4-23-2012

A Miniature-Scale Linear Compressor for Electronics Cooling.

Craig R. Bradshaw
Purdue University - Main Campus, cbradsha@purdue.edu

A MINIATURE-SCALE LINEAR COMPRESSOR FOR ELECTRONICS COOLING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Craig R. Bradshaw

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2012

Purdue University

West Lafayette, Indiana

To Shawna and our family.

ACKNOWLEDGMENTS

I would like to thank my advisors; Profs. Eckhard Groll and Suresh Garimella. You have both challenged me in your own way and never accepted anything but the best I can give. I have come out stronger in the end as a result, and for that, I am eternally grateful. My other committee members, Profs. Charles Krousgrill, and Steve Pekarek have provided wonderful feedback, support, and discussion. I am also thankful to Prof. Doug Adams for his advice and assistance with the compressor short courses, Prof. Patricia Davies for her excellent leadership of Herrick Labs, and Prof. John Abraham for his helpful teaching suggestions.

I also could not have completed this journey without the help and support from the staff at Herrick Lab. Judy, Donna, Ginny, Gil, and Bob, who make Herrick feel like a family. I especially want to thank Frank Lee who was always around with a smile on his face, willing to help, or at least point you in the right direction.

Certainly, I would not have enjoyed, or been as successful, without the help of my fellow labmates. Dr. Jitendra Gupta for your friendship and being an excellent officemate. Drs. Stephan Bertsch and Abhijit Sathe for guiding me through my early years. Drs. Ian Bell and Margaret Mathison for being wonderful friends making one of the best compressor research teams. Christian Bach for entertaining all my coffee break requests. Abhinav Krishna, Bryce Shaffer, Howard Cheung, Brandon Woodland, Seth Holloway, Simbarashe Nyika, Tim Blatchley, Cord Tomforde, Nate Taylor, Steven Caskey, Scott Flueckiger, and Robert Leffler, who have each made my time at Herrick, and Purdue, filled with excellent memories.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

SYMBOLS

A	Area	$[m^2]$
COP	Coefficient of Performance	$[-]$
COP_{carnot}	Carnot Coefficient of Performance	$[-]$
C_D	Drag coefficient for flow past reed valve	$[-]$
C_v	Specific heat capacity at constant volume	$[J\,kg^{-1}\,K^{-1}]$
D	Diameter	$[m]$
$Drift$	Piston drift	$[m]$
E	Total energy in the control volume	$[kW]$
F_{drive}	Driving force from linear motor	$[N]$
F_{shear}	Force generated by shear stress between piston and cylinder	$[N]$
J_{CG}	Rotational moment of inertia of the piston about the CG	$[kg\,m]$
L_1	Reciprocating compressor crank length	$[m]$
L_2	Reciprocating compressor connecting rod length	$[m]$
L_p	Length if linear compressor piston	$[m]$
M	Moving mass	$[kg]$
MAE	Mean absolute error	$[-]$
Ma	Mach number	$[-]$
N	ISO viscosity grade number	$[-]$
N	Normal force	$[N]$
P	Pressure of gas	$[kPa]$
P_{high}	Higher pressure	$[kPa]$
P_{low}	Lower pressure	$[kPa]$
R	Gas constant	$[kJkg^{-1}K^{-1}]$
R_{amb}	Thermal resistance to the ambient	$[WK^{-1}]$

Re	Reynolds number	$[-]$
T	Temperature of gas	$[K]$
T_o	Oil temperature	$[K]$
T_{amb}	Ambient temperature	$[K]$
V	Volume of control volume	$[m^3]$
V_r	Volume ratio	$[-]$
W	Work	$[J]$
$\dot{E}_{d,leak}$	Exergy destroyed due to leakage	$[kW]$
$\dot{Q}$	Total heat transfer into the control volume	$[kW]$
$\dot{Q}_{cool}$	Refrigeration cooling load	$[W]$
$\dot{Q}_{motor}$	Heat transfer from the compressor motor	$[W]$
$\dot{W}$	Time rate of change of work	$[kW]$
$\dot{W}_{friction,o}$	Work dissipated by friction generated between oil and piston	$[W]$
$\dot{W}_{friction}$	Time rate of change of work due to friction	$[kW]$
$\dot{W}_{in,recip}$	Power input to a reciprocating compressor	$[W]$
$\dot{W}_{in}$	Power input to the compressor motor	$[W]$
$\dot{W}_{leak}$	Time rate of change of work due to leakage	$[kW]$
$\dot{m}$	Mass flow rate	$[kg\ s^{-1}]$
$\dot{x}_p$	Instantaneous piston velocity	$[ms^{1-}]$
$\mathbf{T}$	Torque to drive a reciprocating compressor crank	$[N-m]$
$\mathbf{T}_p$	Piston oscillation period	$[s]$
$\mathbf{V}$	Gas velocity	$[m\ s^{-1}]$
$\mathbf{V}_o$	Oil velocity	$[ms^{-1}]$
$\mathbf{f}$	Dry friction coefficient between piston and cylinder	$[-]$
$\mathbf{k}$	Thermal conductivity	$[Wm^{-1}K^{-1}]$
c_{eff}	Effective damping of piston	$[N\ s\ m^{-1}]$
$c_{friction,o}$	Effective damping of piston due to oil film friction	$[Nsm^{-1}]$
f_{res}	Resonant frequency of piston operation	$[Hz]$
h	Specific enthalpy	$[kJ\ kg^{-1}]$

k	Stiffness or spring rate	$[N\,m^{-1}]$
n	Polytropic exponent	$[-]$
q	Heat flux	$[Wm^{-2}]$
r_p	Radius of piston	$[m]$
s	Specific entropy	$[kJkg^{-1}K^{-1}]$
t	Time	$[sec]$
u	Internal energy	$[kJ\,kg^{-1}]$
v	Specific volume of gas	$[m^3\,kg^{-1}]$
x	Displacement	$[m]$
x_e	Equilibrium piston position	$[m]$
x_p	Instantaeous piston displacement	$[m]$
x_s	Total piston stroke	$[m]$
x_{dead}	Dead volume gap	$[m]$

GREEK LETTERS

α	Thermal diffusivity	$[m^2s^{-1}]$
β_1	Crank angle of reciprocating compressor	$[rad]$
β_2	Connecting rod angle of reciprocating compressor	$[rad]$
Δt_{shift}	Difference in period between applied and response frequencies	$[s]$
Δt_{step}	Time step	$[s]$
ΔT_{sub}	Subcooling temperature	$[^\circ C]$
ϵ	Eccentricity of spring force	$[m]$
η	Efficiency	$[-]$
γ	Heat capacity ratio	$[-]$
μ_o	Oil viscosity	$[Pa-s]$
ω_n	Natural frequency	$[rad\,s^{-1}]$
ω_d	Damped natural frequency	$[rad\,sec^{-1}]$
ω_{res}	Resonant frequency	$[rad\,s^{-1}]$
ρ	Density of gas	$[kg\,m^{-3}]$
ρ_o	Oil density	$[kgm^{-3}]$

τ_{zx}	Fluid shear stress	$[Pa]$
θ	Rotational angle of piston	$[rad]$
θ_{max}	Maximum piston rotation	$[rad]$
ε_{carnot}	Second law effectiveness of a refrigeration cycle	$[-]$
ζ	Damping factor	$[-]$

SUBSCRIPTS

amb	ambient
BDC	Bottom dead center
cv	control volume
dis	gas discharged from compression chamber
err	error
gas	quantity resulting from gas
in	into control volume
$leak$	leakage
max	maximum
$mech$	mechanical
o, is	overall isentropic
out	out of control volume
$port$	valve port opening
rms	root mean square
$shell$	compressor shell
$step$	Time step
suc	suction gas
TDC	Top dead center
tr	transitional
v	constant volume process
$valve$	reed valve
vol	volumetric

ABSTRACT

Bradshaw, Craig R. Ph.D., Purdue University, May 2012. A Miniature-Scale Linear Compressor for Electronics Cooling . Major Professors: Eckhard A. Groll and Suresh V. Garimella, School of Mechanical Engineering.

The power density of electronic devices has increased to a level that requires new technology for heat rejection. An alternative to conventional heat rejection techniques is refrigeration. However, current refrigeration technologies cannot meet the packaging constraints, or perform well at the operating conditions encountered by electronics cooling applications. The compressor is critical to the performance of a vapor compression refrigeration system. However, current compressor technology is not well suited for electronics cooling applications. This points to a need for new compressor technology developed specifically for electronics cooling. In this work, a linear compressor applied to electronics cooling is studied.

Linear compressors are appealing for vapor compression refrigeration applications in electronics cooling. A small number of moving components translates to lower frictional losses. Also the potential for this technology to be scaled to smaller physical sizes is better than for conventional compressors. To test the feasibility of a linear compressor applied to electronics cooling, a comprehensive model of a miniature-scale linear compressor for electronics cooling has been developed.

The model developed here incorporates all of the major components of the linear compressor including the dynamics associated with piston motion. The results of the compressor model were validated using experimental data from a prototype linear compressor custom-built for this study as well as a commercially available linear compress from a domestic refrigerator/freezer. The model results showed good agreement with the experimental results. The resonant frequency of the prototype compressor is predicted with 0.5% Mean Absolute Error (MAE) compared with experimental data.

The mass flow rate of the prototype linear compressor is predicted with 16.8% MAE compared with experimental data. The commercial linear compressor mass flow rate, power, and overall isentropic efficiency are predicted to within 4.9%, 6.1%, and 5.2%, MAE, respectively.

A sensitivity study using the model is also presented. This study examined the sensitivity to changes in compressor geometry. The study has identified the leakage gap and piston eccentricity as important parameters to consider when designing a linear compressor. The piston diameter and stroke is also shown to play an important role in overall performance. This study also shows that linear compressor technology scales well to smaller sizes. In addition, the unique ability to provide efficient capacity control is shown. This ability allows a single compressor to cool an electronic device over a wide variety of power input levels with only small changes in overall efficiency.

Using this knowledge, an updated prototype compressor design is presented. This updated design provides 200 W of cooling capacity in an overall cylindrical package size of 50.3 mm diameter and 102 mm length.

CHAPTER 1. INTRODUCTION

1.1 Miniature-Scale Refrigeration Systems

The power density of today's electronics devices is accelerating as a result of an increased number of transistors in silicon chips. The number of transistors on a chip was predicted to double every year by Moore (1965). This trend has slowed in recent years but remains increasing. The International Technology Roadmap for Semiconductors predicts that by 2022 the average power consumption for a typically stationary chip to be over 400W and over 800W by 2026 (2011 ed.).

Higher power density leads to higher heat dissipation rates and higher heat flux. The heat flux rejected from a single silicon chip in certain commercial electronics are approaching the limit of what traditional techniques can facilitate Krishnan et al. (2007). This shows a need for cooling solutions which can facilitate a higher heat flux on a silicon chip, such as vapor compression refrigeration.

Miniature refrigeration systems offer distinct advantages for use in electronics cooling relative to other technologies. Only vapor compression refrigeration and thermoelectric cooling devices (TECs) offers the ability to cool components below the ambient temperature. However, as Phelan et al. (2002) pointed out, the efficiency of TECs compared with vapor compression refrigeration is very poor. This makes vapor compression refrigeration an appealing option if the package size can remain small. In addition, the reliability and performance of semi-conductor devices is improved when operating at lower temperatures. Because vapor compression refrigeration utilizes two-phase heat transfer in the evaporator, it is possible to maintain spatially uniform chip temperatures. The major concerns involving vapor compression refrigeration systems are their cost and reliability, as well as miniaturization of the different components.

Trutassanawin et al. (2006) reviewed the available technologies for vapor compression refrigeration systems in electronics cooling. Several prototype systems were discussed but there appeared to be no commercially viable solutions which are complete miniature systems. Chiriac and Chiriac (2008) concurred with this assessment in a recent review. Other investigations into refrigeration technology have led to the development of system prototypes (Mongia et al., 2006) and system-level models such as ones by Heydari (2002), Possamai et al. (2008), and Nnanna (2006). However, it has been reported that the overall performance and size of these systems is still not at a level that is desired for desktop and portable electronic systems (Cremaschi et al., 2007). In particular, the compressor is a critical component which can greatly affect the overall size and performance of the refrigeration system, as shown by Trutassanawin et al. (2006) and Davies et al. (2010).

Particular emphasis has been placed on the design and performance of cold-plate evaporators for electronics cooling. Lee and Mudawar (2009) evaluated the two-phase flow performance of a microchannel evaporator for an electronics cooling application. This study found that the performance of two phase flow cold plates depends heavily on the flow regime experienced within cold plate channels. Bertsch et al. (2008a) reviewed the available literature and developed general trends in modeling and performance of microchannel evaporators. Bertsch et al. (2008b) then developed a correlation for flow boiling of refrigerants in microchannel evaporators using data from his own study and the literature. Harirchian and Garimella (2010) developed a flow regime map for flow boiling in microchannels followed by flow-regime based models to predict performance in microchannels (Harirchian and Garimella, 2012). While challenges in practical implementation of microchannel heat sinks still exists the driving mechanisms for performance are becoming well understood.

In an electronics cooling application, an atypical challenge for refrigeration systems is the relatively small temperature lift for cycle operation. The temperature lift is limited on the high-side by the environmental conditions and on the low-side by the desire to eliminate possible water vapor condensation from the ambient within the

electronic package. Typically, this leads to a system with a small pressure ratio. This is shown in Figure 1.1, which displays a typical vapor compression refrigeration cycle for electronics cooling using R-134a as the working fluid. The small pressure ratio leads to poor performance for currently available small-scale compressors, which are designed for refrigerator applications with a significantly larger pressure ratio. Some past studies using currently available technology have operated the compressors at a larger pressure ratio, and instead, developed solutions to handle moisture condensation by operating the evaporator and chip assembly in heavily insulated volumes (Heydari, 2002). The disadvantage of this option is a further increase in the cost of the cooling system.

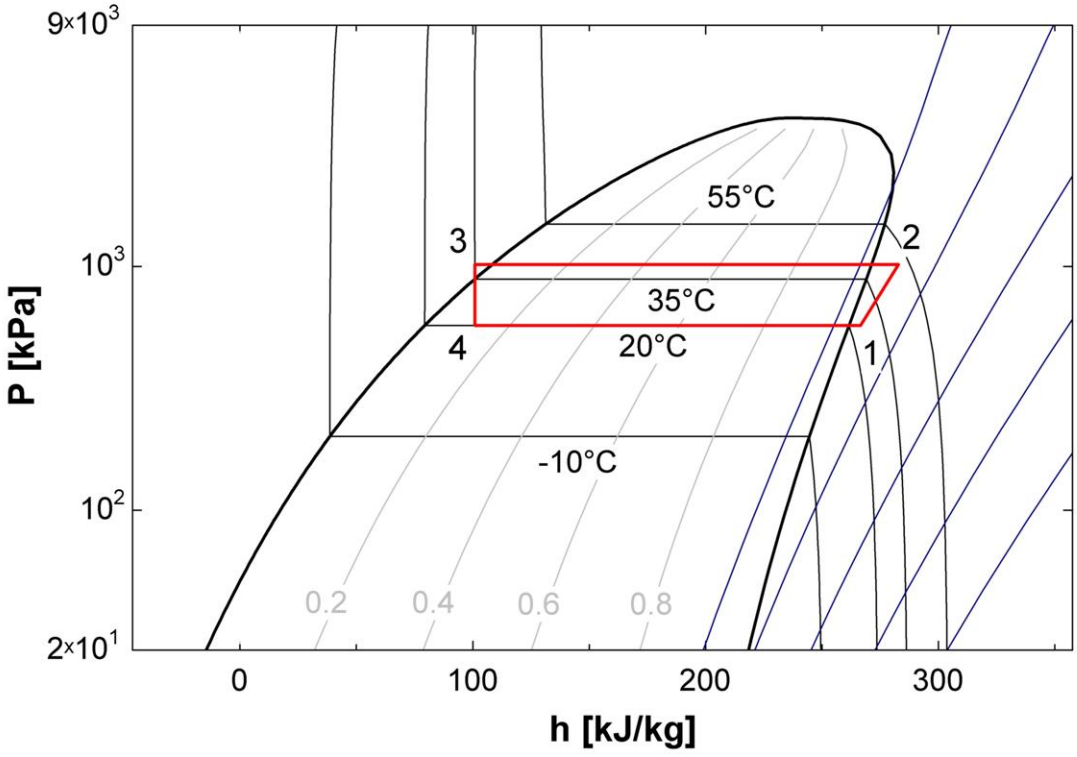


Figure 1.1. Pressure enthalpy diagram of typical R134a miniature-scale refrigeration cycle for electronics cooling.

In addition to condensation the lower pressure ratio experienced also creates a lack of enthalpy increase. This can be seen in Figure 1.1, as the pressure ratio of the refrigeration cycle decreases the change in enthalpy decreases. The change of enthalpy represents the useful energy applied to the system, or working fluid, by the compressor. So at low pressure ratios there is little enthalpy imparted to the fluid but will still require the compressor to utilize power to drive the compressor mechanism (*i.e.* overcome friction, inertia, etc.). At the lower pressure ratios this amount of energy will dominate the enthalpy imparted to the fluid and efficiency will suffer. Therefore, it is important to not only select a fluid that will minimize this burden but also a simple compressor mechanism to minimize friction losses.

Marcinichen et al. (2010) presented an evaluation of working fluids for a vapor compression cycle applied to an electronics cooling application. This work concluded that either R-134a or R-254fa would be suitable candidates for the working fluid in a vapor compression refrigeration system applied to electronics cooling. Sathe (2008) came to the same conclusion. As it is more commonly used, R-134a was selected as the working fluid of choice for this study as a result of these evaluations.

1.2 Linear Compressors

A linear compressor is appealing for electronics cooling applications because it offers several potential advantages over traditional compressor technology. A linear compressor is a positive displacement compressor, similar to a reciprocating compressor. However, a linear compressor does not have a crank mechanism to drive the piston. Instead, the piston is driven directly by a linear motor, as seen in Figure 1.2. The omission of the crank mechanism significantly reduces the frictional losses associated with conventional reciprocating compressors. It also allows oil-free operation which is a significant advantage with respect to the heat transfer performance of the condenser and evaporator in the system.

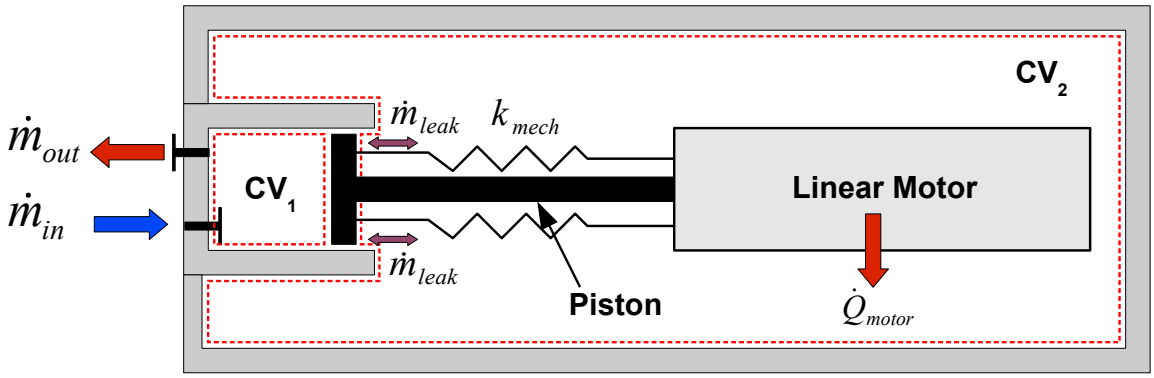


Figure 1.2. Representation of the compression process in a linear compressor.

In addition, a linear compressor is designed as a resonantly operated device. This means that mechanical springs are used to tune the device to operate at a resonant frequency. Operation at a resonant frequency allows a desirable reduction in size of the motor.

Early investigations of a linear compressor were conducted by Cadman and Cohen (1969a, b) for traditional refrigeration systems. Cadman and Cohen observed that the free piston operation of this device created some peculiar effects such as piston drift, which posed a challenge to modeling efforts and limited the practical application of such devices. Pollak et al. (1979) investigated one-dimensional, nonlinear dynamics of the piston and electrical systems and confirmed such confounding effects.

Van der Walt and Unger (1994) assessed the state of the art in linear compressor technology. Unger and Novotny (2002) ,Unger (1998), and Koh et al. (2002) developed several linear compressor prototypes and performed a feasibility study which showed the potential for linear compressors in cryogenic and miniature systems. There have also been studies on the unique differences in the performance of linear compressors compared to conventional reciprocating compressors (Lee et al., 2001, 2004). Recently, Possamai et al. (2008) developed a prototype refrigeration system for a laptop cooling application which utilized a linear compressor. Another recent study on linear compressors investigated the sensitivity of the device to changes in operating

frequency (Kim et al., 2009). This study utilized a single degree of freedom vibration model coupled to an electrodynamic motor model as was done by Pollak et al. (1979).

Minas (1994) also modeled the one-dimensional, nonlinear dynamics of the piston as well as nonlinear effects within a linear motor. The author also presented a relationship between the stroke and resonant frequency of a linear compressor, which provides a unique control challenge that was confirmed by Xia and Chen (2010) and Choe and Kim (2002). In addition to changes in resonant frequency, control of the linear compressor stroke has proven challenging (Cadman and Cohen, 1969a, b; Pollak et al., 1979). Additional attempts to reduce the amount of stroke control needed for a linear compressor have been attempted by Park et al. (2004) as a result of these challenges. Park et al. (2004) concluded that a robust design may not be enough to control the stroke of a compressor effectively.

1.3 Comprehensive Modeling of Hermetic Compressors

Vapor compressors are designed to be some of the most robust positive displacement machines in service. Most compressors have a service life of 10 years, or beyond, with no regular maintenance schedule. This level of robustness requires an intimate knowledge of the working behavior of the compressor. Modeling of hermetic compressors is one mechanism for obtaining this knowledge. A model can be as simple as an efficiency estimation from fundamental thermodynamics (Moran and Shapiro, 2004). Navarro et al. (2007) developed a more advanced compressor model which described the major phenomena within a compressor in terms of its performance metrics, the volumetric and isentropic efficiencies. This model presented a very good fit to experimental data but relied heavily on experimental curve fitting to a specific compressor. A similar modeling approach with similar results was shown by Kim and Bullard (2002). Others have attempted somewhat more detailed and general models, such as Ooi and Wong (1997) who developed an energy and mass balance for a rolling piston compressor using the unsteady first law of thermodynamics. This type of model

utilizes information about the internal workings of the compressor and requires inputs from sub-models for mass flow, heat transfer, geometry, and friction. While this approach is more generic, Ooi and Wong (1997) relied heavily on experimental correlations for calculations within the sub-models.

A recent approach to compressor modeling, called the comprehensive approach, has provided a more complete analysis. The comprehensive modeling approach relies on the unsteady mass and energy balance of a control volume, similar to that of Ooi and Wong (1997). However, the sub-models are incorporated as first-principles based models. Chen et al. (2002b) and Chen et al. (2002a) applied a comprehensive approach to the modeling of scroll compressors. In addition, the entire compressor body is included in the analysis as a collection of lumped masses. This allows a comprehensive mass and energy balance to be imposed on the entire compressor. This method yields insight into the operation and efficiency of the compressor mechanism but also provide information about the shell and oil sump conditions. Kim and Groll (2007) utilized the comprehensive approach for the modeling of a novel bowtie compressor. This type of compressor is based on the Beard-Pennock mechanism, which provides capacity control in a fixed volume device. Jovane (2007) applied the comprehensive approach to a novel rotary compressor, called a z-compressor. Finally, more recently, Mathison et al. (2008) developed a comprehensive model for a two-state rotary compressor, which provided insight into the optimum intermediate pressure for the best cycle and compressor efficiency. This work was continued by applying the approach to a rotary compressor with vapor injection as well as a novel rotary spool compressor (Mathison, 2011). Finally, Bell (2011) presented the most recent addition to the applications of comprehensive modeling by examining the impact of oil flooding on scroll compressors. The comprehensive model provided a deeper insight into the unique internal performance of a scroll compressor flooded with oil. This allowed for a numerical optimization of scroll wraps for use in oil flooding, which may have been costly and time consuming to obtain experimentally.

Past models of linear compressors have often employed linearizing assumptions, were restricted to a single degree of freedom in their analysis, or neglected to include the other compressor components such as valves, leakage losses, or heat transfer. Such analyses have either used a greatly simplified representation of device operation or have focused on specific components of the overall system. The present work develops a comprehensive model of a linear compressor which incorporates all of the major dynamics and nonlinearities as well as a second degree of freedom in the piston motion.

1.4 Objectives and Overview

The objective of this work is to determine the feasibility of miniaturizing linear compressor technology and using it in a vapor compression cycle for electronics cooling applications. In addition, the application of the comprehensive compressor modeling approach to the linear compressor is to be explored. To accomplish this objective a combination of modeling and experimental efforts are used.

A prototype linear compressor is developed specifically for this work to be used for model validation and is shown in Figure 1.3. The linear motor and compression springs were purchased components while the remaining components were designed and built for this work. Details on the prototype compressor design can be found in Appendix A. To test this prototype a miniature-scale compressor load stand is also developed. For additional model validation, a commercially available linear compressor was be used.

The linear compressor is modeled using a comprehensive simulation approach. This approach is centered around mass and energy balance equations for a control volume. As inputs to these equations, several sub-models are incorporated to account for the two degree of freedom piston dynamics, friction, leakage, valve dynamics, and heat transfer. Using the developed and validated model, an array of sensitivity studies

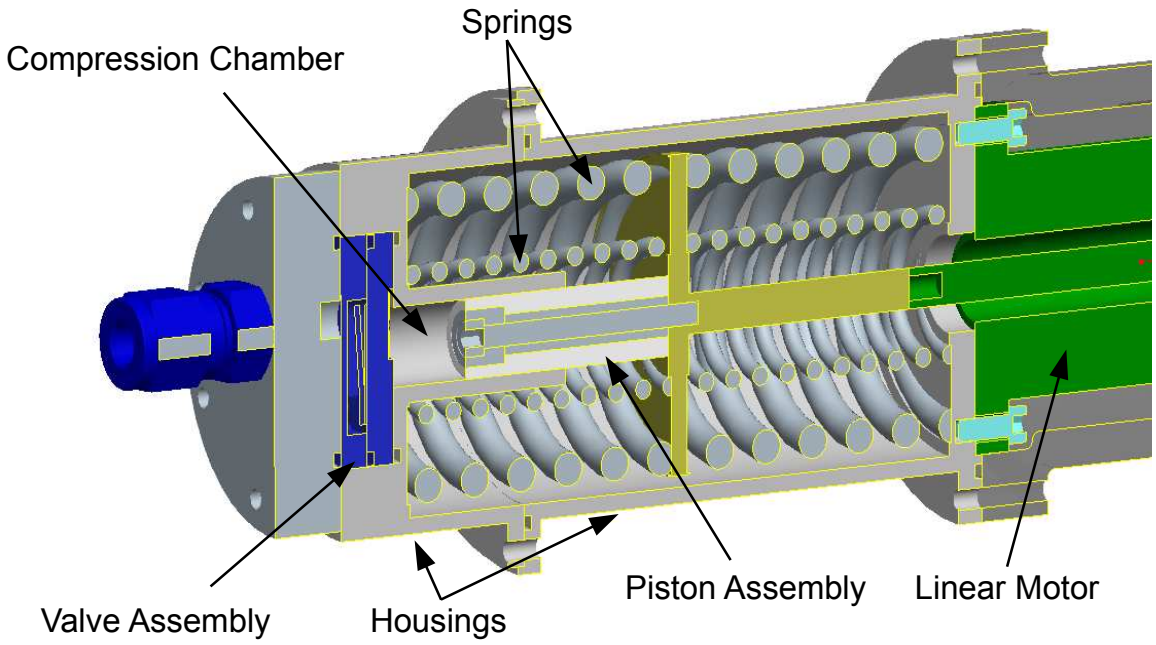


Figure 1.3. Sectional view of the prototype linear compressor built for this work.

determine the design parameters that are the most critical for linear compressor design.

Using the validated and exercised model as a tool the overall feasibility of linear compressor technology can be explored. By understanding the mechanisms that determine operation and superior performance, an assessment is made regarding the miniaturization and application of linear compressors to electronics cooling. An improved linear compressor design is presented which displays the miniaturization capability of linear compressors. For additional model validation a LG linear compressor is tested and modeled. This study examines a linear compressor from a different application serves as an additional verification of the modeling approach.

CHAPTER 2. LINEAR COMPRESSOR MODEL

A comprehensive simulation model for a linear compressor is presented in this Chapter. The comprehensive compressor model consists of a solution to two compression process equations that provide the temperature and density and thus, fix the state within each control volume. These equations require inputs from five sub-models representing the valve flows, leakage flows, motor losses, heat transfer from the cylinder, and piston dynamics. The compression process is discretized and an initial guess of temperature and density is made within each control volume. The compression process equations are then used to step through a compression process using the numerical techniques described in Section 2.8.

2.1 Compression Process Equations

The compression process is modeled using mass and energy conservation equations over a control volume. In order to determine the state of the working fluid in the compression chamber at any point during the compression process, it is necessary to determine two independent fluid properties to fix its state. In this analysis, the compressor is split into two control volumes as illustrated in Chapter 1, Figure 1.2. The first control volume is the compression chamber for the compressor, while the second control volume consists of the remaining volume within the compressor. The compression process equations are solved for each control volume at a particular time step and coupled through the leakage, vibration, and heat transfer sub-models.

The fluid is assumed to be superheated throughout the entire compression process. The equation of state for the chosen working fluid, R-134a, is written in terms of temperature and density (Tillner-Roth and Baehr, 1994); the compression process

equations are therefore cast in terms of these two variables. The conservation of energy for a general control volume can be written as follows:

$$\frac{dE_{cv}}{dt} = \dot{Q} + \dot{W} + \sum_{in} \dot{m}_{in} h_{in} - \sum_{out} \dot{m}_{out} h_{out}. \tag{2.1}$$

Assuming that the fluid in the control volume is well mixed, h_{out} becomes h_{cv} and will be denoted as such in the remainder of this work. With a further assumption that the changes in kinetic and potential energies are negligible, the left-hand side of Equation (2.1) can be expanded in terms of changes in internal energy and mass:

$$\frac{dE_{cv}}{dt} = \frac{d(u_{cv} m_{cv})}{dt} = m_{cv} \frac{du_{cv}}{dt} + u_{cv} \frac{dm_{cv}}{dt}. \tag{2.2}$$

The following thermodynamic relations also hold for the gas in the cylinder:

$$du = C_v dT + \left[T \left(\frac{\partial P}{\partial T} \right)_v - P \right] dv \tag{2.3}$$

$$u = h - Pv. \tag{2.4}$$

Using the relationship in Equations (2.3) and (2.4) the left-hand side of Equation (2.1) can be written as follows:

$$\frac{dE_{cv}}{dt} = m_{cv} \left(C_v \frac{dT}{dt} + \left[T \left(\frac{\partial P}{\partial T} \right)_v - P \right] \frac{dv_{cv}}{dt} \right) + (h - Pv) \frac{dm_{cv}}{dt}. \tag{2.5}$$

The work term from Equation (2.1) can be re-written as boundary work done by the piston:

$$\dot{W} = -P \frac{dV}{dt}. \tag{2.6}$$

In order to eliminate pressure from Equations (2.6) and (2.5), the pressure in the cylinder is expressed as follows:

$$P = T\left(\frac{\partial P}{\partial T}\right)_v. \tag{2.7}$$

By combining Equations (2.5) to (2.7):

$$m_{cv}C_v\frac{dT}{dt} + T\left(\frac{\partial P}{\partial T}\right)_v\left[\frac{dV}{dt} - \frac{1}{\rho}\frac{dm_{cv}}{dt}\right] + h_{cv}\frac{dm_{cv}}{dt} = \dot{Q} + \sum \dot{m}h_{in} - \sum \dot{m}h_{cv} \tag{2.8}$$

in which only temperature is the independent variable. The other independent fluid property is density, which is computed as:

$$\frac{dm_{cv}}{dt} = \frac{d(\rho V)}{dt} = \rho\frac{dV}{dt} + V\frac{d\rho}{dt}. \tag{2.9}$$

The change in mass in the cylinder can then be written as:

$$\frac{dm_{cv}}{dt} = \frac{dm_{in}}{dt} + \frac{dm_{leak,in}}{dt} - \frac{dm_{out}}{dt} - \frac{dm_{leak,out}}{dt}. \tag{2.10}$$

Equations (2.8) - (2.10) can now be solved simultaneously for the temperature and density in the two control volumes. Due to the nonlinear nature of these equations, a numerical solution approach is adopted. A number of inputs to this solution must still be determined from various sub-models as will be discussed in the following sections.

2.2 Valve Model

Reed valves are used in the present work as is typical for piston-cylinder compressors. These valves are essentially cantilevered beams which are displaced when a differential pressure is imposed across them. To ensure that pumping, as opposed to back flow, of the gas occurs, two valves are used with limited ranges of operation. The valve body is constructed to only allow valve motion in the direction of desired flow for each valve; this ensures that pumping occurs and can be seen in Figure 2.1.

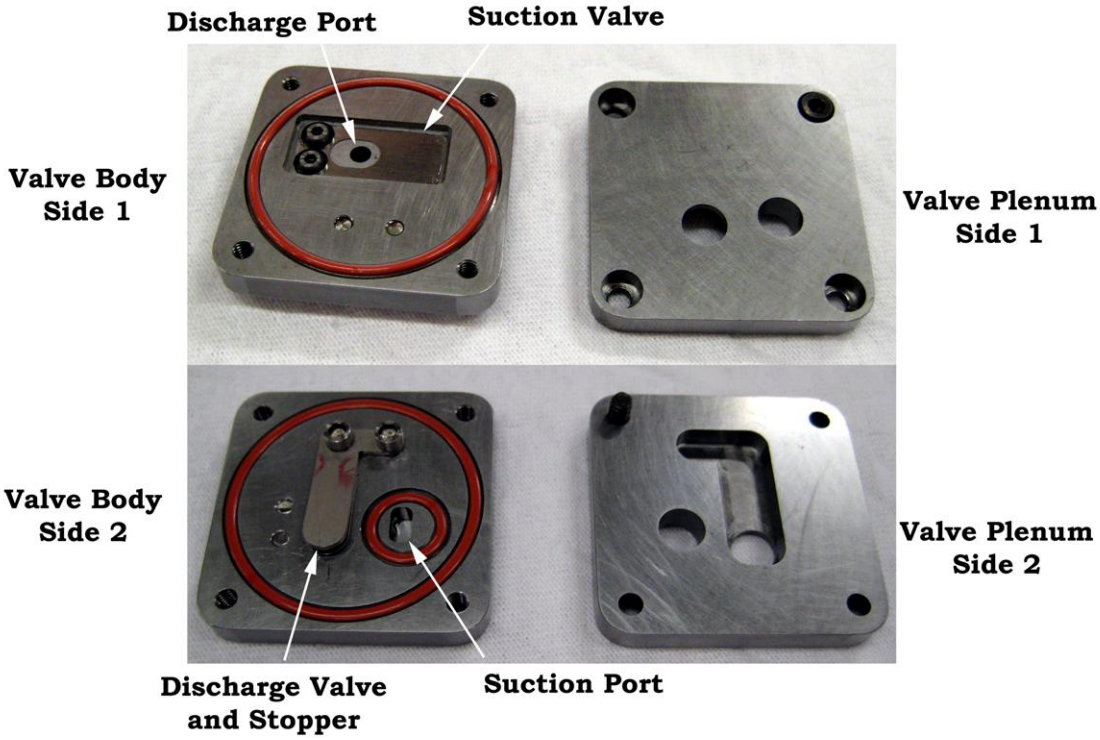


Figure 2.1. Photos of reed valve assembly used in compressor prototype.

Valve analysis is split into two modes of operation depending on the location of the valve reed, similar to the analysis of Kim and Groll (2007). Early in its deflection the stagnation pressure driving the valve reed is assumed equal to the high side pressure. As the valve opens further, this assumption becomes less valid, as the movement of the valve is now dominated by the movement of fluid past the valve and the stagnation pressure becomes equal to the low-side pressure. These two modes of operation are referred to as the pressure-dominant and mass-flux-dominant modes, respectively, with free-body diagrams as shown in Figure 2.2. The port area is a fixed quantity and represents the opening that fluid travels through in the valve body. The valve area is an effective area, taken as a circular area based on the width of each valve reed.

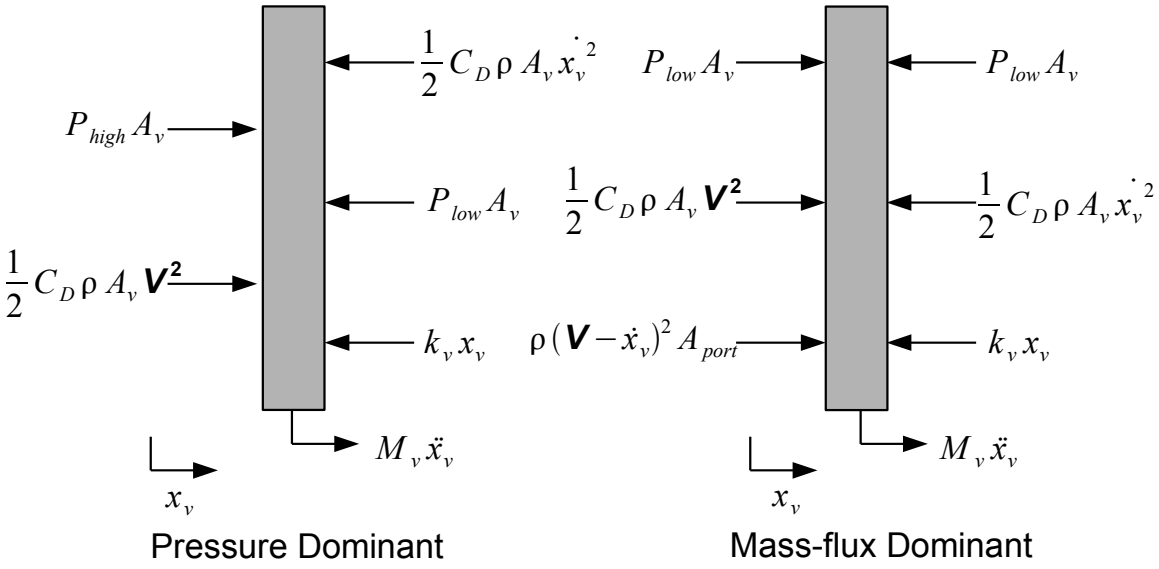


Figure 2.2. Free body diagrams for the valve reed.

For each mode of operation an equation can be written based on the free-body diagrams shown in Figure 2.2. Summing the forces in the direction of displacement, the equation of motion for the pressure-dominated and mass flux-dominated modes may be expressed as:

$$\begin{aligned} M_{valve}\ddot{x}_{valve} + \frac{1}{2}C_D\rho A_{valve}\dot{x}_{valve}^2 + k_{valve}x_{valve} = \\ (P_{high} - P_{low})A_{valve} + \frac{1}{2}C_D\rho \mathbf{V}^2 A_{valve} \end{aligned} \tag{2.11}$$

$$\begin{aligned} M_{valve}\ddot{x}_{valve} + \left(\frac{1}{2}C_D\rho A_{port} + \rho A_{valve}\right)\dot{x}_{valve}^2 + k_{valve}x_{valve} = \\ \frac{1}{2}C_D\rho \mathbf{V}^2 A_{valve} + \rho \mathbf{V} A_{port}. \end{aligned} \tag{2.12}$$

These second-order, nonlinear equations can be solved for the position of the reeds at each time step throughout the compression process. The valve geometry, shown in Figure 2.1, can be used to determine the required valve and port areas and the density

is calculated from the compression process equations. The gas velocity is calculated assuming an isentropic, compressible flow across the valve. The mass and stiffness of the reeds can be calculated by assuming that each reed behaves as an Euler beam. The stiffness is calculated by dividing an arbitrary applied force by the deflection it produces. The effective mass is then found by dividing the total valve mass by a factor of three, as in the case of the effective mass of a spring (Rao, 2004).

