	0.0136	0.00084
4.5	0.016	0.00106
5	0.017	0.00127
5.5	0.0175	0.00145
6	0.0178	0.00162
6.5	0.0181	0.00178
7	0.0182	0.00193
7.5	0.0182	0.00207
8	0.018	0.0022
8.5	0.0179	0.00233
9	0.0176	0.00245
10	0.0169	0.00268
12	0.0147	0.0031
13	0.0132	0.00329
14	0.0116	0.00347
15	0.0099	0.00365
16	0.008	0.00382
17	0.006	0.00399
17.5	0.0058	0.00212
18	0.0057	0.00207
20	0.0054	0.00202

6.IN:
6

3.05	0.0031	0.00018
3.5	0.0101	0.00057
4	0.0144	0.00082
4.5	0.0174	0.00103
5	0.0189	0.00123
5.5	0.0197	0.00141
6	0.0201	0.00159
6.5	0.0203	0.00175
7	0.0204	0.0019
7.5	0.0204	0.00204
8	0.0204	0.00217
8.5	0.0202	0.0023
9	0.0199	0.00242
10	0.0192	0.00265
12	0.0171	0.00307
15	0.0125	0.00363
17	0.0087	0.00397
17.5	0.0076	0.00404
17.6	0.0074	0.00406
17.7	0.0072	0.00408
17.8	0.007	0.0041
17.9	0.0067	0.0041
18	0.0065	0.00228
20	0.0061	0.00208

APPENDIX D

LISTING OF COMPUTER PROGRAMS REGARDING BUFFER ISSUES

D.1: *RIGID2.FOR*

```
*****************************************************************
*
*  RIGID2.FOR
*
*  PROGRAM TO LOOK AT ABSORBING A HEAT PULSE AT 4.2 K BY
*  ADDING HEAT TO A RIGID CONTAINER WITH COMPRESSED LIQUID
*
*
*  BRIAN BOWERS
*  CREATED 8-20-96
******************************************************************
*
*  VARIABLES USED
*
*     h1      -  Initial enthalpy of helium tank (J/kg)
*     h2      -  Final enthalpy of helium in tank (J/kg)
*     P1      -  Initial pressure of tank (Pa)
*     P1best  -  Best initial pressure found (Pa)
*     P2      -  Final pressure of tank (Pa)
*     P1best  -  Final pressure at best condition (Pa)
*     QV      -  Heat absorbed per unit volume of tank
*     QVbest  -  Best QV found (J/m3)
*     rho1    -  Initial density of helium in tank (kg/m3)
*     rho2    -  Final density of helium in tank (kg/m3)
*     T1      -  Initial temperature of tank (K))
*     Tmax    -  Maximum allowable temperature of tank (K)
*     u1      -  Initial internal energy of helium tank (J/kg)
*     u2      -  Final internal energy of helium in tank (J/kg)
*
*
******************************************************************
*
*  MY VARIABLE DECLARATIONS
*
      REAL  h1, h2
      REAL  P1, P2, P1best
      REAL  rho1, rho2
      REAL  QV, Qvbest
      REAL  T1, Tmax
      REAL  u1, u2
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT
*
********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=2
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
               JOUT=11000
*
**********************************************************************
**********************************************************************
**********************************************************************
*
*  Given Values
*
      print *,' Enter Tmax '
      read (*,*) Tmax
      T1 = 4.2

*  First Value to use in the iteration for the Starting Pressure
      P1 = 101325

100   continue

********************************************************************
*
*  FIND CONDITIONS AT START

        JIN1=1
        VALU1=P1
        JIN2=2
        VALU2=T1

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           rho1 = PROP(3,0)
           s1   = PROP(8,0)
```

```
            h1    = PROP(9,0)
            u1    = PROP(11,0)
            IF (IDID .EQ. 3) THEN
*                  This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

*    DENSITY IS CONSTANT IN THE RIGID CONTAINER
           rho2 = rho1

*    FIND FINAL CONDITIONS
         JIN1=3
         VALU1=rho2
         JIN2=2
         VALU2=Tmax

         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
      +     NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            P2    = PROP(1,0)
            s2    = PROP(8,0)
            h2    = PROP(9,0)
            u2    = PROP(11,0)
            IF (IDID .EQ. 3) THEN
*                  This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

**********************************************************************
*
*   FIND HEAT ABSORBED PER UNIT VOLUME
*
        QV = rho1*(u2-u1)
        write(*,5)'P1 =',P1/101325,' QV  = ',QV
5       format(1x,a5,f6.2,a6,f10.1)

        IF  (QV.gt.QVbest) Then
           Qvbest = QV
           P1best = P1
           P2best = P2
        ENDIF

         P1 = P1 + 101325

        IF (P1.LE  .101325*30) then
           goto 100
        ENDIF

        print *,' '
        WRITE(*,10)' QVbest = ',Qvbest,' J/m3'
10      FORMAT(1x,a10,f10.1,A5)

        WRITE(*,15)' 100% PULSE VOLUME ',1800E6/QVbest,' m3'
        WRITE(*,15)'  50% PULSE VOLUME ',900E6/QVbest,' m3'
        WRITE(*,15)'  10% PULSE VOLUME ',180E6/QVbest,' m3'
15      FORMAT(1X,A19,F7.1,A3)

        WRITE(*,20)' P1 = ',P1best/101324,'  T1 = ',T1,' P2 = ',
     ;             P2best/101325,' T2 = ',Tmax

20      FORMAT(1X,A6,F5.2,A7,F6.3,A6,F5.2,A6,f5.2)

        end
```

D2: *RIGID4.FOR*

```
*******************************************************************
*
*  RIGID4.FOR
*
*  PROGRAM TO LOOK AT ABSORBING A HEAT PULSE BY
*  ADDING HEAT TO A RIGID CONTAINER HELIUM STARTING AT 4.2 K
*
*
*  BRIAN BOWERS
*  CREATED 8-20-96
********************************************************************
*
*  VARIABLES USED
*
*     P1      -  Initial pressure of tank (Pa)
*     P1best  -  Best initial pressure found (Pa)
*     P2      -  Final pressure of tank (Pa)
*     P2best  -  Final pressure at best condition (Pa)
*     QV      -  Heat absorbed per unit volume of tank
*     QVbest  -  Best QV found (J/m3)
*     rho    -  Density of helium in tank (kg/m3)
*     rhoBest -  Final density of helium in tank (kg/m3)
*     T1      -  Initial temperature of tank (K)
*     Tmax    -  Maximum allowable temperature of tank (K)
*     u1      -  Initial internal energy of helium tank (J/kg)
*     u2      -  Final internal energy of helium in tank (J/kg)
*     x1      -  Initial quality
*     x1Best  -  Best initial quality
*
*
********************************************************************
*
*  MY VARIABLE DECLARATIONS
*
      REAL  P1, P2, P1best, P2best
      REAL  QV, QVbest
      REAL  rho , rhobest
      REAL  T1, Tmax
      REAL  u1, u2
      REAL  x1, x1Best
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT
*
**********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
*                NPRCIS=2
*
*        Units  (SI units)
*                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
*               JOUT=11000
*
**********************************************************************
**********************************************************************
**********************************************************************
*
*  Given Values
*
      print *,' Enter Tmax '
      read (*,*) Tmax
      T1 = 4.22

*  First Value to use in the iteration for the density
      rho = 5

100   continue

*********************************************************************
*
*  FIND CONDITIONS AT START

        JIN1=3
        VALU1=rho
        JIN2=2
        VALU2=T1

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           P1   = PROP(1,0)
           x1   = PROP(0,0)
           u1   = PROP(11,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF


*     FIND FINAL CONDITIONS
        JIN1=3
        VALU1=rho
        JIN2=2
        VALU2=Tmax
```

```
         CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
      +     NPRCIS, NUNITS)
         IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
            P2    = PROP(1,0)
            u2    = PROP(11,0)
            IF (IDID .EQ. 3) THEN
*                  This code not used in this case
            ENDIF
         ELSE
            CALL ERRMSG (MESSAG, IDID)
            WRITE (*, '(1X,A60)') MESSAG
            STOP
         ENDIF

************************************************************************
*
*    FIND HEAT ABSORBED PER UNIT VOLUME
*
        QV = rho*(u2-u1)


        IF  (QV.gt.QVbest) Then
           Qvbest = QV
           P1best = P1
           P2best = P2
           rhoBest = rho
           x1Best = x1
        ENDIF

        QM = (u2-u1)
        IF  (QM.gt.QMbest) Then
           QMbest = QM
           P1mbest = P1
           P2mbest = P2
           rhomBest = rho
           x1mBest = x1
        ENDIF


        write(*,5)'rho =',rho,' QV  = ',QV,'   QM = ',QM
5       format(1x,a5,f6.2,a6,f10.1,A8,f10.1)
        rho = rho + 1

        IF (rho.LE.110) then
           goto 100
        ENDIF

        print *,' '
        WRITE(*,11)' QVbest = ',Qvbest,' J/m3   rhoQVbest = ',rhoBest,
     ;             ' QM = ',QVbest/rhoBest,' J/kg'
        WRITE(*,11)' QMbest = ',QMbest,' J/kg   rhoQMbest = ',rhomBest,
     ;             ' QV = ',Qmbest*rhomBest,' J/m3'
11      FORMAT(1x,a10,f10.1,A20,f5.1,A7,f10.1,a5)

        WRITE(*,15)' 100% PULSE VOLUME ',1800E6/QVbest,' m3'
        WRITE(*,15)'  50% PULSE VOLUME ',900E6/QVbest,' m3'
        WRITE(*,15)'  10% PULSE VOLUME ',180E6/QVbest,' m3'
15      FORMAT(1X,A19,F7.1,A3)

        WRITE(*,20)' rhoBest = ',rhoBest,'  P1 = ',P1best/101324,
     ;             '  P2 = ', P2best/101325,'  x1 = ',x1Best,
     ;             '  T2 = ',Tmax

20      FORMAT(1X,A11,F5.1,A7,F5.2,A7,F5.2,A7,f6.3,A7,f5.2)

        end
```

D3: DOUBLE3.FOR

```
*******************************************************************
*
*  DOUBLE3.FOR
*
*  PROGRAM TO LOOK AT ABSORBING A HEAT PULSE BY ADDING HEAT TO A
*  TWO PHASE TANK WITH AN AUXILIARY TANK TO HOLD SOME OF THE VAPOR.
*  THIS PROGRAM USES T AS THE INDEPENDENT VARIABLE AND ITERATES ON THE
*  MASS THAT MOVES BETWEEN THE TANKS.
*
*
*  BRIAN BOWERS
*  CREATED 10-8-96
********************************************************************
*
*  VARIABLES USED
*
*     DIR      - Direction of flow of mass between tanks (1 - into
*                tank 1,  2 - into tank 2)
*     DIRFLAG -  Flag variable that tells if you have already switched
*                directions (1 if you have switched, 0 if you have not)
*     dm       - Increment of mass vented moving between the tanks (kg)
*     dQ       - Heat absorbed as dm moves (J)
*     dqinit   - Initial dQ (J)
*     hlv      - Current enthalpy of vapor in tank 1 (J/kg-K)
*     hlvold   - Previous enthalpy of tank 1 (J/kg-K)
*     hV       - Enthalpy of vapor at start
*     h2       - Current enthalpy of tank 2 (J/kg-K)
*     h2old    - Previous enthalpy of tank 1 (J/kg-K)
*     m1       - Current mass in tank 1 (kg)
*     m1old    - Previous mass in tank 1 (kg)
*     m1final -  Final mass in tank 1 at best QV condition (kg)
*     m1start -  Initial mass in tank 1 (kg)
*     m2       - Current mass in tank 2 (kg)
*     m2old    - Previous mass in tank 1 (kg)
*     m2final -  Final mass in tank 2 at best QV condition (kg)
*     m2start -  Initial mass in tank 2 (kg)
*     P        - Current pressure of tanks (Pa)
*     Pold     - Previous pressure of tanks (Pa)
*     Pstart   - Initial pressure of tanks 1 and 2 (Pa)
*     Pfinal   - Final pressure of both tanks at QVbest conditions (Pa)
*     Q        - Total heat absorbed (J)
*     Qbest    - Total heat absorbed at QVbest conditions (J)
*     QV       - Heat absorbed per unit volume of tank (J/m3)
*     QVbest   - Best QV found (J/m3)
*     rhoStart-  Initial overall density of tanks 1 and 2 (kg/m3)
*     rhoBest -  Best initial density(kg/m3)
*     rhoL     - Initial density of liquid (kg/m3)
*     rhoV     - Initial density of vapor (kg/m3)
*     rho1     - Current density of helium in tank 1 (kg/m3)
*     rho1old -  Previous density of helium in tank 1 (kg/m3)
*     rho2     - Current density of helium in tank 2 (kg/m3)
*     rho2old -  Previous density of helium in tank 1 (kg/m3)
*     s1       - Current entropy in tank 1 (J/kg-K)
*     s1old    - Previous entropy in tank 1 (J/kg-K)
*     s1start -  Initial value of entropy in tank 1 (J/kg-K)
*     s2       - Current entropy in tank 2 (J/kg-K)
*     s2old    - Previous entropy in tank 2 (J/kg-K)
*     s2start -  Initial value of entropy in tank 2 (J/kg-K)
*     sV       - Initial entropy of vapor (J/kg-K)
*     T1       - Current temperature of tank 1 (K)
*     T1max    - Maximum allowable temp of tank 1 (K)
*     T1old    - Previous temp of tank 1 (K)
*     T2       - Current temperature of tank 2 (K)
*     T2old    - Previous temp of tank 2 (K)
*     Tstart   - Starting temp of tanks 1 and 2 (K)
*     u1       - Current internal energy of tank 1 (J/kg)
*     u1old    - Previous internal energy (J/kg)
*     u1start -  Starting internal energy of tank 1 (J/kg)
*     u2       - Current internal energy of tank 2(J/kg)
*     u2old    - Previous internal energy in tank 2 (J/kg)
*     u2start -  Starting internal energy of tank 2 (J/kg)
*     uV       - Internal energy of vapor in tank 1
*     V1       - Volume of tank 1 (m3)
*     V1best   - Volume of tank 1 at best QV condition (m3)
*     V2       - Volume of tank 2 (m3)
*     V2best   - Volume of tank 2 at best QV condition (m3)
*     V2V1     - Ratio of V2 to V1
*     V2V1best-  Ratio of V2 to V1 at best QV
*     xStart   - Initial quality
*     xSbest   - Best initial x
*
*
********************************************************************
*
*  MY VARIABLE DECLARATIONS
*

      REAL  dm, dQ
      REAL  hlv, hlvold, h2, hV
      REAL  m1, m1old, m1final, m1start
      REAL  m2, m2old, m2final, m2start
      REAL  P, Pold, Pstart, Pfinal
      REAL  Q, Qbest
      REAL  QV, QVbest
      double precision  rhoStart, rhoL, rhoV
      double precision rho1 , rho1old, rho2, rho2old
      REAL  s1, s1old, s1start, s2, s2old, s2start, Sv
      REAL  T1, T1max, T1old, T2, T2old, Tstart
      double precision  u1, u1old, u1start, u2, u2old, u2start, uV
      REAL  V1, V1best, V2, V2best
      REAL  V2V1, V2V1best
      REAL  x1Start, x1sbest
*
*   Variable assignments needed for HEPROP
```

```
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT
*
*********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=3
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
               JOUT=11000
*
**********************************************************************
**********************************************************************
**********************************************************************
*
*  Given Values
*
      print *,' Enter Tmax in tank 1'
      read(*,*) T1max


      Tstart = 4.22
      V1 = 1000


**********************************************************************
*
*   Choosing a ratio of V2 to V1
*
*       V2V1 = 0.01
      print *,' Enter ratio of V2 to V1'
      read(*,*) V2V1

      V2 = V1 * V2V1

*  CHOOSING AN INCREMENT IN TEMPERATURE AS A FRACTION OF
*  THE TOTAL ALLOWABLE CHANGE IN TEMPERATURE
*
        NT = 50
        dT = (T1max - Tstart) / NT


*     INITIAL QUALITY IN TANK 1
       print *,' enter starting quality in tank 1'
       read(*,*) x1Start
**********************************************************************
*
*   Opening and output file
*
*       OPEN(UNIT = 10, FILE = 'DOUBLE.OUT',STATUS ='UNKNOWN')

*********************************************************************
*
*  STARTING POINT FOR CALCULATION

100   Continue

*  Setting current state
      T1 = Tstart
      T2 = Tstart
      x1 = x1Start
      Q = 0.0

*  FIND INITIAL VAPOR AND LIQUID DENSITY AND CONDITIONS IN TANK 1

        JIN1=2
        VALU1=T1
        JIN2=9
        VALU2=x1

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           P    = PROP(1,0)
           T1   = PROP(2,0)
           rho1 = PROP(3,0)
           s1   = PROP(8,0)
           u1   = PROP(11,0)
           IF (IDID .EQ. 3) THEN
               h1V  = PROP(9,2)
           ELSE
               h1V  = PROP(9,0)
           ENDIF

           rhoL = PROP(3,1)
           rhoV = PROP(3,2)
           sV   = PROP(8,2)
           hV   = PROP(9,2)
           uV   = PROP(11,2)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF

        IF (rhoL.eq.0.OR.rhoV.eq.0) THEN
```

```
          print *,'  Error!  Not 2 phase start'
          stop
       ENDIF

*  FIND INITIAL MASS AND DENSITY IN TANK 1 AND TANK 2
*
       rho2 = rhoV


       m2 = rho2 * V2
       m1 = rho1 * V1

*  OVERALL STARTING DENSITY

       rhoStart = (m1+m2)/(V1+V2)


*  CONDITIONS IN TANK 2 ARE THE SAME AS THE VAPOR
*
       u2 = uV
       h2 = hV
       s2   = sV

       if(T1.LT.4.2) then
          print *,'error  not 2 phase start -  Temp below 4.2 K'
          stop
       endif

*   Save starting conditions

       m1start = m1
       m2start = m2
       s1start = s1
       s2start = s2
       Pstart = P
       u1start = u1
       u2start = u2


*       write(10,24) ' rhoI =  ',rhoStart
*       write(10,25) ' V1 =    ',V1
*       write(10,25) ' V2 =    ',V2
*       write(10,25) ' Vt =    ',V1+V2

*       write(10,20) '  P           m1        s1         T1   ',
*     ;                         '     m2         s2      T2',
*     ;   '  m1/m1init    S1/S1init   S2/S1init    P       dQ/dQinit',
*     ;   '  Q/Vtot '

*20      format (a40,a24,a58,a10)
*24      format (1x,a10,f6.2)
*25      format (1x,a10,f7.1)


*       print *,' m1 = ',m1,'  m2 = ',m2
*       print *,' dm = ',dm
*       print *,' V2 = ',V1,'  V2 = ',V2
*       print *,' u1 = ',u1,'  u2 = ',u2
*       print *,' T1 = ',T1,'  T2 = ',T2
*       print *,' P = ',P
*       print *,' h1V = ',h1V,'  h2 = ',h2
*       print *,' rhoStart = ',rhostart
*       print *,' rho1 = ',rho1,'  rho2 = ',rho2



*  GUESSING AN INCREMENT IN MASS
*
       dm = m2 /1000

**********************************************************************
*  BEGINNING THE ITERATION LOOP

      DO 200, I = 1,NT

*  SAVING PREVIOUS CONDITIONS OF TANK 1 and TANK 2
       Pold = P
       m1old = m1
       m2old = m2
       u1old = u1
       u2old = u2
       h1Vold = h1V
       h2old  = h2
       rho1old = rho1
       rho2old = rho2
       s1old = s1
       s2old = s2
       T1old = T1
       T2old = T2

       n=0

250    continue
*  USING THE TEMP CHANGE AND THE ASSUMED MASS FLOW TO FIND THE NEW CONDITIONS IN TANK 1
*

        T1 = T1 + dT
        rho1 = rho1 - dm/V1


           JIN1=2
           VALU1=T1
           JIN2=3
           VALU2=rho1

           CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +                NPRCIS, NUNITS)
```

```
          IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
             x1    = PROP(0,0)
             P     = PROP(1,0)
             s1    = PROP(8,0)
             u1    = PROP(11,0)
             IF (IDID .EQ. 3) THEN
                h1V  = PROP(9,2)
             ELSE
                h1V  = PROP(9,0)
             ENDIF
          ELSE
             print *,'rho1 error   rho 1 = ',rho1
             CALL ERRMSG (MESSAG, IDID)
             WRITE (*, '(1X,A60)') MESSAG
             STOP
          ENDIF


*  NEW CONDITIONS IN TANK 2

          rho2 = rho2 +dm/V2

          JIN1=3
          VALU1=rho2
          JIN2=1
          VALU2=P

          CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +               NPRCIS, NUNITS)
          IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
             T2    = PROP(2,0)
             s2    = PROP(8,0)
             h2    = PROP(9,0)
             u2    = PROP(11,0)
             IF (IDID .EQ. 3) THEN
*                This code not used in this case
             ENDIF
          ELSE
             print *,'rho2 error  rho2 = ', rho2
             CALL ERRMSG (MESSAG, IDID)
             WRITE (*, '(1X,A60)') MESSAG
             STOP
          ENDIF


*   dq USING TANK 1 ENERGY BALANCE

         if(dm.gt.0) then
            dq = (m1-dm)*u1 - m1*u1old + dm*h1Vold
         else
            dq = (m1+dm)*u1 - m1*m1old - dm*h2old
         endif

*   dm USING OVERALL ENERGY BALANCE

            dm2 = (  m1*(u1old - u1) + m2*(u2old-u2) + dq ) / (u2-u1)


*   COMPARING GUESSED dm and dm2

*            print *,' dm  =      ',dm
*            print *,' dm2 = ',dm2
*            read(*,*)

            if (abs((dm-dm2)/dm).lt.0.01) then
*               print *,'T1 = ',T1

               m1 = m1 - dm
               m2 = m2 + dm
               Q = Q + dq

               n = 0

            else
* if same sign then average, otherwise take negative of dm2
*               if(abs(dm+dm2).gt.abs(dm-dm2) ) then


                 dm = dm + (dm-dm2)/20

                 if (dm.gt.m1.or.dm.gt.m2) then
                     dm = dm/2
                 endif

                n = n + 1
                if (n.gt.3000) then
                   print *,'not converging'
                 print *,' dm  =      ',dm
                 print *,' dm2 = ',dm2
*                 read(*,*)

                  stop
                endif
*              else
*                 dm = dm + (dm-dm2)/2
*              endif
               P     = Pold
               u1    = u1old
               u2    = u2old
               h1V   = h1Vold
               h2    = h2old
               rho1  = rho1old
               rho2  = rho2old
               s1    = s1old
```

```
            s2   = s2old
            T1   = T1old
            T2   = T2old
            goto 250
         endif

200     continue

*
*           write(10,30) P/101325,m1,s1,T1,m2,s2,T2,
*     ;     m1/m1start,
*     ;     m1*s1/(m1start*s1start), m2*s2/(m1start*s1start),
*     ;     P/101325, dq/dqinit, Q/(V1+V2)

*30         format (1x,f6.4,2x,f10.0,2x,f8.1,2x,f6.3,2x,
*     ;                        f10.0,2x,f8.1,1x,f6.3,3x,
*     ;             f7.6,4x,f8.5,3x,f8.5,5x,f6.3,2x,f6.3,
*     ;             4x,f9.1)
*          print *,' P = ',P,' T1 = ',T1,' m1 = ',m1
*         read(*,*)
*           GOTO 200
*        ELSE
          QV = Q/(V1+V2)
          QM = Q/(m1+m2)
          WRITE(*,4)' QV = ',QV,' J/M3   V2 / V1 =',V2/V1,
     ;               '  x1start = ',x1start,' QM = ',QM
4         format(a7,f9.1,a17,f5.2,a11,f4.3,a6,f10.1)
          IF  (QV.GT.QVbest) then
             QVbest = QV
             Qbest  = Q
             V2V1best = V2/V1
             Pfinal = P
             m1final = m1
             m2final = m2
             V1best  = V1
             V2best  = V2
             T1final = T1
             T2final = T2
             x1Sbest  = x1Start
          ENDIF

          if (x1start-.01.gt.0) then
             x1start = x1start - .01
             goto 100
          endif


*          print *, 'rhoStart = ',rhostart
*          rhoStart = rhoStart + 5
*          if (rhoStart.lt.130) then
*             goto 100
*          endif

*        ENDIF



***********************************************************************
*
*   OUTPUT
*
       print *,' '
       WRITE(*,5) ' Tstart = ',Tstart,' rhoStart = ',rhostart
5      FORMAT(1X,A11,F5.2,2x,a13,f5.1)
       WRITE(*,6) ' Pfinal = ',P/101325,' atm '
6      FORMAT(1X,A11,F5.2,A4)
       WRITE(*,7) ' T1final = ',T1final,' K   T2final = ',T2final,' K'
7      FORMAT(1X,A12,F5.2,A15,f5.2,a2)

       WRITE(*,10)' QVbest = ',Qvbest,' J/m3'
10     FORMAT(1x,a11,f10.1,A5)
       WRITE(*,11)' V2 / V1 best = ', V2V1best
11     FORMAT(1X,A16,F6.3)
       WRITE(*,12)' x1startBest = ',x1sBest
12     FORMAT(1X,A15,F5.3)


*       WRITE(*,15)' 100% PULSE VOLUME ',1800E6/QVbest,' m3'
*       WRITE(*,15)'  50% PULSE VOLUME ',900E6/QVbest,' m3'
*       WRITE(*,15)'  10% PULSE VOLUME ',180E6/QVbest,' m3'
*15     FORMAT(1X,A19,F7.1,A3)

       end
```