At a certain valve lift, which is the ratio of the valve port opening diameter to the reed valve diameter, these two analyses provide the same mass flow rate. This valve lift is known as the transitional valve lift because below this value the pressure-dominated model is used and above the mass-flux dominated model is used. When the valve lift is equal to the transitional lift value the pressure-dominant model is used. The transitional valve lift is expressed as:

$$x_{tr} = \frac{D_{port}^2}{4D_{valve}}. \tag{2.13}$$

2.3 Leakage Model

The leakage model only focuses on leakage past the piston. The only other leakage paths, those past the reed valves, are ignored since they are negligibly small compared to the leakage past the piston (Kim and Groll, 2007). It is assumed that the gas velocity is significantly higher than the piston velocity, in general, so the influence of the piston velocity can be ignored. Thus, a compressible, isentropic, flow model is utilized to model the leakage past the piston. To calculate the leakage gas velocity the Mach number of the flow is calculated using the following relationship,

$$\frac{P_{low}}{P_{high}} = \left(1 + \frac{\gamma - 1}{2} Ma^2\right)^{\frac{-\gamma}{\gamma - 1}} \tag{2.14}$$

where the ratio of pressures is found by the difference in pressures between CV_1 and CV_2 and the gas velocity comes from the definition of the Mach number.

$$Ma = \frac{\bar{\mathbf{V}}}{\sqrt{\gamma RT}}. \tag{2.15}$$

2.4 Heat Transfer Model

The instantaneous heat transfer from each of the control volumes is calculated using the empirical approach of Fagotti and Prata (1998).

$$Nu = \frac{q_{cv}D}{\mathbf{k}[T_{cv} - T_{shell}]} = 0.28Re^{0.65} + 0.25L\frac{T_{shell}}{T_{cv} - T_{shell}} \tag{2.16}$$

where:

$$L = \frac{\gamma - 1}{V}\frac{dV}{dt}\sqrt{\frac{D^3}{\alpha \dot{x}_p}}. \tag{2.17}$$

By integrating these instantaneous heat transfer rates over the entire cycle, the total heat transfer from each of the two control volumes is calculated for use in the overall energy balance.

$$\dot{Q} = \frac{1}{t_{step}}\int_0^t \dot{Q}dt. \tag{2.18}$$

2.5 Vibration Model

A linear compressor is a free-piston device for which the stroke is not fixed by a crank mechanism but is instead determined by chosen geometry, the linear motor, and the mechanical springs used. The piston dynamics are modeled using a free body diagram of the piston assembly as shown in Figure 2.3.

Both the desired linear motion of the piston, as well as its undesirable rotation due to eccentricity in the mechanical springs are considered. Any eccentricity causes a perpendicular reactionary load from the mechanical spring as it is compressed, since

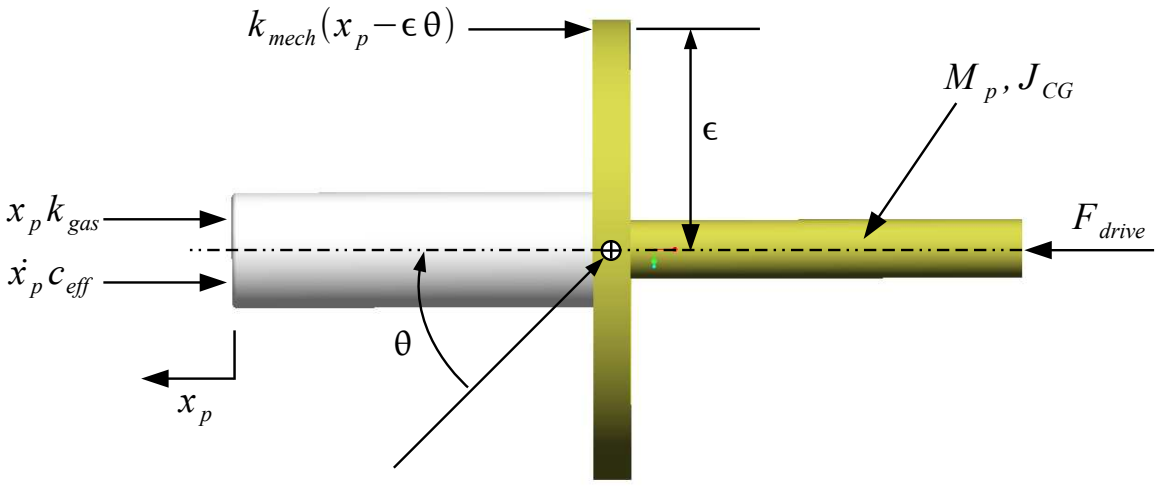


Figure 2.3. Free body diagram of linear compressor piston assembly.

the spring cannot react to a load along its entire circumference. Thus, the effective load acts at a point offset from the center of mass of the piston. In a traditional compression spring as used in the prototype fabricated for this work, this point is assumed to be along the circumference of the spring where the edge of the spring contacts the piston assembly. This point of load is depicted in Figure 2.4.

The linear motion of the piston as well as its rotation, are modeled as a two-degree of freedom vibration system. By assuming that the piston is a rigid body and the motions in the y and z directions as well as the rotation are small, the following equations describe the motion of the piston:

$$M\ddot{x_p} + c_{eff}\dot{x_p} + (k_{gas} + k_{mech})\, x_p = k_{mech}\epsilon\theta + F_{drive} \tag{2.19}$$

$$J_{CG}\ddot{\theta} + k_{mech}\epsilon^2\theta = k_{mech}\epsilon x_p. \tag{2.20}$$

In addition, it is assumed that the driving force and the reactionary forces from the compression chamber are applied directly through the center of mass of the piston.

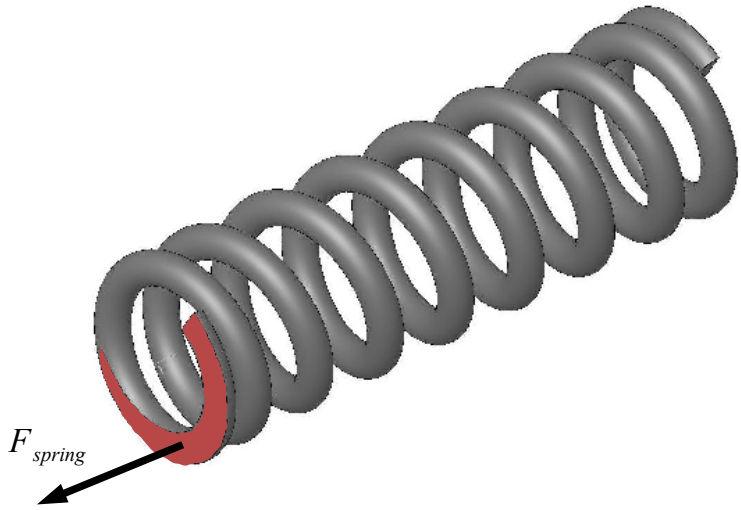


Figure 2.4. Representation of the applied load due to spring eccentricities on a typical ground compression spring.

The driving force, F_{drive}, is the sinusoidal force applied from the motor. The power transmitted to the piston is calculated by assuming an efficiency for the motor. The stiffness associated with the gas, k_{gas}, is determined by estimating the force generated by the gas at each time step over an entire compression cycle. This gas force is then divided by the instantaneous piston displacement of the compressor which yields an effective spring rate of the gas in the compression chamber:

$$k_{gas} = \frac{d(P_{cv,1} - P_{cv,2})A_p}{dx_p}. \tag{2.21}$$

This is similar to the approach adopted by Cadman and Cohen (1969a); however, in the current work k_{gas} is continuously updated as the pressures and stroke change.

The effective damping term is made up of two components: a frictional term and the boundary work performed on the gas (Pollak et al., 1979). Both components are computed from an integration of the energy dissipated over a cycle. The effective damping term is then calculated from summing the contributions described above:

$$c_{eff} = \frac{E_{cycle}}{\omega_d x_p \pi} = \frac{W_{gas}}{\omega_d x_p \pi} + \frac{W_{friction}}{\omega_d x_p \pi}. \tag{2.22}$$

The work done on the gas is calculated by integrating the boundary work expression over the entire compression process (*i.e.* integrating pressure over volume):

$$W_{gas} = -\int P_{cv,1} dV. \tag{2.23}$$

This expression is integrated numerically throughout the compression process. The work done by friction is calculated from (Rao, 2004), and integrated over the entire compression process:

$$W_{friction} = \int_0^{x_s} 4\mathbf{f} N dx \tag{2.24}$$

where the normal force, N, is the reaction force on the piston caused by the cylinder, given by:

$$N = \left(\frac{1}{L_p}\right)\left(k_{mech}\epsilon(x_p - \epsilon\theta)\right). \tag{2.25}$$

2.6 Overall Energy Balance

To ensure that all of the compressor components satisfy an energy balance, a thermal network is constructed to account for heat transfer from the compression chamber (Chen et al., 2002a; Kim and Groll, 2007; Mathison et al., 2008; Jovane, 2007). The energy balance for the compressor assumes that the heat transfer between the two control volumes is negligible and that the heat only flows to the compressor shell. A lumped-mass thermal network can then be constructed consisting of a single lumped mass to represent the compressor shell with two heat inputs and one heat output to the ambient. The heat output path is calculated using a forced convection correlation for flow over a cylinder to determine the thermal resistance between the compressor shell and the ambient (Hilpert, 1933). The thermal network elements are also shown in Figure 2.5. This network adds the following relation which is solved simultaneously with the compression process equations:

$$\frac{T_{shell} - T_{amb}}{R_{amb}} = \dot{Q}_{cv1} + \dot{Q}_{cv2}. \tag{2.26}$$

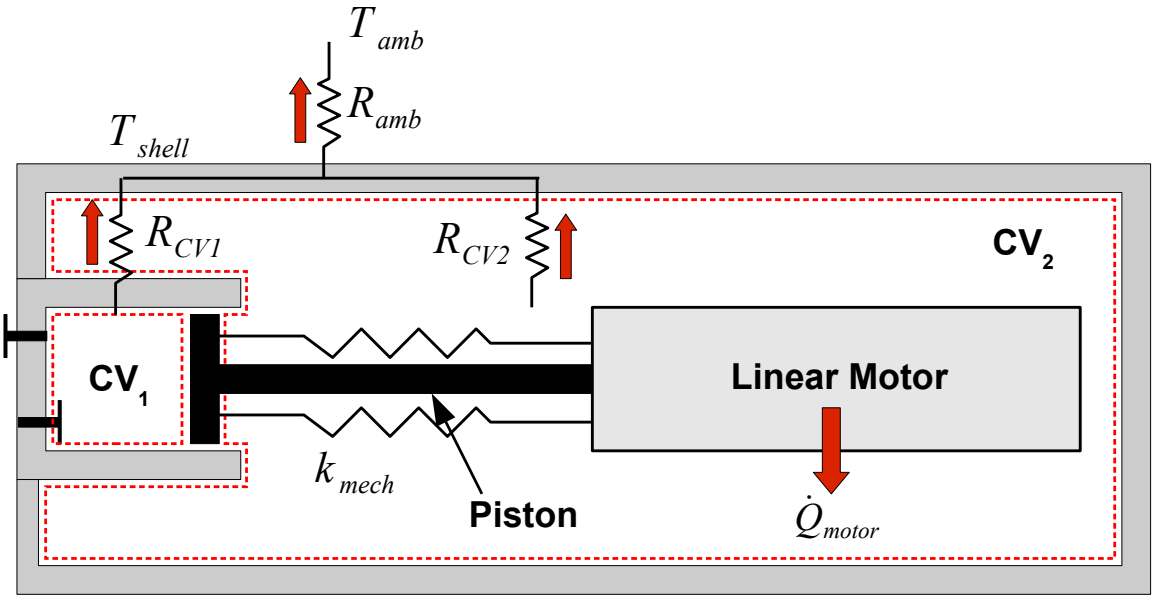


Figure 2.5. Schematic of linear compressor with displayed paths of heat movement.

2.7 Solution Approach

The model developed above for the linear compressor consists of two non-linear first-order differential equations for the compression process and one non-linear equation from the energy balance. An explicit closed-form solution is not available, and the equations are instead solved numerically. The compression process is discretized into small time steps and each equation is solved using the numerical techniques described in Section 2.8.

The order in which the model steps through the series of equations is shown in Figure 2.6. The key input parameters to the model are the compressor inlet pressure and temperature, the desired discharge pressure, and ambient temperature. For each time step, the model sequentially steps through each sub-model and calculates the solution inputs for the next time step. Once each sub-model has been called the

overall compression process solver is called which solves Equations (2.8) to (2.10) and calculates the internal state in the compressor. When the model has iterated through an entire compression cycle Equation (2.26) is solved for T_{shell}. The methods that are used to solve these equations is detailed in the following section.

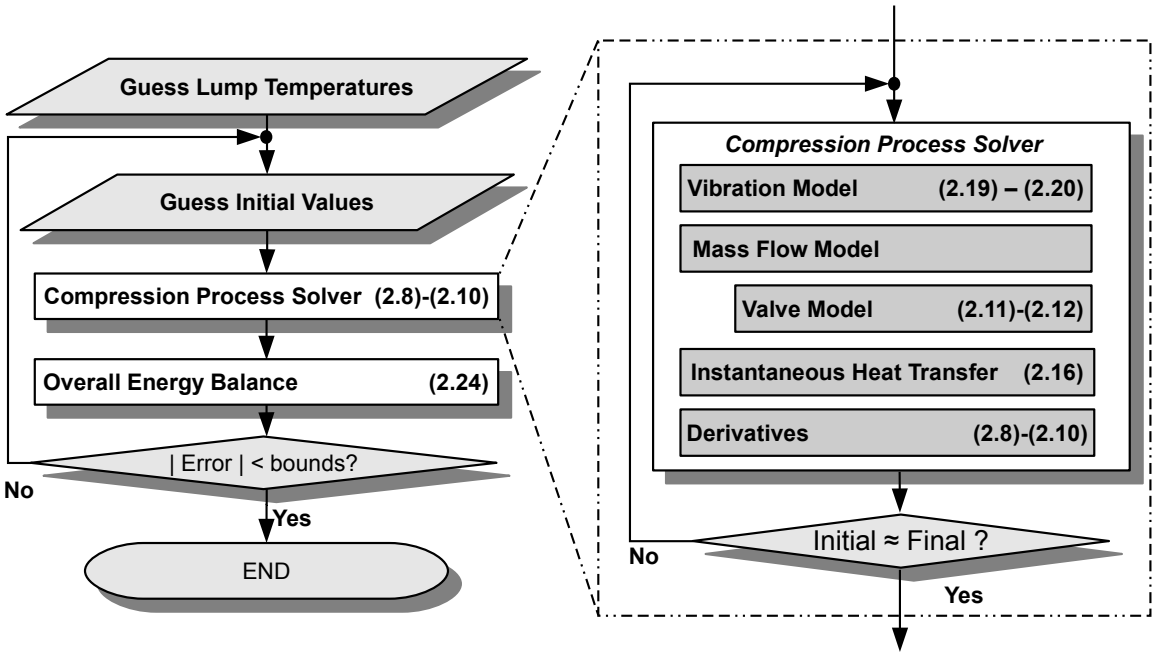


Figure 2.6. Solution flowchart of linear compressor model (equations solved at each step given in blocks).

2.8 Numerical Considerations

2.8.1 Time Shift

The dynamic behavior of the piston presented some unique challenges for the numerical simulation. The piston is excited with a frequency that is input to the model. However, starting from rest, the piston will respond with a slightly different frequency that relates to the transient response of the system. Numerically this is problematic because if the model steps through a fixed number of points using a timestep based on the input frequency, the piston will require a different number

of iterations to complete a full cycle. As an example, Figure 2.7 shows a typical start-up response of the linear compressor piston and the applied sinusoidal force as a function of time. This shows that the piston cycle takes slightly longer time to finish one cycle (oscillation period) compared with the driving force function. This can generate errors in the calculation of power, massflow, and other performance metrics if the compressor piston has not completed an entire cycle. Thus, to remedy this, the response frequency of the piston is allowed to float (via allowing additional time steps) and the system model does not terminate until the piston has completed a full cycle. Then, to ensure that the phase shift from cycle to cycle remains consistent, the calculated phase shift, Δt_{shift}, is calculated and used as an input to the driving force in the next cycle iteration:

$$F_{drive} = F_{drive,max} \cos(\omega_d(t - \Delta t_{shift})). \tag{2.27}$$

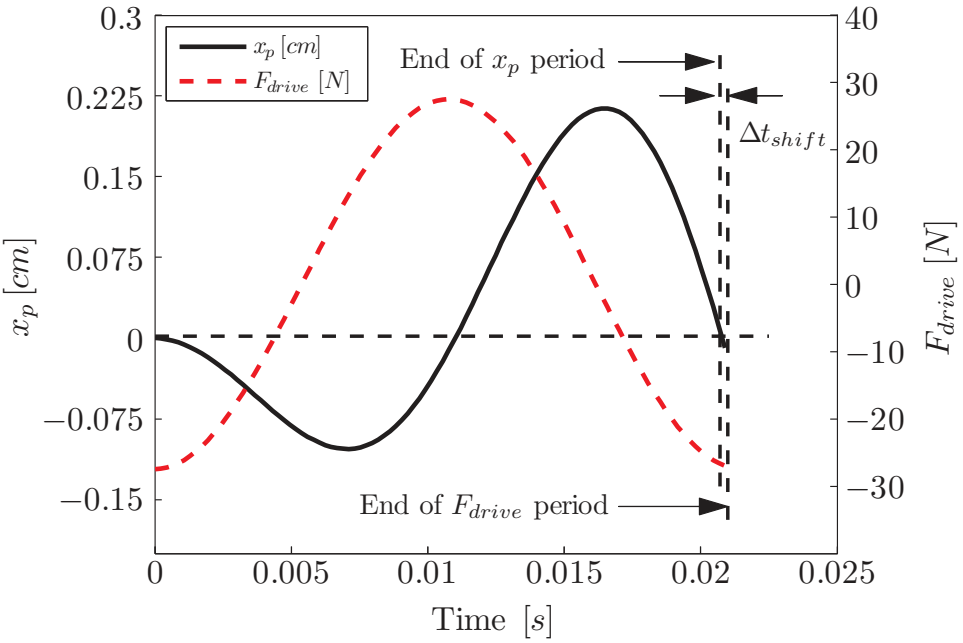


Figure 2.7. A typical first piston cycle showing the response of the compressor piston and the applied motor force highlighting the difference in response period between the two.

2.8.2 Numerical Methods and Stability

To solve Equations (2.8) through (2.10) a numerical approach is required. For this work, a combination of Euler and 4th-order Runge-Kutta methods is utilized. The peculiarities of the piston motion noted in the previous section require that the end of the piston cycle be detected. However, since the data is collected at discrete points the exact time that the piston finishes a cycle (i.e. when x_p crosses zero twice) encounters some error which is proportional to the step size:

$$\Delta t_{shift,err} = f(\Delta t_{step}). \tag{2.28}$$

To maintain stability, this error should be minimized which requires a relatively low step size (*i.e.* high number of steps) for good resolution toward the end of the cycle. This requirement eliminated the possibility of using an adaptive method, as such a method could possibly choose a large step size toward the end of the cycle which would cause large error in Δt_{shift}. Large error in the time shift leads to model instability as the phase shift from cycle to cycle becomes inaccurate.

Since the system stability scales linearly with step size, utilizing the more accurate 4th-order Runge-Kutta method was not advantageous as it operated 4 times slower than Euler's method for a given step size. Therefore, Euler's method was used to solve the compression process equations with a 4th order Runge-Kutta method used to solve the vibration and valve sub-models with a minimum number of steps of 8000. When the changes in temperature and density, calculated by the compression process equations, in each of the two control volumes, as well as the change in T_{shell}, is less than 0.01%, a converged solution is available. The initial conditions in both control volumes were set to the inlet conditions for each operating condition that was tested.

CHAPTER 3. CHARACTERIZATION OF LINEAR COMPRESSOR BEHAVIOR

This chapter explores the unique behavior of a linear compressor. A characterization of the compressor stiffness and damping as well a study of piston stability is presented. These investigations motivate the need for additional control in a linear compressor. A method for calculating the resonant frequency of a linear compressor is developed. In addition, a numerical control algorithm is also provided that ensures compressor operation at the desired stroke. These additions add robustness and numerical stability to the comprehensive model developed in Chapter 2.

3.1 Linear Compressor Behavior

The typical response a linear compressor piston as a function of time is shown in Figure 3.1. The piston position oscillates about an equilibrium position x_e in a sinusoidal manner between the Top Dead Center (TDC) and Bottom Dead Center (BDC) positions. The piston rotation is a result of the moment caused by spring eccentricities. The piston rotation is restricted by the piston making contact with the cylinder wall. Assuming the piston behaves as a rigid body, the maximum rotation is a function of the leakage gap between the piston and cylinder g, and can be calculated as follows:

$$\theta_{max} = tan^{-1}\left(\frac{g}{L_p}\right). \tag{3.1}$$

Since a linear compressor has a fixed piston area the compression chamber volume is proportional to the piston position. Referencing the dimensions given in Figure 3.2 the linear compressor cylinder volume can be calculated as:

$$V = \left(x_p(t) + \frac{x_s}{2}\right) A_p + x_{dead} A_p. \tag{3.2}$$

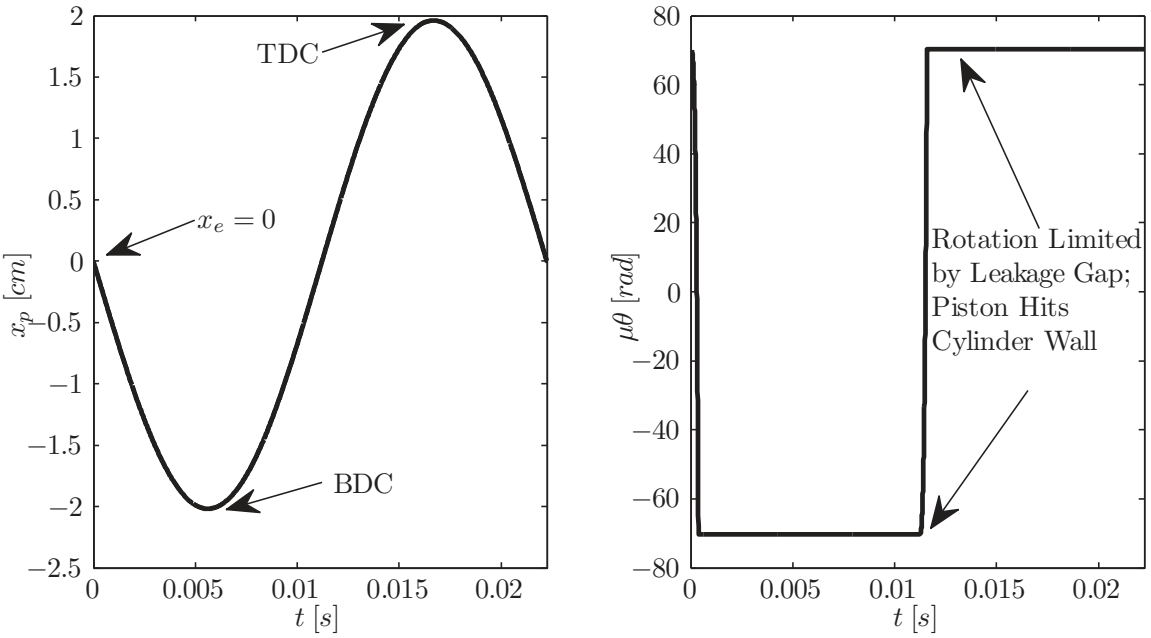


Figure 3.1. A typical piston amplitude (left) and piston rotation response (right) as predicted by the model.

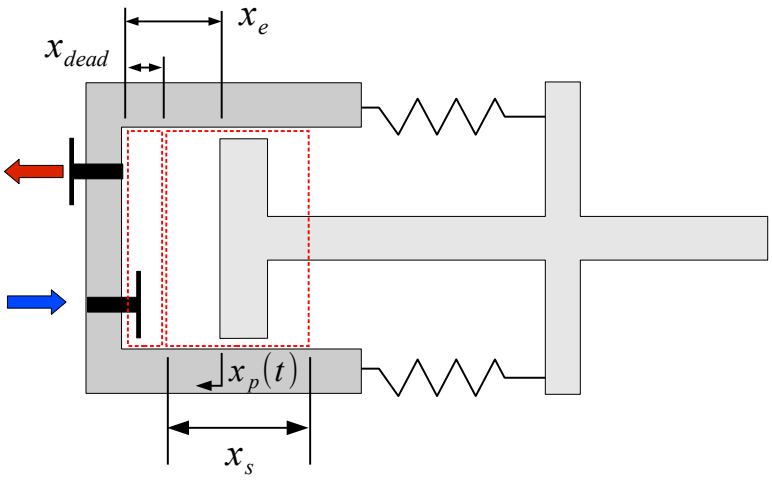


Figure 3.2. Schematic diagram of linear compressor piston dimensions.

The Figure 3.3 shows the refrigerant gas pressure as a function of cylinder volume (*i.e.*, an indicator diagram). This diagram displays a variety of the compressor's behavior. This includes the influence of the valves as seen by the under pressurization of the compressor cylinder caused by the suction valve and the over pressurization

caused by the discharge valve and port. These influences are regarded as losses as additional boundary work must be utilized to return the cylinder to either suction or discharge pressure respectively. Admittedly, the discharge process displays a relatively large amount of over pressurization. This is a result of an undersized discharge port and/or a discharge valve that is too stiff. This is included to illustrate the usefulness of the comprehensive model in combination with the indicator diagram to diagnose potential problems with compressor design. The indicator diagram also displays the compression process, expansion processes, and the dead volume. The dead volume on the indicator diagram is the volume remaining when the piston has reached TDC.

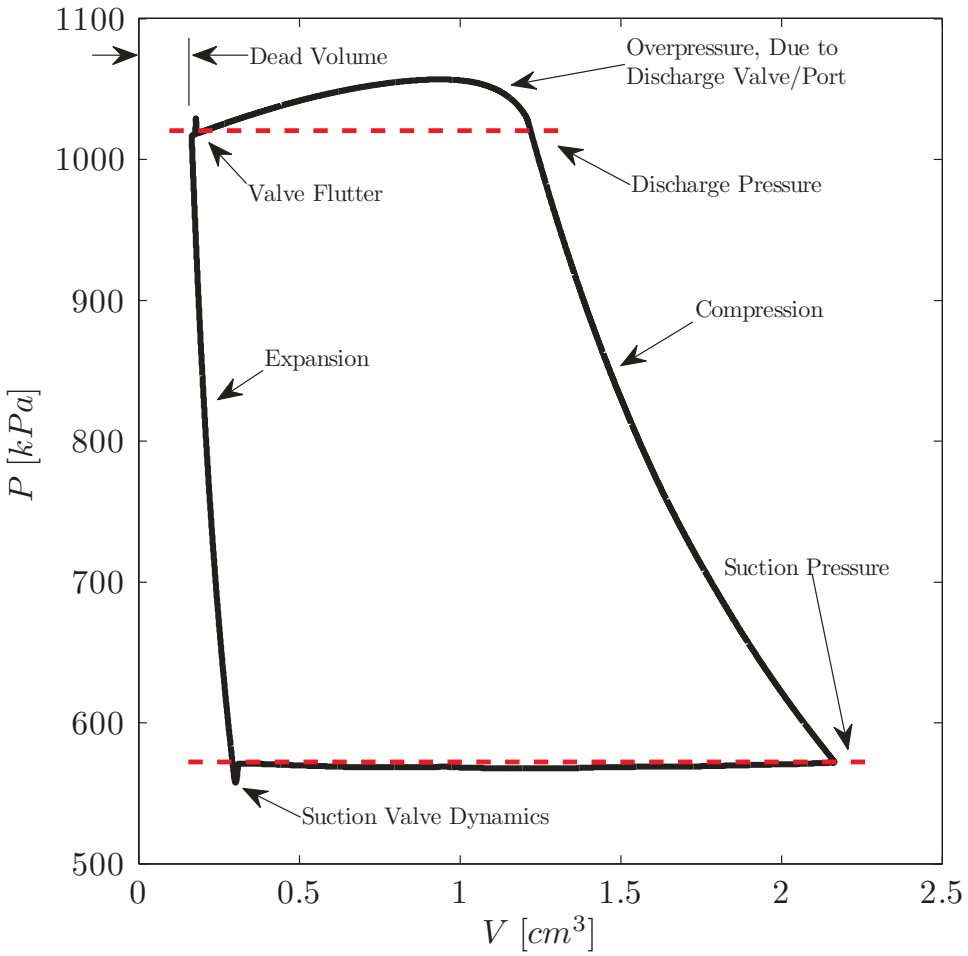


Figure 3.3. A typical pressure volume (indicator) diagram for a linear compressor as predicted by the model.

The behavior presented up to this point is typical of any piston-cylinder compressor. However, the dynamics of the linear compressor piston display behavior which is unique. These are explored further in the next two sections which examine the stiffness and damping associated with the linear compressor piston.

3.1.1 Linear Compressor Stiffness

As presented in Section 2.5, the stiffness associated with the dynamics of the piston motion is composed of two components. One component is the stiffness associated with the mechanical springs, while the other is associated with the compression of the gas in the cylinder. The latter may be visualized in Figure 3.4 which shows the change in pressure across the compressor cylinder as a function of the piston position for a typical operating condition.

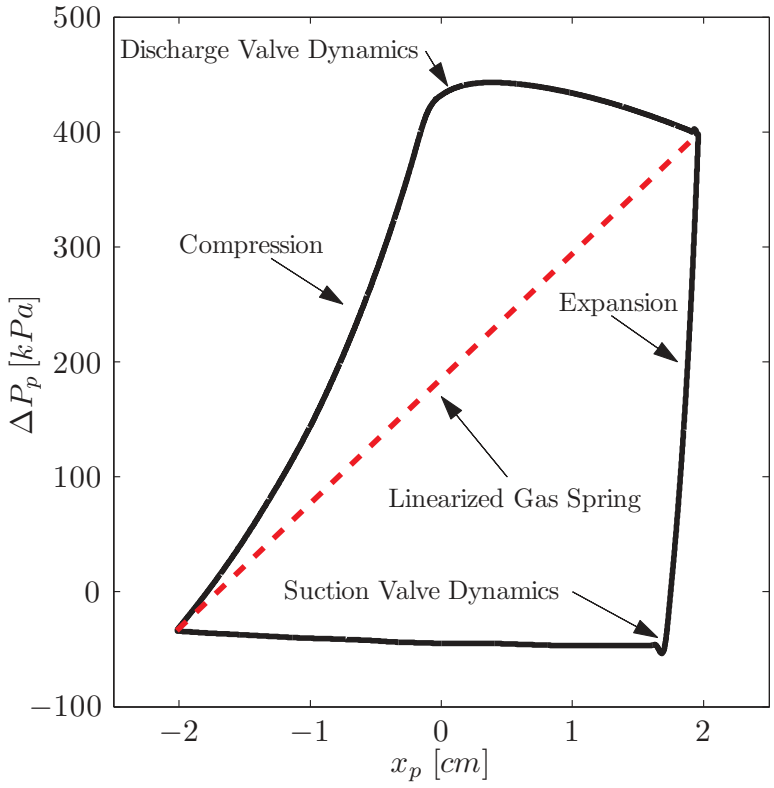


Figure 3.4. A typical representation of the change in pressure across the piston as a function of piston position predicted by the model.

The value of the stiffness in the compressor is proportional to the slope of this curve. The compression process begins at the lower left portion of this figure where the compression chamber is at a lower pressure than the shell so the difference in pressure is negative. Then, following the left-hand side of the curve, the gas is compressed as the piston moves in a positive direction with an increasing slope, meaning the stiffness is increasing. When the discharge valve opens the change in pressure across the piston levels out, so that the stiffness decreases and even becomes negative as the discharge process continues. The stiffness is large again during the expansion process and close to zero during the subsequent suction process. These changes can be abrupt as seen in Figure 3.5 which shows the total effective stiffness and the mechanical stiffness as a function of piston position for a typical operating condition.

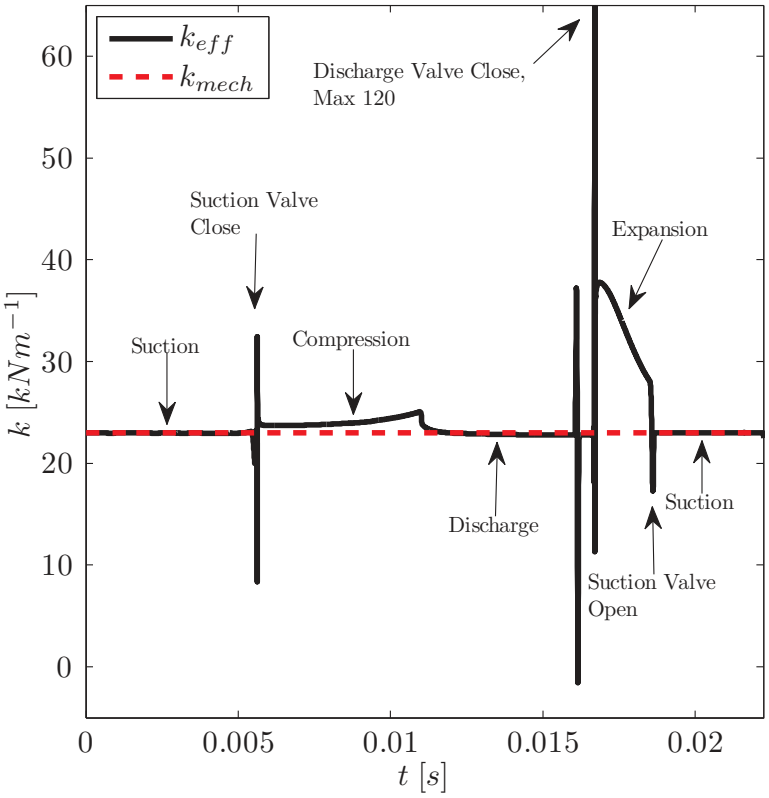


Figure 3.5. A typical effective stiffness of compressor piston as a function of piston position.

The extreme high and low (and even negative) stiffness values throughout a single cycle degrade the overall stability of the numerical system which is why past attempts at linear compressor modeling (Cadman and Cohen, 1969a; Pollak et al., 1979; Kim et al., 2009) have employed a linearizing assumption for the gas stiffness. This assumption estimates the gas stiffness as the line developed on Figure 3.4, which is a constant value based on the operating conditions and stroke. While in general, $k_{mech} > k_{gas}$, Figure 3.5 shows that there are peaks of gas stiffness that are much larger than the mechanical counterpart. As the resonant frequency is related to this stiffness this means that the resonant frequency, and the overall piston dynamics, are changing drastically with stroke.

3.1.2 Linear Compressor Damping

Similar to the stiffness, the damping associated with the dynamics of a linear compressor has two components. One is again associated with the gas compression and the other is associated with friction. The models employed are detailed in Section 2.5. The total effective damping coefficient of the dynamic system as a function of the piston position is shown in Figure 3.6 for a typical set of operating conditions. In addition, the frictional component of the piston damping is also shown.

The nonlinear behavior of the effective damping coefficient has been observed to not cause model instability. Starting at Top Dead Center (TDC) the expansion process begins and the effective damping tends to decrease. As the suction valve opens to begin the suction process the change is cylinder pressure is small so the trend in effective damping tends to follow the frictional damping trend. The damping tends toward a minimum at the piston equilibrium position. At the equilibrium position, where the piston is under no net influence from the mechanical springs there is no side loading, and therefore, no frictional damping occurs. As the piston moves toward Bottom Dead Center (BDC) during the suction process the effective damping increases as the frictional damping increases. As the piston moves from

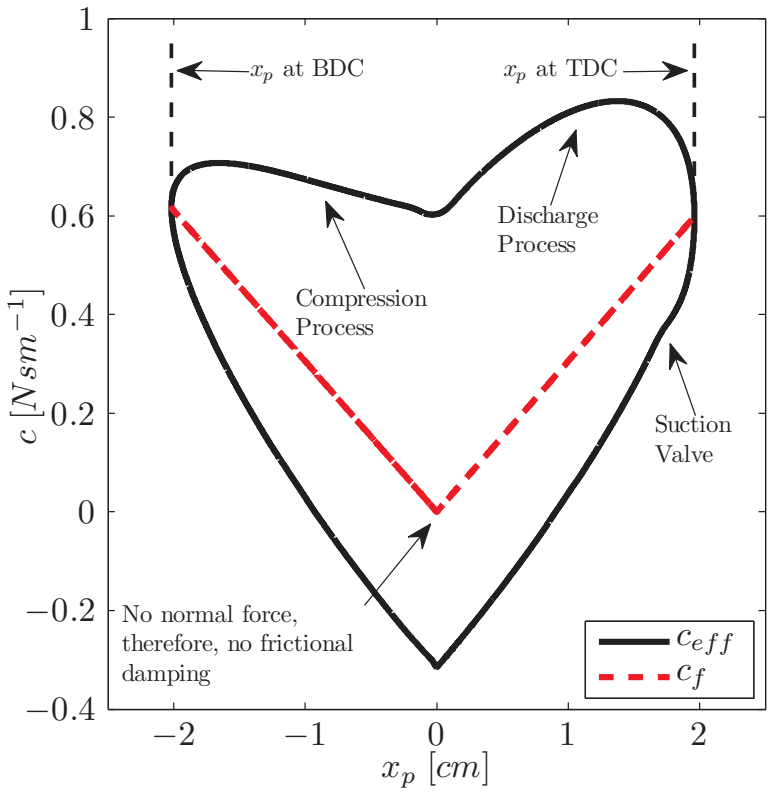


Figure 3.6. A typical effective damping coefficient of compressor piston as a function of piston position.

BDC toward TDC during the compression process the effective damping decreases due to a decrease in frictional damping but remains higher than during the suction process. During the discharge process the effective damping increases further due to the cylinder over-pressurizing which is caused by the dynamics of the discharge valve. As the piston approaches TDC and the cylinder pressure is reduced the effective damping is also reduced.

3.2 Control Considerations

The combination of the stiffness and damping behavior described in the previous section generate a highly nonlinear dynamic system described by the equations of motion explained in Section 2.5. A characteristic of some nonlinear systems is that

it is possible to obtain unexpected responses that are not possible in a linear system. Such an attribute is a sudden change in amplitude, called a jump (Rao, 2004). Other characteristics include bifurcations of the steady-state solution. This means that the system behavior at steady-state not only depends on the initial conditions but also on the conditions along the path to the solution (*i.e.* during the transient). A majority of the sub-models presented in the comprehensive linear compressor model are models of steady-state phenomena and therefore are not accurate in a transient response. Therefore, the (transient) path to solution of the comprehensive linear compressor model may not accurately represent the path to solution. Thus, to ensure model accuracy this dependence on the path to solution must be removed.

A key model output from the comprehensive compressor model is the power required to drive the linear compressor. To accurately calculate this value, the frequency at which the compressor will resonate is needed. To find the resonant frequency, one would typically sweep through frequencies until a maximum amplitude is determined. However, the nonlinearities of this system dictate that the resonant frequency changes based on the stroke and the stroke obtained for a fixed power input changes based on the frequency applied. Therefore,

$$x_p = f(\omega, F_{drive}). \tag{3.3}$$

This coupling generates a path dependence on the final stroke and applied power which needs to be removed to ensure model accuracy. Therefore, the resonant frequency will be decoupled in this relationship so that the compressor output stroke is only dependent on the input driving force from the linear motor.

3.2.1 Calculating the Resonant Frequency

The resonant frequency of the linear compressor depends on the mechanical springs selected in the design as well as the operating conditions. To calculate the resonant frequency of a linear compressor, the stiffness associated with both the mechanical

springs and the operating conditions must be estimated. The stiffness of the mechanical springs is typically reported by the manufacturer. The stiffness associated with the operating conditions is the stiffness from gas compression. Using these stiffness values, an estimate of the resonant frequency of oscillation is obtained from the following expression:

$$\omega_{res} = \omega_n \sqrt{1 - 2\zeta^2} \tag{3.4}$$

where the damping ratio is defined as follows (Rao, 2004)

$$\zeta = \frac{c_{eff}}{2M_{mov}\omega_n} \tag{3.5}$$

and $\omega_n$is given by,

$$\omega_n = \sqrt{\frac{k_{eff}}{M_{mov}}}. \tag{3.6}$$

The effective stiffness is defined as:

$$k_{eff} = k_{mech} + k_{gas} = k_{mech} + \frac{(P_{CV,1} - P_{CV,2})A_p}{dx_p} \tag{3.7}$$

and the effective damping as:

$$c_{eff} = \frac{E_{cycle}}{\omega_{res} x_s \pi} = \frac{W_f}{\omega_{res} x_s \pi} + \frac{W_{gas}}{\omega_{res} x_s \pi} \tag{3.8}$$

with work due to friction and work done on the gas defined as:

$$\begin{aligned} W_{gas} &= -\int P_{CV,1} dV \\ W_f &= 4\mathbf{f} \int_0^{x_s} N(x_p) dx. \end{aligned} \tag{3.9}$$

Equation 3.4 describes the resonant frequency of a one-dimensional system. The model presented previously in Chapter 2 utilizes a two degree of freedom system to describe the piston motion. These two degrees of freedom are the desired piston

translation and the undesired rotation of the piston within the cylinder. However, it has been observed that the resonant frequency of the linear compressor is predicted accurately using a one degree of freedom approach (Pollak et al., 1979; Cadman and Cohen, 1969a, b). In addition, Equations 3.4-3.9 are linearized based on the desired input stroke. This approach estimates the damping and stiffness of the system when the stroke is at a desired condition.

3.2.2 Stroke Control

A consequence of the free-piston design of the linear compressor is that the stroke of the compressor changes with power input and operating conditions. A reciprocating compressor, on the other hand, has a stroke that is fixed by the kinematics of the device and remains constant regardless of operating conditions or power input. The desired piston stroke is one of the necessary specifications in the design of a linear compressor. Therefore, the utility of the linear compressor model developed in Chapter 2 is enhanced here by allowing for the desired stroke to be specified. Since the piston driving force is an input to the two-degree-of-freedom piston vibration model used to determine piston behavior, a numerical algorithm is used to solve for the piston stroke by adjusting the piston driving force. The stroke is controlled to within 0.1 μm for each geometric configuration examined.

The numerical algorithm utilized to control the linear compressor stroke is known as Brent's method (Brent, 1971). This algorithm adjusts the force input to the piston iteratively until the desired stroke is obtained. Brent's method uses a combination of the Secant and Bisection methods for finding the roots. The algorithm depicted in Figure 3.7 presents the modified overall compressor model algorithm. The method requires two different guessed input values of force to begin. For each input value the same calculation algorithm is used. Using the resonant frequency model presented in Section 3.2.1, the compressor resonant frequency is calculated in the first step. Using a guess for force input, the model is solved in the manner previously described

in Chapter 2. The calculated stroke is then compared with the input value and an updated value calculated using Brent's method. This calculated force is then entered into the model, which iterates until the desired stroke is achieved.

3.2.3 Compressor Model Operation

This chapter presented a model for calculating the resonant frequency. In addition, a numerical algorithm is presented which allows stroke as an input to the comprehensive linear compressor model. Using these two additions, the overall compressor model exhibits added robustness for practical operation of the comprehensive linear compressor model. Figure 3.7 depicts the modified linear compressor model algorithm with the resonant frequency calculation and power adjustment algorithm added.

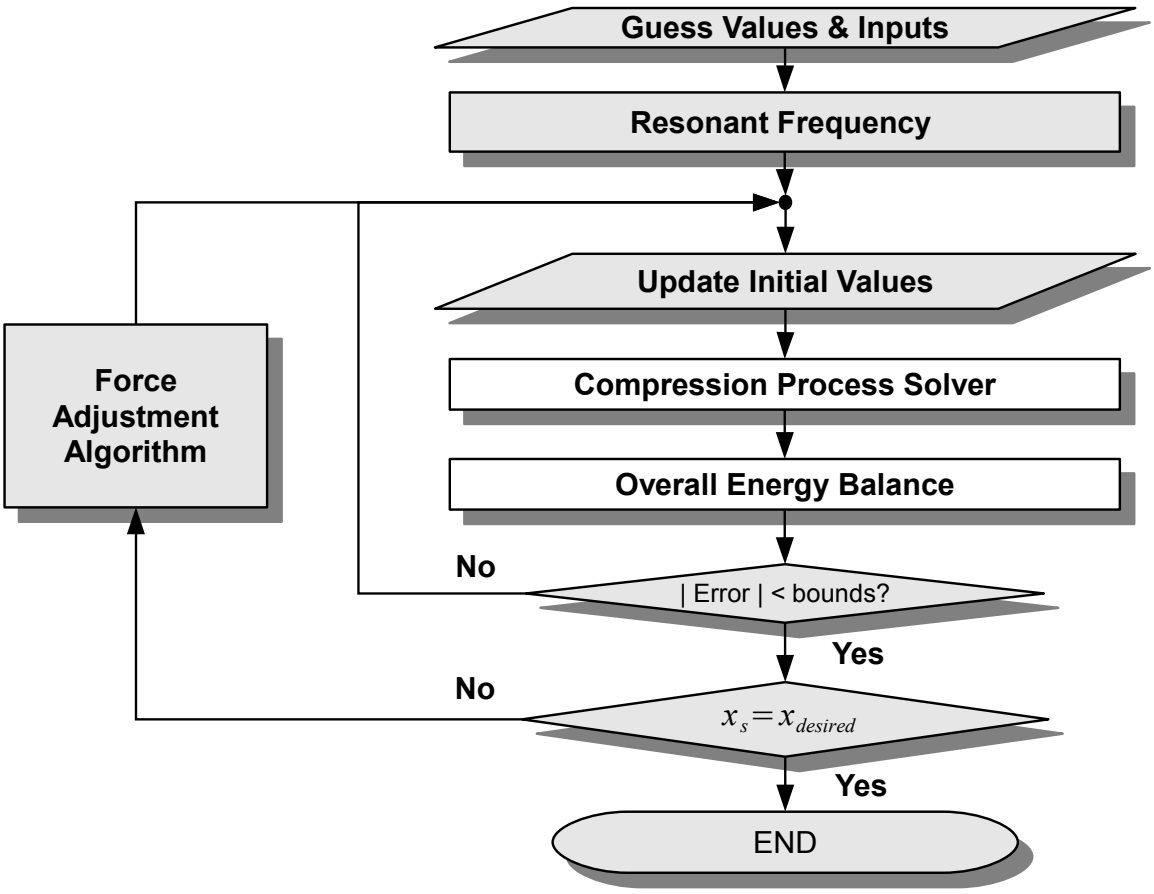


Figure 3.7. Linear compressor model with resonant frequency calculation and force adjustment algorithm added.

CHAPTER 4. PROTOTYPE-COMPRESSOR PERFORMANCE AND MODEL VALIDATION.

4.1 Miniature-Scale Linear Compressor Prototype

At the time this study was initiated there were no commercially available linear compressors to test. In addition, most investigations that utilized prototype compressors restricted themselves to much larger scale applications (Lee et al., 2004, 2008; Unger, 1999). Thus, a custom designed linear compressor for electronics cooling application was built. This design was introduced previously in Figure 1.3 and further details of the design are outlined in Appendix A. Since the initiation of this work a linear compressor became commercially available in a domestic refrigerator/freezer; this compressor is examined in Chapter 6. The remainder of this chapter focuses on the testing and model validation of the prototype compressor built specifically for this work.

4.2 Miniature-Scale Compressor Load Stand

The prototype linear compressor was tested on a compressor load stand specifically built for testing miniature-scale compressors. The compressor load stand is based on a hot-gas bypass design; a schematic is shown in Figure 4.1. This type of load stand has been used successfully in previous studies to conduct compressor performance tests (Sathe et al., 2008; Hubacher et al., 2002). The load stand cycle state points are shown in a pressure enthalpy diagram in Figure 4.2. The compressor operates between state points 1 and 2. From state point 2, the gas is expanded across a set of valves to an intermediate pressure at state point 3. The flow from state point 3 is divided into two flow paths. In one path, the refrigerant flows through the condenser and is cooled to a saturated liquid at state point 4. It is then isenthalpically expanded

to the compressor suction pressure at state point 5. In the hot-gas bypass flow path, the refrigerant is expanded from the intermediate pressure to the compressor suction pressure at state point 6. The flows at state points 5 and 6 are mixed to provide the compressor suction state at state point 1. By controlling the ratio of these two flows the inlet pressure and temperature can be controlled. The discharge pressure of the compressor is controlled by adjusting the valve between state points 2 and 3.

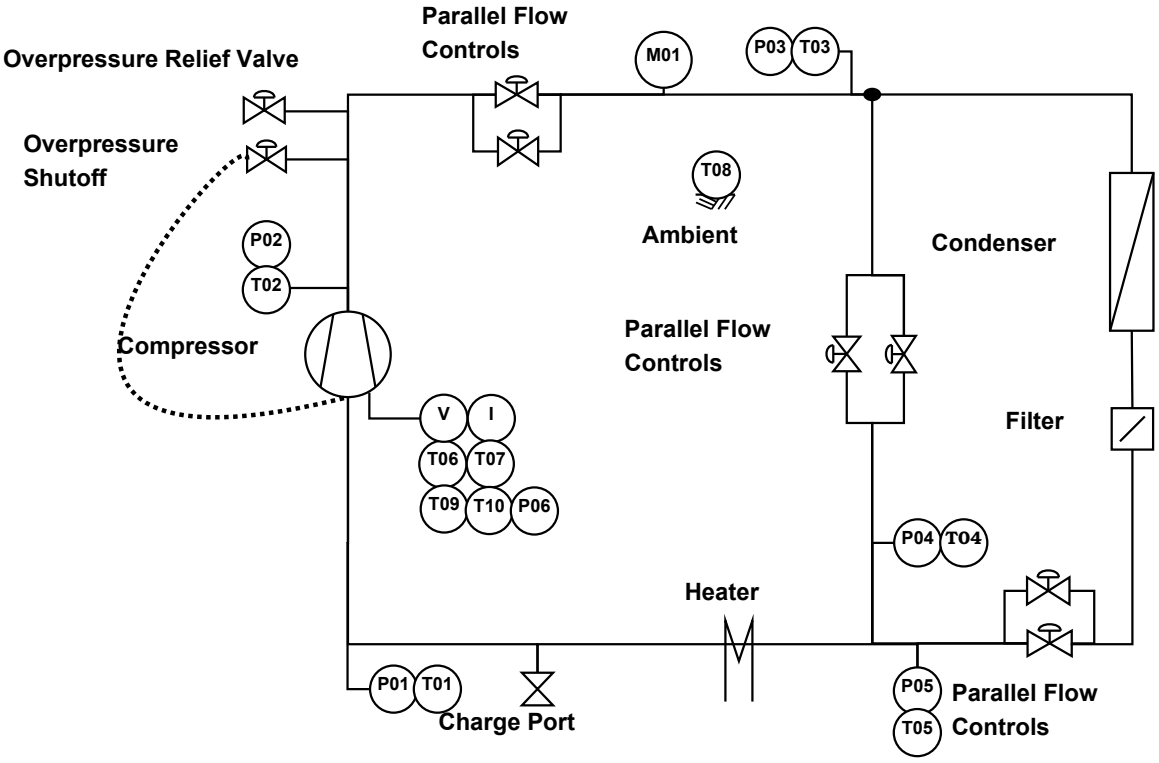


Figure 4.1. Schematic diagram of miniature-compressor hot gas bypass load stand.

The compressor was tested under a range of operating conditions, which simulate an electronics cooling application. At each specific set of compressor inlet conditions, the compressor was operated until a steady state condition was achieved, at which point data was collected. This process was then repeated for subsequent compressor inlet conditions, which are tabulated in Table 4.1. It is noted that the performance testing of a linear compressor is significantly different from that of conventional positive displacement compressors. During the performance testing of conventional com-

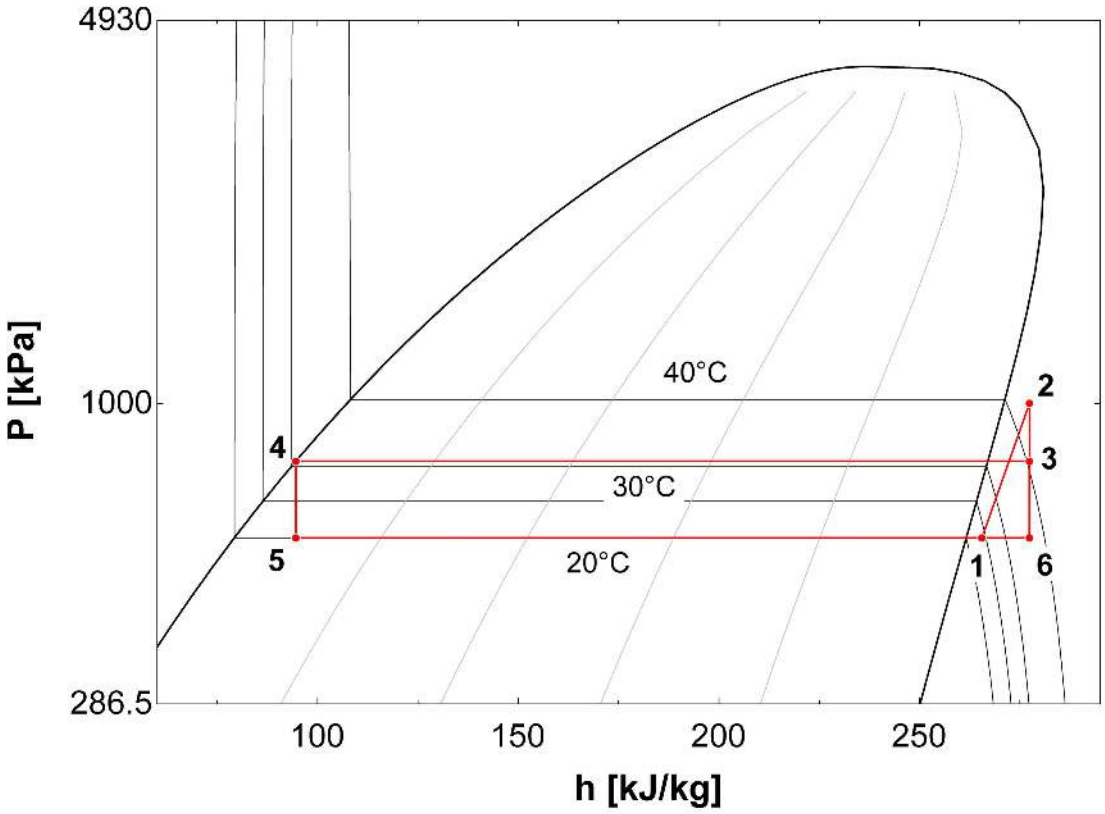


Figure 4.2. Pressure-enthalpy plot of compressor load stand operation for refrigerant R-134a.

pressors, the stroke and frequency of the piston are naturally held constant due to the kinematics of the device. However, a linear compressor requires that the frequency be adjusted to maintain operation at resonance. The power input and operating frequency are varied in these tests so as to maintain similar stroke values throughout the testing. The state points achieved in the experiments are summarized in Table 4.1.

The volumetric efficiency shown in Table 4.1 is defined as follows:

$$\eta_{vol} = \frac{\dot{m}}{\rho_1 x_p f_{res} A_p}. \tag{4.1}$$

Table 4.1. Summary of experimental data.

P_{suc}	T_{suc}	P_{dis}	T_{dis}	x_s	$\dot{m}$	f	$\dot{W}_{in}$	η_{vol}	$\eta_{o,is}$
kPa	K	kPa	K	cm	$g\ s^{-1}$	Hz	W	–	–
557	296.8	641	300.4	0.5784	0.335	44.1	24.7	0.405	0.057
561	297.0	702	299.8	0.5756	0.143	44.1	24.0	0.173	0.040
559	298.1	709	301.1	0.5951	0.120	44.1	23.6	0.141	0.028
534	295.4	587	300.3	0.6445	0.461	44.6	17.5	0.505	0.037
537	295.3	619	300.6	0.6695	0.380	44.6	17.1	0.398	0.048
537	295.3	651	300.2	0.6682	0.267	44.6	22.0	0.280	0.046

This definition accounts for volumetric losses as well as a dependence on the piston dynamics. The overall isentropic efficiency of the device, also listed in Table 4.1, is defined as follows:

$$\eta_{o,is} = \frac{\dot{m}(h_{2,s} - h_1)}{\dot{W}_{in}} \tag{4.2}$$

where $\dot{W}_{in}$ is the power input to the compressor and is calculated as follows:

$$\dot{W}_{in} = \left(F_{drive}\dot{x}_p\right)_{rms} + \dot{Q}_{motor}. \tag{4.3}$$

The definition for overall isentropic efficiency accounts for all the losses in the compressor, and as a result, is used as an overall performance metric for compressor performance.

4.2.1 Experimental Uncertainty

A variety of measurements were obtained in the experiments including temperature and pressure at the suction and discharge ports of the compressor, mass flow rate of refrigerant, piston stroke, frequency, and input power. The measurements of pressure, temperature, and stroke have absolute uncertainties of 4.6 kPa, 0.5 C, and 25.40 μm, respectively. The Coriolis mass flow meter has relative uncertainty values

between 0.350 and 1.25 %, depending on the quantity of mass flow measured. The frequency and input power have absolute uncertainty values which varied from 2.07 x 10^{-4} to 5.10 x $10^{-4} Hz$ and 0.188 and 0.532 W, respectively, which are calculated using a 99.95% confidence interval. The uncertainty in the reported efficiency values is calculated using an uncertainty propagation analysis (Fox et al., 2004). The volumetric and overall isentropic efficiencies have absolute uncertainty value ranges of 0.107 to 0.229 and 0.105 to 0.190, respectively.

4.3 Validation of Model Predictions

The experimental results are compared to the predictions from the model. The model performance is quite sensitive to several parameters: the leakage gap between the piston and cylinder, g, the eccentricity of the piston, ϵ, and the linear motor efficiency, η_{motor}. Since these parameters are difficult to measure, best estimates were used and adjusted within reasonable ranges to provide the best fit with the experimental data. The leakage gap was chosen based on the design dimensions to be in the range of 1 to 23μm. The initial guess of eccentricity was set to the radius of the compression spring used in the prototype compressor, which took values in the range of 9.2 to 11.4mm. The motor efficiency was estimated based on the measured heat loss from the prototype compressor and was varied between 30% and 50%. The final parameter values were selected based on the best agreement to the experimental data. These values are summarized in Table 4.2. The parameters in Table 4.2 along with experimental values for the suction and discharge pressure, suction temperature, frequency, and stroke (Table 4.1) were used as inputs to the model. Where the frequency is calculated in a separate submodel as shown in Section 3.2.1.

4.3.1 Resonant Frequency

Figure 4.3 compares the resonant frequency predicted from this approach and the experimentally obtained resonant frequencies. The model predicted resonant fre-

Table 4.2. Linear compressor prototype parameters.

k_{mech}	A_p	g	η_{motor}	ϵ	$V_{cyl,max}$
$N\,m^{-1}$	cm^2	μm	$-$	cm	cm^3
23000	1.217	13	0.4	1.145	3.091

quencies were calculated using the method desribed in Section 3.2.1. The predictions match the experimental measurements very well, with a Mean Absolute Error (MAE) of 0.5%.

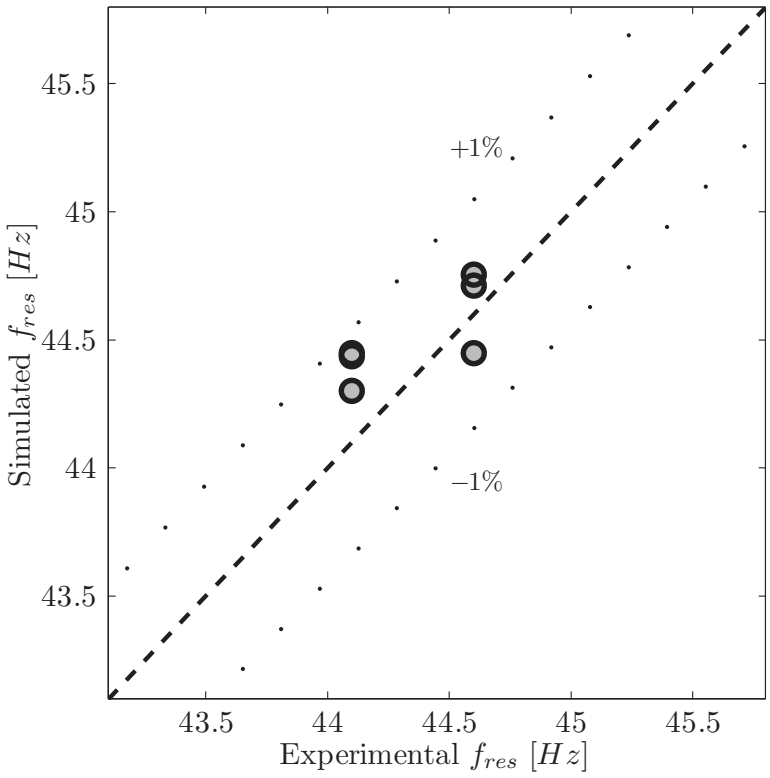


Figure 4.3. Comparison of experimental and simulated resonant frequencies (measurement error within marker width).

These values are then used as inputs along with the experimentally determined stroke and operating conditions to the comprehensive compressor model which calculates the massflow and power required to drive the device.

4.3.2 Massflow Rate, Volumetric, and Overall Isentropic Efficiency

Figure 4.4 shows a comparison of the experimental massflow rate to the model predicted values. The data are generally predicted well with one outlier point and a MAE of 16.8 %. The predicted volumetric efficiency, shown in Figure 4.5, displays similar trends to the mass flow rate. In the performance testing of conventional compressors the volumetric efficiency is simply a dimensionless mass flow rate. However, for a linear compressor, the volumetric efficiency depends not only on the mass flow rate but also the frequency, as seen in Equation (4.1). This accounts for additional discrepancies seen in the volumetric efficiency agreement. Figure 4.6 shows the experimentally determined overall isentropic efficiency compared with the model predicted values. All but one point fall within 0.03 of the predicted overall efficiency.

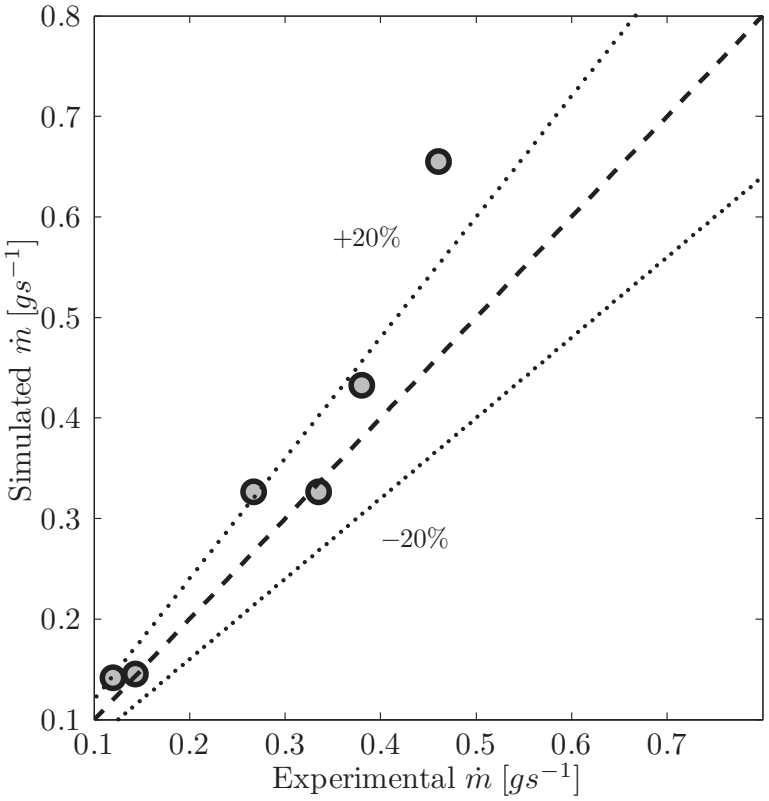


Figure 4.4. Comparison of experimental and simulated mass flow rates (measurement error within marker width).

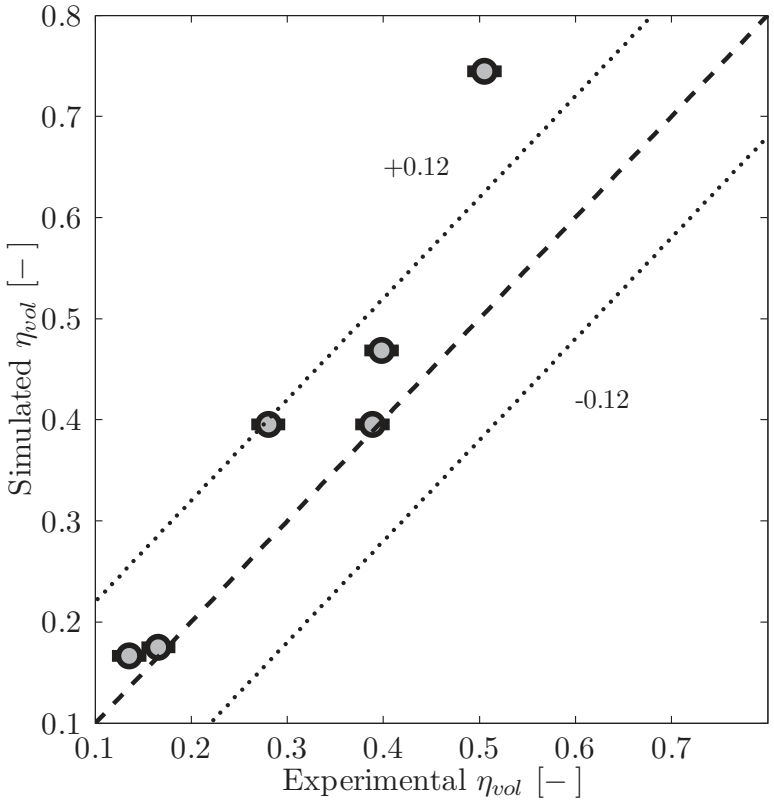


Figure 4.5. Comparison of experimental and simulated volumetric efficiencies.

Figure 4.7 shows the deviation of the experimental efficiencies from the model predicted efficiencies (*i.e.* η_{exp} - η_{model}) compared with experimental power input. This shows that as the power input increases the model performance tends to decrease. This indicates that there is a loss found in the experimental prototype that the model does not account for. However, the prototype compressor displays much room for improvement and this trend could likely be related to the less-than-ideal design. Thus, it is difficult to ascertain from this trend if there some key phenomenon the model does not account for. Therefore, further model validation is needed to determine if the model needs additional modification. This additional model validation is presented in Chapter 6.

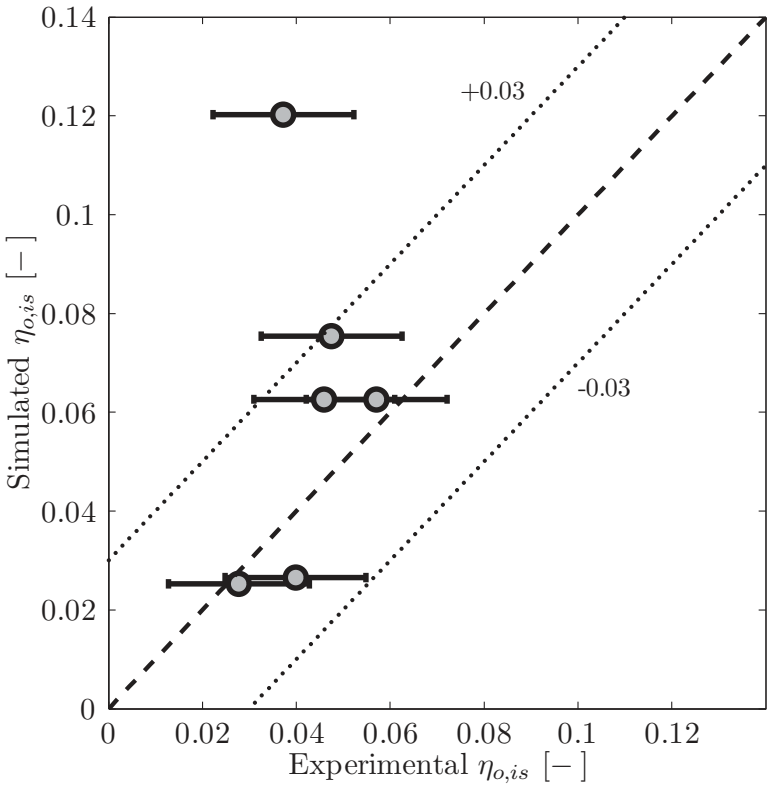


Figure 4.6. Comparison of experimental and simulated overall isentropic efficiencies.

4.4 Conclusions

The overall isentropic efficiency of the prototype compressor is significantly lower than for typical commercially available compressors. However, the purpose of the prototype built in this work was merely to validate the model and not to obtain optimized performance. Since no commercial prototype was available which could be tested for this work, the compressor was made in-house, and could admittedly be significantly improved. However, the operating principles underlying the model and the prototype are identical, and therefore, the experiments serve the purpose of validating model predictions. It is expected that the model can form the basis for more optimized prototypes to be designed. The model developed in this work can be applied in the analysis of any linear compressor application.

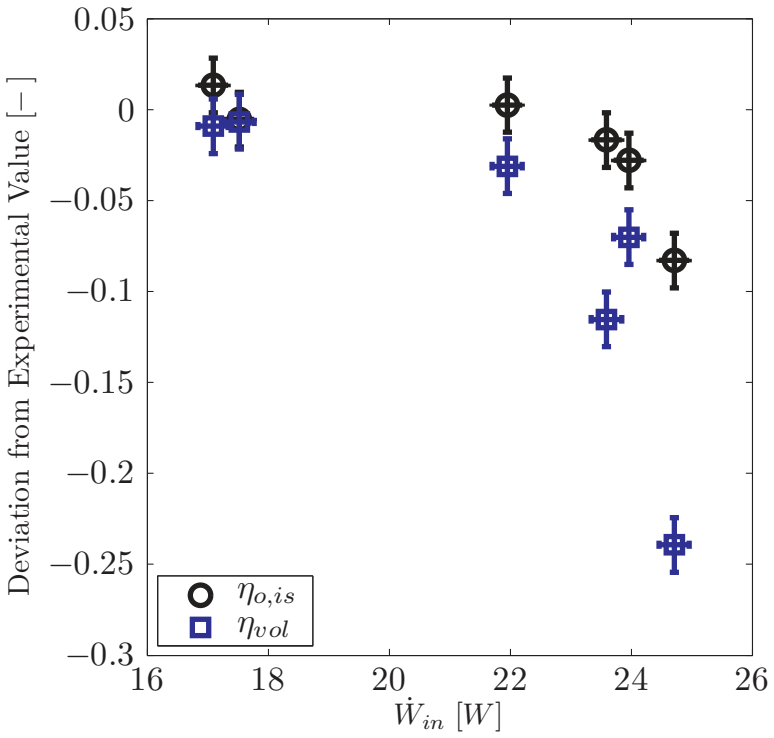


Figure 4.7. Deviation of model predicted results with experimental result ($\eta_{exp} - \eta_{model}$) for volumetric and overall isentropic efficiencies compared with experimental power input.

The agreement with experiment of the value of the resonant predicted by the model is evidence that the one-dimensional approach to calculate the resonant frequency accurately predicts the performance of the device. The mass flow rates show larger discrepancies which suggests that either the valve model or the leakage model needs further improvement. In other compressor models these flow paths are handled using semi-empirical models for good agreement with the performance of a particular compressor (Mathison et al., 2008; Kim and Groll, 2007; Navarro et al., 2007). The models presented here, in contrast, are based on first principles in an effort to add flexibility to the model for use in parametric studies.

Additionally, the sensitivity to small changes in the values of the key parameters identified in this work (i.e., leakage gap, eccentricity, and motor efficiency) may also explain the errors. It was found that given a particular set of operating conditions, a

5% change in the value of eccentricity could lead to a 24% change in overall isentropic efficiency. Each of the identified parameters is difficult to quantify with a high degree of accuracy. Therefore, it may be concluded that there is potentially a significant amount of uncertainty due to the selection of these parameters, which can explain a portion of the error in matching experimental data.

This theory was confirmed with additional modifications to the prototype linear compressor. As detailed in Appendix A, additional modifications to the prototype compressor were performed after model validation had been completed. The purpose of these efforts was to determine how sensitive the prototype compressor is to changes in the key parameters. Using the changes outlined in Section A.6, the modified compressor performance is compared with previous experimental data, as given in Table 4.3.

Table 4.3. Comparison of performance of revision 01 to revision 02 linear compressor.

Comp.	P_{suc}	T_{suc}	P_{dis}	x_p	$\dot{m}$	f	$\dot{W}_{in}$	η_{vol}	$\eta_{o,is}$
[−]	kPa	K	kPa	cm	$g\ s^{-1}$	Hz	W	−	−
Rev 01	561	297.0	702	0.5756	0.143	44.1	24.0	0.173	0.040
Rev 02	581	294.0	681	0.5800	0.910	44.0	24.0	0.891	0.107

The revision 02 prototype compressor incorporated small changes to reduce the dead volume, flow restrictions, friction losses, and leakage gap. These changes were enough to increase the overall isentropic efficiency by over 100% compared with a similar operating point from the previous revision, shown in Table 4.3. The prototype compressor would still require significant effort to raise efficiency values to be comparable with a commercial compressor. However, this has shown that the prototype compressor configuration is highly sensitive to changes in geometric parameters.

CHAPTER 5. SENSITIVITY STUDIES

A loss analysis is presented which quantifies the frictional and leakage losses within the linear compressor. Using this analysis, sensitivity studies conducted using the comprehensive linear compressor model are presented here for three studies. The first is a leakage and friction study, which examines the sensitivity to changes in the leakage gap g, as well as spring eccentricity ϵ. The second study considers changes in the piston geometry and scaling of the compressor, specifically the piston diameter at various displacement volumes. The third considers the impact of changing the stroke within a compressor with a fixed cylinder size by adjusting the compressor dead volume. The additional geometric parameters are shown in Figure 5.1. The ranges of parameters investigated are first discussed, followed by the results of the studies, and finally the discussion of the results. Additional interesting results are also discussed such as piston drift and capacity control. Finally, a comparison to a traditional reciprocating compressor is presented which highlights the major differences between the two technologies.

5.1 Loss Analysis

While the overall isentropic efficiency yields the total losses, a separate estimation of major loss components provides useful insight into the compressor operation. Leakage and friction are typically two sources of major loss within a positive displacement compressor and are quantified here.

The frictional losses in the compressor are determined as the scalar product of the frictional force and the piston velocity. These two quantities oscillate about a mean value of zero. Therefore, to estimate the mean power dissipated via friction over a compression cycle requires the use of the root mean square (rms):

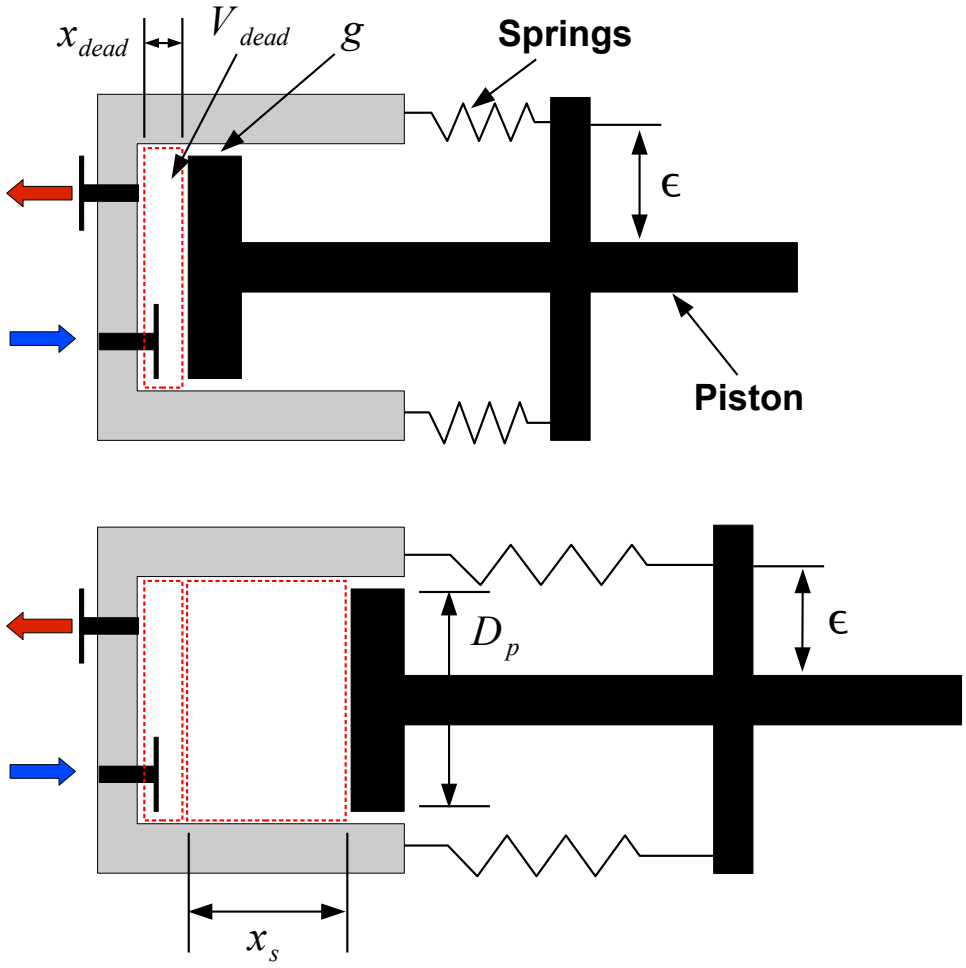


Figure 5.1. Schematic diagram of linear compressor geometric parameters of interest for sensitivity study.

$$\dot{W}_{friction} = (\mathbf{f}N(x))_{rms}(\dot{x}_p)_{rms} \tag{5.1}$$

In a linear compressor, the leakage losses represent mass that has been pushed into the compressor shell; the same amount of mass also returns to the compression chamber due to mass being conserved. The net loss represents the work required to move this mass of gas both into and back out of the compressor shell, and represents a loss in available energy. Therefore, to estimate the leakage loss, an exergy analysis is used. Using the compressor control volumes shown in Figure 1.2, the amount of exergy

destroyed in moving the leakage mass into the compressor shell and subsequently back into the compression volume is:

$$\dot{E}_{d,leak} = \dot{W}_{leak} = T_{amb}\dot{m}_{leak}(s_{cv,2} - s_{cv,1}) + \left(\frac{T_{amb}}{T_{shell}}\right)\dot{m}_{leak}(h_{cv,2} - h_{cv,1}). \qquad (5.2)$$

5.2 Model Sensitivity to Geometric Parameters

The results from the model validation in Chapter 4 highlighted four parameters that significantly impact the overall performance metrics of the compressor: motor efficiency η_{motor} , dry friction coefficient $\mathbf{f}$, spring eccentricity ϵ, and leakage gap g. For simplicity, changes to η_{motor} are not considered here, as changes in this parameter would provide little insight into compressor design; the higher the linear motor efficiency, the better is the compressor overall performance. Thus, the efficiency is fixed at 90% for the rest of this work. The operating conditions are set for all studies at 20 ^oC, 40 ^oC, and 5 ^oC for evaporating, condensing, and superheat temperatures, respectively.