D4: *FLOAT.FOR*

```
***********************************************************************
*   FLOAT.FOR
*
*   Program for finding the energy that can be absorbed per total
*   helium volume when adding heat to a container with liquid helium in
*   the bottom and room temp helium in the top (separated by an insulated float)
*
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   11-7-96
*
*   last modified 11-7-96
**********************************************************************
*
*   VARIABLES USED
*
*     P1      - Initial pressure in both sections (Pa)
*     P2      - Final pressure in both sections (Pa)
*     QVtot   - Ratio of heat entering system to total helium volume (J/m3)
*     R       - Ratio of initial warm volume to cold volume
*     rho1    - Initial density in the liquid section (kg/m3)
*     s       - Change in volume of liquid per initial unit volume
*     T2      - Final temp in cold end (K)
*     u1      - Initial specific internal energy in the liquid section (J/kg)
*     u2      - Final specific internal energy in the liquid section  (J/kg)


*********************************************************************
*
*   My Variable assignments
        real P1, P2, Qvtot, R, rho1, s, T2, u1, u2
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
   .  CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
**********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                 NPRCIS=2
*
*        Units  (SI units)
                 NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
                JOUT=11000
*
***********************************************************************


*        print *,'Enter initial liquid density'
*        read (*,*) rho1
        print *,'Enter final liquid temp'
        read (*,*) T2
*        print *,'Enter change in liquid volume ',
*     ;           'per initial unit volume (S)'
*        read (*,*) S

        rho1 = 10

*    looping through rho terns

        do while (rho1.le.180)

        s = 0.01
        print *,'rho1 = ',rho1

**************************************************************************
*       Finding the liquid conditions at the start
*
        JIN1=2
        VALU1=4.22
        JIN2=3
        VALU2=rho1

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           P1 = PROP(1,0)
           u1 = PROP(11,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF

*    looping through s terms
       do while (s.lt.3)
```

```
***************************************************************************
*       Finding the liquid conditions at the end
*
        JIN1=2
        VALU1=T2
        JIN2=3
        VALU2=rhol/(1+S)

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           P2 = PROP(1,0)
           u2 = PROP(11,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF


        R = S / (1-P1/P2)

       QVtot = rhol*(u2-u1)/(1+R) + P1*log( 1/(1-S/R) ) / (1+1/R)

       if (Qvtot.gt.QVbest.and.R.gt.0) then
             Qvbest = Qvtot
             P1best = P1
             P2best = P2
             Rbest = R
             Rholbest = Rhol
             Sbest = S
       endif

        s= s+0.01

       enddo

        rhol = rhol + 1

       enddo

       write(*,10)' QVbest = ',QVbest,' J/m3    Rholbest = ',
     ;             Rholbest,' kg/m3',' P1best = ',P1best,
     ;        '   P2best = ',P2best,
     ;       '   Rbest = ',Rbest,' Sbest =',Sbest
10     format(a11,f12.1,a25,f5.1,a6,/a14,f12.1,a14,f12.1/
     ;  a11,f5.3,a12,f5.3)

                                                      END
```

APPENDIX E:

COMPUTER PROGRAM FOR TRANSIENT MODELING OF RECIRCULATOR LOOP (*TRANS6.EXE*)

E.1: Flow Chart and Description

A flow chart of the computer program is shown in Figure E.1. The program consists of a main module and 7 subroutines that were written for this project. In addition, a subroutine called HEPROP by Cryodata is used to find properties of helium and several subroutines from Numerical Recipes by Press, et al. are used to solve matrix equations (these are not shown in Figure E.1). Collectively, the main module and subroutines are called *TRANS6.EXE*. The main module, called TRANS6, is used to call the subroutines.

The program *TRANS6.EXE* was written in FORTRAN and compiled using Microsoft FORTRAN 5.00. The compete list of source codes and object codes that are needed is given in Table E.1.

NAME	AUTHOR	DESCRIPTION
TRANS6.FOR	B. Bowers	Main module
BUFFER3.FOR	B. Bowers	Finds initial conditions in buffer tank
INIT7.FOR	B. Bowers	Finds initial (steady state) conditions in recirculator loop
STEADY5.FOR	B. Bowers	Solves for steady conditions in loop for given inlet conditions (m'_{in}, T_{in}, P_{in})
PUMP2.FOR	B. Bowers	Uses pump curve to find mass flow rate that matches a given pressure change
QLOAD3.FOR	B. Bowers	Finds heat rejected to the buffer tank from the recirculator loop
UNSTEAD10.FOR	B. Bowers	Performs iteration to find transient solution consistent with the pump characteristics
POOL.3FOR	B. Bowers	Finds current conditions in buffer tank
HEPROP.OBJ	Cryodata	Finds properties of helium
LINBCG.FOR	Numerical Recipes	Part of sparse matrix solver
ATIMES.FOR	Numerical Recipes	Part of sparse matrix solver
ASOLVE.FOR	Numerical Recipes	Part of sparse matrix solver
DSPRSAX.FOR	Numerical Recipes	Part of sparse matrix solver
DSPRSTX.FOR	Numerical Recipes	Part of sparse matrix solver
SRNM.FOR	Numerical Recipes	Part of sparse matrix solver

Table E.1: FORTRAN Files Needed for *TRANS6.EXE*

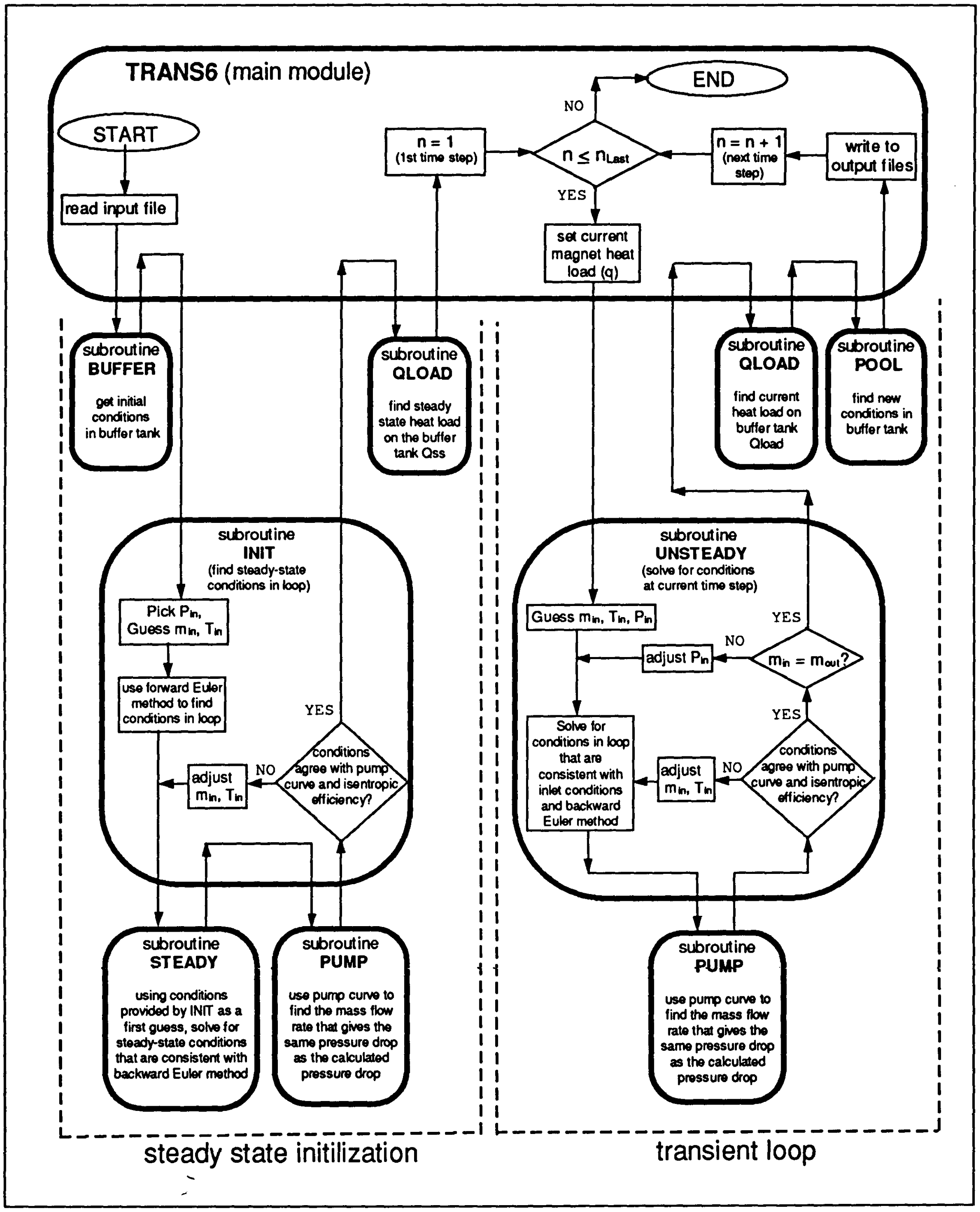


Figure E.1: Flow Chart of *TRANS6.EXE*

The first part of the program finds the initial (steady state) conditions of the recirculator loop and buffer tank. TRANS6 reads the input file and then calls subroutine BUFFER to find the initial conditions in the buffer tank. Subroutine INIT is then called to find the initial steady state conditions in the recirculator loop. The conditions are found for a given inlet pressure (P_{in}). The correct inlet temperature and mass flow rate (T_{in}, m'_{in}) are eventually found, but must be guessed to start the solution. INIT uses a forward Euler method (similar to *CICC.EXE*) to estimate the conditions in the recirculator loop. INIT then calls subroutine STEADY, which uses the values from INIT as a starting point for an iteration that finds conditions consistent with a Backward Euler method. The conditions are then checked against the pump curve using subroutine PUMP. This subroutine finds the mass flow rate that is required for the pressure rise of the pump to match the system pressure drop that was calculated in STEADY. If the current guess for the mass flow rate does not agree with the one found in PUMP, or the inlet and outlet conditions of the pump do not agree with the isentropic efficiency, the inlet conditions (T_{in}, m'_{in}) are changed and the loop is repeated until they do agree. INIT then returns control to TRANS6. Subroutine QLOAD is then called to find the steady state refrigeration load.

The transient portion of the program finds the conditions as the magnet heat load is changed. For each time step, the heat load is set based on the discharge profile that is chosen. The subroutine UNSTEADY is then called to find the new conditions in the recirculator loop. UNSTEADY uses an iteration routine that varies the inlet conditions (P_{in}, m'_{in}, and T_{in}) until the solution 1) is consistent with the pump curve, 2) is consistent with the pump isentropic efficiency, and 3) has the same mass flow rate at the pump inlet and outlet. Subroutine QLOAD is then called to find the rate of heat rejection to the buffer tank. Subroutine POOL is called to find the new conditions in the buffer tank. The data for the current time step is then written to an output file. Finally, the time is incremented to prepare for the next time step. This transient loop is repeated until all time steps have been evaluated.

The program was found to work well for many flow passages. However, it was sometimes unstable with very long or very narrow flow passages. In addition, the program was often unstable when using time steps of less than 0.1 sec. Part of this instability may be caused by using an iteration criteria that is too strict. However, some of the instability is probably due to the finite difference method that was used. This stability problem should be addressed if the code is to ever be used on a regular basis as a SMES design tool.