5.2.1 Leakage and Friction

The dry friction coefficient $\mathbf{f}$ and spring eccentricity both relate to friction between the piston and cylinder and only the spring eccentricity is examined in this study, besides the leakage gap g. The ranges of the values used in the study attempt to represent an extreme set of conditions. For eccentricity, this range spans from 0.1 cm to 0.9 cm, where the upper limit represents an extreme situation. The leakage gap ranged from 1 μm to 23 μm, which spans a realistic range of values for compressor leakage gaps (Kim and Groll, 2007; Jovane, 2007). The stroke is fixed at 2.54 cm (1 in) for these studies.

The leakage mass flow rate as a function of leakage gap at different eccentricity values is shown in Figure 5.2. The leakage flow rate increases as the leakage gap increases and does not display a dependency on piston eccentricity. An increase in leakage mass flow rate results in a decrease in the volumetric efficiency of the device; this is shown in Figure 5.3, which presents the compressor volumetric efficiency for different eccentricities as a function of leakage gap.

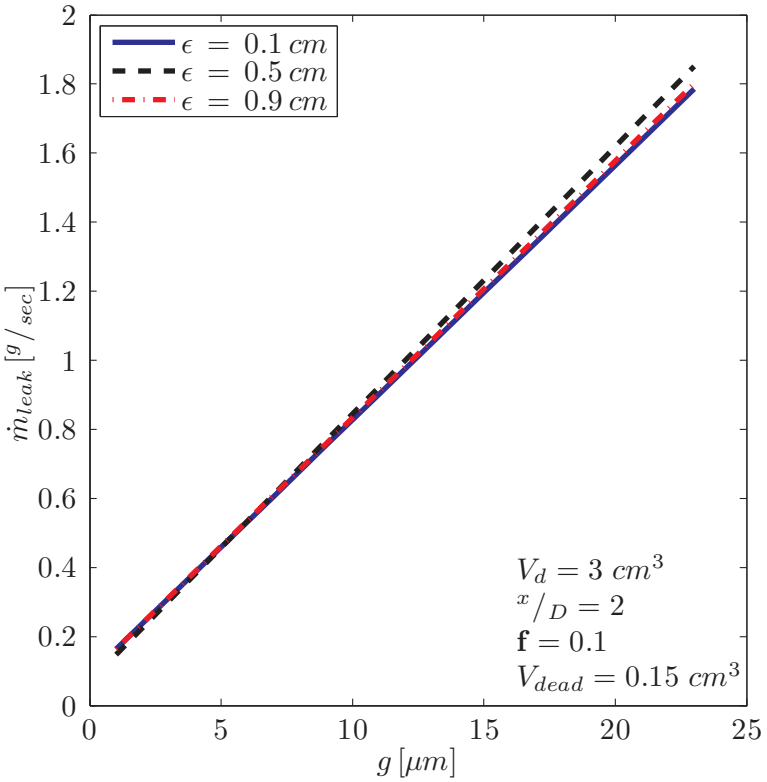


Figure 5.2. Leakage mass flow rate as a function of leakage gap width at different spring eccentricities.

This decrease in efficiency corresponds to a decrease in mass flow rate from the compressor as a result of increased leakage. Both the leakage mass flow rate and the volumetric efficiency show a negligible dependence on eccentricity. It may be concluded that the flow parameters are not affected significantly by changes in frictional characteristics within the compressor but are highly sensitive to changes in leakage gap width.

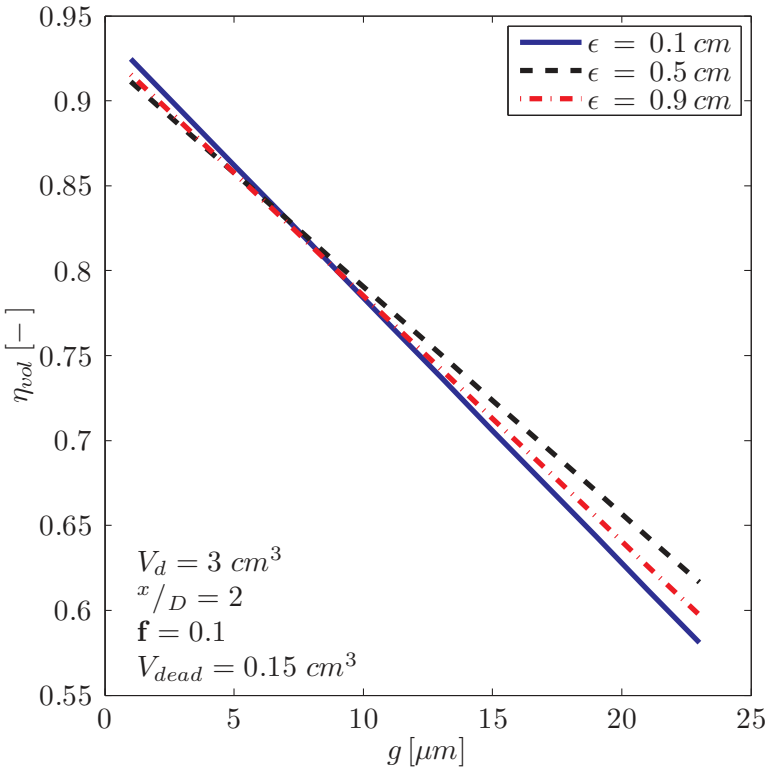


Figure 5.3. Compressor volumetric efficiency as a function of leakage gap width at different spring eccentricities.

The overall compressor performance is presented in Figure 5.4, in terms of the overall isentropic efficiency at different leakage gap and eccentricity values. As the eccentricity decreases, the overall isentropic efficiency increases proportionally. This is a result of increased frictional losses at higher eccentricity values. The compressor performance is also reduced at higher leakage gap widths, since the compressor mass flow rate decreases with larger leakage gaps.

Figure 5.5 shows the net frictional losses as a function of leakage gap width for each of the three eccentricity values considered. The frictional losses are seen to be largely independent of the leakage gap width, with a slight increase in loss as the leakage gap is reduced. However, the frictional losses are strongly dependent on eccentricity. Figure 5.6 shows that leakage losses, on the other hand, increase as the leakage gap increases, with little dependence on eccentricity.

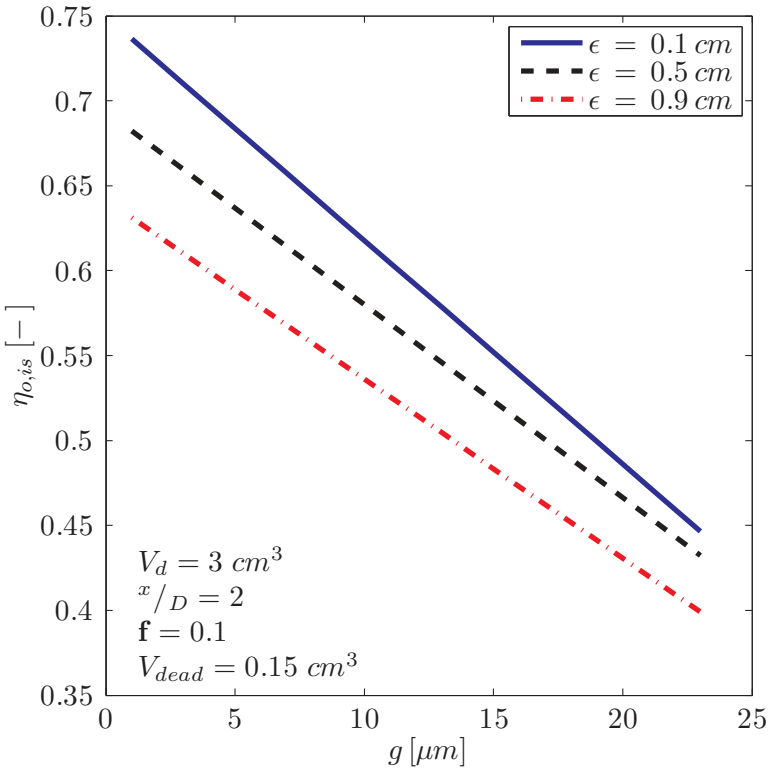


Figure 5.4. Overall isentropic efficiency as a function of leakage gap width at different spring eccentricities.

These results show the high level of sensitivity of the linear compressor performance to both the leakage gap between the piston and cylinder and the eccentricity of the spring. Reducing the leakage gap and the eccentricity of the applied spring force would improve the compressor performance. The lower limit on both these parameters is dictated by manufacturing constraints. It is also noted that at the smallest leakage gap widths, a stronger coupling would exist between leakage and friction that is not accounted for by the dry friction model utilized in the current work.

5.2.2 Piston Geometry and Compressor Scaling

The geometry of a compressor piston can have a large impact on the overall performance of the device (Kim and Groll, 2007; Rigola et al., 2005). The impact of scaling

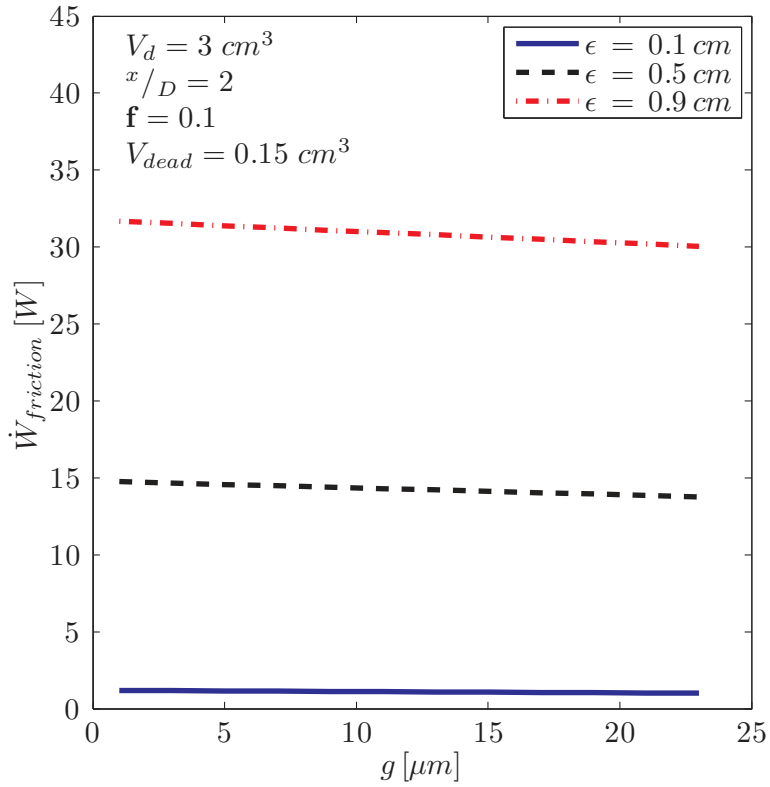


Figure 5.5. Frictional losses for various leakage gap widths at three eccentricities.

of the linear compressor is also of interest to the goal of miniaturizing the compression technology. To explore the impact of scaling, three compressor displacement volumes of 2, 3, and 6 cm^3 are examined. For each displaced volume the piston diameter is varied from 0.8 to 1.7 cm. Fixing the compressor displacement volume and varying the piston diameter requires modification of the compressor stroke. Therefore, the stroke-to-diameter ratio is investigated for three displacement volumes.

The volumetric efficiency of the linear compressor is low at small values of stroke-to-diameter ratio, but increases asymptotically as the value of this ratio is increased, as illustrated in Figure 5.7. This behavior is largely unaffected by the choice of displacement volumes. This behavior is the result of a higher dead volume relative to the displaced volume as the piston diameter increases. Figure 5.1 illustrates the dead volume of the linear compressor at TDC that is generated by the small gap between

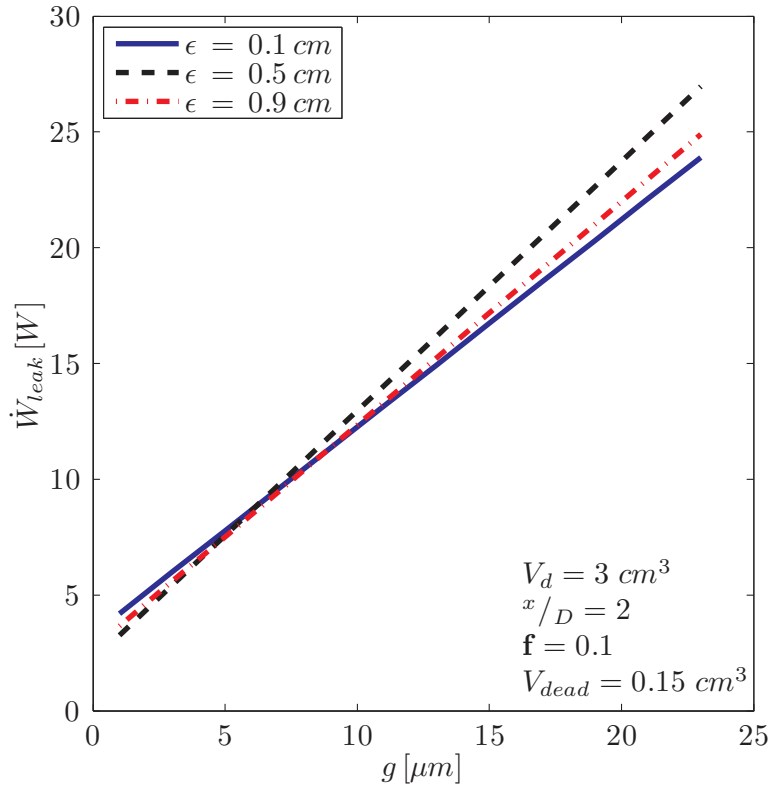


Figure 5.6. Leakage losses as a function of leakage gap at three eccentricities.

the piston and valve head x_{dead}. This gap is fixed at 3 mm for all the sensitivity studies considered here. Therefore, as the stroke-to-diameter ratio decreases, the dead volume increases in relation to the piston surface area. This increase in dead volume results in a reduction in volumetric performance.

The effects of displaced volume and stroke-to-bore ratio on the frictional and leakage losses of the linear compressor are shown in Figure 5.8 and Figure 5.9. The frictional losses in the linear compressor increase at a super-linear (*i.e.* greater than linear increase) rate with an increase in stroke-to-diameter ratio. An increase in displacement volume leads to increased frictional losses. As the stroke-to-diameter ratio increases, the mean piston velocity increases as a result of the increase in stroke. In addition, the frictional force generated by the contact between the piston and cylinder increases. Thus, as seen from Equation (5.1) where two of the three factors

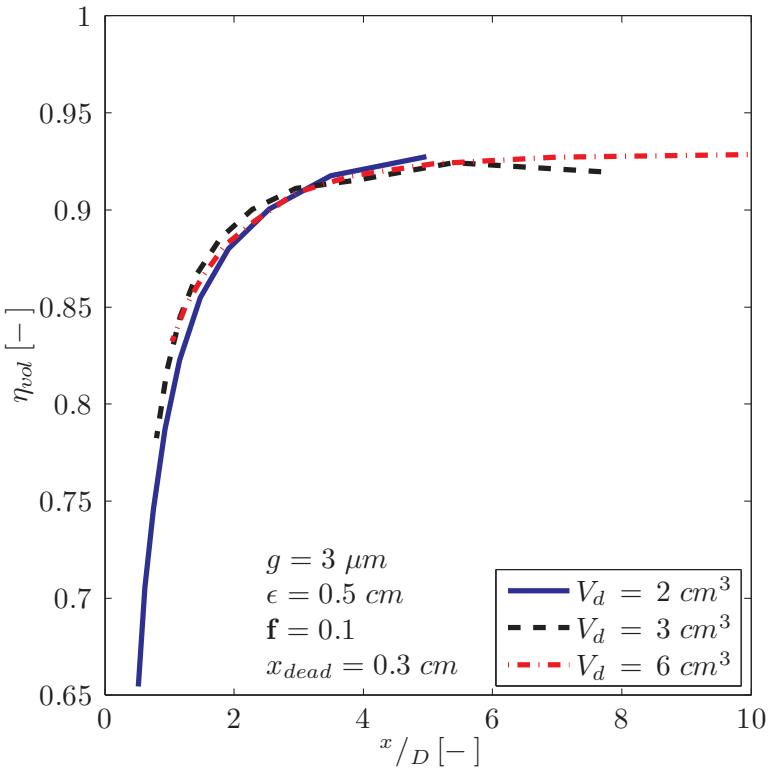


Figure 5.7. Volumetric efficiency as a function of stroke-to-diameter ratio for three displacement volumes.

increase, the frictional losses increase at a super-linear rate as the compressor stroke-to-diameter ratio increases.

The leakage losses show a different behavior with the stroke-to-diameter ratio: they decrease slightly until a stroke-to-diameter ratio of between 3 and 4, and then increase rapidly as this ratio is increased further. As with the frictional losses, the leakage losses increase with an increase in the displacement volume.

The leakage losses display two distinct trends as shown in Figure 5.9, separated at a stroke-to-diameter ratio of between 3 and 4. As this ratio takes increasing smaller values than 3, the leakage losses slightly increase, whereas the losses increase sharply as this ratio increases beyond a value of 4. This behavior is a result of two competing factors: the change in piston resonant frequency, which changes the time period over which leakage can occur, and the overall leakage area. Figure 5.10 shows the

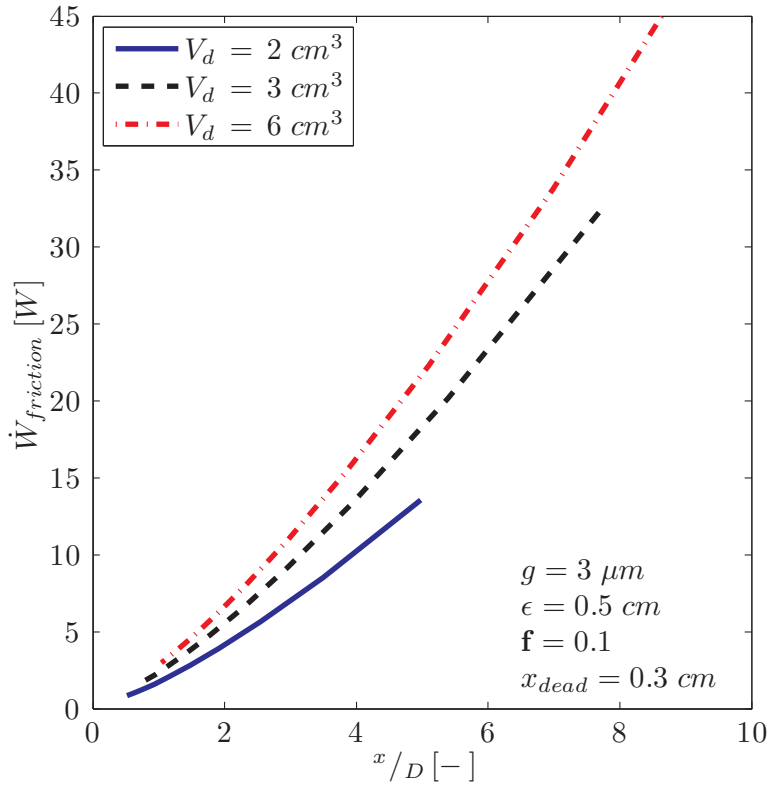


Figure 5.8. Frictional losses of a linear compressor as a function of the stroke-to-diameter ratio for three displacement volumes.

compressor piston resonant frequency as a function of the stroke-to-diameter ratio. As the stroke-to-bore ratio increases, the resonant frequency decreases. Since the resonant frequency of the piston is related to the piston oscillation period by:

$$f_{res} = \frac{1}{\mathbf{T}_p}. \tag{5.3}$$

An increase in stroke-to-bore ratio increases the piston oscillation period. An increase in the piston period generates a larger time period over which leakage can occur. Leakage within the compressor is also proportional to the area within the leakage gap:

$$\dot{m}_{leak} = \rho \mathbf{V} A_{leak}. \tag{5.4}$$

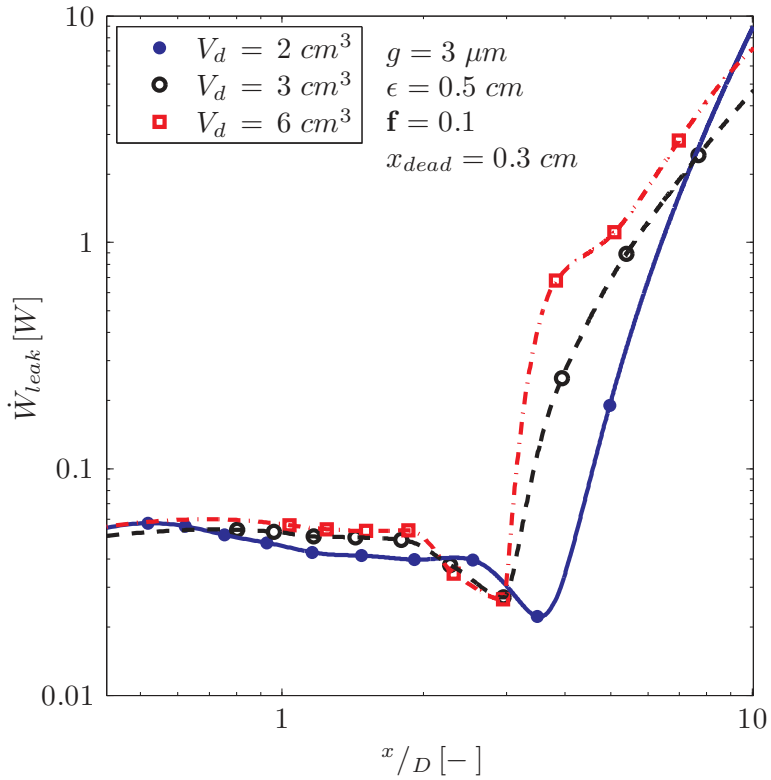


Figure 5.9. Curve fit of leakage losses of a linear compressor as a function of the stroke-to-diameter ratio for three displacement volumes with model data included as points.

The leakage area is determined by the piston diameter, with larger piston diameters corresponding to larger leakage areas. When the stroke-to-diameter ratio is small, piston period is also small, but the leakage area is large. When the stroke-to-diameter ratio is large, on the other hand, the piston period is large but the leakage area is small. This trade-off is largely dominated by piston period but does exhibit an optimum at a stroke-to-diameter ratio of between 3 and 4.

In contrast to the volumetric efficiency, the overall isentropic efficiency decreases with an increase in stroke-to-diameter ratio, as illustrated in Figure 5.11. In addition, the performance improves with a decrease in displacement volume.

Based on the results presented above for the volumetric performance, leakage and friction losses, the overall isentropic performance would also be expected to exhibit

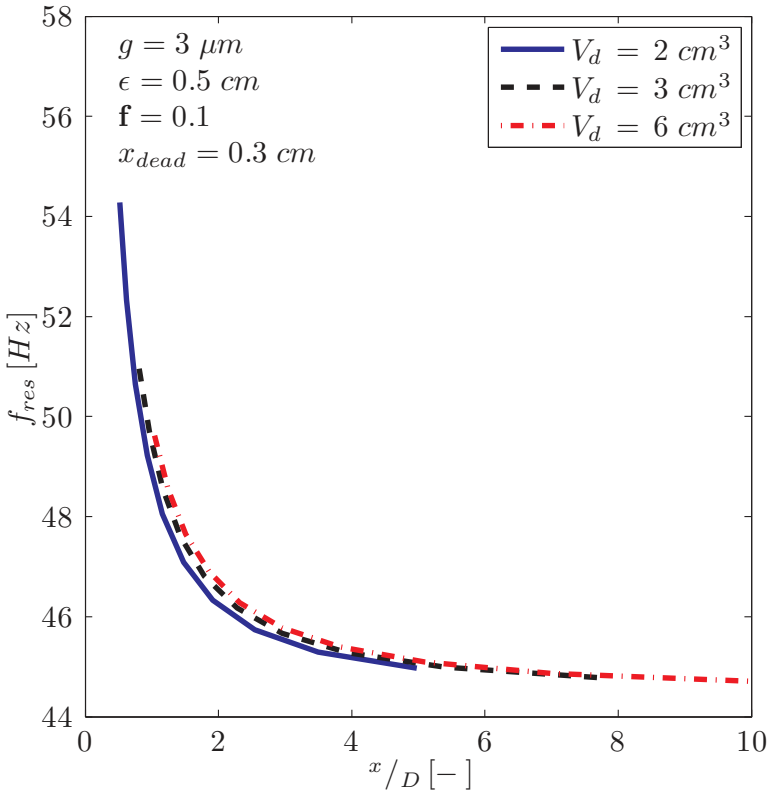


Figure 5.10. Compressor piston resonant frequency as a function of the stroke-to-diameter ratio for three displacement volumes.

a trade-off between volumetric efficiency and frictional and leakage losses. Figure 5.11 shows a degradation in overall isentropic efficiency with an increase in stroke-to-diameter ratio due to increased friction and leakage losses. The degradation in volumetric efficiency seen in Figure 5.7 at small stroke-to-diameter ratios does not seem to manifest as a decrease in the overall isentropic efficiency under these conditions. The volumetric efficiency decreases due to an increase in dead volume, but the overall compressor performance seems to be unaffected by such an increase in dead volume. This peculiar behavior may be attributed to the unique free-piston design of the linear compressor. In a typical positive displacement compressor, the gas trapped in the dead volume is re-expanded during the suction stroke and the energy required to compress this volume is lost. In a free-piston design, the gas trapped in the dead volume is re-expanded during the suction stroke as well. However, in a linear com-

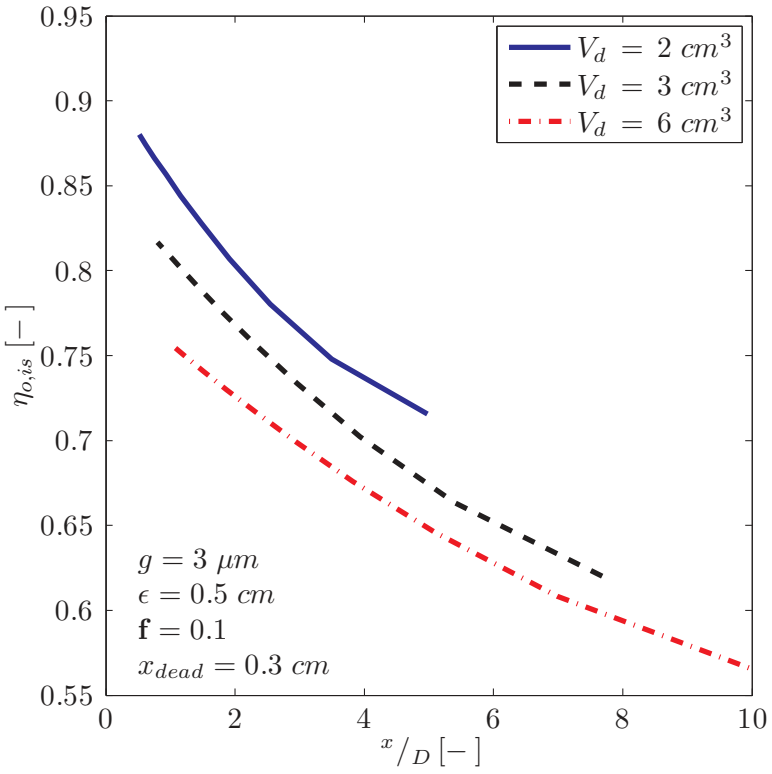


Figure 5.11. Overall isentropic efficiency as a function of the stroke-to-diameter ratio for three displacement volumes.

pressor the spring-mass-damper system of the free piston has the unique ability to almost fully recapture the energy required to compress the gas trapped in the dead volume. This ability to recapture the energy back into the piston mechanism during re-expansion translates into a net reduction in power consumption at the cost of a lower mass flow rate.

The overall isentropic efficiency trends also show that as the displaced volume is decreased the overall performance tends to increase. This suggests that as the compressor is scaled down for lower capacity applications the performance would tend to increase. This trend is a result of the net frictional losses decreasing as the displaced volume decreases; which shows the ability of the linear compressor to scale effectively to miniature-scale for electronics cooling applications. However, it is

acknowledged that this does not represent a comprehensive survey of the impacts of scaling as the miniaturization of the linear motor needs to be explored.

5.2.3 Dead Volume/Stroke Control

To simulate a variation in compressor capacity the compressor model is operated with varying amounts of dead volume. This is used to simulate a compressor with a variable stroke. It is acknowledged that a linear compressor with a fixed cylinder size would change capacity by reducing the compressor stroke. However, since varying the stroke will also change the calculated resonant frequency it becomes difficult to directly compare the compressor performance metrics from one stroke input to the next, as they depend on the stroke and operating frequency. Thus, an increasing compressor dead volume is used to represent a decreasing compressor stroke. The same piston diameter and stroke is used as in the spring eccentricity and leakage gap studies of 1.24 cm and 2.54 cm respectively. This leads to a fixed displaced/swept volume for the compressor. A fixed value of spring eccentricity, motor efficiency, and leakage gap are 0.5 cm, 0.9, and 3 μm, respectively. The dead volume in the compressor is varied from 10% of displaced volume to 120% of displaced volume. In addition, to explore any influence of friction the dry friction coefficient, $\mathbf{f}$, was varied between 0.1 and 0.3.

Figure 5.12 shows the volumetric efficiency as a function of compressor dead volume. As the dead volume increases the volumetric efficiency decreases linearly. With 0.3 cm^3 dead volume the compressor operates at about 90% volumetric efficiency and at 3.6 cm^3 dead volume the compressor operates at about 5% volumetric efficiency. The three separate dry friction coefficients cannot be distinguished in Figure 5.12 as the volumetric efficiency is very weakly dependent on friction.

The overall isentropic efficiency is also explored as shown in Figure 5.13. At a dead volume of 0.3 cm^3 at each dry friction coefficient the efficiency is maximized. The overall isentropic efficiency decreases slightly until a dead volume of roughly 2.5

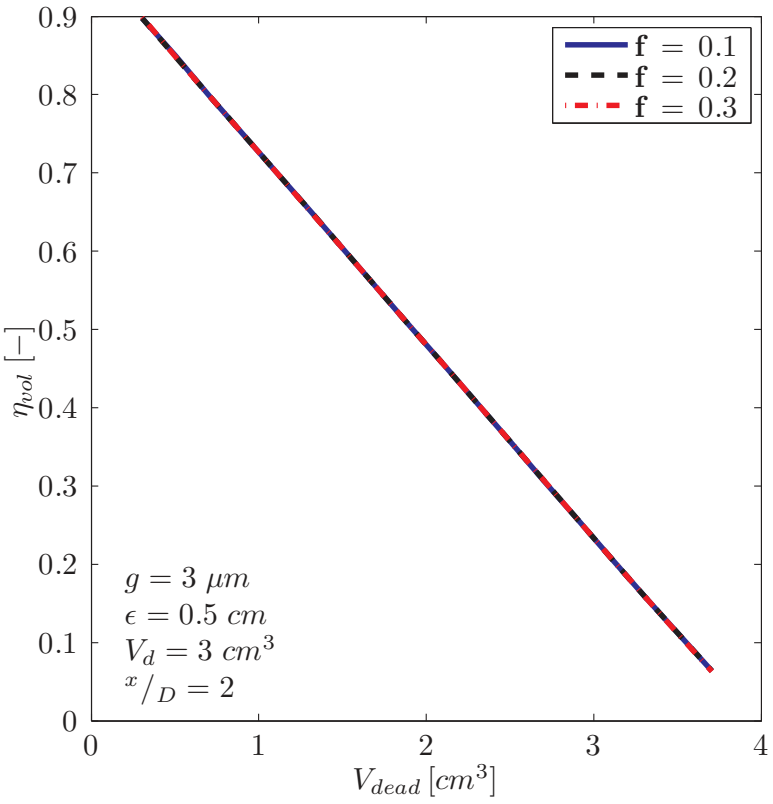


Figure 5.12. Volumetric efficiency as function of compressor dead volume for three dry friction coefficients.

cm^3 where the overall isentropic efficiency degrades rapidly. It can also be noted that as the dry friction coefficient decreases the overall performance increases.

This study yielded results which show that a linear compressor behaves different from a typical positive displacement compressor. Figure 5.12 shows that as the dead volume increases the volumetric efficiency decreases. This is a result of a decay in the amount of massflow provided by the compressor as the dead volume increases. The cause of this relates to the change in volume required to compress a fixed mass of gas. This can be seen by examining an idealized compression process, modeled by a polytropic process:

$$PV^n = const. \tag{5.5}$$

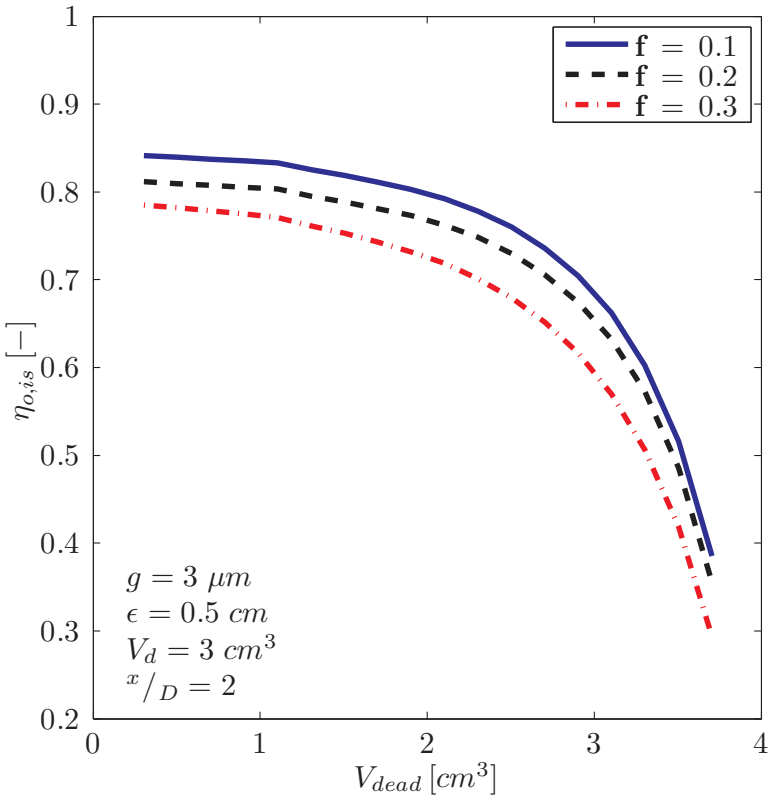


Figure 5.13. Overall isentropic efficiency as a function of compressor dead volume for three dry friction coefficients.

If the compressor is attempting to operate between two pressures P_1 and P_2 then the following can be written:

$$\frac{P_2}{P_1} = \left(\frac{V_1}{V_2}\right)^n \tag{5.6}$$

then introducing a definition for the volumes that includes the dead volume, and moving the polytropic exponent to the other side yields,

$$\left(\frac{P_2}{P_1}\right)^{(1/n)} = \frac{\pi r_p^2 x_{BDC} + V_{dead}}{\pi r_p^2 x_{TDC} + V_{dead}}. \tag{5.7}$$

The volume ratio which defines the ratio required to generate P_2 starting with P_1, defined as,

$$V_r = \frac{A_p x_{BDC}}{A_p x_{TDC}} = \frac{x_{BDC}}{x_{TDC}} \tag{5.8}$$

because the piston area is fixed. Combining Equations (5.7) and (5.8) and solving for the volume ratio one obtains:

$$V_r = \left(\frac{P_2}{P_1}\right)^{(1/n)} + \frac{V_{dead}}{\pi r_p^2 x_{TDC}} \left[\left(\frac{P_2}{P_1}\right)^{(1/n)} - 1\right]. \tag{5.9}$$

This expression shows that as the dead volume increases from zero (dead volume cannot be less than zero) that the volume ratio required to obtain a certain pressure rise increases proportionally. Therefore, for a fixed volume and pressure ratio as the dead volume increases the massflow obtained will decrease proportionally.

Therefore, one would expect that the overall isentropic efficiency to follow a similar trend. However, as seen in Figure 5.13 the trend is not linear. As the dead volume increases from 0.3 cm^3 to about 2.5 cm^3 the overall efficiency shows only minor degradation. However, Figure 5.12 shows that the massflow is drastically reduced with the same change in dead volume. Thus, it can be concluded that the compressor produces less massflow rate at the higher dead volume but also requires less power. This behavior can then be exploited if the system level performance is considered, as presented in the following section.

5.3 Capacity Control

A reduction in massflow translates to a reduction in refrigeration capacity. By analyzing the refrigeration system for electronics cooling as shown in Figure 5.14 the system cooling capacity can be calculated as follows:

$$\dot{Q}_{cool} = \dot{m}(h_1 - h_4). \tag{5.10}$$

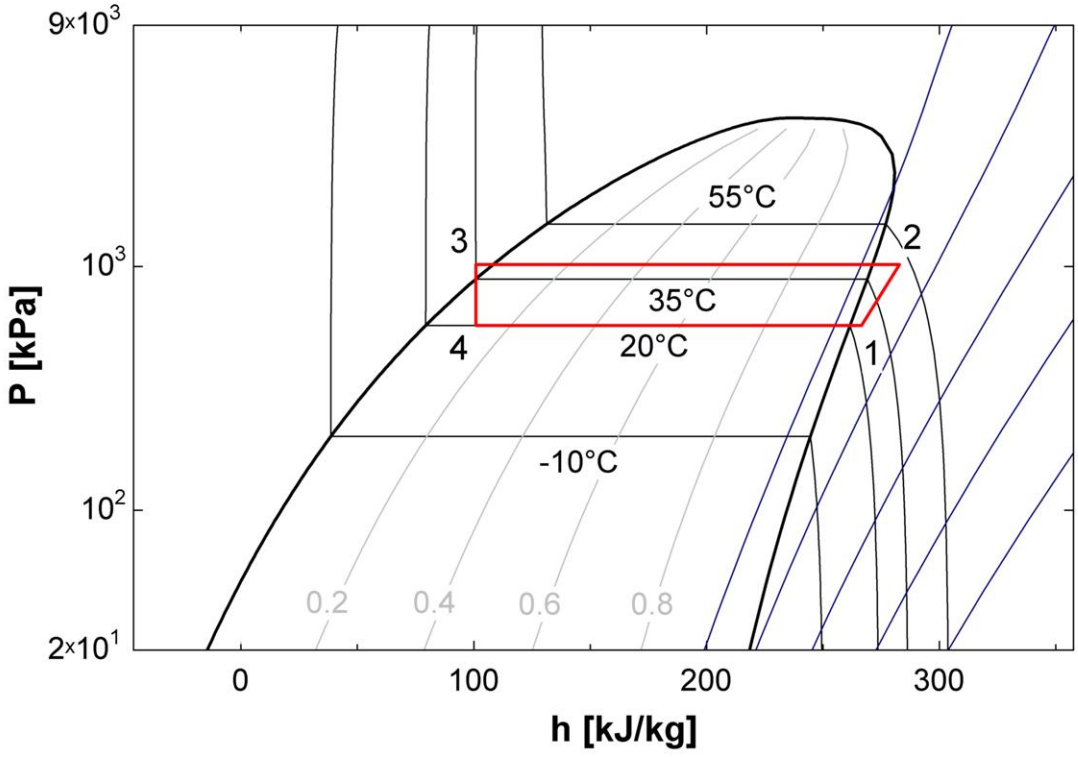


Figure 5.14. Pressure enthalpy diagram of typical R134a miniature-scale refrigeration cycle for electronics cooling.

If the system operates under fixed environmental conditions the enthalpies entering and exiting the system evaporator (h_4 and h_1) will remain constant, where h_4 is calculated using the following relationship:

$$h_4 = h_3 = h(T = T_{sat} - \Delta T_{sub}, P_2) \tag{5.11}$$

where ΔT_{sub} is assumed to be 10 $^{\circ}C$. This means that the system cooling capacity is proportional to the massflow generated by the compressor. The power required to drive the compressor is calculated using the comprehensive linear compressor model, and the system Coefficient of Performance (COP) is defined as:

$$COP = \frac{\dot{Q}_{cool}}{\dot{W}_{in}}. \tag{5.12}$$

For relative comparison to a reversible refrigeration cycle the COP of a Carnot refrigeration cycle is utilized.

$$COP_{carnot} = \frac{T_{evap}}{T_{cond} - T_{evap}}. \tag{5.13}$$

Using the Carnot COP the second law effectiveness of the refrigeration cycle is defined as follows:

$$\varepsilon_{carnot} = \frac{COP}{COP_{carnot}}. \tag{5.14}$$

Using the results from Section 5.2.3 as an input with a dry friction coefficient of 0.1 a variable capacity system can be simulated using the variable massflow simulated by increasing the compressor dead volume. Figure 5.15 shows the predicted system COP and second law effectiveness ε_{carnot} as a function of the cooling capacity.