E.2: Sample Input File

```
      MAGNET FLOW PASSAGE
Total Area of Magnet Flow Passages (m2)
0.0001
Total Hydraulic Diameter of Magnet Flow Passages (m)
0.0004
Length of Magnet Flow Passages
20
Friction Factor of Magnet Flow Passages
0.15
     FIRST HEAT EXCHANGER
Total Flow Area of Heat Exchanger 1 (m2)
0.0039
Total Hydraulic Diameter of Heat Exchanger 1 (m)
0.01
Length of heat exchanger 1 (m)
5
Friction Factor of Heat Exchanger 1
0.15
Overall heat transfer coefficient per unit length of hx1 (W/m-K)
5
     SECOND HEAT EXCHANGER
Total Flow Area of Heat Exchanger 2 (m2)
0.0039
Total Hydraulic Diameter of Heat Exchanger 2 (m)
0.01
Length of heat exchanger 2 (m)
10
Friction Factor of Heat Exchanger 2
0.15
Overall heat transfer coefficient per unit length of hx2 (W/m-K)
5
     RECIRCULATOR PUMP
Isentropic efficiency of pump
0.6
     BUFFER TANK (POOL)
Volume of buffer tank (m3)
0.1
Initial quality of tank
0.09
     INITIAL CONDITIONS
Initial temperature of the pool
4.3
Temperature (K) at HX1 inlet (pump exit) before pulse'
5.0
Pressure (atm) at HX1 inlet (pump exit) before pulse'
8
Initial heat rejection in magnet(W/m)'
0.03
Initial mass flow rate (kg/s)'
0.004
      HEAT LOAD PROFILE
Heat rejection in magnet during pulse (W/m)'
5.0
Heat rejection in magnet during non-pulse times(W/m)'
0.03
Heat rejection in magnet during recovery period (W/m)
0.03
      NUMBER OF NODES BEING CONSIDERED
Number of nodes in the magnet
31
Number of nodes in heat exchanger 1
11
Number of nodes in heat exchanger 2
11
      TIME INFORMATION
Time step during cycling (seconds)
0.1
Time step during recovery after cycling (seconds)
0.1
Number of time steps of pulsed load per cycle (sec)
40
Number of time steps of non-pulse load per cycle (sec)
450
Total number of pulse/non-pulse cycles  (sec)
5
Number of time steps of recovery after cycling (sec)
2000
```

E.3: Main Module TRANS6

```
*********************************************************************
*   TRANS6.FOR
*
*   Main module for evaluating transient load in recirculator loop.
*
*   Note the x vector is used to hold the values of T, P, and m for
*   all nodes in the loop.  Where the T's, P's and m's for a
*   particular node n in the conductor are found by:
*                  T(n) = x(3*n-2)
*                  P(n) = x(3*n-1)
*                  m(n) = x(3*n)
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   10-31-96
*
*   last modified 5-19-97
********************************************************************
*
*   VARIABLES USED
*
*     AcMag   - Cross sectional area of magnet flow passage (m2)
*     AcHx1   - Cross sectional area of heat exchanger 1 flow
*               passage (m2)
*     AcHx2   - Cross sectional area of heat exchanger 2 flow
*               passage (m2)
*     Allo    - Flag to determine whether A, b, fm , fP, and fT arrays
*     Cp(n)   - Specific heat at node n (J/kg-K)
*     DhHx1   - Hydraulic diameter of heat exchanger 1 (m)
*     DhHx2   - Hydraulic diameter of heat exchanger 2 (m)
*     DhMag   - Hydraulic diameter of flow passage (m)s
*     dt      - Time step between calculations during pulsing (sec)
*     dtRecov - Time step between calculations during recovery (sec)
*     dPdR(n) - dP/dRho at constant temperature at node n (Pa-m3/kg)
*     dPdT(n) - dP/dT at constant density at node n (Pa/K)
*     dxHx1   - Distance between nodes in heat exchanger 1
*     dxHx2   - Distance between nodes in heat exchanger 2
*     dxMag   - Distance between nodes in magnet (m)
*     eta     - Isentropic efficiency of recirculator pump
*     fhx1    - Friction factor in heat exchanger 1
*     fhx2    - Friction factor in heat exchanger 2
*     fmag    - Friction factor in magnet
*     Lhx1    - Length of heat exchanger 1 (m)
*     Lhx2    - Length of heat exchanger 2 (m)
*     Lmag    - Length of magnet flow passage (m)
*     h(n)    - Specific enthalpy of helium at node n (J/kg)
*     Line    - Variable used to read past a comment line in input file
*     m       - Mass flow rate at node n (kg/s)
*     min     - Inlet mass flow rate to the loop (kg/s)
*     minit   - Steady state mass flow rate through loop (kg/s)
*     mtank   - Mass of helium in the buffer tank (kg)
*     nCycle  - Current cycle being evaluated
*     nCycTot - Total number of pulse/non-pulse cycles in simulation
*     Nin     - Value of node number n for the conductor inlet
*     Nout    - Value of node number n for the conductor outlet
*     nPulse  - Number of time steps that the pulse lasts
*     nNoPulse- Number of time steps that the non-pulse load lasts
*     nRecover- Number of time steps in the recovery period
*     nTime   - Counter to tell where in the pulse/non-pulse cycle
*               you are at
*     Ntot    - Total number of nodes in system (1/3 of the size of x )
*     P(n)    - Pressure at node n (Pa)
*     Pin     - Initial magnet inlet pressure (Pa)
*     Pmin    - Minimum allowable pressure in conductor (Pa)
*     Ptank   - Pressure in the buffer tank (Pa)
*     q       - Current heat flow rate per unit length in magnet(W/m)
*     qPulse  - Heat rejection in magnet per unit length during pulse (W/m)
*     qNoPulse- Heat rejection in magnet  rate per unit length during
*               non-pulse times (W/m)
*     qRecover- Heat rejection in magnet per unit length during recovery (W/m)
*     Qhx1    - Heat load being rejected to the buffer tank from hx1 (W)
*     Qhx2    - Heat load being rejected to the buffer tank from hx2 (W)
*     Qpool   - Current heat load being rejected to the buffer tank (W)
*     Qss     - Steady state heat load passing through the buffer
*               tank (W)
*     Rho(n)  - Density at node n (kg/m3)
*     s(n)    - Specific entropy at node n (J/kg-K)
*     T(n)    - Temperature at node n (K)
*     Tin     - Initial magnet inlet temperature (K)
*     Tpool   - Temperature in the pool (K)
*     time    - Current time in the magnet cycle (sec)
*     Tmax    - Maximum allowable temperature at any location(K)
*     uaHx1   - Overall heat transfer coefficient PER UNIT LENGTH
*               of HX1 (W/m-K)
```

```
*     uaHx2   - Overall heat transfer coefficient PER UNIT LENGTH
*               of HX1 (W/m-K)
*     Vtank   - Volume of buffer tank pool (m3)
*     x(n)    - Vector containing T, P, and m data for all nodes
*     xp(n)   - Vector containing previous x(n)
*
*********************************************************************
*
*   My Variable assignments
        real AcHx1, AcHx2, AcMag
        integer allo
        DOUBLE PRECISION Cp[allocatable](:)
        real DhHx1, DhHx2, DhMag
        double precision dt
        DOUBLE PRECISION dPdR[allocatable](:),  dPdT[allocatable](:)
        DOUBLE PRECISION dxHx1, dxHx2, dxMag
        real eta
        real fhx1, fhx2, fmag
        real Lhx1, Lhx2, Lmag
        CHARACTER*80 LINE
        REAL h[allocatable](:)
        DIMENSION m[allocatable](:)
        double precision minit, min
        real mtank
        Integer nCycle,nCycTot,nNoPulse, nPulse, nRecover, nTime
        Integer Nin, Nout, Ntot
        DOUBLE PRECISION P[allocatable](:)
        DOUBLE PRECISION Pin, Pmin
        REAL Ptank
        real q, qPulse, qNoPulse, qRecover, Qpool, Qss, Qhx1, Qhx2
        DOUBLE PRECISION Rho[allocatable](:), s[allocatable](:)
        DIMENSION T[allocatable](:)
        double precision Tin
        double precision Tmax, Tpool
        double precision time
        real uaHx1, uaHx2
        real Vtank
        DOUBLE PRECISION x[allocatable](:) , xp[allocatable](:)

*********************************************************************
*
*   OPENING AN INPUT FILE
*
        OPEN(UNIT=10,FILE='TRANS.IN',STATUS='OLD')

*********************************************************************
*
*   Reading Initial Conditions and Physical Description of Loop
*
80      format(a80)

*MAGNET FLOW PASSAGE
*  Total Area of Magnet Flow Passages (m2)
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) AcMag
*  Total Hydraulic Diameter of Magnet Flow Passages (m)
        read (10,80) LINE
        read (10,*) DhMag
*  Length of Magnet Flow Passages
        read (10,80) LINE
        read (10,*) Lmag
*  Friction Factor of Magnet Flow Passages
        read (10,80) LINE
        read (10,*) fmag

*FIRST HEAT EXCHANGER
*  Total Area of Heat Exchanger 1 (m2)
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) AcHx1
*  Total Hydraulic Diameter of Heat Exchanger 1 (m)
        read (10,80) LINE
        read (10,*) DhHx1
*  Length of heat exchanger 1 (m)
        read (10,80) LINE
        read (10,*) Lhx1
*  Friction Factor of Heat Exchanger 1
        read (10,80) LINE
        read (10,*) fhx1
*  Overall heat transfer coefficient per unit length of hx1 (W/m-K)
        read (10,80) LINE
        read (10,*) uaHx1

*SECOND HEAT EXCHANGER
*  Total Area of Heat Exchanger 2 (m2)
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) AcHx2
*  Total Hydraulic Diameter of Heat Exchanger 2 (m)
        read (10,80) LINE
        read (10,*) DhHx2
*  Length of heat exchanger 2 (m)
```

```
        read (10,80) LINE
        read (10,*) Lhx2
*  Friction Factor of Heat Exchanger 2
        read (10,80) LINE
        read (10,*) fhx2
*  Overall heat transfer coefficient per unit length of hx2 (W/m-K)
        read (10,80) LINE
        read (10,*) uaHx2

*RECIRCULATOR PUMP
*  Isentropic efficiency of pump
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) eta

*BUFFER TANK (POOL)
*   Volume of buffer tank (m3)
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) Vtank
*Initial quality of tank
        read (10,80) LINE
        read (10,*) xtank
*INITIAL CONDITIONS
*   Initial temperature in pool
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) Tpool
*   temperature (K) at magnet inlet before pulse'
        read (10,80) LINE
        read (10,*) Tin
*   Pressure (atm) at magnet inlet before pulse'
        read (10,80) LINE
        read (10,*) Pin
        Pin = Pin *101325
*   Initial heat rejection in magnet (W/m)'
        read (10,80) LINE
        read (10,*) q
*   Initial mass flow rate (g/s)'
        read (10,80) LINE
        read (10,*) minit

*HEAT LOAD PROFILE
*   Heat rejection in magnet during pulse (W/m)'
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) qPulse
*   Heat rejection in magnet during non-pulse times (W/m)'
        read (10,80) LINE
        read (10,*) qNoPulse
*   Heat rejection in magnet during recovery period (W/m)
        read (10,80) LINE
        read (10,*) qRecover


*NUMBER OF NODES BEING CONSIDERED
*   Magnet
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) Nmag
*   First HX
        read (10,80) LINE
        read (10,*) Nhx1
*   Second HX
        read (10,80) LINE
        read (10,*) Nhx2
*
*TIME INFORMATION
*   Time step  during cycling (sec)
        read (10,80) LINE
        read (10,80) LINE
        read (10,*) dt
*   Time step during recovery
        read (10,80) LINE
        read (10,*) dtRecov
*   Number of time steps of pulsed load per cycle (sec)
        read (10,80) LINE
        read (10,*) nPulse
*   Number of time steps of non-pulse load per cycle (sec)
        read (10,80) LINE
        read (10,*) nNoPulse
*   Total number of cycles (sec)
        read (10,80) LINE
        read (10,*) nCycTot
*   Number of time steps of recovery after cycling (sec)
        read (10,80) LINE
        read (10,*) nRecover

        CLOSE (10)

***********************************************************************
*   Total number of nodes in system
        Ntot = Nmag + Nhx1 + Nhx2
```

```
*******************************************************************
*
*   Dimension vectors
*
       allocate ( x(3*Ntot),xp(3*Ntot),Cp(Ntot), dPdR(Ntot), dPdT(Ntot),
     ;         h(Ntot), m(Ntot), P(Ntot), Rho(Ntot), s(Ntot), T(Ntot))


*******************************************************************
*
*   Temperature and pressure constraints
*
*   Maximum Temp allowed in magnet
        Tmax = 6.0

*   Minimum Pressure allowed anywhere in loop
        Pmin = 101325*3

*******************************************************************
*
*  Node numbers of magnet inlet and outlet
*
         Nin = Nhx1 + 1
         Nout= Nin + Nmag-1

         print *, 'nin = ',nin,'  nout = ',nout

*******************************************************************
*
*   Initial conditions buffer tank
*
      CALL BUFFER(Vtank, Tpool, xtank, mtank, Ptank, utank)
       PRINT *,'OUT OF BUFFER'
       print *,'Tpool = ',Tpool,'  mtank = ', mtank

*******************************************************************
*
*   Initial conditions in loop

       CALL INIT(Pin, Tin, minit, q, Nin, Nout, Ntot,
     ;               AcMag, DhMag, Lmag, fmag,
     ;               AcHx1, DhHx1, Lhx1, fhx1, uaHx1,
     ;               AcHx2, DhHx2, Lhx2, fhx2, uaHx2,
     ;               Pmin, Tmax, Tpool, eta,
     ;               Cp, Rho, s, h, dPdR, dPdT, x, dxMag, dxhx1, dxhx2)

       PRINT *,'OUT OF INIT'
       print *,'Tpool = ',Tpool

*******************************************************************
*
*   Steady State Heat Load
*
       CALL QLOAD(x, uahx1, uahx2, dxhx1, dxhx2, Tpool, Nout,
     ;          Nin, Ntot, Qss, Qhx1, Qhx2)

       print *, 'Qss = ',Qss, ' qmag*Lmag = ',q*Lmag

**********************************************************************
*
*   Opening and Writing Headings to the Output Files
*
      open(unit = 4,file = 'T.out', status = 'unknown')
      open(unit = 5,file = 'P.out', status = 'unknown')
      open(unit = 6,file = 'm.out', status = 'unknown')
      open(unit = 7,file = 's.out', status = 'unknown')
      open(unit = 8,file = 'q.out', status = 'unknown')

         time = 0.0
         write(4,40) 'location',
     ;       (xLOCAT(Nin,Nout,dxHx1,dxHx2,dxMag,j), j=1,Ntot)
         write(4,41) time, (x(3*j-2), j=1,Ntot)
40       format(A8,250F8.3)
41       format(F8.3,250F8.5)

         write(5,50) 'location',
     ;       (xLOCAT(Nin,Nout,dxHx1,dxHx2,dxMag,j), j=1,Ntot)
         write(5,51) time, (x(3*j-1), j=1,Ntot)
50       format(A8,250F9.3)
51       format(F8.3,250F9.0)

         write(6,60) 'location',
     ;       (xLOCAT(Nin,Nout,dxHx1,dxHx2,dxMag,j), j=1,Ntot)
         write(6,61) time, (x(3*j), j=1,Ntot)
60       format(A8,250F11.3)
61       format(F8.3,250F11.6)

         write(7,70) 'location',
     ;       (xLOCAT(Nin,Nout,dxHx1,dxHx2,dxMag,j), j=1,Ntot)
         write(7,71) time, (x(3*j)*s(j), j=1,Ntot)
70       format(A8,250F10.3)
71       format(F8.3,250F10.4)
```

```
        write(8,81) 'qss - ', qss
        write(8,*)'  time    Qmag      Qhx1      Qhx2    Qpool     Wp',
     ;            '    Tpool'
        write(8,82) time, q*Lmag, Qhx1, Qhx2, Qss,
     ;                x(3*ntot)*(h(1)-h(Ntot)),Tpool
81      format(a8,1x,f8.5)
82      format(f8.3,1x,f8.2,1x,f8.5,1x,f8.5,1x,f8.5,1x,F8.5,1x,F8.6)


*********************************************************************
*
*   Initializing the time to the first time step and the cycle number
*   the first cycle
      nTime - 1
      time - dt
      nCycle - 1

*********************************************************************
*
*   Initializing the inlet mass flow to be the steady state mess
*   flow rate
*
      min - minit


*********************************************************************
*
*   Transient loop

      Do 100 While (nCycle.le.nCycTot+1)

*********************************************************************
*
*
*    Setting the heat load by checking to see where in the cycle you are
*
        write(*,90)' q - ', q

        if(nCycle.le.nCycTot) then
           if (nTime.le.nPulse) then
             q - qPulse
           else
             q - QNoPulse
           endif
        else
           q - qRecover
        endif

        write(*,90)' q - ', q

        write(*,90)' t - ', time
90      format(a5,F7.3)

*********************************************************************
*
*    Saving the current values of the x vector into the xp vector
*
         DO 200 i - 1, 3*nTot
            xp(i) - x(i)
200      continue

*********************************************************************
*
*   Calling subroutine UNSTEADY to evaluate the T, P and m at
*   each node for the current time step
*

250     continue

*       print *, ' dxMag - ',dxMag,' DhMag - ',DhMag,' AcMag - ',AcMag


         CALL UNSTEADY(x, xp, q, Nin, Nout, Ntot,
     ;              AcMag, DhMag, fmag, dxMag,
     ;              AcHx1, DhHx1, fhx1, uaHx1, dxHx1,
     ;              AcHx2, DhHx2, fhx2, uaHx2, dxHx2,
     ;              eta,
     ;              Cp, Rho, s, h, dPdR, dPdT, dt,
     ;              Pin, Tin, min, Tpool, Pmin, Tmax, Allo)



*********************************************************************
*
*    FINDING THE HEAT LOAD BEING ADDED TO THE POOL (BUFFER TANK)
*
       CALL QLOAD(x, uahx1, uahx2, dxhx1, dxhx2, Tpool, Nout, Nin,
     ;         Ntot, Qpool, Qhx1, Qhx2)
```

```
      PRINT *,'OUT OF QLOAD'
      print *,'Tpool = ',Tpool
*******************************************************************
*
*    FINDING THE NEW CONDITIONS IN THE POOL (BUFFER TANK)
*
      CALL POOL(Qss, Qpool, Vtank, mtank, utank, Tpool, Ptank,dt)
      PRINT *,'OUT OF POOL'
      print *, 'Qpool = ',Qpool, ' Qss = ',Qss,'  Tpool = ',Tpool

        write(4,41) time, (x(3*j-2), j=1,Ntot)
        write(5,51) time, (x(3*j-1), j=1,Ntot)
        write(6,61) time, (x(3*j), j=1,Ntot)
        write(7,71) time, (x(3*j)*s(j), j=1,Ntot)
        write(8,82) time, q*Lmag,Qhx1, Qhx2, Qpool,
     ;              x(3*ntot)*(h(1)-h(Ntot)),Tpool

*******************************************************************
*
*    Updating time and nTime
*
        if(nCycle.le.nCycTot) then
          if (nTime.eq.nPulse+nNoPulse) then
              nTime = 1
              nCycle = nCycle +1
              if(nCycle.eq.nCycTot+1) then
                dt = dtRecov
              endif
          else
              nTime = nTime + 1
          endif
        else
          if(nTime.eq.nRecover+1) then
              nCycle = nCycle + 1
          else
              nTime = nTime + 1
          endif
        endif

        time = time + dt

100   continue

      close(5)
      close(6)
      close(7)
      close(8)
********************************************************************
*
*   OUTPUT USED IF YOU WANT TO CHECK THE CONDITIONS AT EACH NODE
*   FOR A PARTICULAR TIME STEP - THIS IS USUALLY DISABLED AND
*   COMMENTED OUT USING THE ASTERISKS
*
*      OPEN(UNIT=20,FILE='MAG.OUT',STATUS='UNKNOWN')
*
*      WRITE(20,5)'   x       T      P       m   '
*      WRITE(20,5)'   m       K     atm     g/s  '
*5     FORMAT(A30)

*       DO 400 n = 1, Ntot
*          WRITE(20,10) (n-Nin)*Lmag/(Nmag-1), x(3*n-2),
*     ;                 x(3*n-1)/101325,x(3*n)*1000
*10        format(1x,f7.2,1x,f6.3,1x,f6.3,1x,f5.3)
*400    CONTINUE


       END


*******************************************************************
*
*   FUNCTION FOR DETERMINING DISTANCE FROM PUMP EXIT
*   (used for placing distances in output files)
*
      REAL FUNCTION xLOCAT(Nin,Nout,dxHx1,dxHx2,dxMag,i)
        double precision dxHx1, dxHx2, dxMAg

        if (i.LT.Nin) THEN
              xLOCAT = (i-1)*dxHx1
        elseif (i.GT.Nout) then
              xLOCAT = (Nin-2)*dxHx1 + (Nout-Nin)*dxMag +
     ;                 (i-Nout-1)*dxHx2
        else
              xLOCAT = (Nin-2)*dxHx1 + (i-Nin)*dxMag
        endif
        return
        end
```

E4: Subroutine Buffer

```
      SUBROUTINE BUFFER(Vtank, Tpool, xtank, mtank, Ptank, utank)

**********************************************************************
*   BUFFER3.FOR
*
*   Subroutine for evaluating the initial conditions of the
*   buffer tank.
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   2-16-97
*
*   last modified 5-19-97
*********************************************************************
*
*   VARIABLES USED
*
*
*     mtank  - Total helium mass in the buffer tank (kg)
*     Ptank  - Pressure in buffer tank (Pa)
*     rho    - Overall density of helium in buffer tank
*     Tpool  - Temperature of buffer tank/pool (K)
*     utank  - Specific internal energy of buffer tank (J/kg-K)
*     Vtank  - Volume of buffer tank (m3)
*     xtank  - Quality of buffer tank
*
********************************************************************
*
*   My Variable assignments
        real mtank
        real Ptank
        double precision Tpool
        real utank
        real Vtank
        real xtank
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
*********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                 NPRCIS=2
*
*        Units  (SI units)
                 NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
                JOUT=11000


      PRINT *,'IN BUFFER'
********************************************************************
*
*   Finding the condition of the tank
*
        JIN1 = 2
        VALU1 = Tpool
        JIN2 = 9
        VALU 2= xtank

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           Ptank = PROP(1,0)
           rho   = PROP(3,0)
           utank = PROP(11,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF
```

```
***********************************************************************
*
*   Mass in the tank
*
       mtank = rho*Vtank

       print *,'mtank = ',mtank,' ptank = ',ptank,' rho = ',rho


       RETURN
       END
```

E5: Subroutine INIT

```
      SUBROUTINE INIT(Pin, Tin, m, q, Nin, Nout, Ntot,
     ;                AcMag, DhMag, Lmag, fmag,
     ;                AcHx1, DhHx1, Lhx1, fhx1, uaHx1,
     ;                AcHx2, DhHx2, Lhx2, fhx2, uaHx2,
     ;                Pmin, Tmax, Tpool, eta,
     ;                Cp, Rho, s, h, dPdR, dPdT, x, dxMag, dxhx1, dxhx2)

**********************************************************************
*    INIT7.FOR
*
*    Subroutine for evaluating the temperature and pressure profile
*    in a recirculator loop under steady state conditions.
*    (Used to find initial values before a transient load is applied)
*    This version has 2 pool-cooled heat exchangers and a cable in
*    conduit conductor between them.
*
*    This version allows for negative mass flows.
*
*    Note the x vector is used to hold the values of T, P, and m for
*    the length of the conductor.  Where T's, P's and m's for a
*    particular node n in the conductor are found by:
*                   T(n) = x(3*n-2)
*                   P(n) = x(3*n-1)
*                   m(n) = x(3*n)
*
*    This program must be linked with HEPROP.OBJ a property routine
*    program by CRYODATA
*
*
*    Brian Bowers
*    original version started on
*    10-31-96
*
*    last modified 5-19-97
**********************************************************************
*
*    VARIABLES USED
*
*      AcHx1    = Cross sectional area of heat exchanger 1 flow passage (m2)
*      AcHx2    = Cross sectional area of heat exchanger 2 flow passage (m2)
*      AcMag    = Cross sectional area of flow passage (m2)
*      Cp(n)    = Specific heat at node n (J/kg-K)
*      DhHx1    = Hydraulic diameter of heat exchanger 1 (m)
*      DhHx2    = Hydraulic diameter of heat exchanger 2 (m)
*      DhMag    = Hydraulic diameter of flow passage (m)
*      dP       = Total pressure drop around loop (atm)
*      dPdR(n)  = dP/dRho at constant temperature at node n (Pa-m3/kg)
*      dPdT(n)  = dP/dT at constant density at node n (Pa/K)
*      dPdx     = dP/dx - Pressure drop due to friction at current
*                 location (Pa/m)(m)
*      dTdx     = dT/dx Temperature change per length at current location (K/m)
*      dxHx1    = Distance between nodes in heat exchanger 1
*      dxHx2    = Distance between nodes in heat exchanger 2
*      dxMag    = Distance between nodes in magnet
*      fmag     = Friction factor in magnet
*      fhx1     = Friction factor in heat exchanger 1
*      fhx2     = Friction factor in heat exchanger 2
*      Lhx1     = Length of heat exchanger 1 flow passage (m)
*      Lhx2     = Length of heat exchanger 2  flow passage (m
*      Lmag     = Length of magnet flow passage (m))
*      h(n)     = Specific enthalpy of helium at node n (J/kg)
*      m        = Mass flow rate through conductor (kg/s)
*      mpump    = Value of m that corresponds to current dP (kg/s)
*      n        = Current node being evaluated
*      Nin      = Value of node number n for the conductor inlet
*      Nout     = Value of node number n for the conductor outlet
*      Ntot     = Total number of nodes in system include those that are
*                 not in the conductor  (ie 1/3 of the size of x )
*      P        = Pressure at current node(Pa)
*      Pin      = Inlet pressure (Pa)
*      Pmin     = Minimum allowable pressure in conductor (Pa)
*      q        = Heat flow rate per unit length (W/m)
*      Rho(n)   = Density at node n (kg/m3)
*      s(n)     = Specific entropy at node n (J/kg-K)
*      T        = Temperature at current node (K)
*      Tin      = Inlet temperature (K)
*      Tmax     = Maximum allowable temperature at any location(K)
*      uaHx1    = Overall heat transfer coefficient PER UNIT LENGTH of HX1 (W/m-K)
*      uaHx2    = Overall heat transfer coefficient PER UNIT LENGTH of HX1 (W/m-K)
*      x(n)     = Vector containing T, P, and m data for all nodes


**********************************************************************
*
*   My Variable assignments
        real AcHx1, AcHx2, AcMag
        DOUBLE PRECISION Cp(Ntot)
        real DhHx1, DhHx2, DhMag
```

```
      double precision dP
      DOUBLE PRECISION dPdR(Ntot), dPdT(Ntot)
      DOUBLE PRECISION dPdx, dTdx
      DOUBLE PRECISION dxhx1, dxhx2, dxMag
      real fhx1, fhx2, fmag
      real Lhx1, Lhx2, Lmag
      REAL h(Ntot)
      double precision m, mpump
      Integer n, Nin, Nout, Ntot
      DOUBLE PRECISION P, Pin, Pmin
      real q
      DOUBLE PRECISION Rho(Ntot), s(Ntot)
      double precision T, T1, Tin
      double precision Tmax, Tpool
      real uaHx1, uaHx2
      DOUBLE PRECISION x(Ntot*3)
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
*********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
              NPRCIS=2
*
*        Units  (SI units)
              NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
             JOUT=11000
*
**********************************************************************

      print *, 'in INIT'

**********************************************************************
*   Setting values at the inlet
*
      n = 1
      T = Tin
      P = Pin
      m = m
      x(3*n-2) = T
      x(3*n-1) = P

**********************************************************************
*   Since the flow is steady, all m's are the same

      DO 100 I = 3*1, 3*Ntot, 3
         x(I) = m
100   CONTINUE

**********************************************************************
*   Length Steps
      dxMag = Lmag/(Nout-Nin)
      PRINT *,' dxMag = ',dxMag

      if (Nin.GT.1)     dxhx1 = Lhx1/(Nin-2)
      print*, 'dxhx1 = ',dxhx1

      if (Nout.NE.Ntot) dxhx2 = Lhx2/(Ntot-Nout-1)
      print*,'dxhx2 = ',dxhx2

**********************************************************************
*   Loop to find conditions at all locations in the loop
*
     DO 200 n = 1, Ntot


**********************************************************************
*
*       Finding the conditions at the current location
        JIN1=1
        VALU1=P
        JIN2=2
        VALU2=T

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           Rho(n) = PROP(3,0)
           s(n)   = PROP(8,0)
           h(n)   = PROP(9,0)
           Cp(n)  = PROP(14,0)
           dPdR(n) = PROP(22,0)
           dPdT(n) = PROP(23,0)

           IF (IDID .EQ. 3) THEN
```

```
*                This code not used in this case
          ENDIF
        ELSE
          CALL ERRMSG (MESSAG, IDID)
          WRITE (*, '(1X,A60)') MESSAG
          STOP
        ENDIF

************************************************************************
*
*   Slopes of temperature and pressure change per length



      IF (n.LT.Nin-1) THEN
        dPdx =  -fhx1* m* abs(m) /(2* Rho(n) * AcHx1**2 * DhHx1)
        dTdx = uaHx1*(Tpool-T)/(m*Cp(n))
     ;     - ( 1/Rho(n) - T*dPdT(n)/(Rho(n)**2 * dPdR(n)) )*dPdx/Cp(n)
      ELSEIF (n.EQ.Nin-1) THEN
         dPdx = 0.0
         dTdx = 0.0
      ELSEIF  (n.LT.Nout) THEN
        dPdx =  -fmag* m* abs(m) /(2* Rho(n) * AcMag**2 * DhMag)
        dTdx = q/(m*Cp(n))
     ;     - ( 1/Rho(n) - T*dPdT(n)/(Rho(n)**2 * dPdR(n)) )*dPdx/Cp(n)
      ELSEIF (n.EQ.Nout) THEN
         dPdx = 0.0
         dTdx = 0.0
      ELSE
        dPdx =  -fhx2* m* abs(m) /(2* Rho(n) * AcHx2**2 * DhHx2)
        dTdx = uaHx2*(Tpool-T)/(m*Cp(n))
     ;     - ( 1/Rho(n) - T*dPdT(n)/(Rho(n)**2 * dPdR(n)) )*dPdx/Cp(n)
      ENDIF

***********************************************************************
*
*   Finding the conditions at the NEXT location
*    note that this is only done if n is less than Ntot since if
*    n = Ntot there is not another node to calculate T and P at.
*    (but the whole loop was needed to calculate the properties of
*     the last node--which is done at the top of the loop)

        IF (n.LT.NtoT) THEN
          IF (n.LT.Nin-1) THEN
             T = T + dTdx*dxHx1
             P = P + dPdx*dxHx1
          ELSEIF (n.EQ.Nin) THEN
             T = T + dTdx*dxHx1
             P = P + dPdx*dxHx1
          ELSEIF (n.LT.Nout) THEN
             T = T + dTdx*dxMag
             P = P + dPdx*dxMag
          ELSEIF (n.EQ.Nout) THEN
             T = T + dTdx*dxMag
             P = P + dPdx*dxMag
          ELSE
             T = T + dTdx*dxHx2
             P = P + dPdx*dxHx2
          ENDIF
          x(3*(n+1)-2) = T
          x(3*(n+1)-1) = P
        ENDIF


************************************************************************
*
*   CHECK TO MAKE SURE THAT THE TEMPERATURE DOES NOT EXCEED THE
*   MAXIMUM ALLOWABLE TEMPERATE AND THAT THE PRESSURE IS ABOVE MINIMUM
*
        IF (T.GT.Tmax) THEN
          print *, ' Temp greater than max allowed temp before pulse'
*          stop
        ENDIF

        IF (P.LT.Pmin) THEN
          print *, ' Pressure below minimum allowable before pulse'
*          stop
        ENDIF

200    Continue

************************************************************************
*
*   CALLING THE SUBROUTINE STEADY TO MAKE VALUES CONSISTENT WITH A
*   BACKWARD EULER METHOD

      CALL STEADY(x, q, m, Nin, Nout, Ntot,
     ;                AcMag, DhMag, fmag, dxMag,
     ;                AcHx1, DhHx1, fhx1, uaHx1, dxHx1,
     ;                AcHx2, DhHx2, fhx2, uaHx2, dxHx2,
```

```
     ;                  Cp, Rho, s, h, dPdR, dPdT,
     ;                  Pin, Tin, Tpool)

*************************************************************************
*
*   USING THE PUMP CHARACTERISTIC EQN TO FIND THE MASS FLOW RATE THAT THE
*   PUMP SHOULD HAVE TO GIVE THE CURRENT TOTAL PRESSURE DROP AROUND THE LOOP
*

*   Converting the pressure drop around the loop to atmospheres

      dP = ( x(2)-x(3*Ntot-1) ) / 101325
*      PRINT *,'dp = ',dp

*   Calling the subroutine PUMP to find the mass flow rate that the pump
*   would need to give the same pressure drop as calculated.
      CALL PUMP(dP,mPump)

**********************************************************************
*
*    CHECKING FOR CONSISTENCY WITH THE PUMP ISENTROPIC EFFICIENCY
*
*    ISENTROPIC EXIT ENTHALPY OF PUMP
        JIN1=1
        VALU1=x(2)
        JIN2=5
        VALU2=s(Ntot)

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           hs   = PROP(9,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF

        h1= h(Ntot) + ( hs - h(Ntot) )/Eta
        print *,'h1 = ',h1

*    TEMPERATURE AT EXIT OF PUMP
        JIN1=1
        VALU1=x(2)
        JIN2=6
        VALU2=h1

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
        IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
           T1   = PROP(2,0)
           IF (IDID .EQ. 3) THEN
*                 This code not used in this case
           ENDIF
        ELSE
           CALL ERRMSG (MESSAG, IDID)
           WRITE (*, '(1X,A60)') MESSAG
           STOP
        ENDIF


*     COMPARING THE CALCULATED TEMPERATURE AT NODE 1 (pump exit) WITH
*     THE CURRENT VALUE IF THEY DISAGREE, THEN ADJUST Tin USING A
*     WEIGHTED AVERAGE OF THE TWO VALUES.
*     ALSO COMPARE THE PRESSURE DROP AND ADJUST m AS NEEDED TO ZERO IN
*     ON THE CORRECT OPERATING POINT.


        if (abs( (T1-Tin) /Tin ) .gt. 1e-7  .OR.
     ;          abs( (mpump-m)/m ) .GT. 1E-6) THEN
              print *,'m = ',m,'   Tin = ',Tin
              Tin = Tin*0.1 + T1*0.9
              m = (mpump + m)/2
              print *,'m = ',m,'   Tin = ',Tin
              goto 200
         endif


      END
```