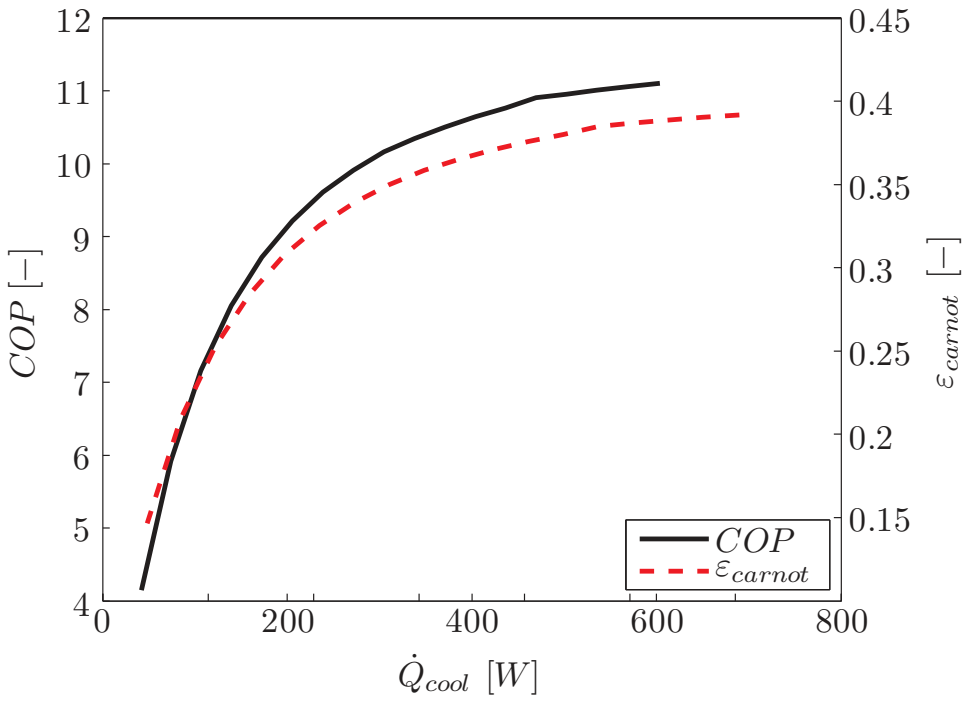


Figure 5.15. Coefficient of Performance and second law effectiveness of a R134a refrigeration cycle operating at a typical electronics cooling condition with 10 °C condenser subcooling with variable capacity predicted by linear compressor model.

At the smallest cooling capacity the linear compressor is producing the smallest massflow rate, which translates to the largest dead volumes in Figures 5.12 and 5.13. As the cooling capacity increases, the dead volume decreases and subsequently the massflow increases. In addition, the overall efficiency of the compressor also increases as the dead volume decreases which generates a better system COP. Figure 5.15 shows that an electronics cooling system can provide a reasonably high system COP over a wide range of cooling capacities. This simulated compressor operates relatively well from roughly 200 W to over 600 W which would provide a sufficient amount of variation to maintain consistent cooling to a future consumer electronic device based on the predictions from the International Roadmap of Semiconductors (2011 ed.). This is another useful feature of the linear compressor for an electronics cooling application.

5.4 Piston Drift

A unique behavior observed from the sensitivity studies is the tendency for the piston to shift its equilibrium, or center of oscillation, during operation. In each initiation of the model the piston begins at rest at an equilibrium position $x_{e,1}$, this is shown in the upper schematic diagram in Figure 5.16. During operation the piston will tend to shift away from the valves to a new equilibrium location, $x_{e,2}$. Therefore, we define the drift of the piston as:

$$Drift = x_{e,1} - x_{e,2}. \tag{5.15}$$

The piston drift depends on several factors, one being the amount of gas in the cylinder and another being the size of the piston. Figure 5.17 shows model results from Sections 5.2.2 and 5.2.3 and the amount of drift the model predicted. The dead volume, or amount of gas in the cylinder, appears to show some relationship to drift as the drift tends to increase as the dead volume increases. However, these changes

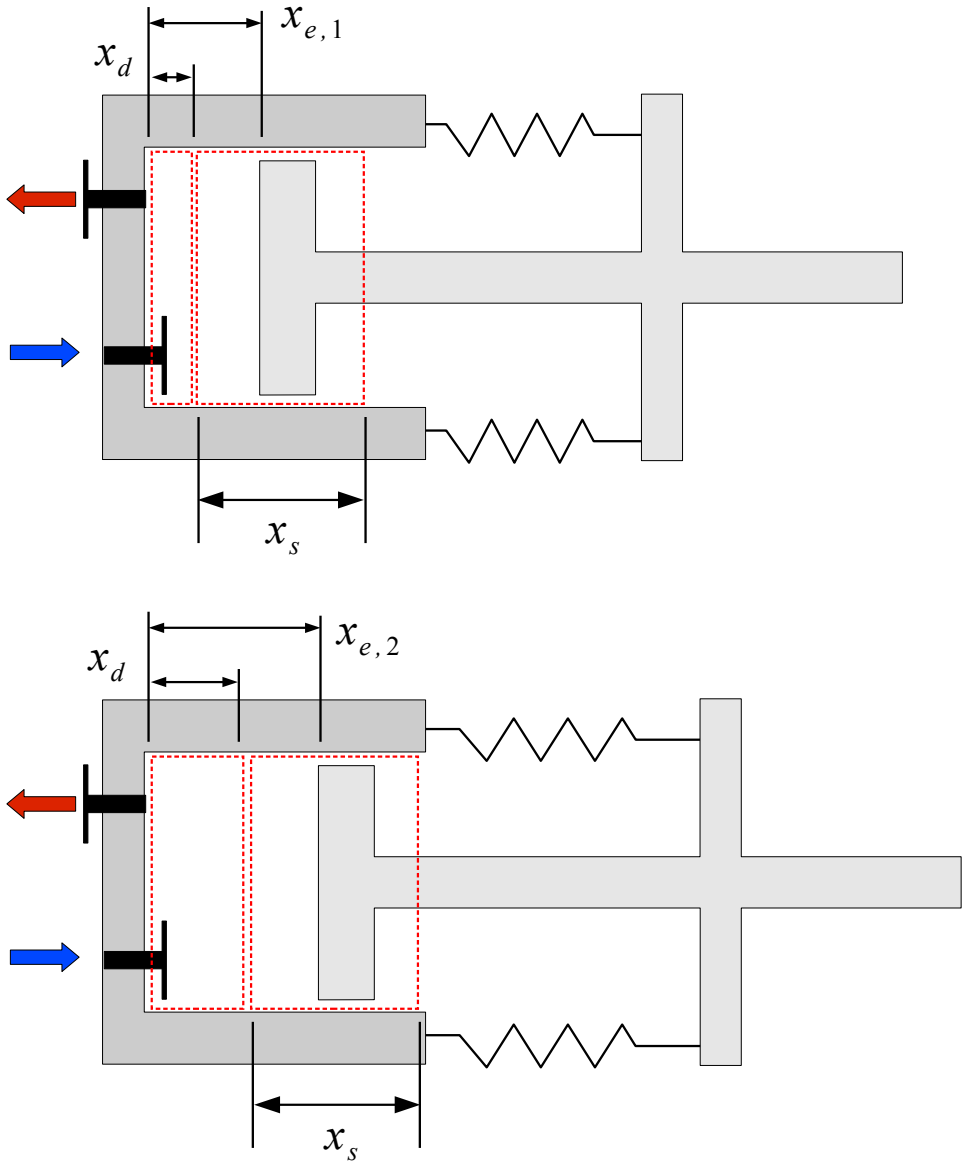


Figure 5.16. Schematic diagram of linear compressor showing the equilibrium position of the compressor piston at two different conditions.

are relatively small compared with the changes due to the piston size in the geometric studies.

The piston drift as a function of the stroke-to-diameter ratio shows a strong functional dependence. At low stroke-to-diameter ratios the piston diameter tends to be large, and the amount of drift is also large. At large stroke-to-diameter ratios the

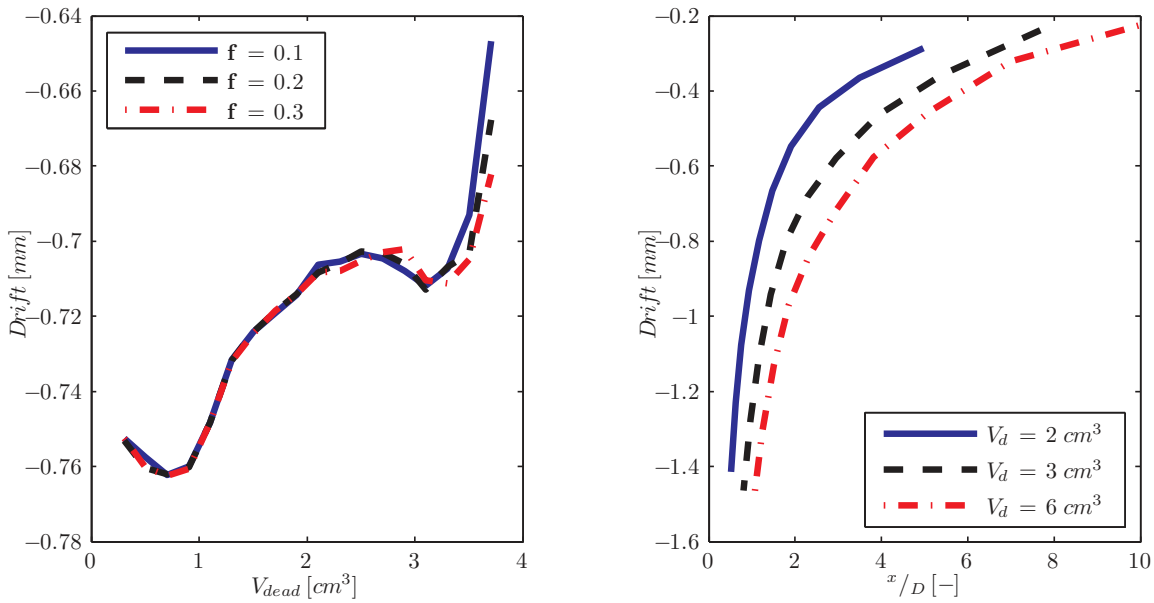


Figure 5.17. Amount of drift as a function of dead volume (left) and stroke to diameter ratio (right).

piston diameter decreases and drift also decreases. Therefore, it is likely that the drift depends on the piston diameter as a larger piston diameter corresponds to a larger amount of drift. When the piston diameter is large the force acting on the piston due to pressure in the cylinder is also large. Assuming that the net pressure differential across the piston does not change drastically for a variety of piston diameters, this generates a higher net force on the piston. The higher net force generates a higher amount of drift of the compressor piston.

While the piston drift was not a focus of characterization in this study, it is an important behavior to consider as a piston with a relatively large piston area will behave with more severe drift. If the drift amount begins to approach the same order of magnitude as the stroke, then piston stability over a wide operating band will be compromised.

5.5 Comparison of a Linear Compressor to Reciprocating Compressor Behavior

The unique behavior of a linear compressor shown in the previous sections serves as the motivation to compare a linear compressor to a more well-known compressor technology such as a reciprocating compressor. A reciprocating compressor provides a good benchmark for comparison to a linear compressor because of the unique ability to directly compare the compression mechanism of the two devices. Fundamentally, a linear and reciprocating compressor are very similar technologies as they are both piston-cylinder devices. However, due to the kinematics of a reciprocating compressor the displaced volume is dictated by the design of the compressor and is not a function of the operating conditions or power input. Figure 5.18 shows a kinematic diagram of a reciprocating compressor driven with a torque T and rotational speed ω.

From this, and assuming the compressor components act as rigid bodies, an expression for the motion of the piston as a function of the crank angle may be written as:

$$x_p = L_1 \cos(\beta_1) + \sqrt{L_2^2 - L_1^2 \sin^2(\beta_1)}. \tag{5.16}$$

Differentiating this expression twice one obtains an expression for the acceleration of the piston may be attained:

$$\ddot{x}_p = -L_1\omega^2 \cos(\beta_1) - \frac{L_1^2\omega^2 \cos(2\beta_1)}{\sqrt{L_2^2 - L_1^2 \sin^2(\beta_1)}} - \frac{L_1^4\omega^2 \cos(2\beta_1)}{2\sqrt[3]{L_2^2 - L_1^2 \sin^2(\beta_1)}}. \tag{5.17}$$

Additionally, an expression for the equation of motion of a reciprocating compressor can be constructed using the free body diagram given in Figure 5.18:

$$\underbrace{M_{mov}\ddot{x}_p}_{\text{Intertia}} + \underbrace{\mathbf{f}F_2 \sin(\beta_2)}_{\text{Damping}} + \underbrace{k_{gas}x_p}_{\text{Stiffness}} = F_2 \cos(\beta_2) \tag{5.18}$$

and noting that,

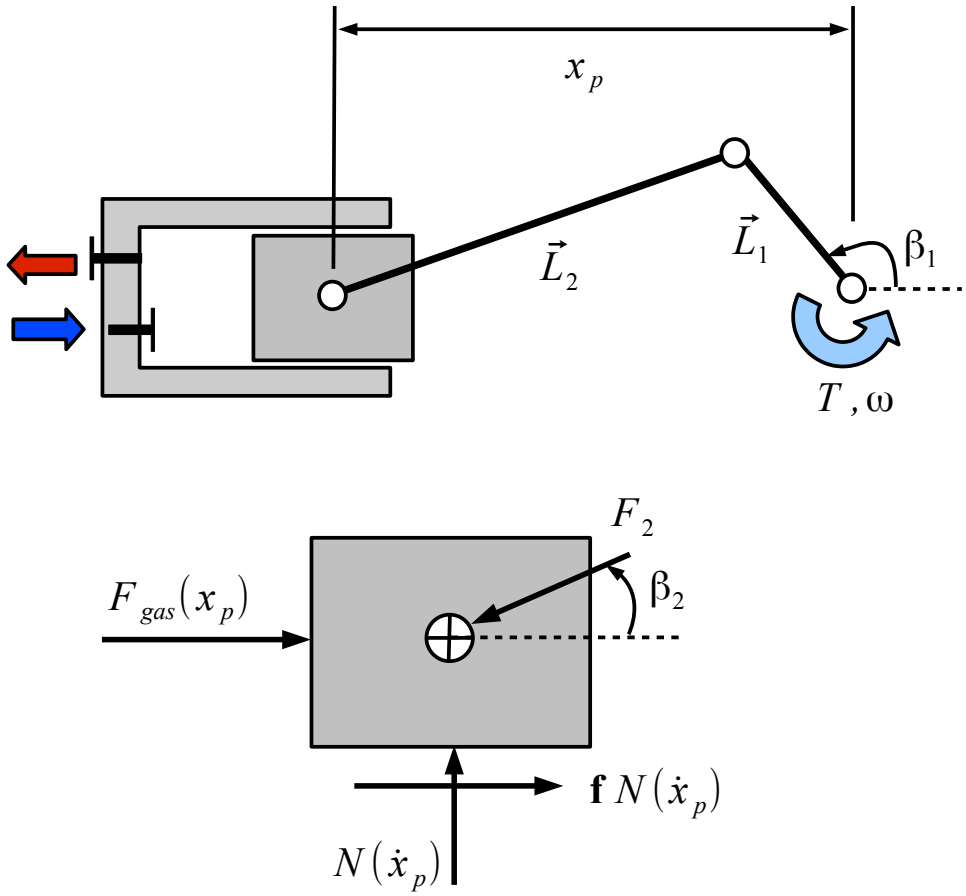


Figure 5.18. Schematic diagram of reciprocating compressor mechanism and free body diagram of compressor piston assuming dry friction contact between piston and cylinder wall.

$$k_{gas} x_p = P_{cv} A_P. \tag{5.19}$$

This equation shows that a reciprocating compressor displays a similar equation of motion compared with the linear compressor. The reciprocating compressor also has inertial, damping, and stiffness terms associated with it. However, unlike a linear compressor a reciprocating compressor has a stiffness that is only associated with the gas compression and is thus much smaller than for a linear compressor (*i.e.* $k_{linear} >> k_{recip}$). The result of this small stiffness term makes it very difficult

to operate a reciprocating compressor at a resonant frequency as it would require operation at a slower speed than A/C rotary motor technology typically operates at (30 - 60 Hz). In addition, the leakage in a compressor is adversely impacted by the result of slowing operational speed which is another reason why operating at lower speeds is typically avoided.

Equation (5.18) is further expanded to determine the required torque from the crank, realizing that β_2 can be written in terms of β_1 as follows:

$$\tan\beta_2 = \frac{\left({}^{L_1}/_{L_2}\right)\sin\beta_1}{\sqrt{1-\left[\left({}^{L_1}/_{L_2}\right)\sin\beta_1\right]^2}}. \tag{5.20}$$

Then the torque required for piston movement can be calculated by solving the equation of motion described in Equation (5.18) for F_2 and using geometry to solve for the torque at the crank:

$$\mathbf{T} = (1+\mu)\left[(M_{mov}\ddot{x}_p + k_{gas}x_p)x_p\tan\beta_2\right]. \tag{5.21}$$

The power required to drive the reciprocating compressor is then given as:

$$\dot{W}_{in,recip} = T\omega. \tag{5.22}$$

Using these calculations a direct comparison between a reciprocating and a linear compressor can be made. With respect to leakage, a linear compressor behaves similar to a reciprocating compressor and are not considered in this comparison. The frictional losses, are also not considered here and the same friction coefficient, **f** of 0.1, is used for the analysis of the reciprocating compressor and linear compressor. Therefore, the sensitivity studies presented in Sections 5.2.1 and 5.2.2 are not reproduced here with reference to reciprocating compressor performance as no notable comparison can be made. However, changes with respect to dead volume, or capacity control, are presented as this displays one of the significant differences between a reciprocating compressor and a linear compressor.

The variation in dead volume for the reciprocating compressor analysis matches that of a linear compressor. It was observed that the volumetric efficiency follows the same trend for both compressors. Therefore, the trends predicted in Figure 5.12 are the same for both compressors as each will tend to lose mass flow as dead volume increases for the reasons explained in Section 5.2.3. However, when comparing the overall isentropic efficiency, as shown in Figure 5.19 one can see that as the dead volume increases the overall isentropic efficiency of a reciprocating compressor tends to decrease linearly as with the volumetric efficiency.

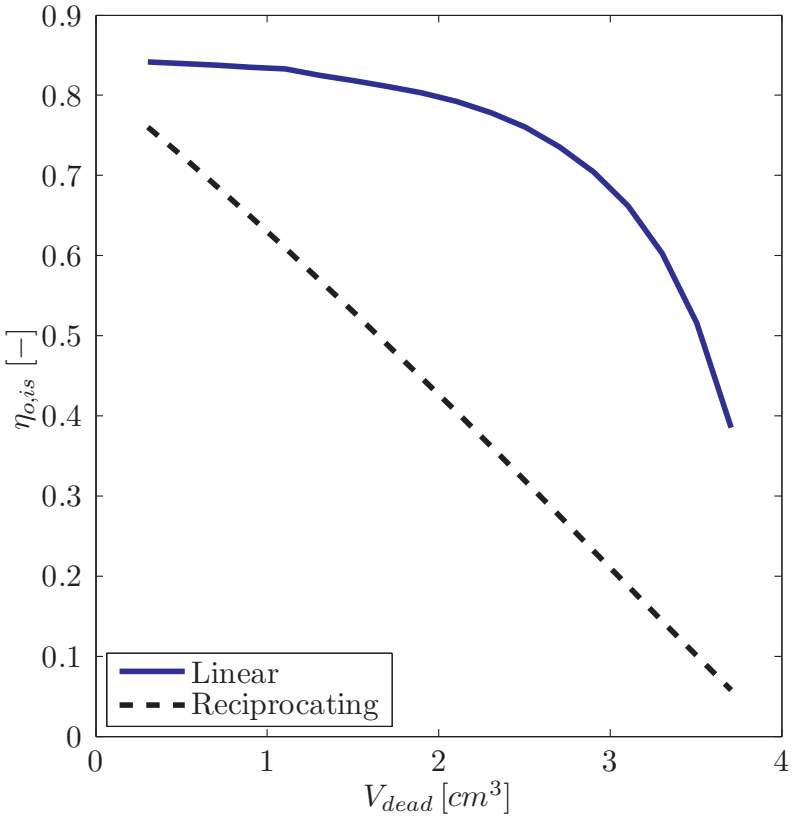


Figure 5.19. Comparison of overall isentropic efficiency as a function of compressor dead volume for a linear compressor compared with a reciprocating compressor.

The reason for the difference is noted in the equations of motion of the two compressors shown in Equations (2.19) and (5.18), where the stiffness term in the linear compressor is much larger due to the mechanical springs. The stiffness in a mechanical system is analogous to a capacitance in an electrical system, and provides a mechanism for a mechanical system to store energy. Therefore, the linear compressor has a higher ability to store energy, or mechanical capacitance. This proves useful when the compressor operates with a variable stroke (or with variable dead volume) as the linear compressor can recapture energy imparted to the gas. The overall isentropic efficiency of the reciprocating compressor decreases linearly with dead volume. This is a result of the power required to drive the compressor mechanism remaining relatively constant, but the net massflow delivered decreasing, as dead volume increases. Conversely, the linear compressor still shows the same reduction in mass flow, but as a result of the high amount of mechanical capacitance, it is able to recapture some of the power typically lost during gas re-expansion and thus operate at a lower power input at smaller capacity levels (*i.e.*, partial load).

5.6 An Improved Design of a Miniature-Scale Linear Compressor for Electronics Cooling

Based on the results obtained in this work, an improved linear compressor design is formulated for an electronics cooling application. A compressor for such an application should be compact, as package size in electronic equipment is often a constraint. The performance of the previous prototype design analyzed in Chapter 4 and detailed in Appendix A can be readily improved, at a lower total compressor package size. The modified design presented here provides an improved overall efficiency and reduced package size.

From the leakage and friction sensitivity studies it was concluded that both the leakage (leakage gap g) and frictional parameters (spring eccentricity ϵ, and dry friction coefficient $\mathbf{f}$) should be minimized. However, practical limitations prohibit the

reduction of these parameters to zero. Therefore, a realistic set of values was selected based on input from compressor manufacturers and previous studies on compressor modeling (Dagilis and Vaitkus, 2009; Kim et al., 2009). A leakage gap width of 4 μm was selected as a lower limit on a clearance fit for a mass produced product. A dry friction coefficient of 0.2 represents a reasonable and low value for an engineering polymer (e.g. PEEK, PTFE, or Rulon) and steel. A value for the spring eccentricity of 0.5 cm also appears to be realistically attainable; lower values may be manufacturable and would correspondingly alter the design. Table 5.1 summarizes these values as well as additional design parameters for the linear compressor.

Table 5.1. Key design parameters of the improved linear compressor design compared with the prototype design.

	k_{mech}	$\mathbf{f}$	D_p	g	f_{res}	ϵ
	$N\ mm^{-1}$	$-$	cm	μm	Hz	cm
Modified Design	30.6	0.2	1.35	4	60	0.5
Prototype Design	23	0.35	1.217	13	44.5	1.145

Similar to the design presented in Chapter 4, the improved linear compressor design utilizes an off-the-shelf motor from HW2 Technologies. First, a cooling capacity of 200 W was selected to represent a desktop computer cooling application. The comprehensive compressor model was then run using the design parameters listed in Table 5.1 with various stroke-to-diameter ratios necessary to achieve the required 200 W of cooling. The required stroke-to-diameter ratio and the corresponding continuous driving force dictate the choice of an appropriate H2W Technologies motor (P/N NCM02-17-035-2F). The modeling results from the present work show an increase in compressor performance with a decrease in the stroke-to-diameter ratio. However, the force required to operate the compressor piston increases with a decrease in this ratio. An increase in force requires a larger linear motor, which translates into a larger overall package size. Thus, there is a trade-off between package size and performance. For the current design, a stroke-to-diameter ratio of 0.4 leads to an acceptable overall package

size. The proposed design of the improved miniature linear compressor is shown in Figure 5.20. The overall performance of the compressor as well as the corresponding refrigeration system performance are listed in Table 5.2. While the predicted cooling capacity of the linear compressor prototype is lower than with the previous design from Chapter 4, the capacity-to-volume ratio has increased significantly. Therefore, this design lends itself to greater miniaturization of linear compressors.

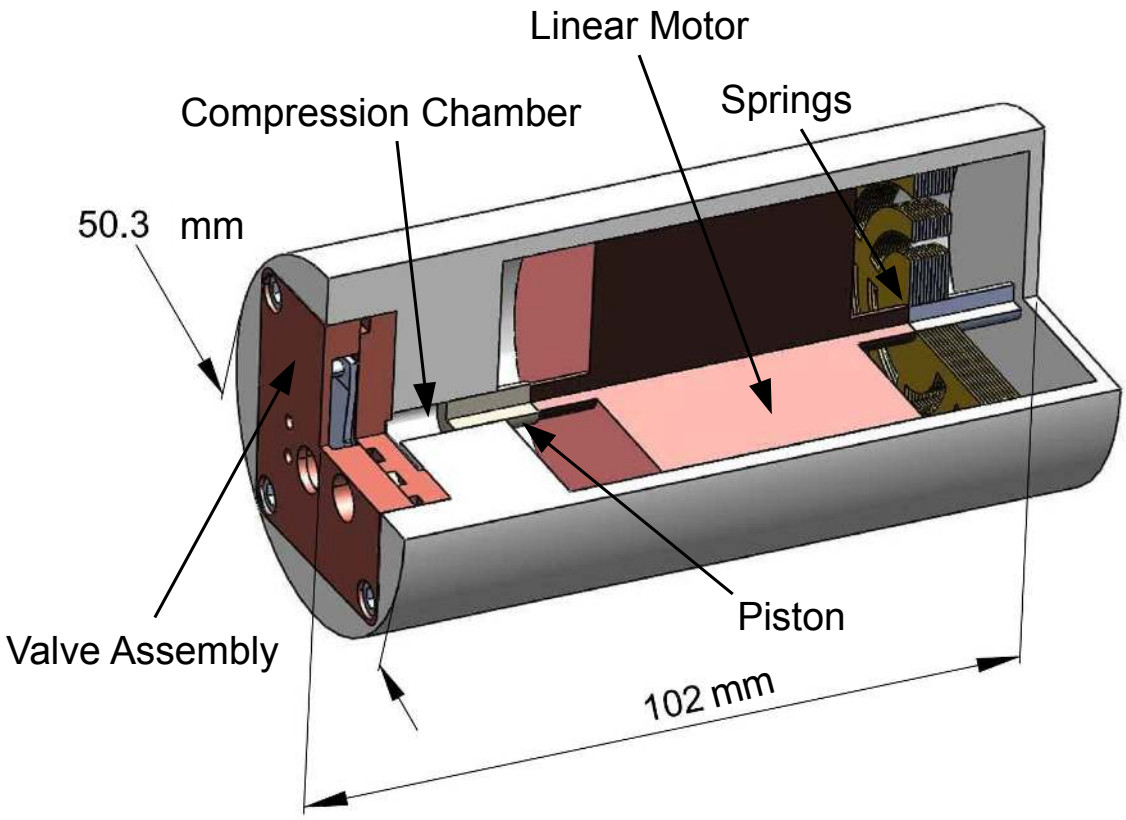


Figure 5.20. Improved linear compressor design with predicted cooling capacity of 200 W.

5.7 Conclusions

The sensitivity of the performance of a linear compressor to changes in its geometric parameters is analyzed, leading to insights into the design methodology of linear

Table 5.2. Predicted performance of the improved linear compressor design compared with the prototype design.

	V_d	$^x/_D$	η_{vol}	$\eta_{o,is}$
	cm^3	–	–	–
Modified Design	2	0.4	0.96	0.86
Prototype Design	3.09	2	0.4	0.08

compressors. These insights allow for a further refinement of the design prototype presented in Chapter 4. In addition, a loss analysis is presented which quantifies the work lost due to friction and leakage.

The sensitivity studies conducted show that the linear compressor is highly sensitive to changes in the leakage gap between the piston and cylinder as well as the spring eccentricity; both parameters should be minimized for optimal performance. Therefore, it is important to quantify and control these parameters in any compressor design that is mass-produced to maximize performance.

This chapter also illustrates the ability of the linear compressor to be readily scaled to smaller capacities. Other types of positive-displacement compressors suffer from performance limitations upon miniaturization due to manufacturing tolerances. The small number of moving parts in the proposed linear compressor design, along with its insensitivity to dead volume, make it an ideal technology for electronics cooling applications. The ability to handle larger amounts of dead volume without performance degradation could also allow this technology to be used to control the capacity of the refrigeration system. Additional interesting behavior called piston drift is noted as a phenomenon that should be avoided to maximize design robustness. Also, the linear compressor is compared to a reciprocating compressor. This study lead to the discovery that a linear compressor has the advantage of efficient capacity control compared with a reciprocating compressor.

The results from the sensitivity analysis are used to inform the scalable design procedure for a linear compressor. An improved linear compressor design for elec-

tronics cooling is presented with an overall package size of 50.3 mm diameter by 102 mm length and a predicted refrigeration capacity of 200 W.

CHAPTER 6. MODELING OF A COMMERCIAL LINEAR COMPRESSOR

At the commencement of this study there were no commercially available linear compressors for testing purposes. However, recently one has become available from LG in a domestic refrigerator/freezer. The refrigerator/freezer was purchased (LGC25776SW) to test its linear compressor. This refrigerator also provides the additional feature of capacity control which means the compressor remains operational for longer when the refrigerator cycles on. This feature will be utilized later in the chapter when estimating the compressor stroke. This chapter details modifications made to the comprehensive linear compressor model presented in Chapter 2 as well as the experimental testing of the commercial linear compressor. Finally, the modified comprehensive linear compressor model results are compared with the experimental results from the commercial linear compressor. The chapter begins by examining the compressor design and operating mechanism.

6.1 Description of Commercial Linear Compressor

The commercial linear compressor is a hermetic compressor used to operate a domestic refrigerator/freezer. The operating mechanism is the same as presented in previous chapters with refinements for cost and/or performance benefit. Figure 6.1 shows a full schematic diagram of the commercial linear compressor. This diagram highlights the major components of the commercial linear compressor including the linear motor, piston, suction muffler, discharge valve, and discharge muffler. Figure 6.2 shows a detailed view of the rear of the compressor with the shell removed and highlights the location of the suction muffler, mechanical compression springs, and mechanical spring retainers.

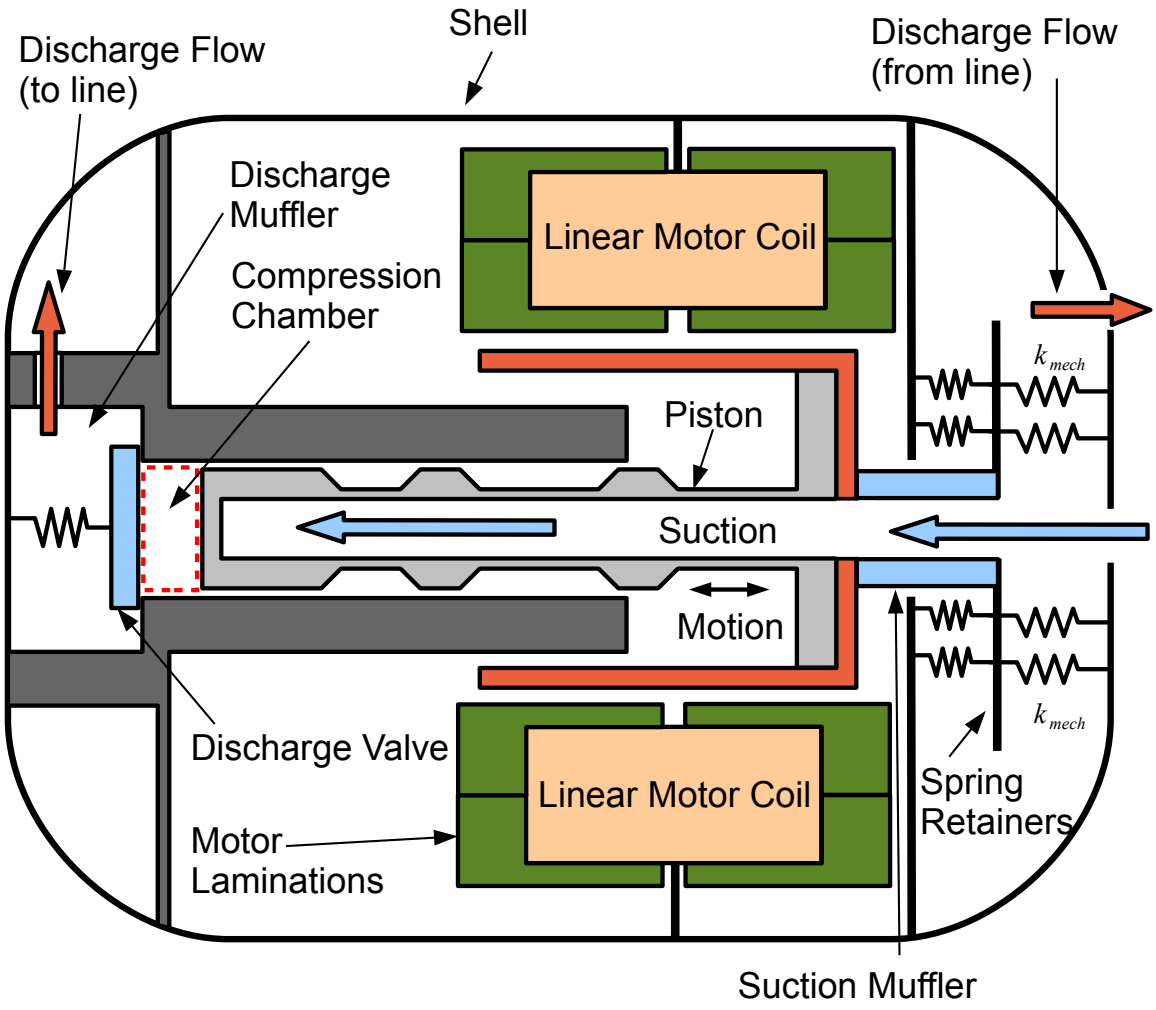


Figure 6.1. Schematic diagram of a commercial linear compressor (LG) highlighting the major components of the design and flow of refrigerant through the compressor.

The linear compressor operates with a low-side shell, meaning the hermetic shell is pressurized at the suction pressure. The suction gas from the refrigerator/freezer evaporator empties directly into the compressor shell. This generates some gas movement within the shell which keeps the motor cool. The suction gas travels from the shell of the compressor into the suction muffler, then into the center of the piston. As shown in Figure 6.1, the suction gas travels from the center of the piston and exits at

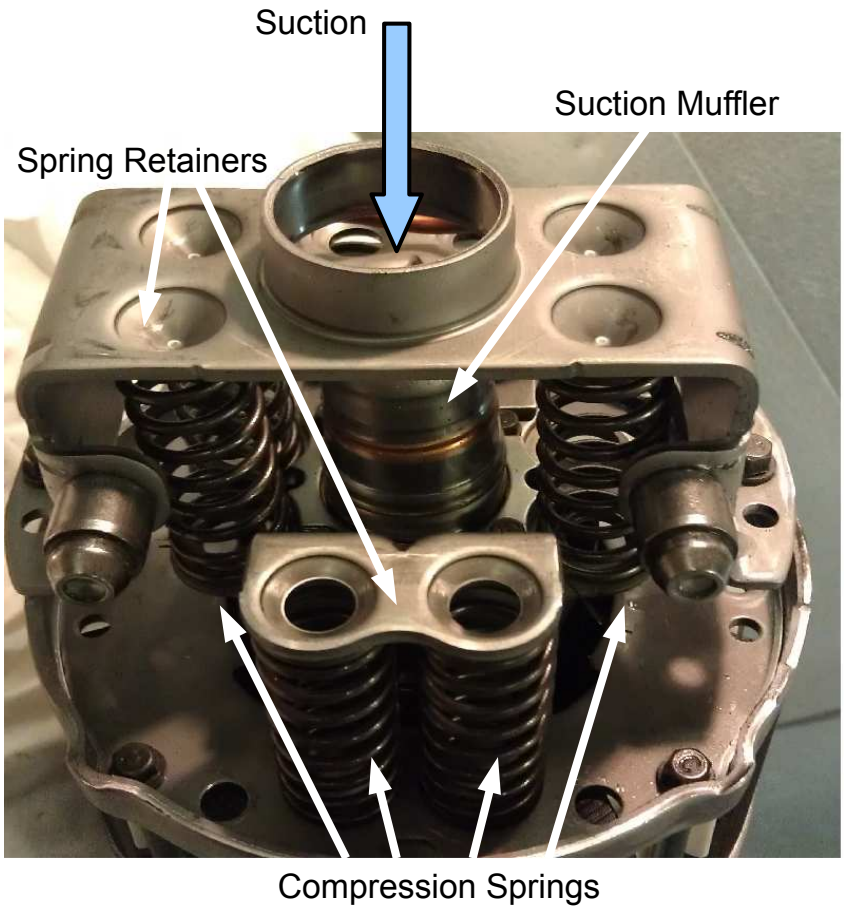


Figure 6.2. Rear assembly of commercial (LG) linear compressor, with hermetic shell removed, showing the array of mechanical springs and the entrance to the compressor suction muffler.

the end of the piston where it passes through the suction ports via the suction reed valve.

Figure 6.3 shows the piston within the compressor cylinder. The suction reed valve is clearly shown as well as outlines for the three suction ports. As gas passes through the suction ports and past the valves it travels into the compression chamber, shown in Figure 6.1. Figure 6.3 presents a view looking into the compression chamber.

Once compressed, gas pushes the discharge valve out of the way and enters a series of two discharge mufflers. Figure 6.4 shows where the gas exits the cylinder and enters the discharge muffler past the discharge valve. In this figure, the discharge assembly

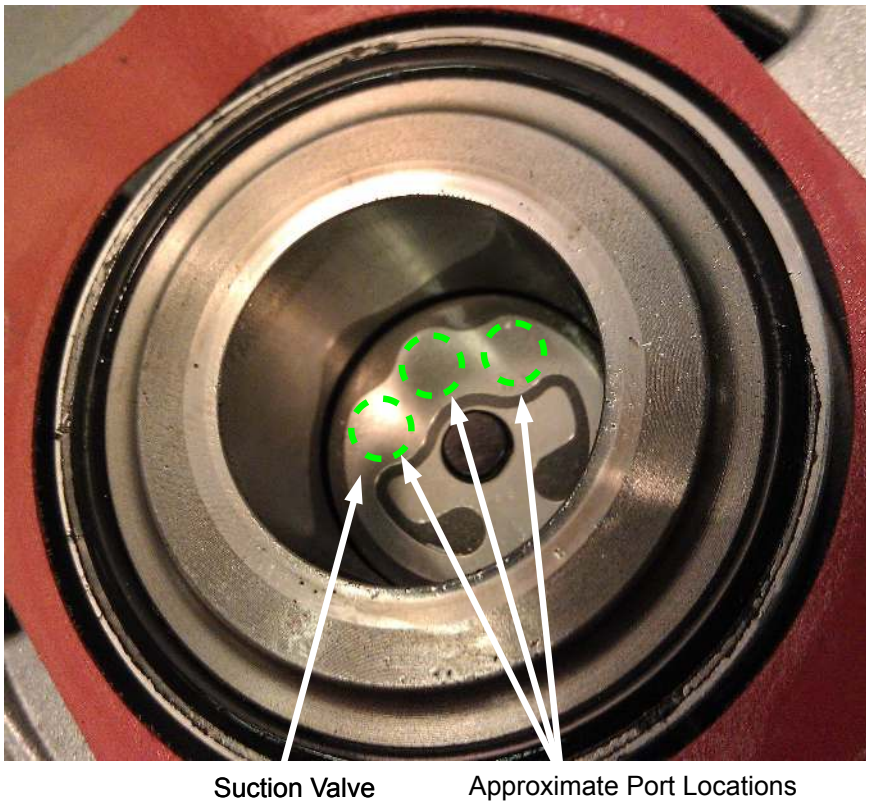


Figure 6.3. View of commercial linear compressor piston within the cylinder showing the suction valve.

has been removed from the cylinder head to display the discharge valve and piston cylinder. The gas then travels from the first discharge muffler into a second and finally into a discharge tube. This discharge tube takes a serpentine path from the second discharge muffler toward the rear of the compressor and exits the compressor shell near the suction line entrance. Figure 6.4 also shows part of the linear motor assembly. The darker pieces on the motor in the figure are the linear motor laminations which cover the copper coils. These laminations, which help generate the electromagnetic field, come in close proximity to the permanent magnet assembly shown attached to the piston in Figure 6.1.