E.6: Subroutine STEADY

```
      SUBROUTINE STEADY(x, q, m, Nin, Nout, Ntot,
     ;                 AcMag, DhMag, fmag, dxMag,
     ;                 AcHx1, DhHx1, fhx1, uaHx1, dxHx1,
     ;                 AcHx2, DhHx2, fhx2, uaHx2, dxHx2,
     ;                 Cp, Rho, s, h, dPdR, dPdT,
     ;                 Pin, Tin, Tpool)

***********************************************************************
*   STEADY5.FOR  - sparse matrix version
*
*   Subroutine for evaluating the temperature and pressure profile
*   in a recirculator loop under STEADY STATE conditions.  This routine
*   uses the T, P and m values that are found in subroutine INIT as a
*   starting point.  It then iterates to make sure that the values are
*   all consistent with a BACKWARD EULER estimation for spatial
*   derivatives.
*   (INIT uses a FORWARD EULER technique that will give very similar
*    results, but will be slightly different.)
*
*   Note the x vector is used to hold the values of T, P, and m for
*   the length of the conductor.  Where T's, P's and m's for a
*   particular node n in the conductor are found by:
*                  T(n) = x(3*n-2)
*                  P(n) = x(3*n-1)
*                  m(n) = x(3*n)
*
*   Note that the program originally used an LU decompostion method
*   of solution that put the coefficients into an A matrix.  This
*   method was slow and memory intensive so a sparse matrix method
*   was used.  However, the sparse matrix solution compacts the
*   A matrix into two vector: SA to store the diagonal and non-zero
*   elements, and IJA to store the indices.  Since the locations in
*   these vectors are very difficult to interpret, the original A matrix
*   assignments were left in, but commented out, to show where in the
*   A matrix the values are being assigned.
*
*   The routines needed to solve the sparse matrix are all from
*   Numerical Recipes: the Art of Scientific Computing, version 2.
*   The routines that were used are:
*        linbcg
*        atimes
*        asolve
*        dsprsax
*        dsprstx
*        snrm
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*
*   Brian Bowers
*   original version started on
*   1-30-97
*
*   last modified 5-19-97
**********************************************************************
*
*   VARIABLES USED
*
*     A        = Matrix containing derivatives - used for Newton's method
*     AcHx1    = Cross sectional area of heat exchanger 1 flow
*                passage (m2)
*     AcHx2    = Cross sectional area of heat exchanger 2 flow
*                passage (m2)
*     AcMag    = Cross sectional area of flow passage (m2)
*     b        = Vector used for solving using Newton's method
*     Cp(n)    = Specific heat at node n (J/kg-K)
*     D        = Flag used by program LUDCMP and LUBKSB
*     DhHx1    = Hydraulic diameter of heat exchanger 1 (m)
*     DhHx2    = Hydraulic diameter of heat exchanger 2 (m)
*     DhMag    = Hydraulic diameter of flow passage (m)
*     dhdP(n) = dh/dP at constant temperature at node n (m3/kg)
*     dPdR(n) = dP/dRho at constant temperature at node n (Pa-m3/kg)
*     dPdT(n) = dP/dT at constant density at node n (Pa/K)
*     dRdT(n) = dRho/dT at constant pressure at node n (kg /m3-K)
*     dRdP(n) = dRho/dP at constant temperature at node n ( kg/m3-Pa)
*     dudT(n) = du/dT at constant pressure at node n (J/kg-K)
*     dudP(n) = du/dP at constant temperature at node n (J/kg-Pa)
*                or (m3/kg)
*     dxHx1    = Distance between nodes in heat exchanger 1
*     dxHx2    = Distance between nodes in heat exchanger 2
*     dxMag    = Distance between nodes in magnet (m)
*     Em       = Total error in m equations
*     EP       = Total error in P equations
*     ET       = Total error in T equations
*     err      = Error value returned from sparse matrix solver (linbcg)
*     fhx1     = Friction factor in heat exchanger 1
*     fhx2     = Friction factor in heat exchanger 2
```

```
*     fmag     - Friction factor in magnet
*     ija      - Vector used to store column information for sparse
*                version of the A matrix (called SA)
*     iter     - Number of iterations used by linbcg
*     itmax    - Maximum allowable number of iterations in linbcg
*     itol     - Type of convergence check being used in linbcg
*     indx     - Vector used by program LUDCMP and LUBKSB
*     LenSA    - Length of SA and ija (vectors used to store sparse matrix)
*     h(n)     - Specific enthalpy of helium at node n (J/kg)
*     m        - Mass flow rate at all locations (kg/s)
*     Nin      - Value of node number n for the conductor inlet
*     Nout     - Value of node number n for the conductor outlet
*     Ntot     - Total number of nodes in system include those that are
*                not in the conductor  (ie 1/3 of the size of x )
*     Pin      - Inlet pressure (Pa)
*     Pmin     - Minimum allowable pressure in conductor (Pa)
*     q        - Heat flow rate per unit length (W/m)
*     Rho(n)   - Density at node n (kg/m3)
*     s(n)     - Specific entropy at node n (J/kg-K)
*     SA       - Sparse version of A matrix (stored in a vector)
*     Tin      - Inlet temperature (K)
*     Tmax     - Maximum allowable temperature at any location(K)
*     tol      - Tolerance variable required for sparse matrix solver
*     Tpool    - Temperature of pool (K)
*     uaHx1    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     uaHx2    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     x(n)     - Vector containing T, P, and m data for all nodes
*     xt(n)    - Values of x at last iteration
*
*******************************************************************
*
*   My Variable assignments
*        DOUBLE PRECISION A[allocatable](:,:)
        real AcHx1, AcHx2, AcMag
        DOUBLE PRECISION b[allocatable](:)
        DOUBLE PRECISION Cp(Ntot)
*        REAL D
        real DhHx1, DhHx2, DhMag
        DOUBLE PRECISION dPdR(Ntot), dPdT(Ntot)
        DOUBLE PRECISION dRdT[allocatable](:)
        DOUBLE PRECISION dRdP[allocatable](:)
        DOUBLE PRECISION dhdP[allocatable](:)
        DOUBLE PRECISION dudT[allocatable](:)
        DOUBLE PRECISION dudP[allocatable](:)
        DOUBLE PRECISION dxHx1, dxHx2, dxMag
        real Em, ET, EP
        real fhx1, fhx2, fmag
        real h(Ntot)
        integer ija[allocatable](:)
*        INTEGER indx[allocatable](:)
        DOUBLE PRECISION m
        Integer Nin, Nout, Ntot
        DOUBLE PRECISION Pin
        real q
        DOUBLE PRECISION Rho(Ntot), s(Ntot)
        DOUBLE PRECISION SA[allocatable](:)
        double precision Tin
        double precision Tpool
        real uaHx1, uaHx2
        DOUBLE PRECISION x(Ntot*3)
        DOUBLE PRECISION xt[allocatable](:)

*
*   Variable assignments needed for sparse matrix solver linbcg
*
        integer itmax, itol, iter
        double precision tol, err

*
*   Variable assignments needed for HEPROP
*
       DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
       CHARACTER MESSAG*60
       INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=2
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
               JOUT=11000
*
*********************************************************************

       print *, 'in STEADY'
```

```
*********************************************************************
*
*   Length of SA and ija (vectors used to store sparse matrix)
*   (there are 3*Ntot diagonal elements, 1 blank space,  4 off diagonal
*    elements for each of the Ntot-3 "elements", and 3 off diagonal
*    elements from continuity at both the HX1-Mag and Mag-HX2 interfaces.)
*
      LenSA = 3*Ntot + 1 + 4*(Ntot-3) + 3 * 2


*********************************************************************
*
*   Allocating memory to the variables not being
*   passed back to main routine
*

*          allocate(A(3*Ntot,3*nTot), b(3*Ntot), SA(LenSA), ija(LenSA),
         allocate(b(3*Ntot), SA(LenSA), ija(LenSA),
     ;      dRdT(Ntot), dRdP(Ntot), dhdP(Ntot), dudT(Ntot), dudP(Ntot),
     ;      xt(3*Ntot))
*     ;      indx(3*Ntot),xt(3*Ntot))



*********************************************************************
*
*  Specifying ija vector for sparse matrix storage
*
*     PATTERN IS:
*
*          FROM HX1:
*               first 3 rows  :  no off diagonals
*               Next 3*(Nin-2) rows have pattern [rows 4 to 3*(Nin-1) ]
*                   row 3*n-2   :  off diagonals at columns
*                                    3*n-5
*                                    3*n-4
*                                    3*n-1
*                   row 3*n-1 :  off diagonal at column 3*n-4
*
*          FROM MAGNET:
*               row 3*Nin-2 : off diagonal at column 3*(Nin-1)-2
*               row 3*Nin-1 : off diagonal at column 3*(Nin-1)-1
*               row 3*Nin   : off diagonal at column 3*(Nin-1)
*               Next 3*(Nout-Nin) rows  [ rows 3*Nin+1 to 3*Nout ]
*                   row 3*n-2 :  off diagonals at columns
*                                    3*n-5
*                                    3*n-4
*                                    3*n-1
*                   row 3*n-1 :  off diagonal at column 3*n-4
*
*          FROM HX2:
*               row 3*Nout+1 : off diagonal at column 3*Nout-2
*               row 3*Nout+2 : off diagonal at column 3*Nout-1
*               row 3*Nout+3 : off diagonal at column 3*Nout
*               Next 3*(Ntot-Nout-1) rows  [rows 3*Nout+4 to 3*Ntot]
*                   row 3*n-2 :  off diagonals at columns
*                                    3*n-5
*                                    3*n-4
*                                    3*n-1
*                   row 3*n-1 :  off diagonal at column 3*n-4
*
*    Setting up the first 3*Ntot+1 elements of ija   (from diagonal components)

          ija(1) = 3*Ntot+2
          ija(2) = 3*Ntot+2
          ija(3) = 3*Ntot+2
          ija(4) = 3*Ntot+2

          do i = 5, 5 + 3*(Nin-3), 3
            ija(i)   = ija(i-1) + 3
            ija(i+1) = ija(i) + 1
            ija(i+2) = ija(i+1)
          enddo

          ija(3*Nin-1) = ija(3*Nin-2)+1
          ija(3*Nin)   = ija(3*Nin-1)+1
          ija(3*Nin+1) = ija(3*Nin)  +1

          do i = 3*Nin+2, 3*Nin+2 + 3*(Nout-Nin-1), 3
            ija(i)   = ija(i-1) + 3
            ija(i+1) = ija(i) + 1
            ija(i+2) = ija(i+1)
          enddo

          ija(3*Nout+2) = ija(3*Nout+1)+1
          ija(3*Nout+3) = ija(3*Nout+2)+1
          ija(3*Nout+4) = ija(3*Nout+3)+1

          do i = 3*Nout+5, 3*Nout+5 + 3*(Ntot-Nout-2), 3
            ija(i)   = ija(i-1) + 3
            ija(i+1) = ija(i) + 1
            ija(i+2) = ija(i+1)
```

```
          enddo
*      setting up the off diagonal components of ija

          ija(3*Ntot+2) = 1
          ija(3*Ntot+3) = 2
          ija(3*Ntot+4) = 5
          ija(3*Ntot+5) = 2

          do i = 3*Ntot+6, 3*Ntot+6 + 4*(Nin-3) - 1
            ija(i) = ija(i-4) + 3
          enddo

          ija(3*Ntot+6 + 4*(Nin-3)  ) = 3*Nin-5
          ija(3*Ntot+6 + 4*(Nin-3)+1) = 3*Nin-4
          ija(3*Ntot+6 + 4*(Nin-3)+2) = 3*Nin-3

          ija(3*Ntot+6 + 4*(Nin-3)+3) = 3*Nin-2
          ija(3*Ntot+6 + 4*(Nin-3)+4) = 3*Nin-1
          ija(3*Ntot+6 + 4*(Nin-3)+5) = 3*Nin+2
          ija(3*Ntot+6 + 4*(Nin-3)+6) = 3*Nin-1

          do i = 3*Ntot+6 + 4*(Nin-3)+7, 3*Ntot+4*(Nout-3)+8
            ija(i) = ija(i-4) + 3
          enddo

          ija(3*Ntot+4*(Nout-3)+9)  = 3*Nout-2
          ija(3*Ntot+4*(Nout-3)+10) = 3*Nout-1
          ija(3*Ntot+4*(Nout-3)+11) = 3*Nout

          ija(3*Ntot+4*(Nout-3)+12) = 3*Nout+1
          ija(3*Ntot+4*(Nout-3)+13) = 3*Nout+2
          ija(3*Ntot+4*(Nout-3)+14) = 3*Nout+5
          ija(3*Ntot+4*(Nout-3)+15) = 3*Nout+2

          do i = 3*Ntot+4*(Nout-3)+16, LenSA
              ija(i) = ija(i-4) + 3
          enddo

*          print *,' ija(1) = ',ija(1), ' 3*Ntot = ',3*Ntot
*          print *,'  LenSA = ',LenSA
*           k=0
*           do i = 1, LenSA
*              k = k+1
*              print *,'i = ',i, '  ija = ',ija(i)
*              if (k.eq.20) then
*                  pause ' <RETURN> for more'
*                  k=0
*              endif
*           enddo
*           pause ' <RETURN> to continue '


************************************************************************
*
*   DERIVATIVES
*

      DO 30 n = 1, Ntot
        dRdT(n) = -dPdT(n) / dPdR(n)
        dRdP(n) = 1/dPdR(n)
        dhdP(n) = 1/Rho(n) - x(3*n-2)*dPdT(n) /( dPdR(n)*(Rho(n))**2 )
        dudT(n) = Cp(n) - x(3*n-1)*dPdT(n) /( dPdR(n)*(Rho(n))**2 )
        dudP(n) = ( x(3*n-1) - x(3*n-2)*dPdT(n) ) /
     ;                           ( dPdR(n)*(Rho(n))**2 )
30    CONTINUE

************************************************************************
*
*  CLEARING THE A MATRIX AND THE b VECTOR TO START THE SOLUTION
*   (A matrix was commented out when the sparse matrix solution was added)
*
50    CONTINUE
      DO 200 i = 1, 3*Ntot
         b(i) = 0
*          DO 300 j = 1, 3*Ntot
*            A(i,j) = 0
*300       CONTINUE
200   CONTINUE

************************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE FIRST HEAT EXCHANGER
*  (note the A matrix assignments are commented out since the sparse
*   matrix method of solution is being used.  But these original A matrix
*   statements are left in for it to be easier to see where the values
*   are being assigned to since the sparse matrix vector is difficult
*   to understand.)

*
```

```
*   SETTING UP THE FIRST THREE EQNS IN THE MATRIX
*   NODE 1 IS SAME AS PUMP
*   ASSUMING P1 = Pin and T1 = Tin and m1 = msteady


*        A(1,1)  = 1
         SA(1)   = 1
        b(1) = Tin

*        A(2,2) = 1
         SA(2) = 1
        b(2) = Pin

*        A(3,3) = 1
         SA(3) = 1
        b(3) = m

*  starting a counter for filling in the off diagonal elements at the
*  end of the SA matrix:
      k = 3*Ntot + 2


*  REMAINING HX1 EQUATIONS

       DO 400 n = 2, Nin-1

*    First Law equations
*        A(3*n-2,3*n-5)    = -m*Cp(n) /dxHx1
               SA(k)      = -m*Cp(n) /dxHx1
               k = k+1
*        A(3*n-2,3*n-4)    = -m*dhdP(n)/dxHx1
               SA(k)      = -m*dhdP(n)/dxHx1
               k = k+1
*        A(3*n-2,3*n-2)    = m*Cp(n)/dxHx1 + uaHx1
               SA(3*n-2) = m*Cp(n)/dxHx1 + uaHx1
*        A(3*n-2,3*n-1)    = m*dhdP(n)/dxHx1
               SA(k)      = m*dhdP(n)/dxHx1
               k = k+1

         b(3*n-2)           = uaHx1*Tpool

*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4)    = -1/dxHx1
              SA(k)       = -1/dxHx1
              k = k+1
*        A(3*n-1,3*n-1)    =  1/dxHx1
              SA(3*n-1)   =  1/dxHx1

         b(3*n-1)           = -fHx1*m**2/(2*Rho(n)*DhHx1*AcHx1**2)

*    Mass Conservation equations
*        A(3*n,3*n)         = 1
              SA(3*n)      = 1
         b(3*n)             = m

400   ENDDO


************************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE CICC MAGNET CABLE
*

*   ASSUMING CONTINUITY OF m, P and T BETWEEN THE HX1 AND THE MAGNET INLET
*   THE FIRST 3 EQUATION FOR THE MAGNET ARE THEN
*
*   temperature continuity
*      A(3*Nin-2,3*(Nin-1)-2)   = -1
            SA(k)             = -1
            k = k+1
*      A(3*Nin-2,3*Nin-2)       =  1
            SA(3*Nin-2)       =  1
*   pressure continuity
*      A(3*Nin-1,3*(Nin-1)-1)   = -1
            SA(k)             = -1
            k = k+1
*      A(3*Nin-1,3*Nin-1)       =  1
            SA(3*Nin-1)       =  1

*   mass continuity
*      A(3*Nin,3*(Nin-1))       = -1
            SA(k)             = -1
            k = k+1
*      A(3*Nin,3*Nin)           =  1
            SA(3*Nin)         =  1


*   REMAINING MAGNET EQUATIONS
       DO 600 n = Nin+1, Nout

*    First Law equations
*        A(3*n-2,3*n-5)    = -m*Cp(n) /dxMag
            SA(k)         = -m*Cp(n) /dxMag
```

```
          k = k+1
*        A(3*n-2,3*n-4)    = -m*dhdP(n)/dxMag
            SA(k)          = -m*dhdP(n)/dxMag
            k = k+1
*        A(3*n-2,3*n-2)    = m*Cp(n)/dxMag
            SA(3*n-2)      = m*Cp(n)/dxMag
*        A(3*n-2,3*n-1)    = m*dhdP(n)/dxMag
            SA(k)          = m*dhdP(n)/dxMag
            k = k+1

        b(3*n-2)           =  q

*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4) = -1/dxMag
            SA(k)       = -1/dxMag
            k = k+1
*        A(3*n-1,3*n-1) =  1/dxMag
            SA(3*n-1)   =  1/dxMag

        b(3*n-1)        = -fmag*m**2 / (2*Rho(n)*DhMag*AcMag**2)

*    Mass Conservation equations (all nodes have same mass flow)
*        A(3*n,3*n)   = 1
            SA(3*n)   = 1
        b(3*n)        = m

600     ENDDO

************************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE SECOND HEAT EXCHANGER
*
*

*   ASSUMING CONTINUITY OF m, P and T BETWEEN THE MAGNET EXIT AND HX2 INLET
*   THE FIRST 3 EQUATION FOR HX2 ARE THEN
*
*   temperature continuity
*      A(3*Nout+1,3*Nout-2)      = -1
            SA(k)               = -1
            k = k+1
*      A(3*Nout+1,3*(Nout+1)-2) =  1
            SA(3*Nout+1)        =  1

*   pressure continuity
*      A(3*Nout+2,3*Nout-1)      = -1
            SA(k)               = -1
            k = k+1
*      A(3*Nout+2,3*(Nout+1)-1) =  1
            SA(3*Nout+2)        =  1

*   mass continuity
*      A(3*Nout+3,3*Nout)        = -1
            SA(k)               = -1
            k = k+1
*      A(3*Nout+3,3*(Nout+1))    =  1
            SA(3*Nout+3)        =  1


*  REMAINING HX2 EQUATIONS

      DO 700 n = Nout+2, Ntot

*    First Law equations
*        A(3*n-2,3*n-5)    = -m*Cp(n) /dxHx2
            SA(k)          = -m*Cp(n) /dxHx2
            k = k+1
*        A(3*n-2,3*n-4)    = -m*dhdP(n)/dxHx2
            SA(k)          = -m*dhdP(n)/dxHx2
            k = k+1
*        A(3*n-2,3*n-2)    = m*Cp(n)/dxHx2 + uaHx2
            SA(3*n-2)      = m*Cp(n)/dxHx2 + uaHx2

*        A(3*n-2,3*n-1)    = m*dhdP(n)/dxHx2
            SA(k)          = m*dhdP(n)/dxHx2
            k = k+1

        b(3*n-2)           = uaHx2*Tpool
*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4) = -1/dxHx2
            SA(k)       = -1/dxHx2
            k = k+1
*        A(3*n-1,3*n-1) =  1/dxHx2
            SA(3*n-1)   =  1/dxHx2
        b(3*n-1)        = -fHx2*m**2/(2*Rho(n)*DhHx2*AcHx2**2)

*    Mass Conservation equations
*        A(3*n,3*n)   = 1
            SA(3*n)   = 1

        b(3*n)        = m
```

```
700   ENDDO

******************************************************************
*
*  Before Solving, save the previous values of the x vector in xt
*
        DO 800 i = 1, 3*Ntot
             xt(i) = x(i)
800     CONTINUE

******************************************************************
*
*  Solving for the new x vector - the sparse matrix solver LINBCG
*  from Numerical Recipes is used.  The LU decomposition matrix solver
*  (LUDCMP and LUBKSB also from Numerical Recipes) that was originally
*  used is shown, but is commented out.
* tolerance and maximum number of iteration needed for LINBCG
      itol = 1
      tol  = 1e-7
      itmax = 200

       call linbcg(SA,ija,3*Ntot,b,x,itol,tol,itmax,iter,err)

*       CALL LUDCMP(A,3*Ntot,3*Ntot,INDX,D)
*       CALL LUBKSB(A,3*Ntot,3*Ntot,INDX,b)


*       DO 750 n = 1, Ntot
*        write(*,*) ' n = ',n
*        write(*,11)' T(n) =    ',x(3*n-2), ' P(n) =   ',x(3*n-1),
*     ;     ' m(n) =    ', b(3*n)
*        write(*,11)' Tt(n) =   ',xt(3*n-2),' Pt(n) =   ',xt(3*n-1),
*     ;     ' mt(n) =   ', xt(3*n)
*        read(*,*)
*11      format(A10,f9.3,a9,f10.0,a10,f7.5)
*750     CONTINUE

******************************************************************
*
*  Saving the previous values of the x vector in xt and
*  Assigning the new values to the x vector
*  (this section is currently commented out- it is used if the solution
*  is unstable and a relaxation method is needed. The new values
*  are averaged with the old to let the solution converge slower)
*
*        DO 800 i = 1, 3*Ntot
*             xt(i) = x(i)
*             x(i) = (b(i)*.5 + x(i)*.5)
*800     CONTINUE


************************************************************************
*
*  CHECKING THE ERROR
        Em=0
        EP = 0
        ET = 0

        DO 900 n = 1, Ntot
          Em = Em + abs( x(3*n) - xt(3*n) )

          EP = EP + abs( x(3*n-1) - xt(3*n-1) )

          ET = ET + abs( x(3*n-2) - xt(3*n-2) )

900     CONTINUE

************************************************************************
*
*  IF ERROR IS GOOD, THEN RETURN, OTHERWISE GET THE PROPERTIES AT
*  ALL NODES USING THE NEW X VECTOR

        IF (Em.GT.0.0000001*Ntot. or. EP.GT.Ntot/10 .
     ;                     or .ET.GT.0.0000005*Ntot) THEN
*          print *,'em = ',em,' ep = ',eP,' eT = ',eT
          DO 1000 n = 1, Ntot

*          Finding the conditions at the current location
            JIN1=1
            VALU1=x(3*n-1)
            JIN2=2
            VALU2=x(3*n-2)

            CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +      NPRCIS, NUNITS)
            IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
               Rho(n) = PROP(3,0)
               s(n)   = PROP(8,0)
               h(n)   = PROP(9,0)
               Cp(n)  = PROP(14,0)
```

```
              dPdR(n) = PROP(22,0)
              dPdT(n) = PROP(23,0)

              dRdT(n) = -dPdT(n) / dPdR(n)
              dRdP(n) = 1/dPdR(n)
              dhdP(n) = 1/Rho(n) - x(3*n-2)*dPdT(n) /
     ;                            ( dPdR(n)*(Rho(n))**2)
              dudT(n) = Cp(n) - x(3*n-1)*dPdT(n) /
     ;                            ( dPdR(n)*(Rho(n))**2 )
              dudP(n) = ( x(3*n-1) - x(3*n-2)*dPdT(n) ) /
     ;                      ( dPdR(n)*(Rho(n))**2 )

              IF (IDID .EQ. 3) THEN
*                 This code not used in this case
              ENDIF
           ELSE
              CALL ERRMSG (MESSAG, IDID)
              WRITE (*, '(1X,A60)') MESSAG
              print *,n
              STOP
           ENDIF
1000     CONTINUE
         GOTO 50

       ELSE
         print *,' error good'
*          print *,'em = ',em,' ep = ',eP,' eT = ',eT
       ENDIF

************************************************************************
*
*   Deallocating the memory of the variables not being
*   passed back to main routine
*

* this line commented out when program was switched from using full A matrix
* to only using the SA and IJA vectors of the sparse matrix method
*       deallocate(A, b, SA, ija, dRdT, dRdP, dhdP, dudT, dudP, indx, xt)


      deallocate(b, SA, ija, dRdT, dRdP, dhdP, dudT, dudP, xt)

      END
```