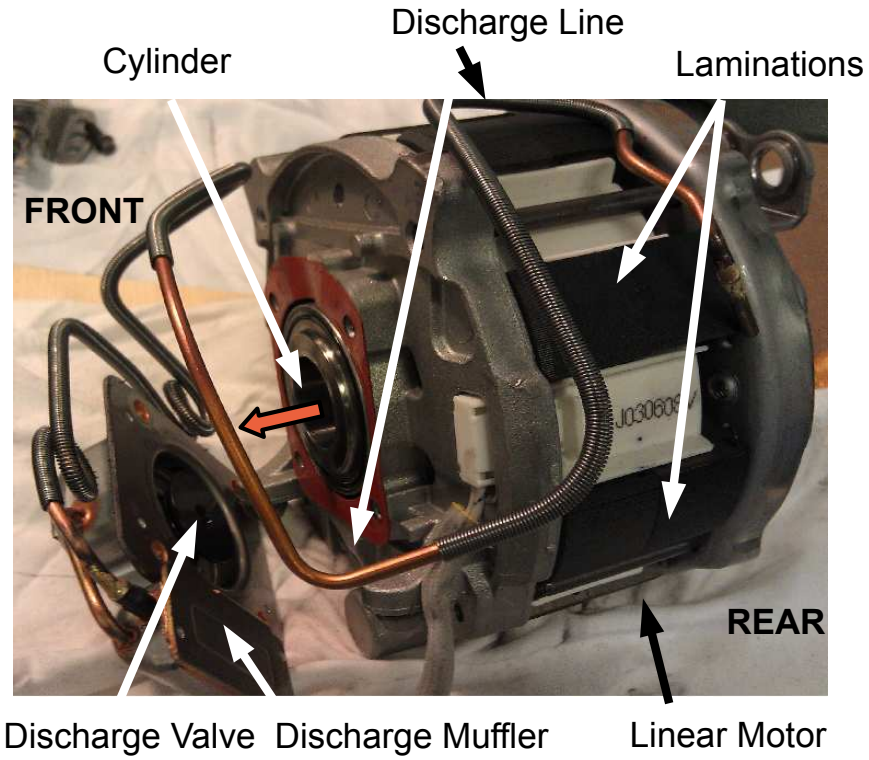
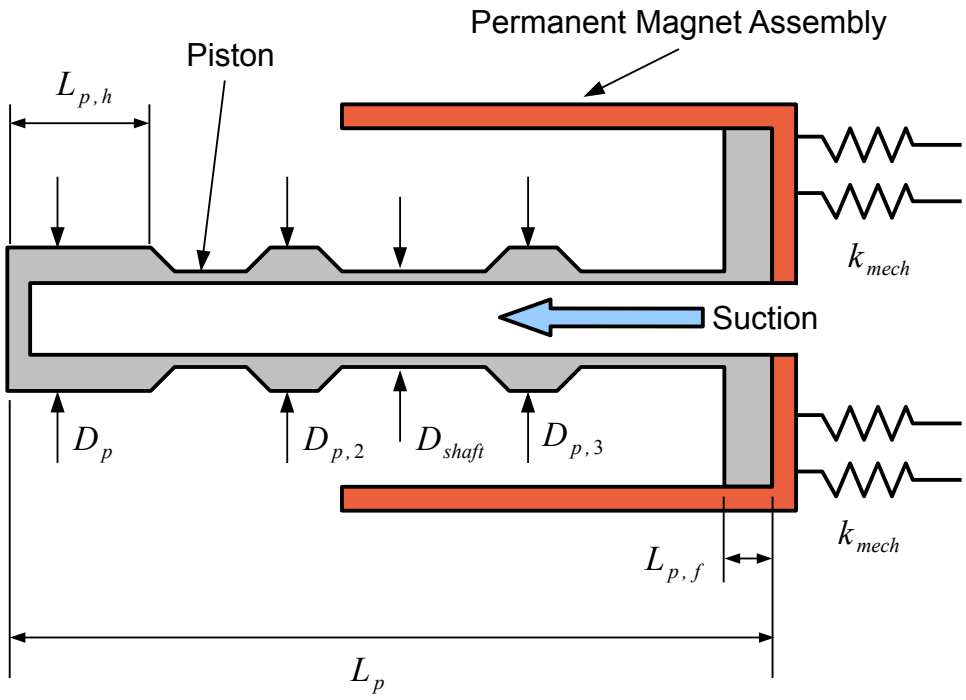


Figure 6.4. Overall view of the commercial linear compressor with the discharge valve and muffler assembly removed.

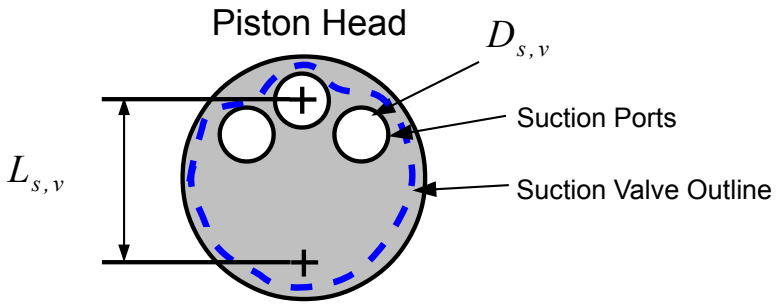
6.1.1 Measurement of Compressor Geometry and Other Comprehensive Compressor Model Inputs

During dis-assembly many key dimensions of the commercial linear compressor were gathered. These included the dimensions of the compressor piston, cylinder, and valves. Additionally, the masses of the compression springs, piston assembly, and discharge valve were measured. Table 6.1 shows the results of the measurements with reference to Figure 6.5 which shows the compressor piston with key dimensions highlighted. The linear dimensions were collected using a Mitutoyo 500-341 digital caliper and mass measurements were made using a Acculab SV-30 digital scale.

The mechanical spring rate could not be measured directly; thus an experimental method for measuring the mechanical spring rate of the compression springs was developed. Figure 6.6 shows the test apparatus which is composed of one of the



(a) Side view of compressor piston.



(b) Front view of compressor piston.

Figure 6.5. Schematic diagrams of commercial linear compressor piston with key geometric dimensions highlighted.

compressor spring retainer plates from the commercial linear compressor with 4 compression springs attached and pressed against a steel tabletop.

Lead weights were incrementally added to the apparatus to add force to the compression springs. At each incremental amount of load the height between the apparatus and the steel table was measured. Figure 6.7 shows the results of this experiment as a plot showing applied load to the springs as a function of the change in height.

Table 6.1. List of measured quantities from the commercial linear compressor referencing the nomenclature in Figure 6.5

Measurement	*Value*	*Unit*
D_p	2.6505(1.0435)	cm (in)
L_p	7.9629(3.135)	cm (in)
$L_{p,h}$	1.4478(0.5700)	cm (in)
$L_{p,f}$	0.6375(0.2510)	cm (in)
$D_{p,2}$	2.6378(1.0385)	cm (in)
$D_{p,3}$	2.6518(1.0440)	cm (in)
D_{shaft}	2.6212(1.0320)	cm (in)
$D_{s,v}$	0.5207(0.2050)	cm (in)
$D_{d,v}$	2.9489(1.1610)	cm (in)
$L_{s,v}$	1.8491(0.7280)	cm (in)
$h_{s,v}$	0.1778(7.0)	mm $(thou)$
M_{piston}	0.632(1.39)	kg (lbm)
$M_{springs}$	0.164(0.362)	kg (lbm)
$M_{d,v}$	4(0.009)	g (lbm)
k_{mech}	58.66 (335)	${}^{N}/_{mm}$ $\left({}^{lbf}/_{in}\right)$
g	1(0.04)	μm $(thou)$

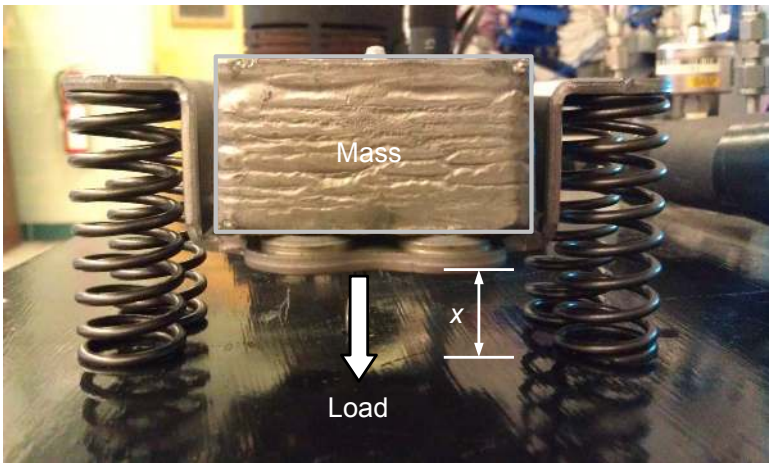


Figure 6.6. Test apparatus used to measure stiffness of mechanical springs in commercial linear compressor.

This figure shows that as the load is increased, the change in height of the springs decreases linearly. The slope of this curve describes the stiffness of the 4 compression

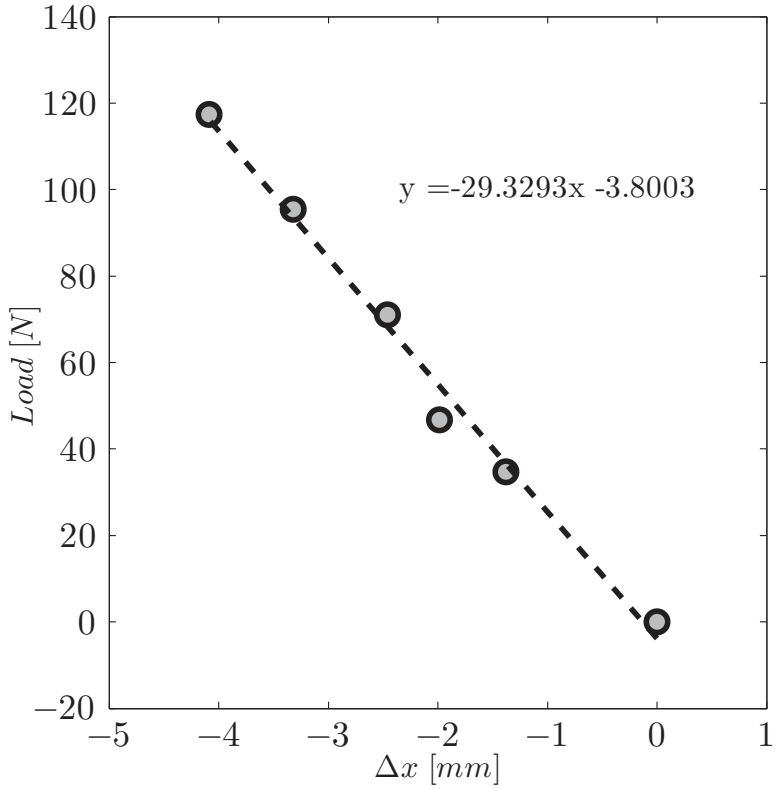


Figure 6.7. Load as a function of the change of displacement for four compression springs from the commercial linear compressor.

springs (29.33 $^{N}/_{mm}$). However, the commercial linear compressor utilized 2 opposing sets of 4 compression springs for a total of 8 compression springs in parallel, which means the total stiffness of the commercial linear compressor is double the value measured by considering only four springs (58.99 $^{N}/_{mm}$).

During dis-assembly of the compressor, it was also determined that the compressor utilized 160 mL of a synthetic refrigerant oil, which was drained from the compressor oil sump and measured for volume. While the type of oil is unknown, a typical pairing with R134a refrigerant is a POE oil of viscosity grade (ISO-VG) of ISO 32 or 68. Therefore, POE ISO 68 oil is assumed to be the lubricating medium in the remainder of this study.

6.2 Linear Compressor Model Modifications

The comprehensive model developed in Chapter 2 required some minor modifications to better represent the commercial linear compressor tested. In particular, since the commercial compressor utilized oil for lubrication a modification to the friction sub-model within the vibration sub-model is required. The geometry parameters measured in the previous section are also quite different compared with the prototype compressor, so these input parameters had to be modified. The latter set of changes did not require the modification of any of the governing equations.

6.2.1 Thin Film Friction Model

The friction modeling approach utilized is based on the shear force generated between two flat plates sliding past each other with a thin oil film between them. Beginning with the Navier-Stokes equations (*i.e.*, momentum equations) and assuming that the motion of the piston dominates the expression such that $\frac{dP}{dx} = 0$, a velocity profile of the oil film between the piston and cylinder can be modeled as:

$$\mathbf{V}_o = \dot{x}_p \left(\frac{z}{g} \right) \tag{6.1}$$

where a constant leakage gap, g is assumed and z is the vertical distance between the piston and cylinder. An expression for the viscous shear stress between the piston and cylinder is then obtained as:

$$\tau_{zx}|_{z=g} = \left(\mu_o \frac{\partial \mathbf{V}_o}{\partial z} \right)_{z=g} = \frac{\mu_o \dot{x}_p}{g}. \tag{6.2}$$

The force required to move the piston through the cylinder due to shear stress is given by:

$$F_{shear} = \int_0^A \tau_{zx}|_{z=g} \, dA = \frac{\mu_o \dot{x}_{p,avg} A_p}{g}. \tag{6.3}$$

The frictional power is then calculated as:

$$\dot{W}_{friction,o} = F_{shear}\dot{x}_{p,avg}. \tag{6.4}$$

Using the same technique as in Section 2.5 the damping associated with the friction of the oil film is recast in terms of an effective damping as follows:

$$c_{friction,o} = \frac{4F_{shear}}{\pi\omega x_p}. \tag{6.5}$$

Oil viscosity changes in inverse proportionality to temperature with a negligible dependence on pressure (Kim, 2005). Kauzlarich (1998) developed expressions for density ρ, and kinematic viscosity $\mu\rho$, for five classes of lubricants. These lubricant families were paraffines, maphthenics, PFPE, diesters, and polyol esters (*i.e.*, POE lubricants). For POE lubricants the relationships are as follows:

$$\mu_o\rho_o = \left\{0.0400\exp\left[\frac{473.9}{T_o+123.7}\right]\right\} VG^{\left\{0.2097\exp\left[\frac{458.4}{T_o+240.5}\right]\right\}}. \tag{6.6}$$

where the oil density ρ is equal to,

$$\rho_o = (0.8753 - 0.00062T_o)\, VG^{0.0374}. \tag{6.7}$$

VG is the ISO-VG number, ρ_o is the oil density in ${}^{kg}/_{m^3}$, T_o is the oil temperature in $^\circ C$ and μ_o is the oil viscosity in cP. These expressions have an accuracy of $\pm 5\%$ at reference temperatures of $40^\circ C$ and $100^\circ C$ at atmospheric pressure.

6.2.2 Model Components

After modification, the comprehensive linear compressor was split into three components. First, the resonant frequency model is utilized to calculate the stroke of the linear compressor. Next, assuming a sinusoidal piston motion using this calculated stroke the mass flow and leakage is calculated using the compression process

equations. Finally, to calculate the power required to operate the compressor the information from the first two steps is passed to the vibration sub-model.

The necessity for treatment in three components stems from some of the differences in operating conditions and geometry of the commercial linear compressor compared with the prototype compressor developed for electronics cooling. Recall from Equation (2.21) that the gas stiffness during the compression process is proportional to the difference in pressure across the piston as well as the piston area. The commercial linear compressor operates with much higher pressure differences across the piston and has a larger piston area. As a result, the gas stiffness in the commercial linear compressor is significantly higher than that in the prototype compressor. In fact, the gas stiffness is large enough to be of the same order of magnitude as the mechanical stiffness. This causes numerical instability in the model, as the highly non-linear nature of the gas stiffness and damping, as described in Section 3.1.1 and 3.1.2, generates even more extreme changes in piston dynamics as a function of stroke. In addition, the coupling between the compression process equations and the vibration model presents an additional level of nonlinearity that is amplified due to the differences between the prototype and commercial linear compressor. For example, as the stroke increases from rest, the mean pressure in the piston increases and thus the piston drift, as described in Section 5.4, increases. As the drift increases the dead volume increases, which reduces the mean cylinder pressure and the drift. This coupling behavior poses a challenge to model convergence for many operating points and impossible for others. To avoid this, the vibration model is decoupled from the comprehensive compressor model. While this limits the ability of the model to characterize peculiar performance behavior, such as drift, it does not impact the ability of the modeling approach to predict overall performance metrics.

6.2.3 Stroke Calculation

The first step in the comprehensive compressor model presented in Chapter 2 and modified in Chapter 3 involves the calculation of the resonant frequency based on the input stroke to the device and a linearization of the stiffness and damping in the linear compressor piston. The operating stroke of the commercial linear compressor could not be measured, as this would have required opening the hermetic shell and likely destroying the compressor. However, the frequency delivered to the device was measured at each operating point to be 60 Hz. The model presented in Section 3.2.1 to calculate resonant frequency depends on the stroke input and the operating conditions. Since the experimentally obtained operating conditions are known along with the driving frequency, this model can be used in reverse to estimate the compressor stroke. This assumes that the commercial linear compressor is controlled in such a way as to ensure that the compressor operates at or close to a resonant frequency at all operating conditions.

Using Equation (3.4) and inputs from Equations (3.5) through (3.9) a secant-method numerical routine is used to calculate the required stroke by first guessing stroke values and adjusting the guess of the next step until the appropriate resonant frequency is calculated. Figure 6.8 shows the calculated stroke using the method described as a function of compressor pressure ratio.

Comparing the calculated results to the experimental mass flow measurements, some observations can be made. Generally, as the pressure ratio decreases the compressor responds with a lower mass flow rate. However, at a pressure ratio of about 5 and below, the mass flow remained relatively constant or even increased. Therefore, it may be concluded that there is some change in the feedback control that maintains the stroke in the refrigerator. Similar results were noticed at the highest recorded pressure ratios as the mass flow began to decrease as the pressure ratio increased. Therefore, an assumption is made that the stroke is controlled between two extreme values, a low value of roughly 0.57 cm and a maximum value of roughly 1.55 cm.

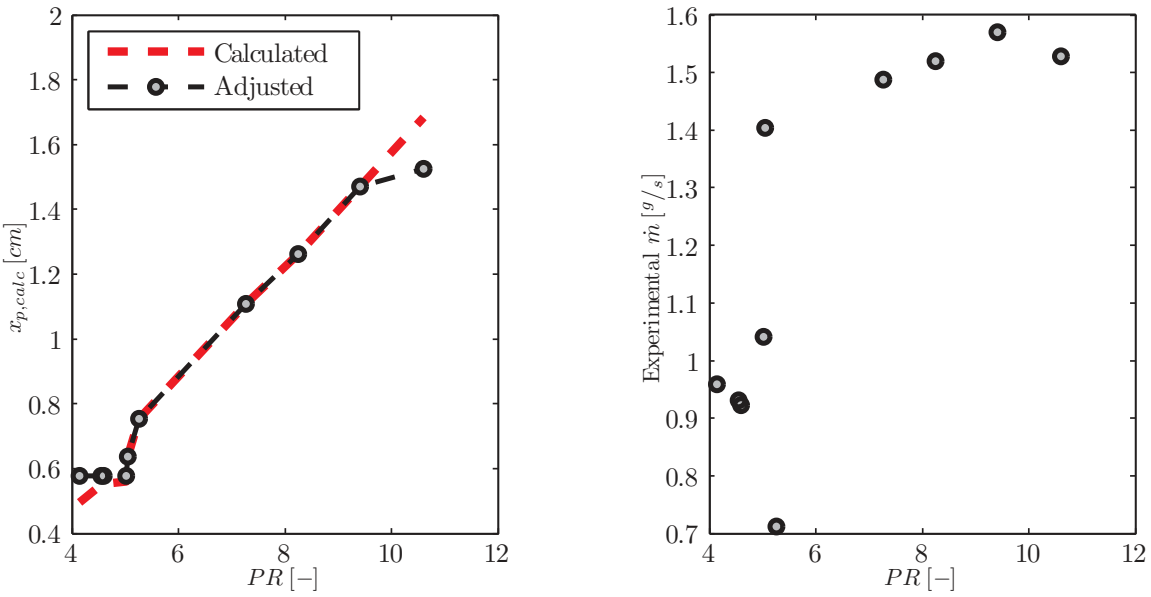


Figure 6.8. Estimated stroke of the commercial linear compressor as predicted by resonant frequency model (left) and the experimental mass flow rate as a function of pressure ratio (right).

6.2.4 Comprehensive Compressor Model

Using the calculated stroke from the previous section the next step in the model utilizes the comprehensive compressor model. This operates just as outlined in Chapter 2 with one exception, the vibration model is replaced with a simple geometry model. This model will represent the piston motion as a sinusoidal driving force,

$$x_p = x_{stroke} \cos(\omega t). \tag{6.8}$$

This modification to the comprehensive compressor model algorithm, which is shown in Figure 6.9. The leakage gap, valve, and piston dimensions were utilized as inputs to this model. This model returns mass flow, leakage, temperature, and pressure as key output information. This information is then passed into the vibration model.

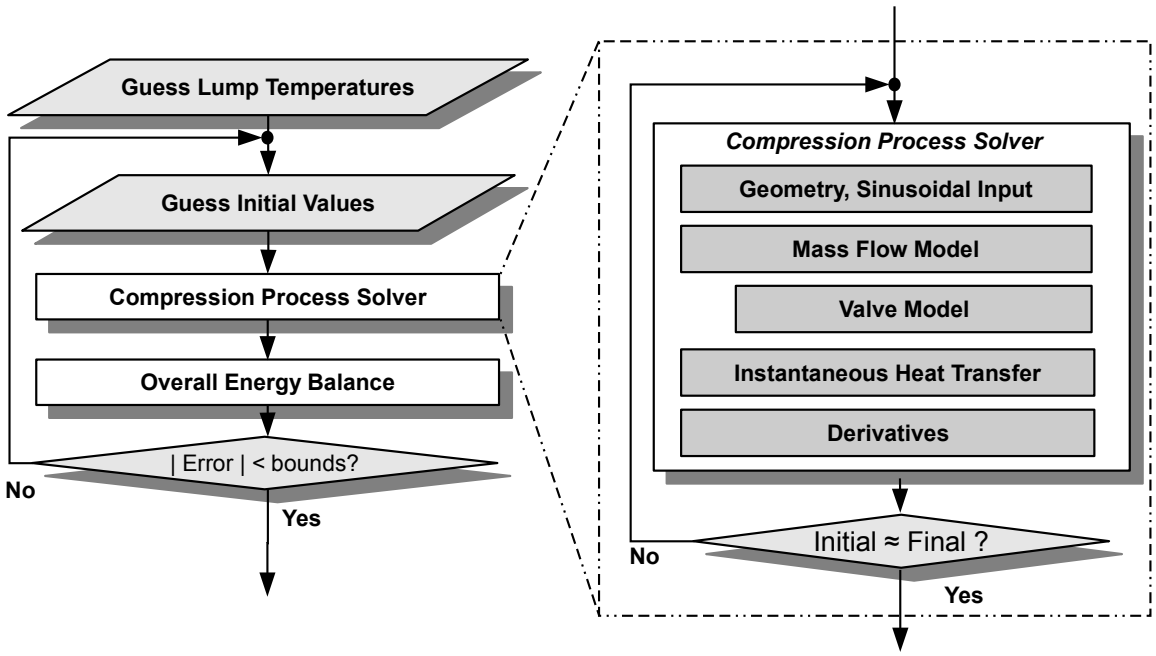


Figure 6.9. Modified computational algorithmn of comprehensive linear compressor model with a simple geometry model.

6.2.5 Vibration Model

Using the inputs passed from the comprehensive compressor model with a simple geometric model for the piston motion, the vibration model is called. Figure 6.10 shows the overall modified model flowchart. The vibration model is now uncoupled from the compression process equations. This eliminates the nonlinearities associated with the coupling of these equations (*i.e.*, drift) which allows for a more robust operation of the comprehensive model.

Using the vibration model, as before, the force required to drive the piston at the input stroke is calculated. Using this information the power required to operate the compressor is calculated as:

$$\dot{W}_{in} = \left(F_{drive}\dot{x}_p\right)_{rms} + \dot{Q}_{motor}. \tag{6.9}$$

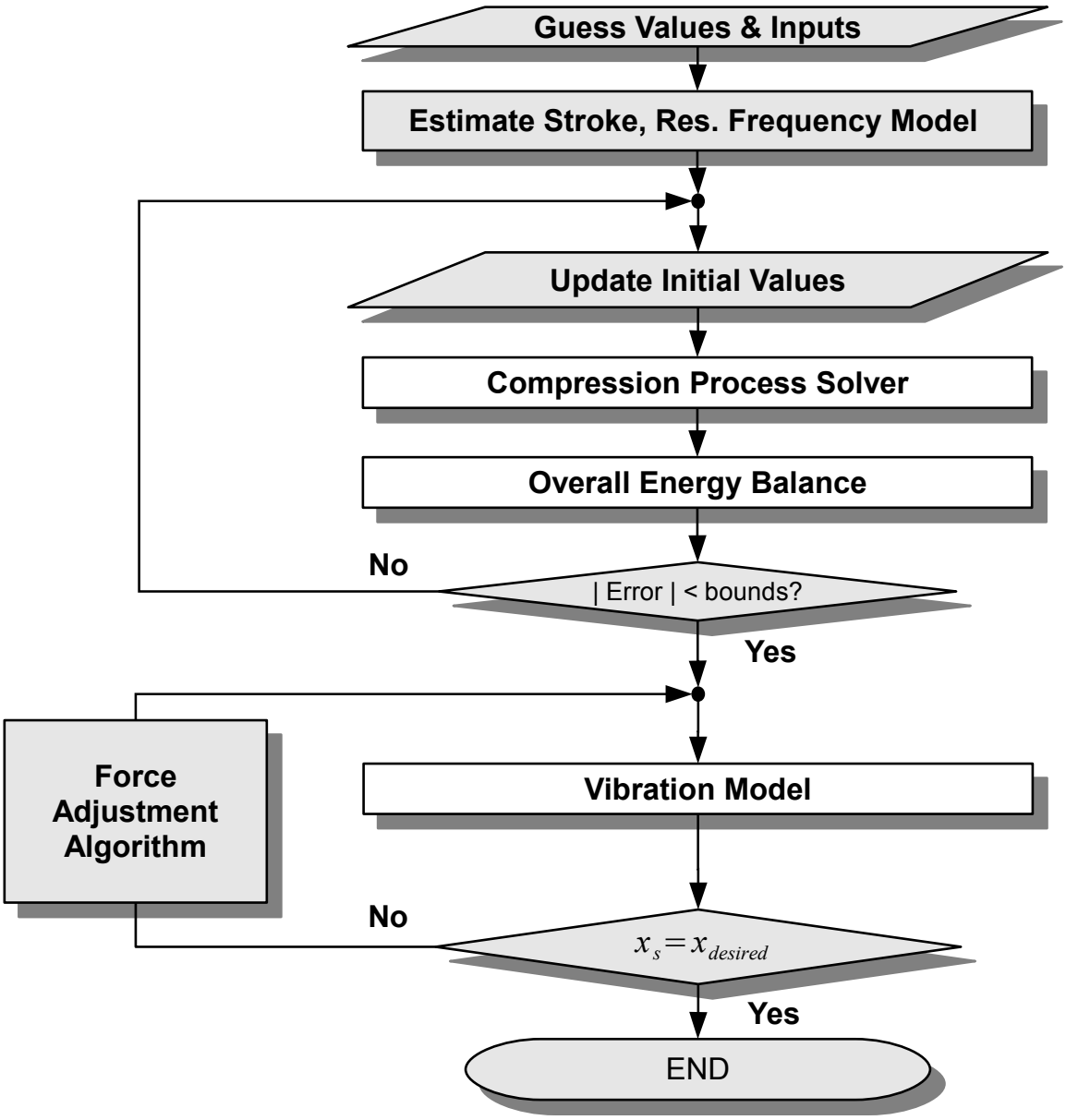


Figure 6.10. Model flowchart for comprehensive model of a linear compressor applied to a commercial linear compressor.

6.3 Modification of Domestic Refrigerator/Freezer for Testing of a Commercial Linear Compressor

A domestic refrigerator/freezer was purchased for testing of the enclosed commercial linear compressor. This refrigerator/freezer (P/N: LGC25776SW) has a unique

feature of capacity control. This is accomplished by adjusting the compressor stroke by varying the input voltage to the device. While the feature is useful for the product, it poses additional difficulty for testing the commercial linear compressor as removing the linear compressor from the refrigerator requires intimate knowledge of the controllers operating the device. Therefore, the refrigerator/freezer was instead placed into an environmental chamber. Figure 6.11 shows a schematic diagram of the modifications to the refrigerator within the environmental chamber.

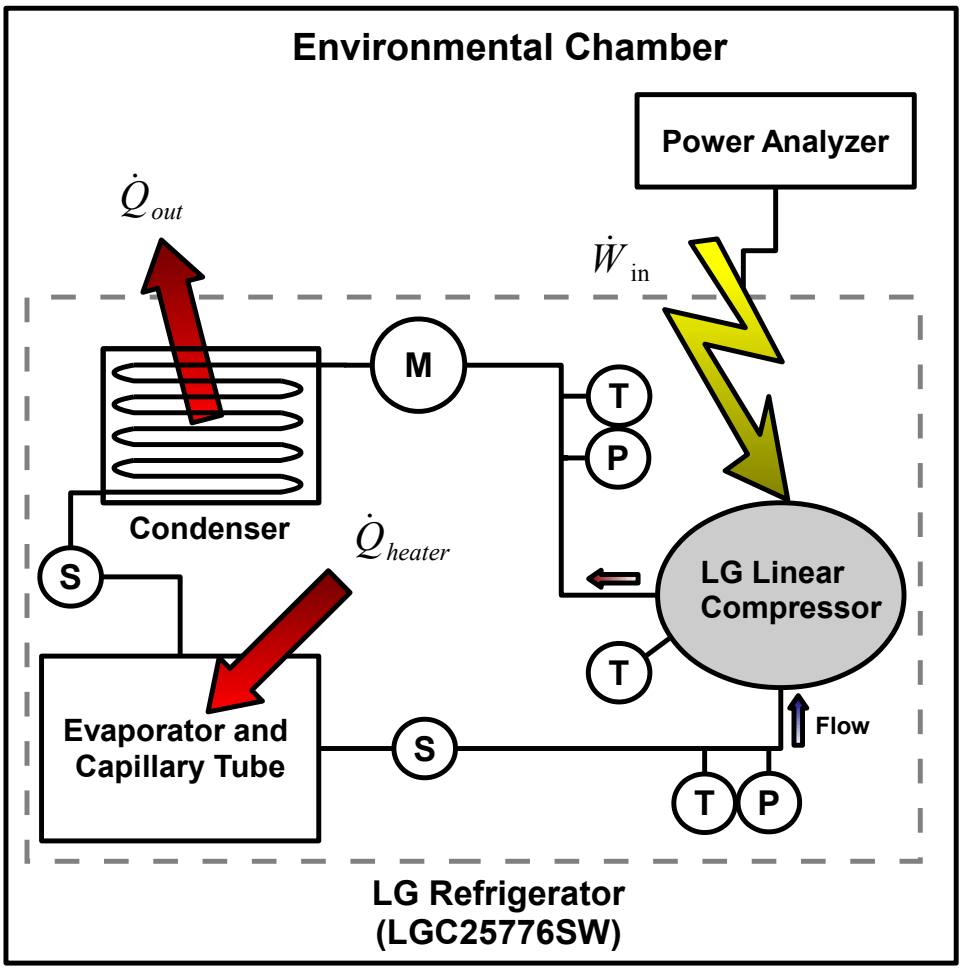


Figure 6.11. Schematic diagram of modified domestic refrigerator/freezer used to test a commercial linear compressor in an environmental chamber.

The environmental chamber is used to control the pressure ratio across the compressor by controlling the condenser temperature. A heater was also placed in the freezer of the refrigerator to act as a fixed load. This served to extend compressor cycles and aid in steady-state data acquisition. The actual refrigeration cycle was modified with additional sensors including thermocouples at the suction and discharge of the linear compressor, one to measure ambient conditions and three to measure surface temperatures on the linear compressor shell. Pressure is measured at the compressor suction and discharge of the linear compressor. Finally, a mass flow meter is placed in the discharge line between the linear compressor and the condenser coil in addition to sight glasses on either side of the evaporator coil. Linear compressor and heater power were measured with power analyzers and the compressor frequency with an oscilloscope. Table 6.2 gives the product, part numbers, and absolute uncertainty associated with each measurement used in this experiment.

Table 6.2. Experimental measurement devices used for testing commercial linear compressor and their uncertainty values.

Measurement	Device	Uncertainty
Temperature	T-Type Thermocouple	0.2 K
Pressure	Omega Model PX176	4 kPa
Mass Flow Refrigerant	MicroMotion Model CMF010	0.35%
Compressor Power	Valhallah 2100 Power Analyzer	0.20%
Compressor Frequency	Textronix TPS 2024	0.50%
Heater Power	Kill A Watt P4400	3.00%

6.3.1 Refrigerator/Freezer Operation

The modified refrigerator/freezer operated in a cyclic fashion. Figure 6.12 shows the freezer temperature changing with time for a typical set of operating conditions the freezer heater powered off. This behavior is expected, and would be similar to any refrigerator/freezer one would purchase.

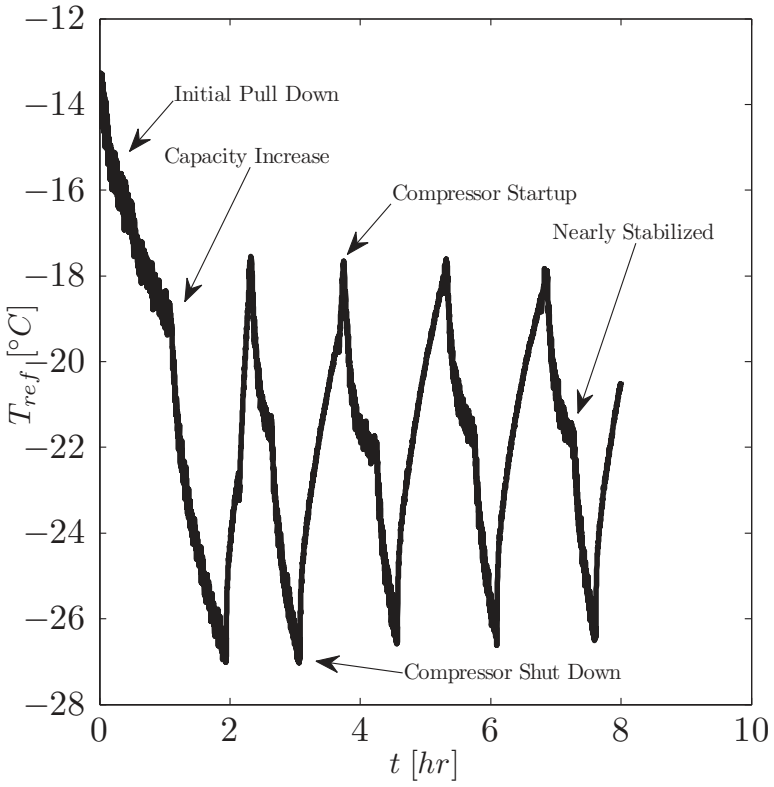


Figure 6.12. Freezer temperature over time for a typical operation of the domestic refrigerator/freezer.

At time equal to zero, the refrigerator/freezer is operating in the initial cool down period after first being turned on (*i.e*, pull down). The freezer temperature pulls down to about -19 $^{\circ}C$ and then a discontinuity occurs which represents a change in the capacity of the compressor. This change in capacity represents an increase in stroke and subsequent increase in mass flow rate. The increase in mass flow rate allows the temperature of the freezer to be pulled down further until it reaches roughly -27$^{\circ}C$. As it begins to stabilize at the lowest temperature, the compressor reduces stroke (and capacity) again and the freezer temperature jumps back to roughly -23 $^{\circ}C$. Finally, when temperature is nearly steady at this condition the compressor is shut down when the freezer temperature jumps to up to roughly -18 $^{\circ}C$. The refrigerator/freezer controller then detects that this is too warm and the compressor turns back on and the freezer begins to cool again. This cycle then continues with the freezer temperature

'settling in' at various temperatures as intermediate steps. Utilizing the heater in the freezer the period of these cycles could be somewhat extended to allow the compressor to achieve a performance closer to steady state.

6.3.2 Testing Conditions

The compressor was tested using three heater power levels and ambient temperatures ranging from 6 $^{\circ}C$ to 39 $^{\circ}C$. Tables 6.3 and 6.4 show the results of 10 sets data points collected from the modified refrigerator/freezer.

Table 6.3. Measured temperatures from testing of the commercial linear compressor.

Run	T_{suc}	T_{dis}	T_{ref}	T_{freeze}	$T_{shell,1}$	$T_{shell,2}$	$T_{shell,3}$	T_{amb}
–	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$	$^{\circ}C$
1	8.08	26.51	4.79	-4.42	16.78	19.79	18.23	6.45
2	5.80	33.36	4.68	-6.09	21.80	25.87	25.14	12.90
3	9.03	29.73	2.61	-8.66	19.74	23.49	22.48	10.16
4	-0.13	29.33	1.94	-11.68	18.55	22.84	21.88	10.21
5	17.84	61.84	1.25	-4.97	41.59	48.30	48.50	29.51
6	18.52	66.37	2.85	-5.88	44.90	52.14	52.58	33.48
7	27.52	74.99	1.18	-4.81	51.84	59.63	60.17	38.42
8	30.82	82.92	2.45	-5.07	58.12	66.51	67.51	43.32
9	13.24	48.33	1.28	-5.14	31.30	36.77	35.75	19.10
10	-0.93	26.97	1.85	-19.37	18.34	22.26	21.90	10.99

6.3.3 Validation of the Commercial Linear Compressor Model Predictions

The refinement of the commercial linear compressor required special attention to certain features that were not present in the prototype compressor. This includes a variation of motor efficiency with operating condition as well as the additional superheating of the suction gas as it moves from the shell entrance to the compression chamber. Utilizing these considerations the modified comprehensive linear compressor

Table 6.4. Additional experimental results from testing of a commercial linear compressor.

Run	P_{suc}	P_{dis}	PR	$\dot{m}$	$\dot{W}_{in}$	V	I	$\dot{W}_{heater}$	$\eta_{o,is}$
$-$	kPa	kPa	$-$	$^{g}/_{s}$	W	V	A	W	$[-]$
1	111.81	463.70	4.15	0.959	62.15	167.80	0.684	151	0.517
2	112.38	564.75	5.03	1.041	62.20	167.80	0.684	151	0.635
3	108.90	500.27	4.59	0.923	55.82	172.60	0.663	150	0.599
4	109.66	499.63	4.56	0.931	56.09	172.47	0.663	99	0.576
5	125.47	910.96	7.26	1.487	112.30	227.28	0.875	150	0.650
6	123.23	1017.05	8.25	1.520	123.12	229.91	0.936	150	0.650
7	123.77	1165.43	9.42	1.570	138.41	253.91	1.032	150	0.658
8	124.02	1315.35	10.61	1.528	152.13	268.41	1.129	150	0.623
9	132.31	667.89	5.05	1.404	87.38	201.03	0.763	150	0.624
10	93.28	490.33	5.26	0.712	47.46	165.66	0.636	51	0.575

model is applied and the predicted results for mass flow, power, and overall isentropic efficiency are presented.