E.7 Subroutine PUMP

```
      SUBROUTINE PUMP(dP, mPump)

**********************************************************************
*   PUMP2.FOR
*
*   Subroutine that uses the pump curve of a given pump to give the
*   mass flow for a given pressure change.
*
*   Brian Bowers
*   original version started on
*   2-27-97
*
*   last modified 2-27-97
*********************************************************************
*
*   VARIABLES USED
*
*
*     dP      - Total pressure drop around loop (atm)
*     mPump   - Value of m that corresponds to current dP (kg/s)
*
*
********************************************************************
*
*   My Variable assignments
        double precision dP
        double precision mPump

********************************************************************
*
*   Using the pump curve to find mPump
*   this curve is a very rough fit of data from
*   Lue, J.W., et al. Test of a Cryogenic Helium Pump.
*       Advances in Cryogenic Engineering, Vol. 27,
*       Plenum Press, New York, 1982.


* 200 rpm curve
*      mpump = (-1.4 * dp**2  +  1.2 * dp  +  10.1) / 1000

* 100 rpm curve
      mpump = (-1.29 * dp**2  +  0.61 * dp  +  6) / 1000

      RETURN
      END
```

E.8 Subroutine QLOAD

```
      SUBROUTINE Qload(x, uahxl, uahx2, dxhxl, dxhx2, Tpool, Nout, Nin,
     ;                Ntot, Qpool, Qhxl, Qhx2)

***********************************************************************
*   Qload3.FOR
*
*   Subroutine for finding the heat being rejected to the buffer tank
*
*   Note the x vector is used to hold the values of T, P, and m for
*   the length of the conductor.  Where T's, P's and m's for a
*   particular node n in the conductor are found by:
*                   T(n) = x(3*n-2)
*                   P(n) = x(3*n-1)
*                   m(n) = x(3*n)
*
*
*   Brian Bowers
*   original version started on
*   2-16-97
*
*   last modified 2-16-97
**********************************************************************
*
*   VARIABLES USED
*
*     DhHxl    - Hydraulic diameter of heat exchanger 1 (m)
*     DhHx2    - Hydraulic diameter of heat exchanger 2 (m)
*     Nin      - Node number of magnet inlet
*     Nout     - Node number of magnet outlet
*     Ntot     - Total number of nodes in loop
*     Qhxl     - Heat load being rejected to the buffer tank from hxl (W)
*     Qhx2     - Heat load being rejected to the buffer tank from hx2 (W)
*     Qpool    - Total heat load being rejected to the buffer tank (W)
*     Tpool    - Current temperature of buffer tank (K)
*     uaHxl    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     uaHx2    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     x(n)     - Vector containing T, P, and m data for all nodes
*
*********************************************************************
*
*   My Variable assignments

      double precision dxhxl, dxhx2
      integer Nin, Nout, Ntot
      real Qhxl, Qhx2
      real Qpool
      double precision Tpool
      real uahxl, uahx2
      double precision x(*)

*      print *, 'uahxl = ',uahxl,'uahx2 = ',uahx2
*      print *, 'dxhxl = ',dxhxl,'dxhx2 = ',dxhx2
*      print *, 'T*pool = ',tpool


       PRINT *,'IN QLOAD'

*********************************************************************
*
*   HEAT LOAD DUE TO HEAT EXCHANGER 1
*
      Qhxl = 0.0000000
      do 100 n = 2, Nin-1
         Qhxl = Qhxl + uahxl*(x(3*n-2)-Tpool)*dxhxl
100   enddo


*********************************************************************
*
*   HEAT LOAD DUE TO HEAT EXCHANGER 2
*
      Qhx2 = 0.0000000
      do 200 n = Nout+2, Ntot
         Qhx2 = Qhx2 + uahx2*(x(3*n-2)-Tpool)*dxhx2
200   enddo

      Qpool = Qhxl + Qhx2

      return
      end
```

E.9 Subroutine POOL

```
      SUBROUTINE POOL(Qss, Qpool, Vtank, mtank, utank, Tpool, Ptank,dt)

**********************************************************************
*   POOL3.FOR
*
*   Subroutine for evaluating the current temperature of the
*   buffer tank
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   2-16-97
*
*   last modified 2-16-97
*********************************************************************
*
*   VARIABLES USED
*
*
*     dt     - time step
*     mtank  - Total helium mass in the buffer tank (kg)
*     Ptank  - Current pressure in buffer tank (Pa)
*     Qpool  - Current heat load being rejected to buffer tank (W)
*     Qss    - Steady state heat load passing through buffer tank (W)
*     Tpool  - Temperature of buffer tank/pool (K)
*     utank  - Specific internal energy of buffer tank (J/kg-K)
*     Vtank  - Volume of buffer tank (m3)
*
*
********************************************************************
*
*   My Variable assignments
        double precision dt
        real mtank
        real Ptank
        real Qpool, Qss
        double precision Tpool
        real utank
        real Vtank
*
*   Variable assignments needed for HEPROP
*
      DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
      CHARACTER MESSAG*60
      INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
*********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
                NPRCIS=2
*
*        Units  (SI units)
                NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
                JOUT=11000


       PRINT *,'IN POOL'
       print *,'mtank = ',mtank

********************************************************************
*
*   Finding the new specific internal energy of the tank
*
*        print *,'qpool = ',qpool,'  Qss = ',Qss
*        print *,'Vtank = ',Vtank,'  Tpool = ',Tpool,' Ptank = ',Ptank
*        print *,'utank = ',utank
        utank = utank + (Qpool - Qss)*dt/mtank
*        print *,'utank = ',utank

********************************************************************
*
*   Finding the new temperature of the tank
*
        JIN1=3
        VALU1= mtank/Vtank
        JIN2= 7
        VALU2= utank

        CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +     NPRCIS, NUNITS)
```

```
      IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
         Ptank = PROP(1,0)
         Tpool = PROP(2,0)
         IF (IDID .EQ. 3) THEN
*               This code not used in this case
         ENDIF
      ELSE
         CALL ERRMSG (MESSAG, IDID)
         WRITE (*, '(1X,A60)') MESSAG
         STOP
      ENDIF

      RETURN
      END
```