Motor Efficiency

The commercial linear compressor tested has undergone much refinement and optimization, which is clear from the construction of the device. The linear motor is also expected to operate at near optimum efficiency at its Energy Star rated operating conditions (70 $^{\circ}F$, with periodic door openings) (Energy Star, 2007).

Therefore, it is assumed that as the environmental temperature increases, the motor efficiency will decrease as shown by several studies on linear motors applied to linear compressors (Redlich et al., 1996; Redlich, 1995). As such, the motor is assumed to be 95% efficient at its rated conditions with an efficiency that decreases linearly to 60% at 110 $^{\circ}F$ environmental conditions.

Suction Gas Superheating

An additional consideration for the modeling of the commercial linear compressor is the heat interactions within the compressor shell. Figure 6.13 shows the serpentine path taken by the discharge line going from the compressor discharge muffler to the exit of the compressor shell.

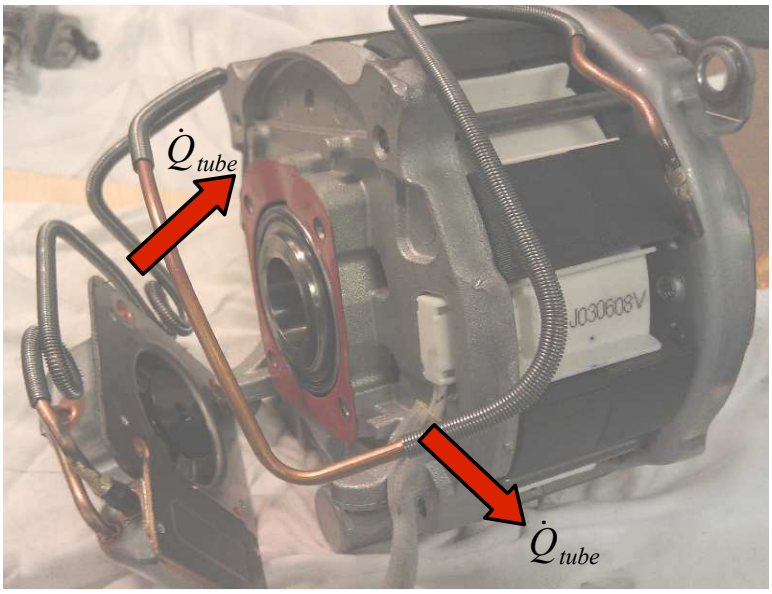


Figure 6.13. Global view of commercial linear compressor highlighting the heat transfer from the discharge tube into the compressor shell.

Along this path the discharge line is surrounded by refrigerant vapor at suction conditions. Figure 6.14 shows the measured discharge, average shell, and suction temperatures as well as an adjusted suction temperature. The difference between measured suction and discharge temperature ranges from roughly 20 $°C$ to 50 $°C$. Since both the suction and discharge measurements are taken outside the compressor shell it can be assumed that the temperature in the discharge line within the compressor is significantly higher and cools slightly due to heat transfer to the incoming suction gas. This is visualized in Figure 6.13 which shows the transfer of heat from the discharge line.

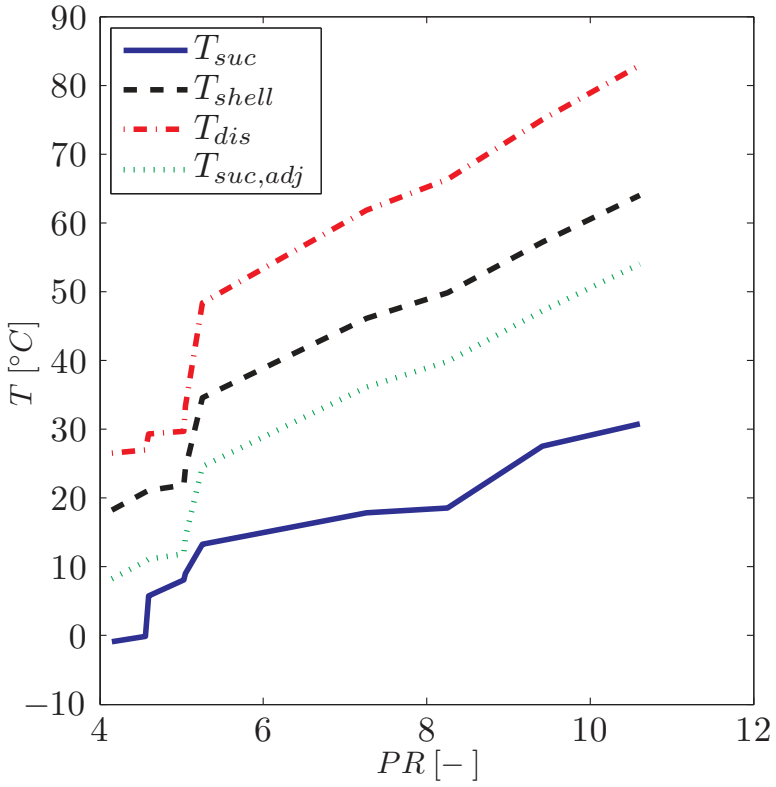


Figure 6.14. Experimental suction, discharge, and shell temperatures along with the modified suction temperatures as a function of pressure ratio.

Examining Figure 6.14 it can first be concluded that the actual temperature of the gas entering the compression chamber is between the measured inlet temperature T_{suc} and the measured discharge temperature T_{dis}. In addition, the average shell temperature T_{shell}, is also between the suction and discharge. Since the shell temperature will also likely be slightly warmer than the suction gas entering the compressor, a rough assumption was made that the suction gas entering the compression chamber $T_{suc,adj}$ is given by:

$$T_{suc,adj} = T_{shell} - 10°C. \tag{6.10}$$

While this approach is simplistic and relies on experimental data, the goal of the study is to validate the comprehensive modeling approach against a commer-

cially available linear compressor. In contrast, if the goal is to accurately predict the performance of the commercial linear compressor further rigor would be needed to incorporate additional sub-models for the heat interaction within the compressor shell.

Validation Results

Figure 6.15 shows a plot comparing the simulated mass flow rate compared with the experimental mass flow rate. The Mean Absolute Error (MAE) for these results is 4.9% with all points falling within ± 8% of the experimental value. The lowest mass flow rates correspond with the data points at lower pressure ratios and have mass flow rates which are generally underpredicted. In contrast, the mass flow rates at the highest pressure ratios tend to be overpredicted. Figure 6.16 shows a plot of the simulated input power compared with the experimental input power. The MAE of this fit is 6.1% with all points falling within roughly 18%. Unlike the mass flow rate fit the power input does not display any distinct trends as most points are scattered about the parity line. Figure 6.17 shows a plot of the simulated overall isentropic efficiency, as defined in Equation (4.2), compared with experimentally obtained values. The MAE of these results is 5.2% with all points falling within ± 12%. The outlier points in this diagram represent points where the error from the previous fits overlapped. In other words, the point with the over predicted overall isentropic efficiency has an under predicted power input and an over predicted mass flow rate.

6.4 Commercial Linear Compressor Summary

A description, model modifications, and experimental validation of a commercial linear compressor is presented in this chapter. This chapter highlighted the unique capacity control operation of the commercial linear compressor. The comprehensive linear compressor model was modified and utilized to analyze the commercial linear compressor. These modifications are: the addition of a thin-film friction model for

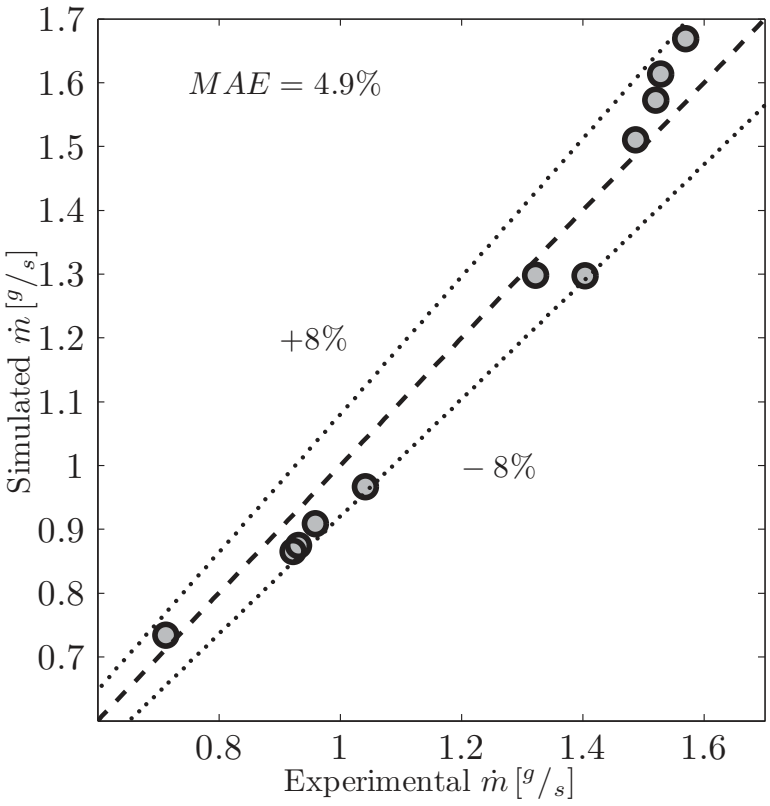


Figure 6.15. Simulated compared against experimental mass flow rate for a commercial linear compressor (measurement error within marker width).

the friction interaction between the piston and cylinder and a restructuring of the model algorithm in an effort to improve model robustness.

The domestic refrigerator/freezer that contained the commercial linear compressor was utilized as a test platform for the commercial linear compressor. This test platform was placed in an environmental chamber and the compressor loaded by changing the environmental temperature. The test results were compared with model predictions and it was found that the mass flow and compressor input power matched the experimental results within 4.9 and 6.1% MAE, respectively. The mass flow displayed a trend which tended to under predict the mass flow at lower pressure ratios and over predict at higher. The compressor input power is predicted less accurately but with-

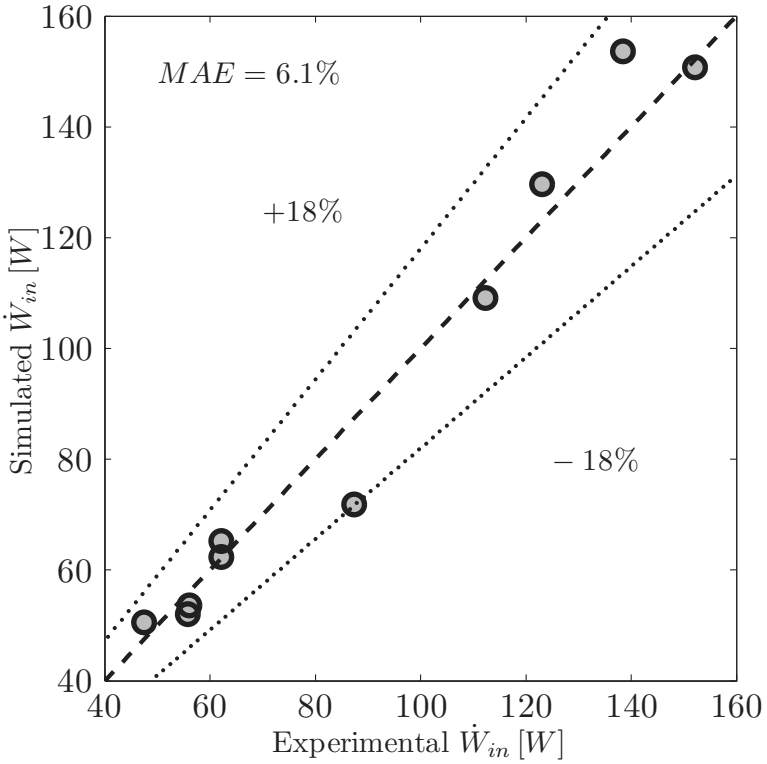


Figure 6.16. Simulated compared against experimental input power for a commercial linear compressor (measurement error within marker width).

out a specific trend. The overall isentropic efficiency is predicted to within 5.2% and shows a reasonable distribution.

This study has shown that a comprehensive approach can indeed be used for predicting the performance of a commercial linear compressor. In combination with the previous validation of the prototype linear compressor in Chapter 4, this shows that the general approach for modeling linear compressors is strong as relatively few modifications to the model presented in Chapter 2 were needed to represent the commercial linear compressor.

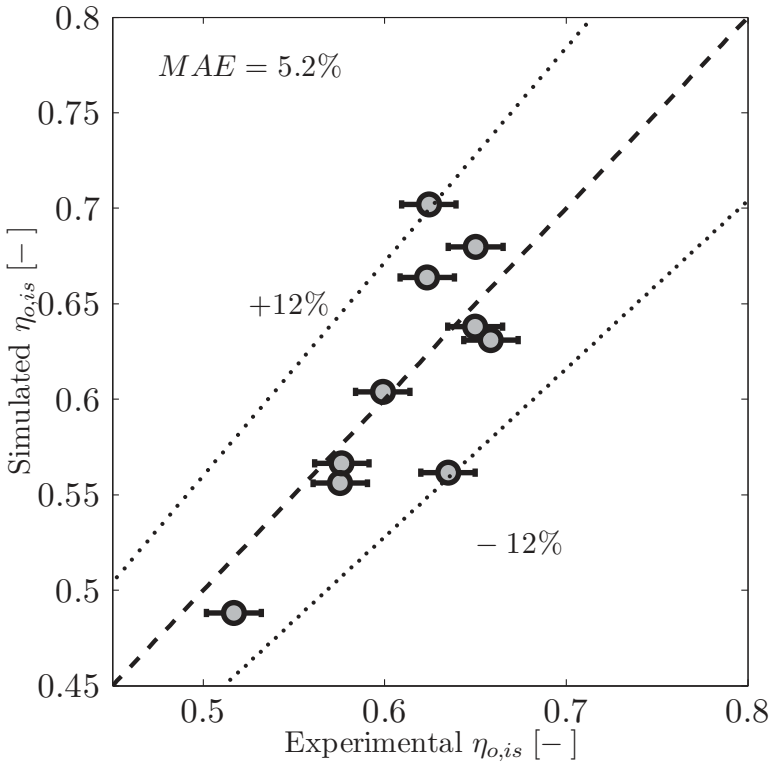


Figure 6.17. Simulated compared against experimental overall isentropic efficiency for a commercial linear compressor.

CHAPTER 7. CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

Linear compressor technology is reviewed with a particular focus on an electronics cooling application. The need for active cooling solutions for electronics to handle the increasing heat demands serves as motivation for the development of vapor compression refrigeration for electronics cooling employing linear compressors.

A comprehensive model of a miniature-scale linear compressor is presented. This model includes a limited number of assumptions which provides potential for this approach to be applied to any positive displacement device. It also includes sub-models for piston dynamics, valves, leakage, and heat transfer. The nonlinear dynamics of the linear compressor piston presented a unique set of challenges. These challenges necessitated revisions to the comprehensive modeling approach, specific to a linear compressor, which allowed for a more robust overall model. These revisions included the ability of the compressor frequency to change between iterations as well as the decoupling of the vibration model from the compression process equations.

The modeling approach was validated using two separate linear compressors. One linear compressor was custom-built for this work and designed for an electronics cooling application. The other is a commercially available linear compressor used in a domestic refrigerator/freezer. While the two compressors were designed for very different applications, the same modeling techniques were applied successfully to predict the performance of both linear compressors. This provides confirmation for the techniques applied specifically to the linear compressors, but also additional confirmation for the validity of the general comprehensive modeling approach.

This model was exercised to determine the important geometric parameters when designing a linear compressor. It was discovered that the linear compressor is highly

sensitive to the leakage gap between the piston and cylinder and moderately sensitive to the changes in frictional parameters such as spring eccentricity. Changes to the piston geometry show that as the displaced volume of the compressor decreases, the performance tends to improve. This means that linear compressor technology would scale well to smaller scales, which is appealing for electronics cooling. The linear compressor also displays some interesting characteristics that set it apart from other compressor technologies. One of its more useful characteristics is its ability for providing capacity control. Compared with a reciprocating compressor a linear compressor has a much stronger ability to control capacity by varying its piston stroke. It has also been shown that doing so results in very little degradation in performance over a wide range of cooling capacities for a given compressor.

A revised linear compressor design is presented to meet the increasing cooling needs for high power electronics. This design represents a collection of lessons learnt that resulted in a design that is efficient, small, and powerful. This shows that the linear compressor provides a promising solution for electronics cooling applications. However, implementation in real applications needs further commercial development.

7.2 Recommendations

Several areas related to linear compressors, compressor modeling, and electronics cooling should be investigated further, as discussed below.

Linear compressors require the following investigations:

- An investigation of appropriate spring types to minimize compressor piston side-loading.
- Further investigation into the performance benefits of energy recovery within linear compressors.
- An investigation into a double-sided compressor/expander which utilizes the energy recovery mechanism in a linear compressor.

- An investigation into the control strategy of a linear compressor and how it can be done to minimize parasitic power draw.

The comprehensive compressor modeling approach requires specific attention with investigations in the following areas:

- An investigation into improvements of mass flow and leakage modeling which incorporate multiple fluid phases/types.
- Numerical techniques for improving computational speed.

Finally, with respect to electronics cooling the following areas should be further investigated:

- A reduced system-level package size of a vapor compression refrigeration system for electronics cooling.
- An application-based study to determine the appropriate compressor technology for a variety of electronics cooling applications.
- Further investigation into the impact of oil on microchannel heat transfer performance.

LIST OF REFERENCES

LIST OF REFERENCES

Energy Star program requirements for residential refrigerators and/or freezers, August 2007.

allorings.com. O-ring gland design charts. Internet, March 2011.

I. H. Bell. *Theoretical and Experimental Analysis of Liquid Flooded Compression in Scroll Compressors*. PhD thesis, Purdue University, 2011.

S. Bertsch, E.A. Groll, and S.V. Garimella. Review and comparative analysis of studies on saturated flow boiling in small channels. *Nanoscale Microscale Thermophys. Eng*, 12(3):187–227, 2008a.

S.S. Bertsch, E.A. Groll, and S.V. Garimella. Refrigerant flow boiling heat transfer in parallel microchannels as a function of local vapor quality. *International Journal of Heat and Mass Transfer*, 51(19-20):4775–4787, 2008b.

R.P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, 14(4):422–425, 1971.

R.V. Cadman and R. Cohen. Electrodynamic oscillating compressors: Part 1 Design based on linearized loads. *ASME J. Basic Eng*, December:656–663, 1969a.

R.V. Cadman and R. Cohen. Electrodynamic oscillating compressors: Part 2 Evaluation of specific designs for gas load. *ASME J. Basic Eng.*, December:664–670, 1969b.

Y. Chen, N. P. Halm, J. E. Braun, and E. A. Groll. Mathematical modeling of scroll compressors - part ii: Overall scroll compressor modeling. *International Journal of Refrigeration*, 25(6):751–764, 2002a.

Y. Chen, N. P. Halm, E. A. Groll, and J. E. Braun. Mathematical modeling of scroll compressors - part i: Compression process modeling. *International Journal of Refrigeration*, 25(6):731–750, 2002b.

V. Chiriac and F. Chiriac. An overview and comparison of various refrigeration methods for microelectronics cooling. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety conference on*, pages 618–625, 2008.

G.S. Choe and K.J. Kim. Theoretical and experimental analysis of nonlinear dynamics in a linear compressor. *Journal of vibration and acoustics*, 124(1):152–154, 2002.

L. Cremaschi, E. A. Groll, and S. V. Garimella. Performance potential and challenges of future refrigeration-based electronics cooling approaches. In *THERMES 2007: Thermal Challenges in Next Generation Electronic Systems*, pages 119–128, 2007.

V. Dagilis and L. Vaitkus. Parametric analysis of hermetic refrigeration compressors. *Mechanika*, 6(80):23–29, 2009.

G.F. Davies, I.W. Eames, P. Bailey, M.W. Dadd, A. Janiszewski, R. Stone, G.G. Maidment, and B. Agnew. Cooling microprocessors using vapor compression refrigeration. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, pages 1–8. IEEE, 2010.

F. Fagotti and A.T. Prata. A new correlation for instantaneous heat transfer between gas and cylinder in reciprocating compressors. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, pages 871–876, 1998.

R.W. Fox, A.T. McDonald, and P.J. Pritchard. *Introduction to Fluid Mechanics*. John Wiley and Sons, 6th. edition, 2004.

T. Harirchian and S. V. Garimella. A comprehensive flow regime map for microchannel flow boiling with quantitative transition criteria. *International Journal of Heat and Mass Transfer*, 53(1314):2694 – 2702, 2010. ISSN 0017-9310.

T. Harirchian and S. V. Garimella. Flow regime-based modeling of heat transfer and pressure drop in microchannel flow boiling. *International Journal of Heat and Mass Transfer*, 55(4):1246 – 1260, 2012. ISSN 0017-9310.

R. Hellinger and P. Mnich. Linear motor-powered transportation: History, present status, and future outlook. *Proceedings of the IEEE*, 97(11):1892–1900, 2009.

A. Heydari. Miniature vapor compression refrigeration systems for active cooling of high performance computers. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2002. ITHERM 2002. The Eighth Intersociety Conference on*, pages 371–378, 2002.

R. Hilpert. Wärmeabgabe von geheizten Drähten und Rohren im Luftstrom (Heat transfer from heated wires and pipes in the air). *Forsch. Geb. Ingenieurwes.*, 4:215, 1933.

B. Hubacher, E.A. Groll, and C. Hoffinger. Performance measurements of a semihermetic carbon dioxide compressor. In *Proceedings of the International Refrigeration and Air Conditioning Conference, Purdue University, West Lafayette, IN USA*, number R11-10, 2002.

The Math Works Inc. Matlab, 2010a.

M.E. Jovane. *Modeling and analysis of a novel rotary compressor*. PhD thesis, Purdue University, 2007.

J. J. Kauzlarich. Viscosity times density equations for five classes of lubricants. *Proceedings of the Institution of Mechanical Engineers, Part J: Journal of Engineering Tribology*, 212 no. 3:235–239, 1998.

H. Kim, C. Roh, J. Kim, J. Shin, Y. Hwang, and J. Lee. An experimental and numerical study on dynamic characteristic of linear compressor in refrigeration system. *International Journal of Refrigeration*, 32(7):1536–1543, 2009.

J.H. Kim. *Analysis of a bowtie compressor with novel capacity modulation*. PhD thesis, Purdue University, 2005.

J.H. Kim and E.A. Groll. Feasibility study of a bowtie compressor with novel capacity modulation. *International Journal of Refrigeration*, 30(8):1427–1438, 2007.

M.H. Kim and C.W. Bullard. Thermal performance analysis of small hermetic refrigeration and air-conditioning, compressors. *JSME International Journal Series B-Fluids and Thermal Engineering*, 45(4):857–864, NOV 2002. ISSN 1340-8054.

W.J. Kim and B.C. Murphy. Development of a novel direct-drive tubular linear brushless permanent-magnet motor. *International Journal of Control Automation and Systems*, 2:279–288, 2004.

D.Y. Koh, Y.J. Hong, S.J. Park, H.B. Kim, and K.S. Lee. A study on the linear compressor characteristics of the Stirling cryocooler. *Cryogenics*, 42(6-7):427–432, 2002.

S. Krishnan, S.V. Garimella, G.M. Chrysler, and R.V. Mahajan. Towards a thermal moore's law. *IEEE Transactions on Advanced Packaging*, 30(3):462–474, 2007.

H. Lee, S.S. Jeong, C.W. Lee, and H.K. Lee. Linear compressor for air-conditioner. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number C047, 2004.

H. Lee, S. Ki, S. Jung, and W. Rhee. The innovative green technology for refrigerators-development of innovative linear compressor. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number 1419, 2008.

H.K. Lee, J.T. Heo, G.Y. Song, K.B. Park, S.Y. Hyeon, and Y.H. Jeon. Loss analysis of linear compressor. In *Compressors and their Systems: 7th International Conference*, page 305, 2001.

J. Lee and I. Mudawar. Low-temperature two-phase microchannel cooling for high-heat-flux thermal management of defense electronics. *IEEE Transactions on Components and Packaging Technologies*, 32(2):453 – 465, 2009.

J. B. Marcinichen, J. R. Thome, and B. Michel. Cooling of microprocessors with micro-evaporation: A novel two-phase cooling cycle. *International Journal of Refrigeration*, 33(7):1264 – 1276, 2010.

M. Mathison. *Modeling and Evaluation of Advanced Compression Techniques for Vapor Compression Equipment*. PhD thesis, Purdue University, 2011.

M.M. Mathison, J.E. Braun, and E.A. Groll. Modeling of a two-stage rotary compressor. *HVAC&R Research*, 14(5):719–748, 2008.

C. Minas. Nonlinear dynamics of an oilless linear drive reciprocating compressor. *Journal of Vibration and Acoustics, Transactions of the ASME*, 116(1):79 – 84, 1994.

R. Mongia, K. Masahiro, E. DiStefano, J. Barry, W. Chen, M. Izenson, F. Possamai, A. Zimmermann, and M. Mochizuki. Small scale refrigeration system for electronics cooling within a notebook computer. In *Thermal and Thermomechanical Phenomena in Electronics Systems, 2006. ITHERM'06. The Tenth Intersociety Conference on*, pages 751–758, 2006.

G.E. Moore. Cramming more components onto integrated circuits, Electronics. *April*, 19:114–117, 1965.

M.J. Moran and H.N Shapiro. *Fundamentals of Engineering Thermodynamics*. John Wiley and Sons, 5th. edition, 2004.

E. Navarro, E. Granryd, J. F. Urchueguia, and J. M. Corberan. A phenomenological model for analyzing reciprocating compressors. *International Journal of Refrigeration*, 30(7):1254–1265, 2007.

A.G. Agwu Nnanna. Application of refrigeration system in electronics cooling. *Applied Thermal Engineering*, 26(1):18 – 27, 2006.

K. T. Ooi and T. N. Wong. A computer simulation of a rotary compressor for household refrigerators. *Applied Thermal Engineering*, 17(1):65–78, 1997.

K. Park, E. Hong, K. Choi, and W. Jung. Linear compressor without position controller. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, 2004.

P.E. Phelan, V.A. Chiriac, and T. Tom Lee. Current and future miniature refrigeration cooling technologies for high power microelectronics. *IEEE Transactions on Components and Packaging Technologies*, 25(3):356 – 365, 2002.

E. Pollak, W. Soedel, R. Cohen, and F.J. Friedlaender. On the resonance and operational behavior of an oscillating electrodynamic compressor. *Journal of Sound and Vibration*, 67:121–133, 1979.

F.C Possamai, Lilie D.E.B., A.P. Zummermann, and R.K. Mongia. Miniature vapor compression system. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number 2932, 2008.

S.S. Rao. *Mechanical Vibrations*. Prentice Hall, 4th edition, 2004.

R. Redlich. A summary of twenty years experience with linear motors and alternators. In *LDIA*, 1995.

R. Redlich, R. Unger, and N. Van der Walt. Linear motors: motor configuration, modulation and systems. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number 68, 1996.

J. Rigola, C.D. Perez-Segarra, and A. Oliva. Parametric studies on hermetic reciprocating compressors. *International Journal of Refrigeration*, 28(2):253–266, 2005.

A.A. Sathe. *Miniature-scale diaphragm compressor for electronics cooling*. PhD thesis, Purdue University, 2008.

A.A. Sathe, E.A. Groll, and S.V. Garimella. Experimental evaluation of a miniature rotary compressor for application in electronics cooling. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number 1115, 2008.

R. Tillner-Roth and H.D. Baehr. An international standard formulation for the thermodynamic properties of 1, 1, 1, 2-tetrafluoroethane (HFC-134a) for temperatures from 170 k to 455 k and pressures up to 70 mpa. *Journal of Physical and Chemical Reference Data*, 23:657, 1994.

S. Trutassanawin, E. A. Groll, S. V. Garimella, and L. Cremaschi. Experimental investigation of a miniature-scale refrigeration system for electronics cooling. *IEEE Transactions on Components and Packaging Technologies*, 29(3):678, 2006.

R. Unger. Development and testing of a linear compressor sized for the european market. In *In Proceedings at the International Appliance Technology Conference, Purdue University, West Lafayette, IN*, number 74, 1999.

R. Unger and S. Novotny. A high performance linear compressor for cpu cooling. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, number C23-3, 2002.

R.Z. Unger. Linear compressors for clean and specialty gases. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, volume 1, page 51, 1998.

N.R. Van der Walt and R. Unger. Linear compressors-a maturing technology. In *Proceedings of the International Compressor Engineering Conference, Purdue University, West Lafayette, IN, USA*, pages 239–246, 1994.

M. Xia and X. Chen. Analysis of resonant frequency of moving magnet linear compressor of stirling cryocooler. *International Journal of Refrigeration*, 33(4):739 – 744, 2010.

APPENDICES

Appendix A: Prototype Linear Compressor

No linear compressors are commercially available in the capacity and pressure ranges desired for electronics cooling. Therefore, a prototype compressor was custom-designed and built for the purpose of conducting experiments that could serve to validate the model developed in this work. The aim was the development of a prototype that would operate in a vapor compression refrigeration system which has a cooling capacity of at least $400W$. This represents a high performance desktop PC total heat rejection. Additional requirements of the design are that the compressor needs to operate oil-free and should minimize the total volume occupied by the device. The design of the prototype is described in this chapter and final design drawings are shown in Appendix B.

A.1 Prototype Design Model

In an effort to determine what size components are needed for the linear compressor prototype a simple model was developed. This model incorporates a single degree of freedom piston dynamics model which was used to estimate the power and spring rate required to operate the prototype compressor. This model was coupled with an idealized vapor compression refrigeration system model to determine the required flow rate of the prototype compressor.

The idealized vapor compression refrigeration model used an evaporation temperature of $20°C$, condensing temperature of $40°C$, and a fixed compressor superheat of $5\ K$. Using these assumptions and R-134 refrigerant properties from Tillner-Roth and Baehr (1994), the required volume flow rate of refrigerant required to obtain $400\ W$ of total cooling was determined to be $6.95\ {}^{L}/_{m}$.

Realizing that the volume flow rate is defined as,

$$\dot{V} = x_p f_{res} A_p \tag{A.1}$$

the prototype design model is needed to estimate both the stroke and frequency of operation. The stroke was to be determined by motor selection and the frequency by spring selection. Thus, the piston area was selected to ensure a proper flow rate is achieved. This selection continued iteratively using a variety of springs and motors until a suitable overall size, piston aspect ratio, and operating frequency were obtained from the selected components.

A.2 Component and Material Selection

In order to develop the most robust design possible, an effort was made to purchase as many off-the-shelf (i.e. pre-engineered) components as possible. This included the motor and compression springs. In addition to specific components, material selection for the piston-cylinder interface and valve reeds were considered of equally critical importance.

Linear Motor

Linear motors have matured significantly in recent years, mainly within the transportation industry (Hellinger and Mnich, 2009). These changes have specifically targeted large-scale applications such as mass transit, or theme park rides. This trend is partially a result of advances in permanent magnet motors. Permanent magnet motors are brush-less and thus, a lower reliability concern than other varieties of motors. A reliable linear motor has expanded the applications from larger-scale transportation applications to smaller devices such as the linear compressor developed in this work (Redlich, 1995; Redlich et al., 1996).

A plate type linear motor, typically used in mass transit, would provide a design challenge which is unnecessary for a hermetic compressor application. For this reason a tubular style linear motor was selected. A tubular-style linear motor consists of a permanent magnet moving core surrounded by a stator with a single coil winding. The excitation of the coil with an electric potential this generates a force onto the

permanent magnet as described by the Lorentz-force law from electro-magnetics. The operation of a linear motor is shown by Kim and Murphy (2004) in Figure A.1. The motor selected for the prototype design is manufactured by H2W Technologies Inc., and is shown in Figure A.2. This motor was selected as it met the requirement for rated force needed to operate a compressor with a rated cooling capacity of at least 400 W. Additionally, this motor comes pre-assembled with two linear bearings to support the drive shaft.

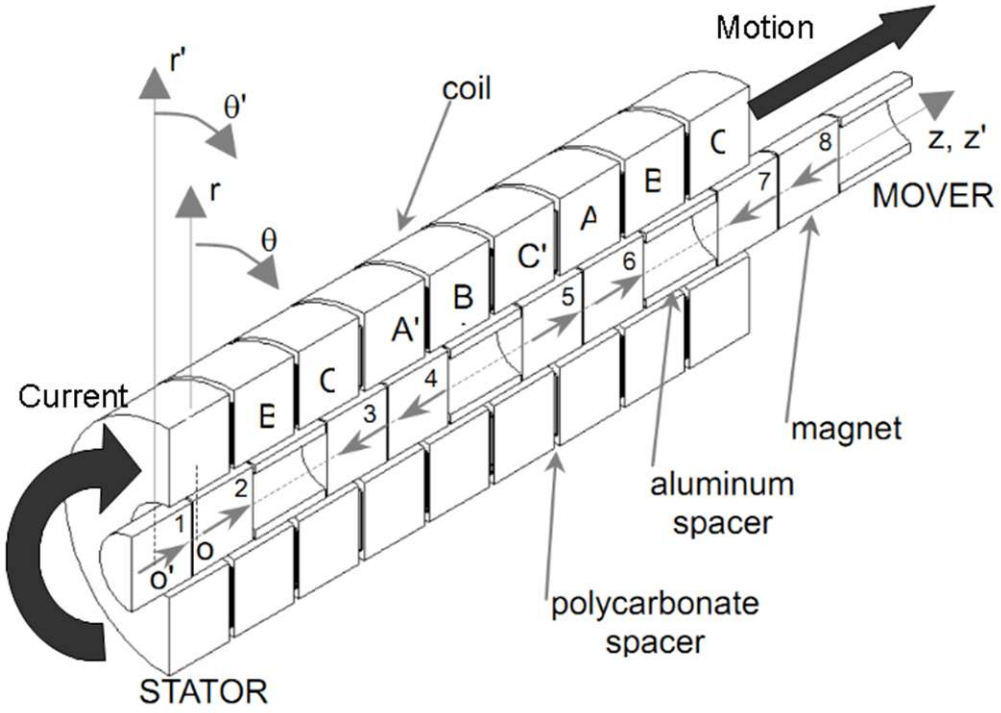


Figure A.1. Linear motor operation from Kim and Murphy (2004).

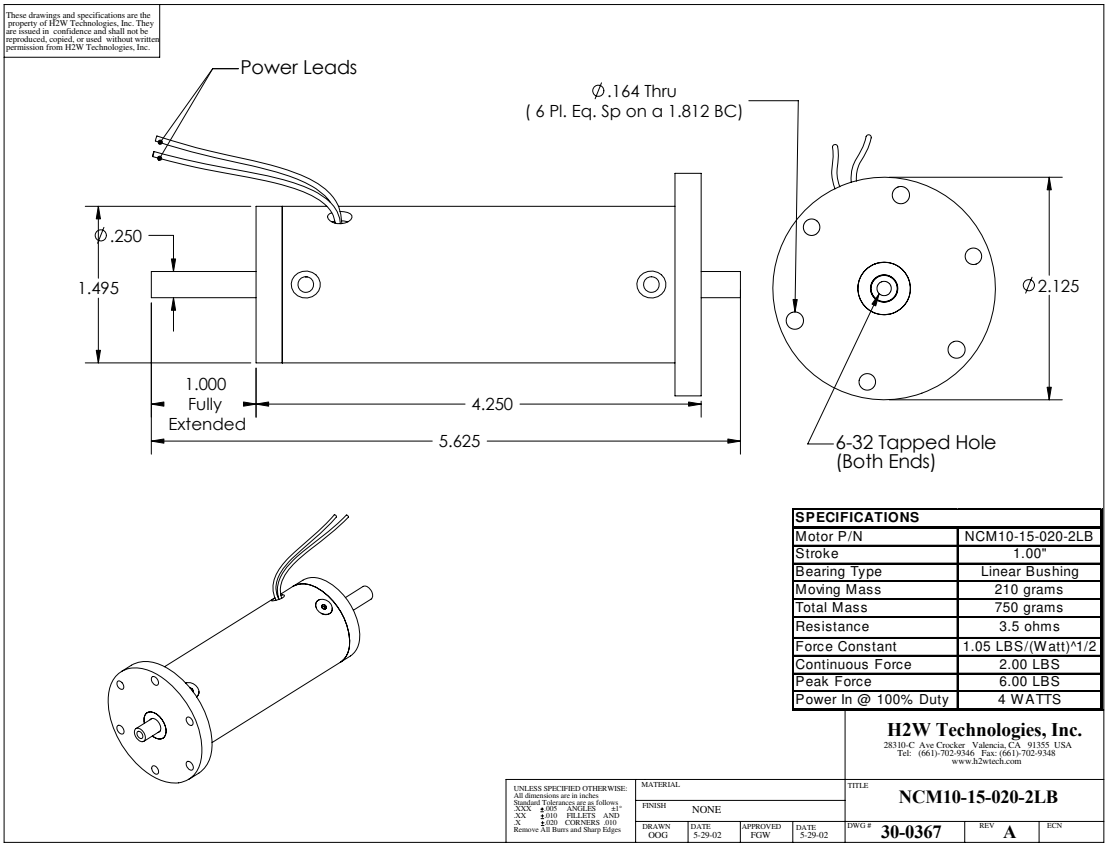


Figure A.2. Linear motor drawing.

Compression Springs

To achieve the desired resonant frequency anticipated from the design model a series of compression springs were selected based on spring rate of each spring. The spring rate desired required two springs (part numbers MCQ11507-MW and MCW20701-MW). These springs are manufactured by Murphy and Reed Spring Company.

Piston-Cylinder Materials

The requirement for the compressor to operate without lubrication requires special consideration when selecting the piston and cylinder materials. Raw Teflon (PTFE) was selected as the piston material to contact the piston wall because of the relatively low friction coefficient. In addition, Teflon tends to wear and develop a fine powder of Teflon particles which act as a type of dry lubrication. The cylinder surface needs to have significantly different hardness to ensure the piston and cylinder do not damage themselves. A T6 aluminum alloy was selected as it was readily available and provided an easy material to machine. The aluminum compliments the PTFE surface as it is much harder which helps reduce overall wear.

Valve Reed Material

While the valve body and plenum only required strength, the valve reeds required strength and a predictable amount of elasticity. This is necessary because the elasticity dictates the dynamic motion of the valve reeds, which has an impact on the overall performance of the compressor. The material selected is a spring steel manufactured in sheets, which is specifically designed for planar-type springs. The spring steel selected is a blue-tempered 0.017 in thick stock.

O-ring Materials

In order to ensure that the compressor would be the most robust machine possible, it was designed with seals that could handle any fluid or lubricant. These seals are PTFE coated silicone o-rings which have the following advantages. The first, is the increased flexibility due to the silicone material. This flexibility allows for increased sealing ability with minimal gland pressure. The second advantage is the increased range of fluid compatibility afforded by the PTFE coating.

A.3 Valve Design

The design of the valves include an overlapping reed design. The suction and discharge reeds are orthogonal to each other in orientation with the discharge port traveling through the suction reed. This orientation was utilized in an attempt to place the suction and discharge ports as close to the center of the cylinder as possible.

The suction and discharge reeds are attached on opposite sides of a valve body which sits flush into the cylinder. This valve body is then attached to the compressor body using a valve plenum and screws. The size of the ports are sized to ensure that the Mach number of the incoming or outgoing fluid remains at approximately 0.2 or lower. In addition to the port size, another consideration is flow through the opening between port and reed. To ensure there would be sufficient flow through this opening an estimation of the valve opening height was performed. This analysis also allowed for a determination of a proper discharge stopper height.