E.10 Subroutine UNSTEADY

```
      SUBROUTINE UNSTEADY(x, xp, q, Nin, Nout, Ntot,
     :                 AcMag, DhMag, fmag, dxMag,
     :                 AcHx1, DhHx1, fhx1, uaHx1, dxHx1,
     :                 AcHx2, DhHx2, fhx2, uaHx2, dxHx2,
     :                 eta,
     :                Cp, Rho, s, h, dPdR, dPdT, dt,
     :                Pin, Tin, min, Tpool, Pmin, Tmax, Allo)

*********************************************************************
*   Unstead10.FOR    - spare matrix version
*
*   Subroutine for evaluating the temperature and pressure profile
*   in a recirculator loop under transient conditions.
*   (Used to for pulsed load cycling from a discharging cycle)
*   This version has 2 pool-cooled heat exchangers and a cable in
*   conduit conductor between them.
*
*   This version estimates spatial derivatives using the current node
*   and the previous node instead of the current node and next node.
*
*   Note the x vector is used to hold the values of T, P, and m for
*   the length of the conductor.  Where T's, P's and m's for a
*   particular node n in the conductor are found by:
*                  T(n) = x(3*n-2)
*                  P(n) = x(3*n-1)
*                  m(n) = x(3*n)
*
*   Note that the program originally used an LU decompostion method
*   of solution that put the coefficients into an A matrix.  This
*   method was slow and memory intensive so a sparse matrix method
*   was used.  However, the sparse matrix solution compacts the
*   A matrix into two vector: SA to store the diagonal and non-zero
*   elements, and IJA to store the indices.  Since the locations in
*   these vectors are very difficult to interpret, the original A matrix
*   assignments were left in, but commented out, to show where in the
*   A matrix the values are being assigned.
*
*   The routines needed to solve the sparse matrix are all from
*   Numerical Recipes: the Art of Scientific Computing, version 2.
*   The routines that were used are:
*        linbcg
*        atimes
*        asolve
*        dsprsax
*        dsprstx
*        snrm
*
*   This program must be linked with HEPROP.OBJ a property routine
*   program by CRYODATA
*
*
*   Brian Bowers
*   original version started on
*   11-3-96
*
*   last modified 5-19-97
*********************************************************************
*
*   VARIABLES USED
*
*     A       - Matrix containing derivatives - used for Newton's method
*     AcHx1   - Cross sectional area of heat exchanger 1 flow
*               passage (m2)
*     AcHx2   - Cross sectional area of heat exchanger 2 flow
*               passage (m2)
*     AcMag   - Cross sectional area of flow passage (m2)
*     Allo    - Flag to determine whether A, b, fm , fP, and fT arrays
*               have already been allocated
*     b       - Vector used for solving using Newton's method
*     Cp(n)   - Specific heat at node n (J/kg-K)
*     D       - Flag used by program LUDCMP and LUBKSB
*     DhHx1   - Hydraulic diameter of heat exchanger 1 (m)
*     DhHx2   - Hydraulic diameter of heat exchanger 2 (m)
*     DhMag   - Hydraulic diameter of flow passage (m)
*     dhdP(n) - dh/dP at constant temperature at node n (m3/kg)
*     dP      - Pressure change across pump (Pa)
***     dPin    - Amount to change the guess value of Pin in next iteration (Pa)
*     dPdR(n) - dP/dRho at constant temperature at node n (Pa-m3/kg)
*     dPdT(n) - dP/dT at constant density at node n (Pa/K)
*     dRdT(n) - dRho/dT at constant pressure at node n (kg /m3-K)
*     dRdP(n) - dRho/dP at constant temperature at node n ( kg/m3-Pa)
*     dudT(n) - du/dT at constant pressure at node n (J/kg-K)
*     dudP(n) - du/dP at constant temperature at node n (J/kg-Pa)
*               or (m3/kg)
*     dxHx1   - Distance between nodes in heat exchanger 1
*     dxHx2   - Distance between nodes in heat exchanger 2
*     dxMag   - Distance between nodes in magnet (m)
*     dt      - Time step between calculations (sec)
*     Em      - Total error in m equations
```

```
*     EP       - Total error in P equations
*     ET       - Total error in T equations
*     err      - Error value returned from sparse matrix solver (linbcg)
*     eta      - Isentropic efficiency of recirculator pump
*     fhx1     - Friction factor in heat exchanger 1
*     fhx2     - Friction factor in heat exchanger 2
*     fmag     - Friction factor
*     indx     - Vector used by program LUDCMP and LUBKSB
*     h(n)     - Specific enthalpy of helium at node n (J/kg)
*     ija      - Vector used to store column information for sparse
*                version of the A matrix (called SA)
*     iter     - Number of iterations used by linbcg
*     itmax    - Maximum allowable number of iterations in linbcg
*     itol     - Type of convergence check being used in linbcg
*     LenSA    - Length of SA and ija (vectors used to store sparse matrix)
*     mErr     - Difference between outlet and inlet mass flow rates (kg/s)
*     mErr1    - Value of mErr corresponding to Pin1
*     mErr2    - Value of mErr corresponding to Pin2
******     mErrLast- Previous value of mErr (last iteration)
*     min      - Inlet mass flow rate to the loop (kg/s)
*     mpump    - Value of m that corresponds to current dP (kg/s)
*     Nin      - Value of node number n for the conductor inlet
*     Nout     - Value of node number n for the conductor outlet
*     nRange   - Flag used for bisection search of Pin:
*                      1 - on first pick for Pin
*                      2 - on second pick for Pin
*                      3 - searching for range where   merr2 < 0 < merr1
*                     -1 - range has been found
*     Ntot     - Total number of nodes in system include those that are
*                not in the conductor  (ie 1/3 of the size of x )
*     Pin      - Inlet pressure (Pa)
*     Pin1     - Used to store a value of Pin when iterating on Pin
*                if nRange - -1 then we know  Pin is between Pin1 and Pin2
*     Pin2     - Used to store a value of Pin when iterating on Pin
*                if nRange - -1 then we know  Pin is between Pin1 and Pin2
*     PinLast  - Previous guess value for Pin (from last iteration)
*     Pmin     - Minimum allowable pressure in conductor (Pa)
*     q        - Heat flow rate per unit length (W/m)
*     Rho(n)   - Density at node n (kg/m3)
*     s(n)     - Specific entropy at node n (J/kg-K)
*     SA       - Sparse version of A matrix (stored in a vector)
*     scale    - Scaling factor used to search for range of Pin during
*                nRange - 3 mode
*     Tin      - Inlet temperature (K)
*     Tmax     - Maximum allowable temperature at any location(K)
*     tol      - Tolerance variable required for sparse matrix solver
*     Tpool    - Temperature of pool (K)
*     uaHx1    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     uaHx2    - Overall heat transfer coefficient PER UNIT LENGTH
*                of HX1 (W/m-K)
*     x(n)     - Vector containing T, P, and m data for all nodes
*     xp(n)    - Vector containing T, P, and m data for previous
*                time step
*     xt(n)    - Vector containing T, P, and m data for last iteration
*
******************************************************************
*
*   My Variable assignments
*        DOUBLE PRECISION A[allocatable](:,:)
       real AcHx1, AcHx2, AcMag
       integer allo
       DOUBLE PRECISION b[allocatable](:)
       DOUBLE PRECISION Cp(Ntot)
*        REAL D
       real DhHx1, DhHx2, DhMag
       DOUBLE PRECISION dP
*        DOUBLE PRECISION dPin
       DOUBLE PRECISION dPdR(Ntot), dPdT(Ntot)
       DOUBLE PRECISION dRdT[allocatable](:)
       DOUBLE PRECISION dRdP[allocatable](:)
       DOUBLE PRECISION dhdP[allocatable](:)
       DOUBLE PRECISION dudT[allocatable](:)
       DOUBLE PRECISION dudP[allocatable](:)
       DOUBLE PRECISION dxHx1, dxHx2, dxMag
       DOUBLE PRECISION dt
       real Em, ET, EP
       real fhx1, fhx2, fmag
       real h(Ntot)
*        INTEGER indx[allocatable](:)
       INTEGER ija[allocatable](:)
       INTEGER LenSA
       double precision mErr, mErr1, mErr2
*        double precision mErrLast
       double precision min, mpump
       Integer Nin, Nout, Ntot
       integer nRange
       DOUBLE PRECISION Pin, Pin1, Pin2
*        DOUBLE PRECISION PinLast
       DOUBLE PRECISION Pmin
       real q
       DOUBLE PRECISION Rho(Ntot), s(Ntot)
       DOUBLE PRECISION SA[allocatable](:)
```

```
      real scale
      double precision  Tin
      double precision Tmax, Tpool
      real uaHx1, uaHx2
      DOUBLE PRECISION x(Ntot*3), xp(Ntot*3)
      DOUBLE PRECISION xt[allocatable](:)

*
*   Variable assignments needed for sparse matrix solver linbcg
*
      integer itmax, itol, iter
      double precision tol, err

*
*   Variable assignments needed for HEPROP
*
     DOUBLE PRECISION VALU1, VALU2, PROP(0:41,0:2)
     CHARACTER MESSAG*60
     INTEGER IDID, NUNITS, NPRCIS, JIN1, JIN2, JOUT

*
**********************************************************************
*  Defining variable values for using HEPROP property routine
*
*       Precision (moderate)
               NPRCIS=2
*
*        Units  (SI units)
               NUNITS=1
*
*       Which properties to calculate (state variables and derivatives)
              JOUT=11000
*
************************************************************************
      print *, 'in UNSTEADY'

************************************************************************
*
*   Length of SA and ija (vectors used to store sparse matrix)
*   (there are 3*Ntot diagonal elements, 1 blank space,  7 off diagonal
*    elements for each of the Ntot-3 "elements", and 3 off diagonal
*    elements from continuity at both the HX1-Mag and Mag-HX2 interfaces.)
*
      LenSA = 3*Ntot + 1 + 7*(Ntot-3) + 3 * 2


************************************************************************
*
*   Allocating memory to the variables not being
*   passed back to main routine
*

     if (allo.eq.0) then
*          allocate(A(3*Ntot, b(3*Ntot), SA(LenSA), ija(LenSA),
          allocate(b(3*Ntot), SA(LenSA), ija(LenSA),
    ;       dRdT(Ntot), dRdP(Ntot), dhdP(Ntot), dudT(Ntot), dudP(Ntot),
    ;       xt(3*Ntot) )
*    ;       indx(3*Ntot), xt(3*Ntot) )
*          allo = 1
     endif


************************************************************************
*
*   Setting the variable PinLast to zero so that the logic at the
*   end of the program will know that this is the first time through
*   the iteration
*
*      PinLast = 0


      min = min
************************************************************************
*
*   Setting the variable nRange to 1 so that the logic at the
*   end of the program will know that this is the first time through
*   the iteration.  Also set scale variable for the bisection logic.
*
      nRange = 1
      scale =  1.05

***********************************************************************
*
*  Specifying ija vector
*
*     PATTERN IS:
*
*          FROM HX1:
*               first 3 rows  :  no off diagonals
*               Next 3*(Nin-2) rows have pattern [rows 4 to 3*(Nin-1) ]
```

```
*                  row 3*n-2 :  off diagonals at columns
*                                   3*n-5
*                                   3*n-4
*                                   3*n-1
*                  row 3*n-1 :  off diagonal at column 3*n-4
*                  row 3*n   :  off diagonals at columns
*                                   3*n-3
*                                   3*n-2
*                                   3*n-1
*
*         FROM MAGNET:
*              row 3*Nin-2 : off diagonal at column 3*(Nin-1)-2
*              row 3*Nin-1 : off diagonal at column 3*(Nin-1)-1
*              row 3*Nin   : off diagonal at column 3*(Nin-1)
*              Next 3*(Nout-Nin) rows  [ rows 3*Nin+1 to 3*Nout ]
*                  row 3*n-2 :  off diagonals at
*                                   3*n-5
*                                   3*n-4
*                                   3*n-1
*                  row 3*n-1 :  off diagonal at column 3*n-4
*                  row 3*n   :  off diagonals at columns
*                                   3*n-3
*                                   3*n-2
*                                   3*n-1
*
*         FROM HX2:
*              row 3*Nout+1 : off diagonal at column 3*Nout-2
*              row 3*Nout+2 : off diagonal at column 3*Nout-1
*              row 3*Nout+3 : off diagonal at column 3*Nout
*              Next 3*(Ntot-Nout-1) rows  [rows 3*Nout+4 to 3*Ntot]
*                  row 3*n-2 :  off diagonals at
*                                   3*n-5
*                                   3*n-4
*                                   3*n-1
*                  row 3*n-1 :  off diagonal at column 3*n-4
*                  row 3*n   :  off diagonals at columns
*                                   3*n-3
*                                   3*n-2
*                                   3*n-1
*
*     Setting up the first 3*Ntot+1 elements of ija   (from diagonal components)

        if (allo.eq.0) then

            ija(1) = 3*Ntot+2
            ija(2) = 3*Ntot+2
            ija(3) = 3*Ntot+2
            ija(4) = 3*Ntot+2

            do i = 5, 5 + 3*(Nin-3), 3
              ija(i)   = ija(i-1) + 3
              ija(i+1) = ija(i) + 1
              ija(i+2) = ija(i+1) + 3
            enddo

            ija(3*Nin-1) = ija(3*Nin-2)+1
            ija(3*Nin)   = ija(3*Nin-1)+1
            ija(3*Nin+1) = ija(3*Nin)  +1

            do i = 3*Nin+2, 3*Nin+2 + 3*(Nout-Nin-1), 3
              ija(i)   = ija(i-1) + 3
              ija(i+1) = ija(i) + 1
              ija(i+2) = ija(i+1) + 3
            enddo

            ija(3*Nout+2) = ija(3*Nout+1)+1
            ija(3*Nout+3) = ija(3*Nout+2)+1
            ija(3*Nout+4) = ija(3*Nout+3)+1

            do i = 3*Nout+5, 3*Nout+5 + 3*(Ntot-Nout-2), 3
              ija(i)   = ija(i-1) + 3
              ija(i+1) = ija(i) + 1
              ija(i+2) = ija(i+1) + 3
            enddo

*       setting up the off diagonal components of ija

           ija(3*Ntot+2) = 1
           ija(3*Ntot+3) = 2
           ija(3*Ntot+4) = 5
           ija(3*Ntot+5) = 2
           ija(3*Ntot+6) = 3
           ija(3*Ntot+7) = 4
           ija(3*Ntot+8) = 5


           do i = 3*Ntot+9, 3*Ntot+9 + 7*(Nin-3) - 1
             ija(i) = ija(i-7) + 3
           enddo

           ija(3*Ntot+9 + 7*(Nin-3)  ) = 3*Nin-5
           ija(3*Ntot+9 + 7*(Nin-3)+1) = 3*Nin-4
           ija(3*Ntot+9 + 7*(Nin-3)+2) = 3*Nin-3
```

```
         ija(3*Ntot+9 + 7*(Nin-3)+3) = 3*Nin-2
         ija(3*Ntot+9 + 7*(Nin-3)+4) = 3*Nin-1
         ija(3*Ntot+9 + 7*(Nin-3)+5) = 3*Nin+2
         ija(3*Ntot+9 + 7*(Nin-3)+6) = 3*Nin-1
         ija(3*Ntot+9 + 7*(Nin-3)+7) = 3*Nin
         ija(3*Ntot+9 + 7*(Nin-3)+8) = 3*Nin+1
         ija(3*Ntot+9 + 7*(Nin-3)+9) = 3*Nin+2


         do i = 3*Ntot+9 + 7*(Nin-3)+10, 3*Ntot+7*(Nout-3)+11
           ija(i) = ija(i-7) + 3
         enddo

         ija(3*Ntot+7*(Nout-3)+12) = 3*Nout-2
         ija(3*Ntot+7*(Nout-3)+13) = 3*Nout-1
         ija(3*Ntot+7*(Nout-3)+14) = 3*Nout

         ija(3*Ntot+7*(Nout-3)+15) = 3*Nout+1
         ija(3*Ntot+7*(Nout-3)+16) = 3*Nout+2
         ija(3*Ntot+7*(Nout-3)+17) = 3*Nout+5
         ija(3*Ntot+7*(Nout-3)+18) = 3*Nout+2
         ija(3*Ntot+7*(Nout-3)+19) = 3*Nout+3
         ija(3*Ntot+7*(Nout-3)+20) = 3*Nout+4
         ija(3*Ntot+7*(Nout-3)+21) = 3*Nout+5



         do i = 3*Ntot+7*(Nout-3)+22, LenSA
             ija(i) = ija(i-7) + 3
         enddo
         allo = 1

      endif

*         print *,' ija(1) = ',ija(1), ' 3*Ntot = ',3*Ntot
*         print *,'  LenSA = ',LenSA
*          k=0
*          do i = 1, LenSA
*             k = k+1
*             print *,'i = ',i, '  ija = ',ija(i)
*             if (k.eq.20) then
*                 pause ' <RETURN> for more'
*                 k=0
*             endif
*          enddo
*          pause ' <RETURN> to continue '
*           stop



************************************************************************
*
*   DERIVATIVES
*


      DO 30 n = 1, Ntot
        dRdT(n) = -dPdT(n) / dPdR(n)
        dRdP(n) = 1/dPdR(n)
        dhdP(n) = 1/Rho(n) - x(3*n-2)*dPdT(n) /( dPdR(n)*(Rho(n))**2 )
        dudT(n) = Cp(n) - x(3*n-1)*dPdT(n) /( dPdR(n)*(Rho(n))**2 )
        dudP(n) = ( x(3*n-1) - x(3*n-2)*dPdT(n) ) /
     ;                          ( dPdR(n)*(Rho(n))**2 )

30    CONTINUE

*       DO 40 n = 1, Ntot
*       write(*,*) ' n = ',n
*       write(*,11)' T(n) =    ',x(3*n-2), ' P(n) =   ',x(3*n-1),
*     ;    ' m(n) =    ', x(3*n)
*       write(*,11)' Tp(n) =   ',xp(3*n-2),' Pp(n) =   ',xp(3*n-1),
*     ;    ' mp(n) =  ', xp(3*n)
*       read(*,*)
*40    CONTINUE



************************************************************************
*
*  CLEARING THE A MATRIX AND THE b VECTOR TO START THE SOLUTION
*
*
*
50    CONTINUE
      DO 200 i = 1, 3*Ntot
         b(i) = 0.0
*         DO 300 j = 1, 3*Ntot
*           A(i,j) = 0
*300      CONTINUE
200   CONTINUE
```

```
      DO 300 i = 1, LenSA
         SA(i) = 0.0
300   CONTINUE

**********************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE FIRST HEAT EXCHANGER
*  (the A matrix was commented out when the sparse matrix solution was added)
*
*   SETTING UP THE FIRST THREE EQNS IN THE MATRIX
*   NODE 1 IS SAME AS PUMP
*   ASSUMING P1 = Pin and T1 = Tin and m1 = msteady


*        A(1,1) = 1
      SA(1)  = 1
      b(1) = Tin

*        A(2,2) = 1
      SA(2)  = 1
      b(2) = Pin

*        A(3,3) = 1
      SA(3)  = 1
      b(3) = min


*  starting a counter for filling in the off diagonal elements at the
*  end of the SA matrix:

       k = 3*Ntot + 2


*  REMAINING HX1 EQUATIONS
       DO 400 n = 2, Nin -1

*    First Law equations
*        A(3*n-2,3*n-5)   = -x(3*n)*Cp(n) /dxHx1
               SA(k)     = -x(3*n)*Cp(n) /dxHx1
               k = k+1
*        A(3*n-2,3*n-4)   = -x(3*n)*dhdP(n)/dxHx1
               SA(k)     = -x(3*n)*dhdP(n)/dxHx1
               k = k+1
**        A(3*n-2,3*n-3)   = -x(3*n-1) / (Rho(n)*dxHx1)
*        A(3*n-2,3*n-2)  = x(3*n)*Cp(n)/dxHx1 + Rho(n)*AcHx1*dudT(n)/dt
**    ;                             + uaHx1
*    ;                        -AcHx1*x(3*n-1)*dRdT(n)/(Rho(n)*dt) + uaHx1
               SA(3*n-2)= x(3*n)*Cp(n)/dxHx1 + Rho(n)*AcHx1*dudT(n)/dt
     ;                        -AcHx1*x(3*n-1)*dRdT(n)/(Rho(n)*dt) + uaHx1
*        A(3*n-2,3*n-1)  = x(3*n)*dhdP(n)/dxHx1 + Rho(n)*AcHx1*dudP(n)/dt
*    ;                      -AcHx1*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
               SA(k)    = x(3*n)*dhdP(n)/dxHx1 + Rho(n)*AcHx1*dudP(n)/dt
     ;                      -AcHx1*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
               k = k+1
**        A(3*n-2,3*n)     = x(3*n-1)/(Rho(n)*dxHx1)

        b(3*n-2)          = Rho(n)*AcHx1*dudT(n)*xp(3*n-2)/dt +
     ;                   Rho(n)*AcHx1*dudP(n)*xp(3*n-1)/dt + uaHx1*Tpool
     ;                   -AcHx1*x(3*n-1)*dRdT(n)*xp(3*n-2)/(Rho(n)*dt)
     ;                   -AcHx1*x(3*n-1)*dRdP(n)*xp(3*n-1)/(Rho(n)*dt)

*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4)   = -1/dxHx1
               SA(k)     = -1/dxHx1
               k = k+1
*        A(3*n-1,3*n-1)   =  1/dxHx1
               SA(3*n-1) =  1/dxHx1
        b(3*n-1)        = -fHx1*x(3*n)*abs(x(3*n))/
     ;                          (2*Rho(n)*DhHx1*AcHx1**2)


*    Mass Conservation equations
*        A(3*n,3*n-3)     = -1/dxHx1
               SA(k)     = -1/dxHx1
               k = k+1
*        A(3*n,3*n-2)     = AcHx1*dRdT(n) / dt
               SA(k)     = AcHx1*dRdT(n) / dt
               k = k+1
*        A(3*n,3*n-1)     = AcHx1*dRdP(n) / dt
               SA(k)     = AcHx1*dRdP(n) / dt
               k = k+1
*        A(3*n,3*n)       =  1/dxHx1
               SA(3*n)   =  1/dxHx1
        b(3*n)           =  (AcHx1*dRdT(n)/dt)*xp(3*n-2)+
     ;                      (AcHx1*dRdP(n)/dt)*xp(3*n-1)


400   ENDDO
```

```
************************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE CICC MAGNET CABLE
*  (the A matrix was commented out when the sparse matrix solution was added)
*
*   ASSUMING CONTINUITY OF m, P and T BETWEEN THE HX1 AND THE MAGNET INLET
*   THE FIRST 3 EQUATION FOR THE MAGNET ARE THEN
*
*   temperature continuity
*      A(3*Nin-2,3*(Nin-1)-2)    = -1
           SA(k)              = -1
           k = k+1
*      A(3*Nin-2,3*Nin-2)        =  1
           SA(3*Nin-2)        =  1

*   pressure continuity
*      A(3*Nin-1,3*(Nin-1)-1)    = -1
           SA(k)              = -1
           k = k+1
*      A(3*Nin-1,3*Nin-1)        =  1
           SA(3*Nin-1)        =  1

*   mass continuity
*      A(3*Nin,3*(Nin-1))        = -1
           SA(k)              = -1
           k = k+1
*      A(3*Nin,3*Nin)            =  1
           SA(3*Nin)          =  1

*   REMAINING MAGNET EQUATIONS
      DO 600 n = Nin+1, Nout

*    First Law equations
*        A(3*n-2,3*n-5) = -x(3*n)*Cp(n) /dxMag
           SA(k)       = -x(3*n)*Cp(n) /dxMag
           k = k+1
*        A(3*n-2,3*n-4) = -x(3*n)*dhdP(n)/dxMag
           SA(k)       = -x(3*n)*dhdP(n)/dxMag
           k = k+1
**         A(3*n,3*n-3)   = -x(3*n-1) / (Rho(n)*dxMag)
*        A(3*n-2,3*n-2) = x(3*n)*Cp(n)/dxMag + Rho(n)*AcMag*dudT(n)/dt
*     ;                    - AcMag*x(3*n-1)*dRdT(n)/(Rho(n)*dt)
           SA(3*n-2)   = x(3*n)*Cp(n)/dxMag + Rho(n)*AcMag*dudT(n)/dt
     ;                     - AcMag*x(3*n-1)*dRdT(n)/(Rho(n)*dt)
*        A(3*n-2,3*n-1) = x(3*n)*dhdP(n)/dxMag + Rho(n)*AcMag*dudP(n)/dt
*     ;                    - AcMag*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
           SA(k)       =x(3*n)*dhdP(n)/dxMag + Rho(n)*AcMag*dudP(n)/dt
     ;                     - AcMag*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
           k = k+1
**         A(3*n,3*n)     = x(3*n-1)/(Rho(n)*dxMag)
         b(3*n-2)      = Rho(n)*AcMag*dudT(n)*xp(3*n-2)/dt +
     ;                  Rho(n)*AcMag*dudP(n)*xp(3*n-1)/dt + q
     ;                  - AcMag*x(3*n-1)*dRdT(n)*xp(3*n-2)/(Rho(n)*dt)
     ;                  - AcMag*x(3*n-1)*dRdP(n)*xp(3*n-1)/(Rho(n)*dt)

*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4) = -1/dxMag
           SA(k)       = -1/dxMag
           k = k+1
*        A(3*n-1,3*n-1) =  1/dxMag
           SA(3*n-1)   =  1/dxMag
         b(3*n-1)      = -fmag*x(3*n)*abs(x(3*n))/
     ;                          (2*Rho(n)*DhMag*AcMag**2)

*    Mass Conservation equations
*        A(3*n,3*n-3) = -1/dxMAg
           SA(k)     = -1/dxMag
           k = k+1
*        A(3*n,3*n-2) =  AcMag*dRdT(n) / dt
           SA(k)     =  AcMag*dRdT(n) / dt
           k = k+1
*        A(3*n,3*n-1) =  AcMag*dRdP(n) / dt
           SA(k)     =  AcMag*dRdP(n) / dt
           k = k+1
*        A(3*n,3*n)   =  1/dxMag
           SA(3*n)   =  1/dxMag
         b(3*n)      = (AcMag*dRdT(n)/dt)*xp(3*n-2)+
     ;                   (AcMag*dRdP(n)/dt)*xp(3*n-1)

 600     ENDDO

************************************************************************
*
*  SETTING UP THE A and b COMPONENTS DUE TO THE SECOND HEAT EXCHANGER
*
*
*   ASSUMING CONTINUITY OF m, P and T BETWEEN THE MAGNET EXIT AND HX2 INLET
*   THE FIRST 3 EQUATION FOR HX2 ARE THEN
*
*   temperature continuity
```

```
*     A(3*Nout+1,3*Nout-2)      = -1
           SA(k)              = -1
           k = k+1
*     A(3*Nout+1,3*(Nout+1)-2) =  1
           SA(3*Nout+1)       =  1

*   pressure continuity
*     A(3*Nout+2,3*Nout-1)      = -1
           SA(k)              = -1
           k = k+1
*     A(3*Nout+2,3*(Nout+1)-1) =  1
           SA(3*Nout+2)       =  1

*   mass continuity
*     A(3*Nout+3,3*Nout)        = -1
           SA(k)              = -1
           k = k+1
*     A(3*Nout+3,3*(Nout+1))   =  1
           SA(3*Nout+3)       =  1

*  REMAINING HX2 EQUATIONS

      DO 700 n = Nout+2, Ntot

*    First Law equations
*        A(3*n-2,3*n-5) = -x(3*n)*Cp(n) /dxHx2
            SA(k)       = -x(3*n)*Cp(n) /dxHx2
            k = k+1
*        A(3*n-2,3*n-4) = -x(3*n)*dhdP(n)/dxHx2
            SA(k)       = -x(3*n)*dhdP(n)/dxHx2
            k = k+1
**        A(3*n-2,3*n-3)    = -x(3*n-1) / (Rho(n)*dxHx2)
*        A(3*n-2,3*n-2) = x(3*n)*Cp(n)/dxHx2 + Rho(n)*AcHx2*dudT(n)/dt
**      ;                         + uaHx2
*      ;                        -AcHx2*x(3*n-1)*dRdT(n)/(Rho(n)*dt) + uaHX2
            SA(3*n-2)   = x(3*n)*Cp(n)/dxHx2 + Rho(n)*AcHx2*dudT(n)/dt
     ;                          -AcHx2*x(3*n-1)*dRdT(n)/(Rho(n)*dt) + uaHX2
*        A(3*n-2,3*n-1) = x(3*n)*dhdP(n)/dxHx2 + Rho(n)*AcHx2*dudP(n)/dt
*      ;                        -AcHx2*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
            SA(k)       = x(3*n)*dhdP(n)/dxHx2 + Rho(n)*AcHx2*dudP(n)/dt
     ;                         -AcHx2*x(3*n-1)*dRdP(n)/(Rho(n)*dt)
            k = k+1
**        A(3*n-2,3*n)      = x(3*n-1)/(Rho(n)*dxHx2)

       b(3*n-2)         = Rho(n)*AcHx2*dudT(n)*xp(3*n-2)/dt +
     ;                    Rho(n)*AcHx2*dudP(n)*xp(3*n-1)/dt + uaHx2*Tpool
     ;                    -AcHx2*x(3*n-1)*dRdT(n)*xp(3*n-2)/(Rho(n)*dt)
     ;                    -AcHx2*x(3*n-1)*dRdP(n)*xp(3*n-1)/(Rho(n)*dt)

*    Momentum (pressure drop) equations
*        A(3*n-1,3*n-4) = -1/dxHx2
            SA(k)       = -1/dxHx2
            k = k+1
*        A(3*n-1,3*n-1) =  1/dxHx2
            SA(3*n-1)  =  1/dxHx2
       b(3*n-1)         = -fHx2*x(3*n)*abs(x(3*n))/
     ;                              (2*Rho(n)*DhHx2*AcHx2**2)

*    Mass Conservation equations
*        A(3*n,3*n-3) = -1/dxHx2
            SA(k)     = -1/dxHx2
            k = k+1
*        A(3*n,3*n-2) =  AcHx2*dRdT(n) / dt
            SA(k)     =  AcHx2*dRdT(n) / dt
            k = k+1
*        A(3*n,3*n-1) =  AcHx2*dRdP(n) / dt
            SA(k)     =  AcHx2*dRdP(n) / dt
            k = k+1
*        A(3*n,3*n)    =  1/dxHx2
            SA(3*n)   =  1/dxHx2
       b(3*n)         = (AcHx2*dRdT(n)/dt)*xp(3*n-2)+
     ;                    (AcHx2*dRdP(n)/dt)*xp(3*n-1)

700   ENDDO

************************************************************************
* logic used if you ever want to check the A or b matrix
*        open (unit = 8, file = 'A.out', status = 'unknown')
*        open (unit = 9, file = 'A.prn', status = 'unknown')
*        open (unit = 10, file = 'b.prn', status = 'unknown')
*          write(8,87)'  dt = ',dt,'  dx = ',dxMag,'    q = ',q,
*      ;              '  AcMag =',AcMag,'  DhMag = ',DhMag,'  fmag = ',fmag
*87        format(a8,f5.3,a8,f6.2,a8,f6.5,a8,f7.6,a8,f6.5,a8,f5.4)
*
*          do i = 1, 3*Ntot
*                write(8,88) (A(i,j), j=1,3*Ntot)
*                write(9,88) (A(i,j), j=1,3*Ntot)
*88              format(15E10.4)
*          enddo
*
*          do i = 1, 3*Ntot
*                write(8,89) b(i)
```

```
*              write(10,89) b(i)
*89            format(E9.3)
*         enddo
*
*         write(8,*)'  T        P       m      Cp     rho',
*     ;              '      dPDdR     dPdT'
*         do n = 1,Ntot
*            write(8,90) x(3*n-2),x(3*n-1),x(3*n),cp(n),rho(n),
*     ;                 dPdR(n), dPdT(n)
*90          format(1x,f5.2,2x,f7.0,1x,f6.5,2x,f6.0,2x,f5.1,2x,
*     ;           f8.1,2x,f8.1)
*         enddo
*        close (8)
*        close (9)
*        close (10)

******************************************************************
*
*  Before Solving, save the previous values of the x vector in xt
*
        DO 750 i = 1, 3*Ntot
             xt(i) = x(i)
750     CONTINUE

******************************************************************
*
*  Solving for the new x vector -- the sparse matrix solver LINBCG
*  from Numerical Recipes is used.  The LU decomposition matrix solver
*  (LUDCMP and LUBKSB also from Numerical Recipes) that was originally
*  used is shown, but is commented out.
*
* tolerance and maximum number of iteration needed for LINBCG
*
      itol = 1
      tol  = 1e-10
      itmax = 200

       call linbcg(SA,ija,3*Ntot,b,x,itol,tol,itmax,iter,err)

*       CALL LUDCMP(A,3*Ntot,3*Ntot,INDX,D)
*       CALL LUBKSB(A,3*Ntot,3*Ntot,INDX,b)

* lines used if you wan to watch values as the solution progresses
*       DO 760 n = 1, Ntot
*        write(*,*) ' n = ',n
*        write(*,11)' T(n) =   ',b(3*n-2), ' P(n) =  ',b(3*n-1),
*     ;    ' m(n) =   ', b(3*n)
*        write(*,11)' Ts(n) =  ',x(3*n-2),' Ps(n) =  ',x(3*n-1),
*     ;    ' ms(n) =  ', x(3*n)
*        read(*,*)
*11      format(A10,f9.3,a9,f10.0,a10,f7.5)
*760     CONTINUE


******************************************************************
*
**  Saving the previous values of the x vector in xt and
*  Assigning the new values to the x vector
*
*  this section uses a weighted average of the new solution and
*  the old values (A relaxation technique) to improve stability
        DO 800 i = 1, 3*Ntot
             x(i) = (x(i)*.9 + xt(i)*.1)
800     CONTINUE



*       DO 850 n = 1, Ntot
*        write(*,*) ' n = ',n
*        write(*,11)' T(n) =   ',x(3*n-2), ' P(n) =  ',x(3*n-1),
*     ;    ' m(n) =   ', x(3*n)
*        write(*,11)' Tp(n) =  ',xp(3*n-2),' Pp(n) =  ',xp(3*n-1),
*     ;    ' mp(n) =  ', xp(3*n)
*        read(*,*)
*11      format(A10,f9.3,a9,f10.0,a10,f7.5)
*850     CONTINUE




************************************************************************
*
*  CHECKING THE ERROR
        Em=0
        EP = 0
        ET = 0


         DO 900 n = 1, NTOT
             if (  abs( (x(3*n)-xt(3*n)) / x(3*n) ).GT.Em ) then
                      Em = abs( (x(3*n)-xt(3*n)) / x(3*n) )
             endif
```

```
            if (  abs( (x(3*n-1)-xt(3*n-1)) / x(3*n-1) ).GT.EP ) then
                     EP = abs( (x(3*n-1)-xt(3*n-1)) / x(3*n-1) )
            endif
            If (  abs( (x(3*n-2)-xt(3*n-2)) / x(3*n-2) ).GT.ET ) then
                     ET = abs( (x(3*n-2)-xt(3*n-2)) / x(3*n-2) )
            endif
900       CONTINUE


      IF (Em.GT.5E-5. or. EP.GT.1E-5 .or .ET.GT.1E-5) THEN

*         print *,'em = ',em,' ep = ',eP,' eT = ',eT

*****************************************************************
*    CHECK FOR NEGATIVE PRESSURES
*     if pressure is negative then mass flow rate is too high
*      lower min, reset x vector and return to top
*      otherwise, continue finding the new properties


      do n = 2, 3*Ntot-1,3

        if ( x(n).lt.0) then
          print *,'negative pressure found - lowering min'

          min = 0.8*min
          do i = 1, 3*Ntot
             x(i) = xp(i)
          enddo
          goto 50
        endif
      enddo


*****************************************************************
*    CHECK FOR EXTRA LOW TEMPERATURES
*     if temperature is below 0.8 K then inlet temp is too low
*      raise Tin, reset x vector and return to top
*      otherwise, continue finding the new properties


      do n = 1, 3*Ntot-2,3

        if ( x(n).lt.0.8) then
          print *,'temperature below 0.8 K - raising Pin'
          print *,'n =',(n+2)/3
*          print *,'Pin = ',Pin,'  Tin =  ',Tin
           write(*,11)'  Tin =   ',Tin, ' Pin =  ',Pin,
     ;       ' min =   ', min
11         format(A10,f9.5,a9,f10.0,a10,f8.6)
          Pin = Pin+50
          min = min*1.125
           write(*,11)'  Tin =   ',Tin, ' Pin =  ',Pin,
     ;       ' min =   ', min

*          print *,'Pin = ',Pin,'  Tin =  ',Tin

          do i = 1, 3*Ntot
             x(i) = xp(i)
          enddo
          goto 50
        endif
      enddo


*****************************************************************
*   CALCULATING THE PROPERTIES AT ALL NODES USING THE NEW X VECTOR
       DO 1000 n = 1, Ntot

*      Finding the conditions at the current location
       JIN1=1
       VALU1=x(3*n-1)
       JIN2=2
       VALU2=x(3*n-2)


       CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
     +    NPRCIS, NUNITS)
       IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
          Rho(n) = PROP(3,0)
          s(n)   = PROP(8,0)
          h(n)   = PROP(9,0)
          Cp(n)  = PROP(14,0)
          dPdR(n) = PROP(22,0)
          dPdT(n) = PROP(23,0)

          dRdT(n) = -dPdT(n) / dPdR(n)
          dRdP(n) = 1/dPdR(n)
          dhdP(n) = 1/Rho(n) - x(3*n-2)*dPdT(n) /( dPdR(n)*(Rho(n))**2)
          dudT(n) = Cp(n) - x(3*n-1)*dPdT(n) /( dPdR(n)*(Rho(n))**2 )
          dudP(n) = ( x(3*n-1) - x(3*n-2)*dPdT(n) ) /
     ;                         ( dPdR(n)*(Rho(n))**2 )
```

```
          IF (IDID .EQ. 3) THEN
*                This code not used in this case
          ENDIF
       ELSE
          CALL ERRMSG (MESSAG, IDID)
          WRITE (*, '(1X,A60)') MESSAG
          print *,n,x(3*n-1),x(3*n-2)
          STOP
       ENDIF

1000    CONTINUE



         GOTO 50
       else
*         print *,' error good'
*         print *,'em = ',em,' ep = ',eP,' eT = ',eT

*         DO 950 n = 1, Ntot
*           write(*,*) ' n = ',n
*           write(*,11)' T(n) =   ',x(3*n-2), ' P(n) =  ',x(3*n-1),
*    ;       ' m(n) =   ', x(3*n)
*           write(*,11)' Tp(n) =  ',xp(3*n-2),' Pp(n) =  ',xp(3*n-1),
*    ;       ' mp(n) =  ', xp(3*n)
*           write(*,*) ' dRdT = ',dRdT(n) ,'  dRdP = ',dRdP(n)
*           write(*,*) ' rho = ',rho(n),'  cp = ',cp(n)
*           write(*,*) ' energy equation:',
*    ;               x(3*n)*Cp(n) * ( x(3*n-2) - x(3*n-5) ) / dxMag+
*    ;               x(3*n)*dhdP(n)*( x(3*n-1) - x(3*n-4) ) / dxMag+
*    ;               x(3*n-1)*( x(3*n)-x(3*n-3) )/Rho(n)/dxMag+
*    ;               Rho(n)*AcMag*dudT(n)*( x(3*n-2) - xp(3*n-2) ) /dt+
*    ;               Rho(n)*AcMag*dudP(n)*( x(3*n-1) - xp(3*n-1) ) /dt - q
*
*           read(*,*)
*11         format(A10,f9.5,a9,f10.0,a10,f8.6)
950      CONTINUE
       ENDIF



************************************************************************
*
*   USING THE PUMP CHARACTERISTIC EQN TO FIND THE MASS FLOW RATE THAT THE
*   PUMP SHOULD HAVE TO GIVE THE CURRENT TOTAL PRESSURE DROP AROUND THE LOOP
*


      dP = ( x(2)-x(3*Ntot-1) ) / 101325
*      PRINT *,'dp = ',dp

      CALL PUMP(dP,mPump)
*      print *,'mpump = ',mpump


**********************************************************************
*
*    CHECKING FOR CONSISTENCY WITH THE PUMP ISENTROPIC EFFICIENCY
*
*    ISENTROPIC EXIT ENTHALPY OF PUMP
       JIN1=1
       VALU1=x(2)
       JIN2=5
       VALU2=s(Ntot)

       CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
    +     NPRCIS, NUNITS)
       IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
          hs   = PROP(9,0)
          IF (IDID .EQ. 3) THEN
*                This code not used in this case
          ENDIF
       ELSE
          CALL ERRMSG (MESSAG, IDID)
          WRITE (*, '(1X,A60)') MESSAG
          print *,'failure from pump lookup: 1'
          STOP
       ENDIF

       hl= h(Ntot) + ( hs - h(Ntot) )/Eta
*       print *,'hl = ',hl

*    TEMPERATURE AT EXIT OF PUMP
       JIN1=1
       VALU1=x(2)
       JIN2=6
       VALU2=hl

       CALL CALC (IDID, PROP, JOUT, JIN1, VALU1, JIN2, VALU2,
    +     NPRCIS, NUNITS)
       IF ((IDID .EQ. 1) .OR. (IDID .EQ. 3)) THEN
```

```
          T1    = PROP(2,0)
          IF (IDID .EQ. 3) THEN
*                This code not used in this case
          ENDIF
       ELSE
          CALL ERRMSG (MESSAG, IDID)
          WRITE (*, '(1X,A60)') MESSAG
          print *,'failure from pump lookup: 2'
          STOP
       ENDIF


*   Double iteration logic
*     COMPARING THE CALCULATED TEMPERATURE AT NODE 1 WITH THE CURRENT VALUE
*     IF THEY DISAGREE, THEN ADJUST Tin.
*     ALSO COMPARE THE PRESSURE DROP AND ADJUST m AS NEEDED


       if (abs( (T1-Tin) /Tin ) .gt. 5e-4  .OR.
     ;          abs( (mpump-min)/min ) .GT. 5E-4) THEN
*            print *,'min = ',min,'   Tin = ',Tin
           Tin = Tin*0.1 + T1*0.9

           if ( 0.3*mpump + 0.7*min .lt. 0.5*min) then
              min = min * 0.9
           else
              min = 0.3*mpump + 0.7*min
           endif
           print *,'min = ',min,'   Tin = ',Tin
           goto 50
        endif





*     Comparing the inlet and outlet mass flow rates
*     If they disagree, adjust Pin using a bisection method

       mErr = x(3*Ntot) - min
       print *,'mErr/min = ',mErr/min
       Print *,'Pin = ',Pin
*       print *,'nRange = ',nRange

      IF (  abs( mErr/min ).gt.5e-3) THEN
         if (nRange.ne.-1) then
*        finding a range for Pin
             IF (nRange.ne.3) THEN
*               getting and storing Pin and mErr values for 2 guesses of Pin
                if(nRange.eq.1) then
*                 nRange = 1 : first guess
                  Pin1   = Pin
                  mErr1  = mErr
                  Pin    = Pin + 20
                  nRange = 2
                else
*                 nRange must be = 2 : second guess
                  Pin2  = Pin
                  merr2 = mErr
                  Pin    = Pin2 + (Pin1-Pin2)*merr2/(merr2-mErr1)*scale
*                 range is found if merr1 and merr2 have opposite signs
*                 therefore start bisection on next iteration (nRange = -1)
*                 otherwise need to search using nRange =3 logic
                  IF (merr2*merr1.lt.0) THEN
                    nRange  = -1
                  ELSE
                    nRange= 3
                  ENDIF
                endif
             ELSE
*               nRange = 3 : searching for range of Pin where merr changes sign
                if ( mErr*mErr1.GT.0) then
*                  mErr and mErr1 must have same sign - increase range
                   scale = scale *1.5
                   Pin   = Pin2 + (Pin1-Pin2)*merr2/(merr2-mErr1)*scale
                else
*                  mErr and mErr1 must have opposite sign - Range found
*                  store value with merr < 0 as Pin2 and mErr2
                   nRange = -1
                   IF (mErr.lt.0) then
                       mErr2 = mErr
                       Pin2 = Pin
                   ELSE
                       mErr2 = mErr1
                       Pin2  = Pin1
                       mErr1 = mErr
                       Pin1  = Pin
                   ENDIF
                   Pin = Pin2 + (Pin1-Pin2)*merr2/(merr2-mErr1)
                endif
             ENDIF
         else
```

```
*         nrange = -1 : bisection routine
              IF (mErr.gt.0) THEN
                 Pin1 = Pin
                 mErr1 = mErr
              ELSE
                 Pin2 = Pin
                 mErr2 = mErr
              ENDIF
*            either use pure bisection or do and interopolation
*            (comment out the one you not using)

*             Pin = (Pin1+Pin2)/2
              Pin = Pin1 + (Pin1-Pin2)*mErr1/(mErr2-mErr1)

          endif
*          Print *,'Pin 1 = ',Pin1, '  Pin2  = ',Pin2
          Print *,'Pin = ',Pin
*          Read(*,*)

          GOTO 50

       ENDIF

*  note there is no need to deallocate the vectors and matrices since this is the
*  last thing being done in the program.  There is no reason to try to free up
*  the memory for something else since nothing else is being done.

       END
```

35 12 - 46