While the majority of the valve assembly was manufactured using traditional manufacturing techniques the valve reeds required special treatment due to the thin gage of the material. The process used is known as Electric Discharge Machining (EDM).

A.4 Prototype Design, Revision 00

The final design of the first revision of the linear compressor prototype is shown in Figure A.3. The motor and compression springs were the key purchased components. As a result, these components dictated a majority of the overall component orientation. The motor served as a central hub of assembly and the compression springs are not of sufficient diameter to fit outside of the motor, which forced their placement on one side of the motor.

Compression Spring Orientation

In an effort to keep the compressor design as simple as possible both sets of compression springs were positioned on the compression chamber side of the motor. A piston assembly was manufactured and attached on one side of the motor shaft which included a plate for the springs to seat themselves on. To maintain contact with this place the springs had to be assembled partially compressed. In this way, at top and bottom dead center each set of opposing springs still maintains contact with some amount of force on the plate. This eliminates any movement of the springs within the compressor during operation.

Piston Assembly

The first revision of piston assembly utilized a two-piece design with two PTFE piston rings. The rear piston is a solid piece of 1040 steel alloy which connects directly to the motor shaft with a threaded coupling. This piece is machined with an extra wide plate, seen between the two sets of springs in Figure A.3, for seating the compression springs as mentioned in Section A.4. This shaft continues toward the compression chamber and ends with a flange for seating the piston rings as well as a female thread for attaching the front of the piston. The piston rings are slid onto the rear of the piston and attached with a front piston face which is screwed into the rear piston using a counter-sunk machine screw.

O-ring Gland Design

O-rings offer significant advantages over other sealing techniques but require significantly more precise manufacturing techniques when designing a device to use them. In an effort to reduce potential sealing problems the number of sealed joints was minimized. The linear compressor prototype retains only six o-ring sealed joints in the

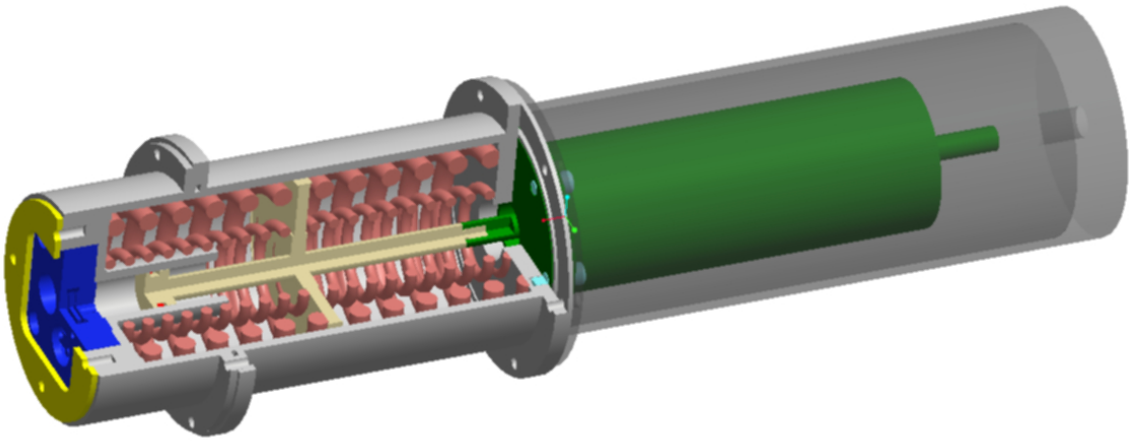


Figure A.3. Linear compressor prototype Revision 00.

device. Four of these sealing joints are located within the valve assembly, and two additional are located along the body of the compressor.

To ensure correct sealing of the o-rings, the o-ring glands must be properly sized. This is done using the data from Tables A.1 and A.2 for static radial and axial seals, respectively. The dimensions in Tables A.1 and A.2 reference Figures A.4 to A.6.

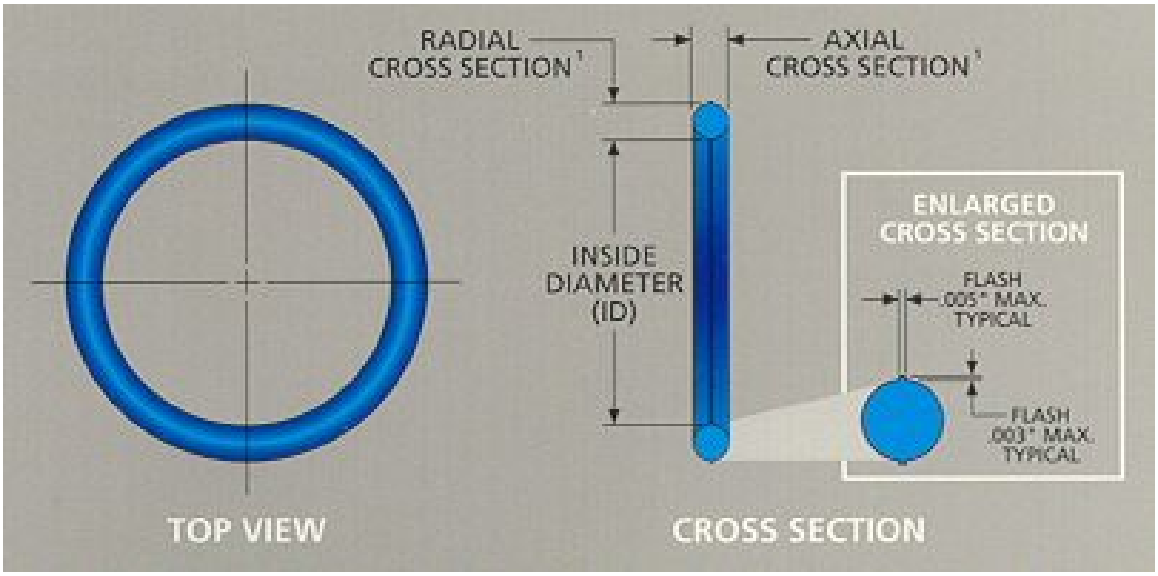


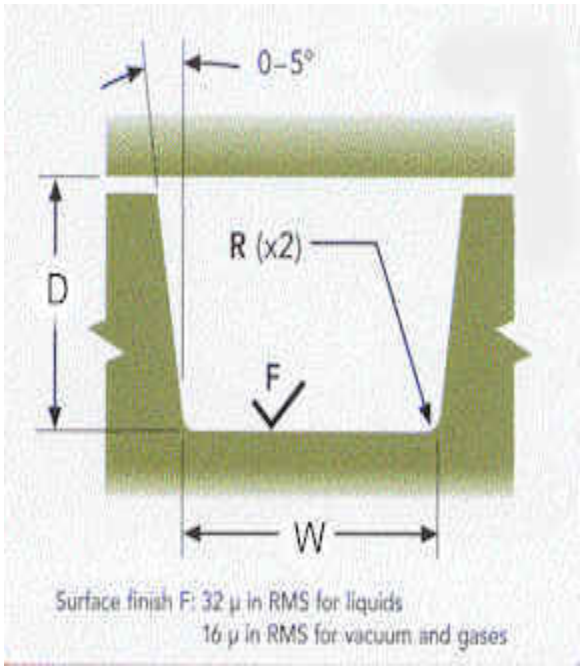
Figure A.4. Typical o-ring dimensions allorings.com (2011).

Table A.1. Design guidelines of o-ring gland dimensions for a static radial seal from allorings.com (2011).

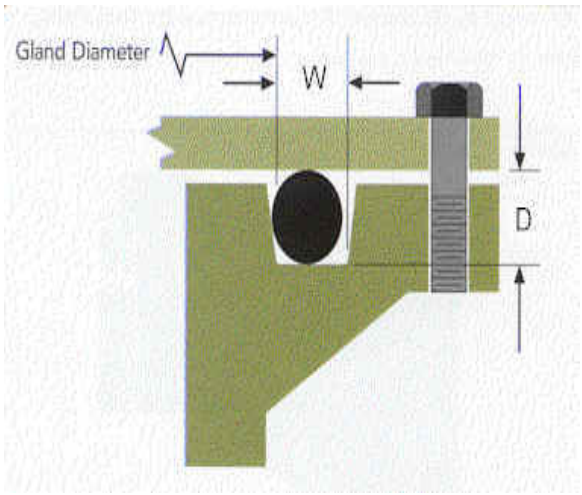
O-Ring Size	**O-Ring**	**Gland**	**Actual**	**%**	**(C)**	**Gland Width**			**Groove**
AS568A-	**Cross Section**	**Depth**	**Squeeze**	**Squeeze**	**Clearance**	**No Back-Up**	**One Back-Up**	**Two Back-Ups**	**Radius**
004 to 050	.070" +/-.003	.050 to .052	.015 to .023	22 to 32	.002 to .005	.093 to .098	.138 to .143	.205 to .210	.005 to .015
102 to 178	.103" +/-.003	.081 to .083	.017 to .025	17 to 24	.002 to .005	.140 to .145	.171 to .176	.238 to .243	.005 to .015
201 to 284	.139" +/-.004	.111 to .113	.022 to .032	16 to 23	.003 to .006	.187 to .192	.208 to .213	.275 to .280	.010 to .025
309 to 395	.210" +/-.005	.170 to .173	.032 to .045	15 to 21	.003 to .006	.281 to .286	.311 to .316	.410 to .415	.020 to .035
410 to 475	.275" +/-.006	.226 to .229	.040 to .055	15 to 20	.004 to .007	.375 to .380	.408 to .413	.538 to .543	.020 to .035

Table A.2. Design guidelines of o-ring gland dimensions for a static axial seal from allorings.com (2011).

O-Ring Size	**O-Ring**	**Gland**	**Actual**	**%**	**Groove Width**		**Groove**
AS568A-	**Cross Section**	**Depth**	**Squeeze**	**Squeeze**	**Liquids**	**Gas/Vacuum**	**Radius**
004 to 050	.070" +/-.003	.050 to .054	.013 to .023	19 to 32	.101 to .107	.084 to .089	.005 to .015
102 to 178	.103" +/-.003	.074 to .080	.020 to .032	20 to 30	.136 to .142	.120 to .125	.005 to .015
201 to 284	.139" +/-.004	.101 to .107	.028 to .042	20 to 30	.177 to .187	.158 to .164	.010 to .025
309 to 395	.210" +/-.005	.152 to .162	.043 to .063	21 to 30	.270 to .290	.239 to .244	.020 to .035
410 to 475	.275" +/-.006	.201 to .211	.058 to .080	21 to 29	.342 to .362	.309 to .314	.020 to .035

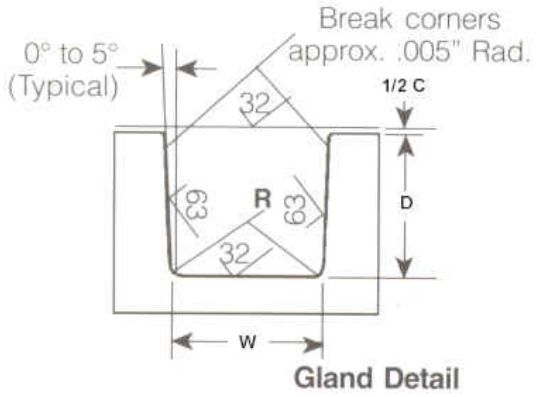


(a) Close-up of axial gland.

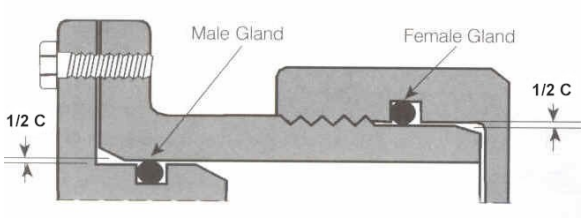


(b) Global view of axial gland.

Figure A.5. Axial sealing gland dimensions allorings.com (2011).



(a) Close-up of radial gland.



(b) Global view of radial gland.

Figure A.6. Radial sealing gland dimensions allorings.com (2011).

Design Challenges

The first iteration of the prototype, seen in Figure A.3, presented a series of design challenges. These challenges resulted in the need to re-design specific components of the compressor, particularly the piston assembly.

When first operated the compressor was found to resonate at approximately $68Hz$. This value was significantly higher than the design model prediction of 30Hz. In addition, the stroke achieved by the device was found to be only a fraction (approximately 5-10%) of the design stroke. Finally, the noise from the device was observed to be significantly louder than expected. The latter two observations prompted a disassembly of the compressor to investigate potential internal damage.

When the compressor prototype was disassembled, it was discovered that both the piston and cylinder had been severely damaged. As seen in Figures A.7 and A.8

the cylinder bore had been scored from the piston movement. The magnitude of this scoring was considered severe as a significant effort was required to separate piston from cylinder after operation. The piston also displayed similar damage as shown in Figure A.9. The PTFE piston rings on the piston displayed worn to the point that the steel of the piston had directly contacted the cylinder. The steel and aluminum had subsequently rubbed together which caused a destruction of both surfaces. This iteration of the prototype design prompted a redesign effort to address the destruction of the piston and cylinder.

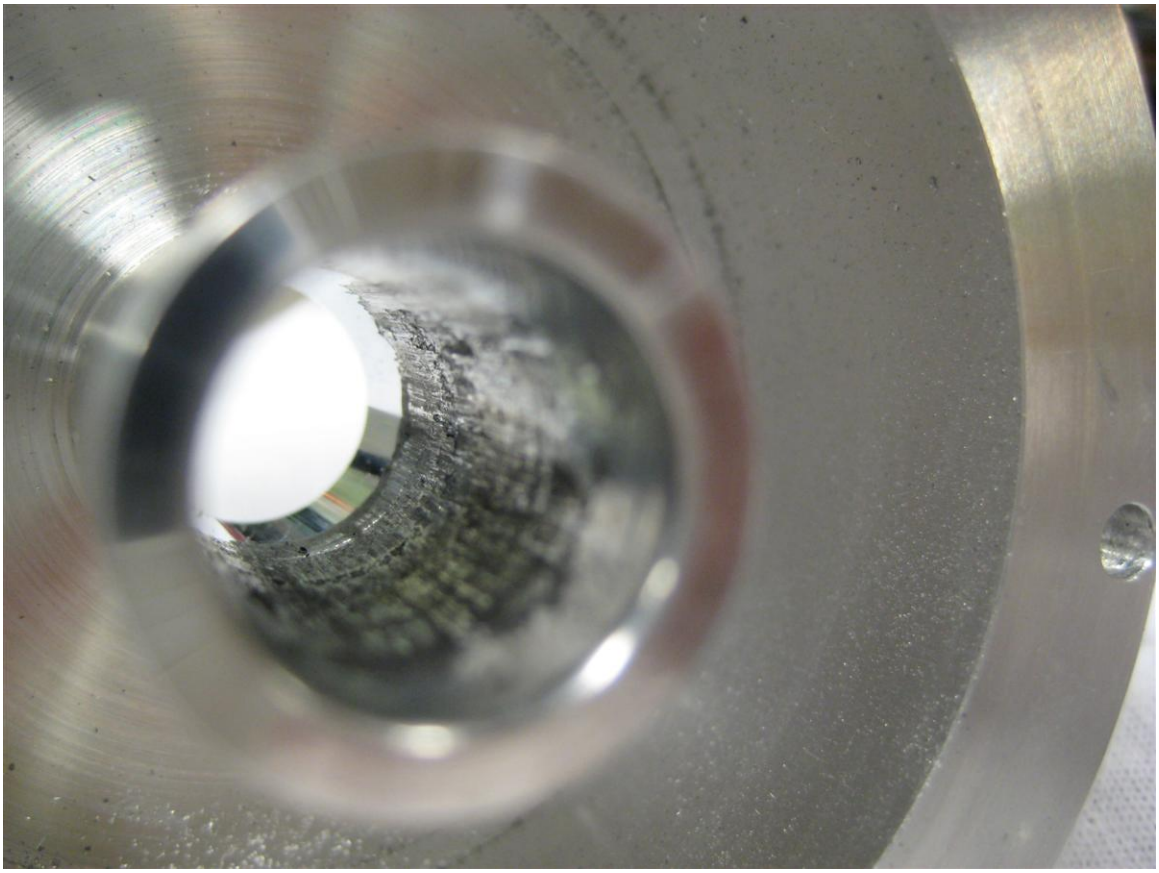


Figure A.7. Damage to compressor cylinder, view from inside compressor, Revision 00.

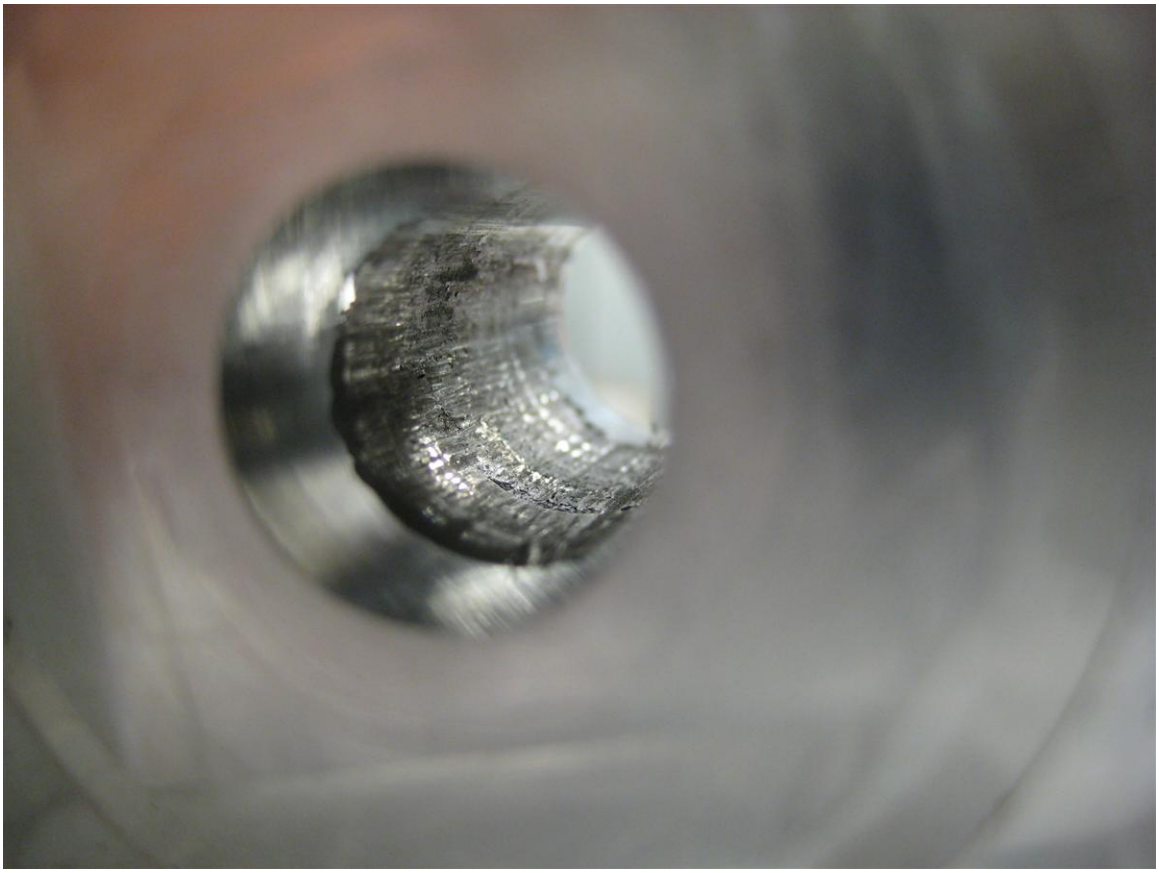


Figure A.8. Damage to compressor cylinder, view from outside compressor, Revision 00.

A.5 Prototype Design, Revision 01

Piston Redesign

To address the destruction of the piston and cylinder, the piston assembly was redesigned. The root cause of the contact and wear of the piston and cylinder surfaces was determined to be caused excessive side loading of the piston on the cylinder wall caused by spring eccentricities. The excessive loading was theorized to have caused the PTFE rings to deteriorate rapidly. Without any rings the steel and aluminum contacted each other directly and as the hardness of each material is similar, the un-lubricated contact caused damage to each surface. To resolve this issue a revised

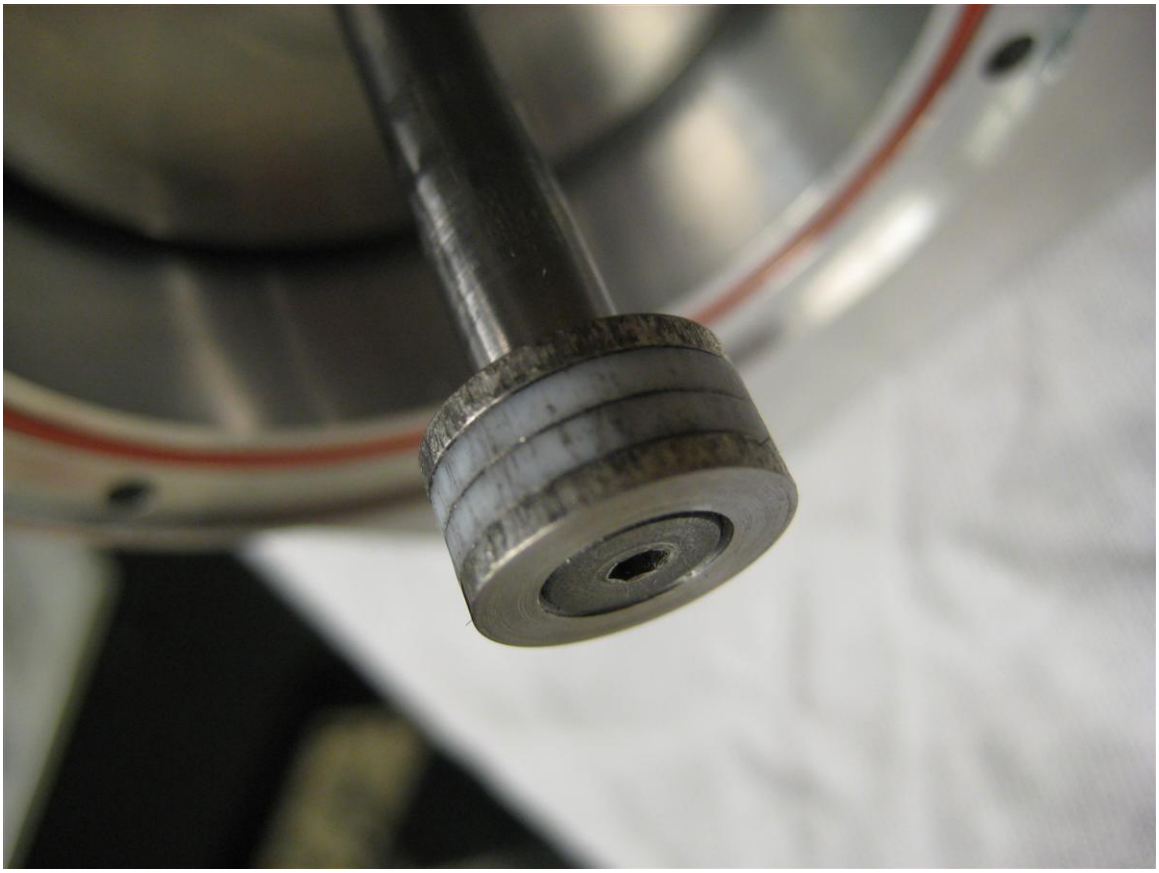


Figure A.9. Damage to compressor piston, Revision 00.

piston design was developed to ensure that aluminum and steel contact could be avoided. This modification involved a piston manufactured entirely out of PTFE. While this did not address the root of the problem it would eliminate the two surfaces from destroying each other. The updated piston design is shown in Figure A.10.

Design Challenges

Upon rebuilding the prototype compressor the device operated with a large reduction in noise but still failed to achieve full design stroke. It was hypothesized that the side loading caused by the compression springs was still causing excessive dry friction between the piston and cylinder. In an attempt to reduce this, the larger diameter

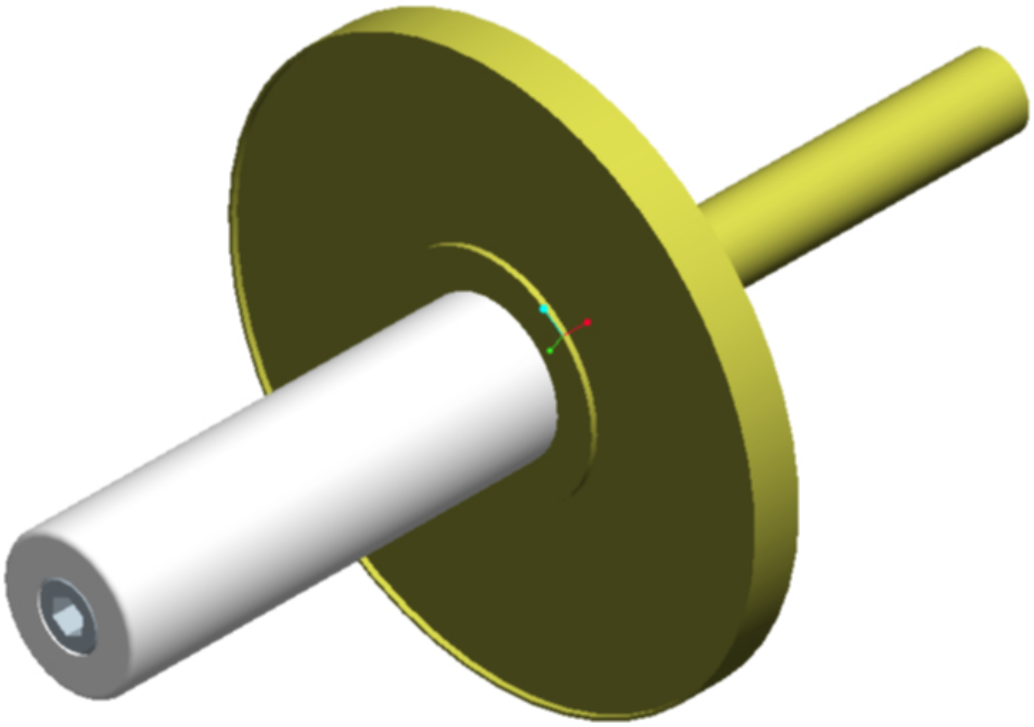


Figure A.10. Modified piston assembly with a complete PTFE piston, Revision 01.

outer springs were removed. The outer springs are stiffer and also produce a larger moment arm from the eccentricities. This modification can be seen in Figure A.11.

With the larger spring removed and the redesigned piston assembly the compressor was able to operate for a significant amount of time and produce some useful output. This allowed for data collection which will be explained in more detail in Chapter 4. However, this iteration of the design still performed relatively poor. It is hypothesized that the poor performance is based on significant side loading due to spring eccentricities. While this friction did not destroy the compressor as it did in the first iteration of the design, a break down of the PTFE piston was noted.

After final testing the compressor was taken apart for examination. and wear on the piston was noted as large quantities of black dust. This dust had collected on

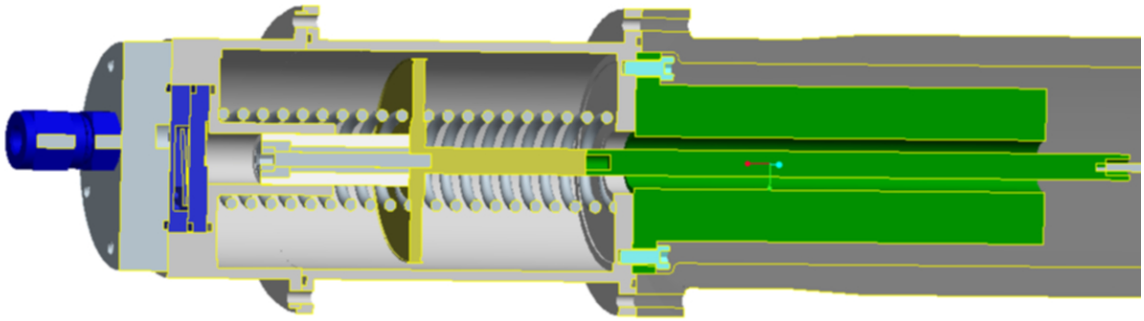


Figure A.11. Global section view of prototype compressor with revised piston design, Revision 01.

the inside of the cylinder, the piston, valves and left trace amounts throughout the inside of the compressor. This is seen in Figures A.12 to A.14. This dust is believed to be from a breakdown of the PTFE piston and is a sign that the friction between the piston and cylinder is still significant.

A.6 Design Revision 02

After completion of model validation and further testing of a commercial linear compressor (Chapter 6) additional modifications were made to the prototype linear compressor. The design changes are summarized below:

- The compressor was charged with POE 32 oil for lubrication and sealing purposes.
- The valves were modified to allow for less restricted gas flow into and out of the compressor.
- A squeeze ring was placed inside the PTFE piston sleeve to allow a degree of control over the leakage gap as shown in Figure A.15. By tightening the piston

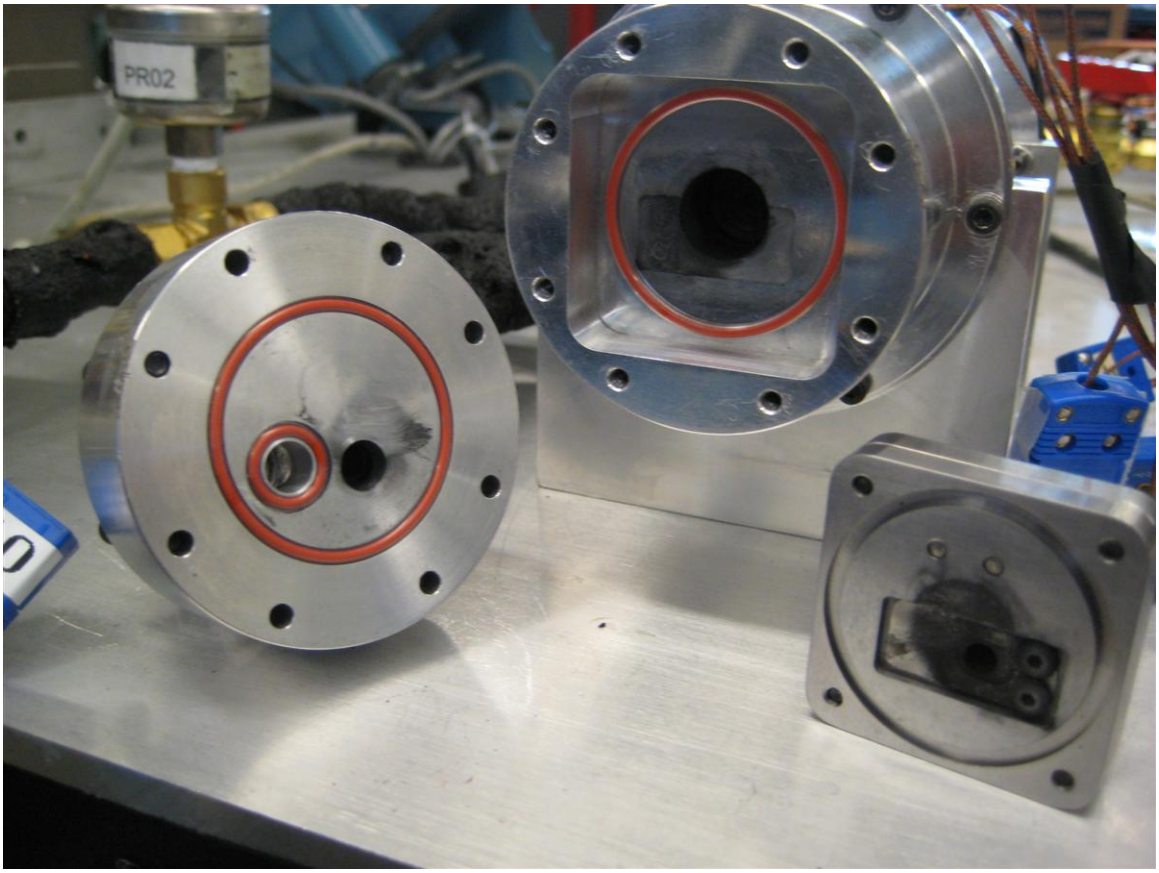


Figure A.12. Typical dust from compressor operation on valves and body, Revision 01.

screw this engaged this squeeze ring and deformed the piston by expanding it radially. The slightly increased the piston diameter and thus decreases the leakage gap.

- A spacer was placed behind the piston sleeve to reduce the dead volume in the compressor during operation as seen in Figures A.16, and A.17. This spacer directly reduced the amount of dead volume in the machine during normal operation.

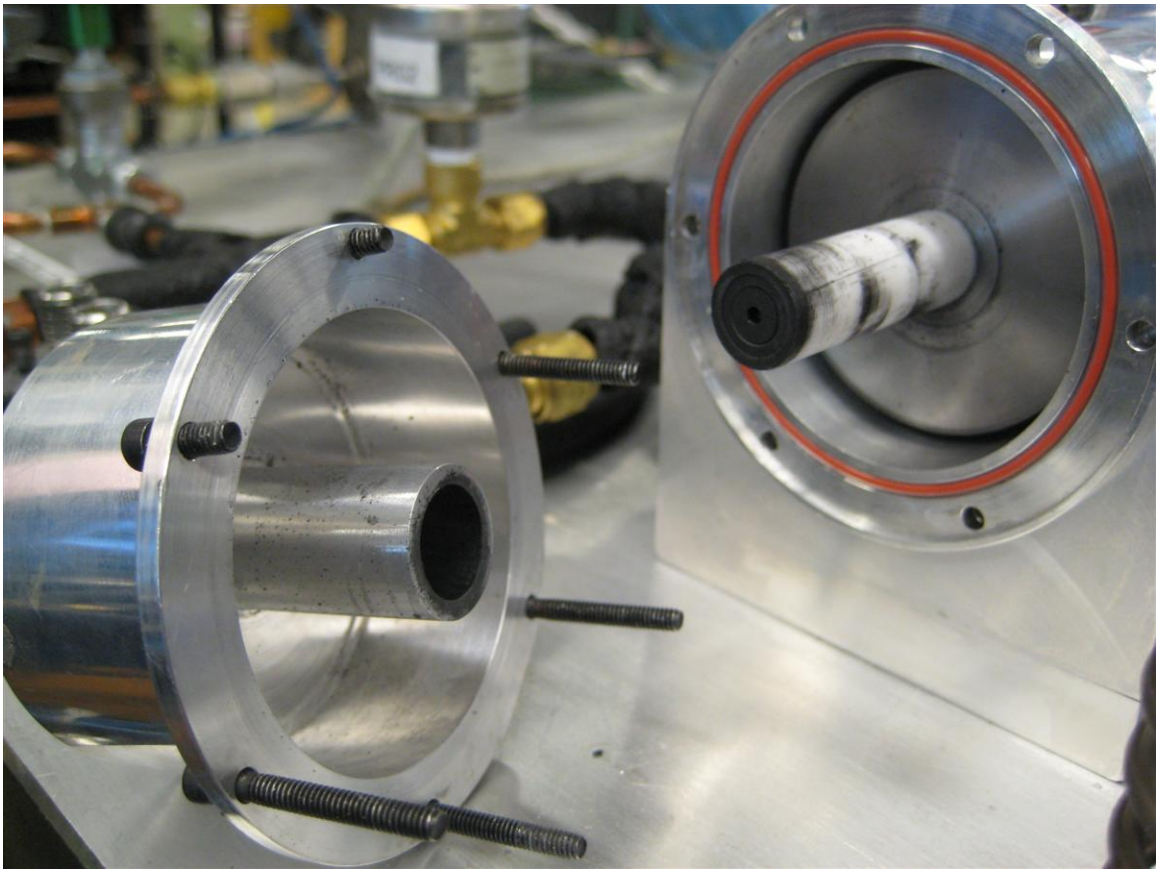


Figure A.13. Typical dust from compressor operation on piston and cylinder, Revision 01.

A.7 Design Conclusions

The final design of the linear compressor prototype yielded enough data to assist validating the model. An additional design modification was attempted and displayed how sensitive the linear compressor prototype is to design modifications. This highlighted the potential for improvement of this device. However, as noted in previous sections the current design is still flawed in many ways and would require significant modifications to operate at comparable efficiency to a commercial compressor. One modification would simply be finding springs with smaller potential for side loading. The side loading of the piston caused by the spring eccentricities is the root cause for the lack of performance and thus, should be addressed in iterations moving forward.

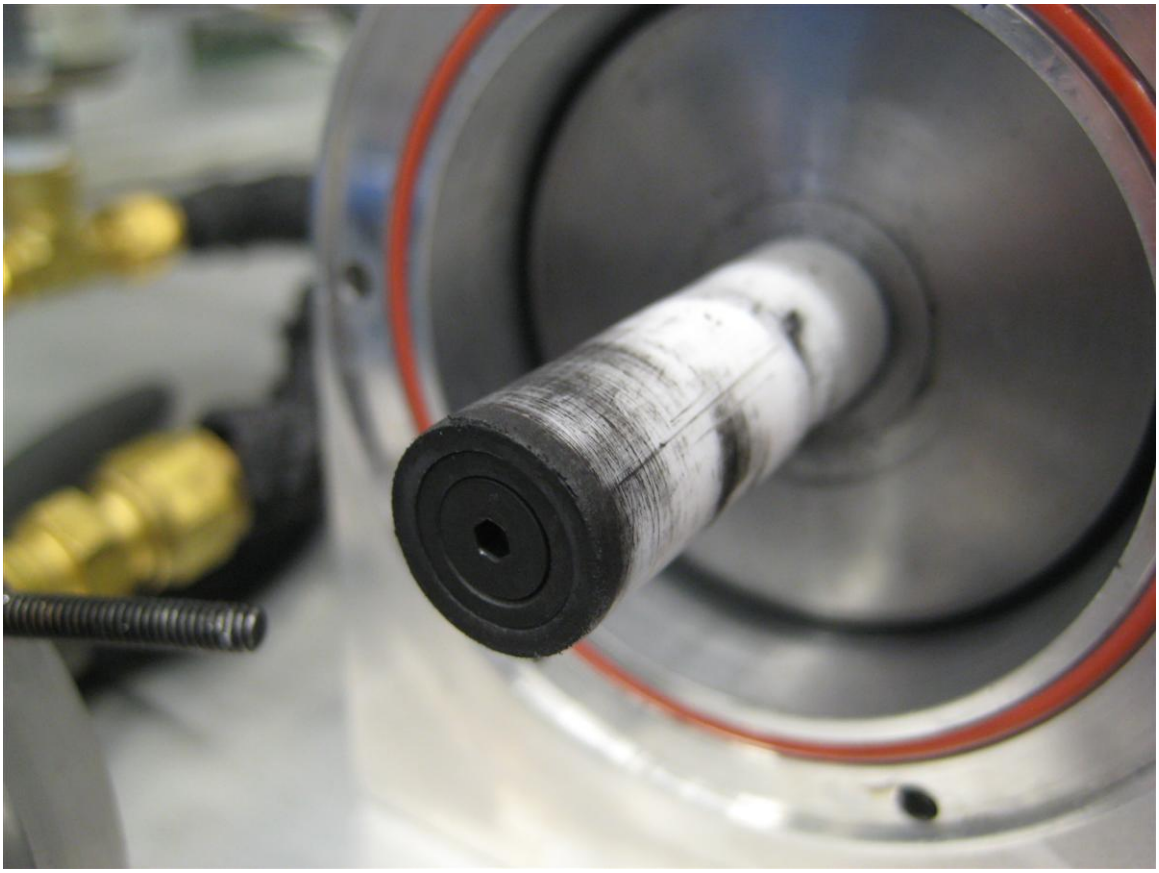


Figure A.14. Close-up view of typical dust from compressor operation on piston, Revision 01.

Figure A.15. Piston squeeze ring resting inside PTFE piston sleeve, Revision 02.

Figure A.16. Piston spacer on the bottom of assembled piston, Revision 02.

Figure A.17. Piston nut and screw installed, untightened, showing full piston assembly, Revision 02.

Appendix B: Compressor Prototype Drawings

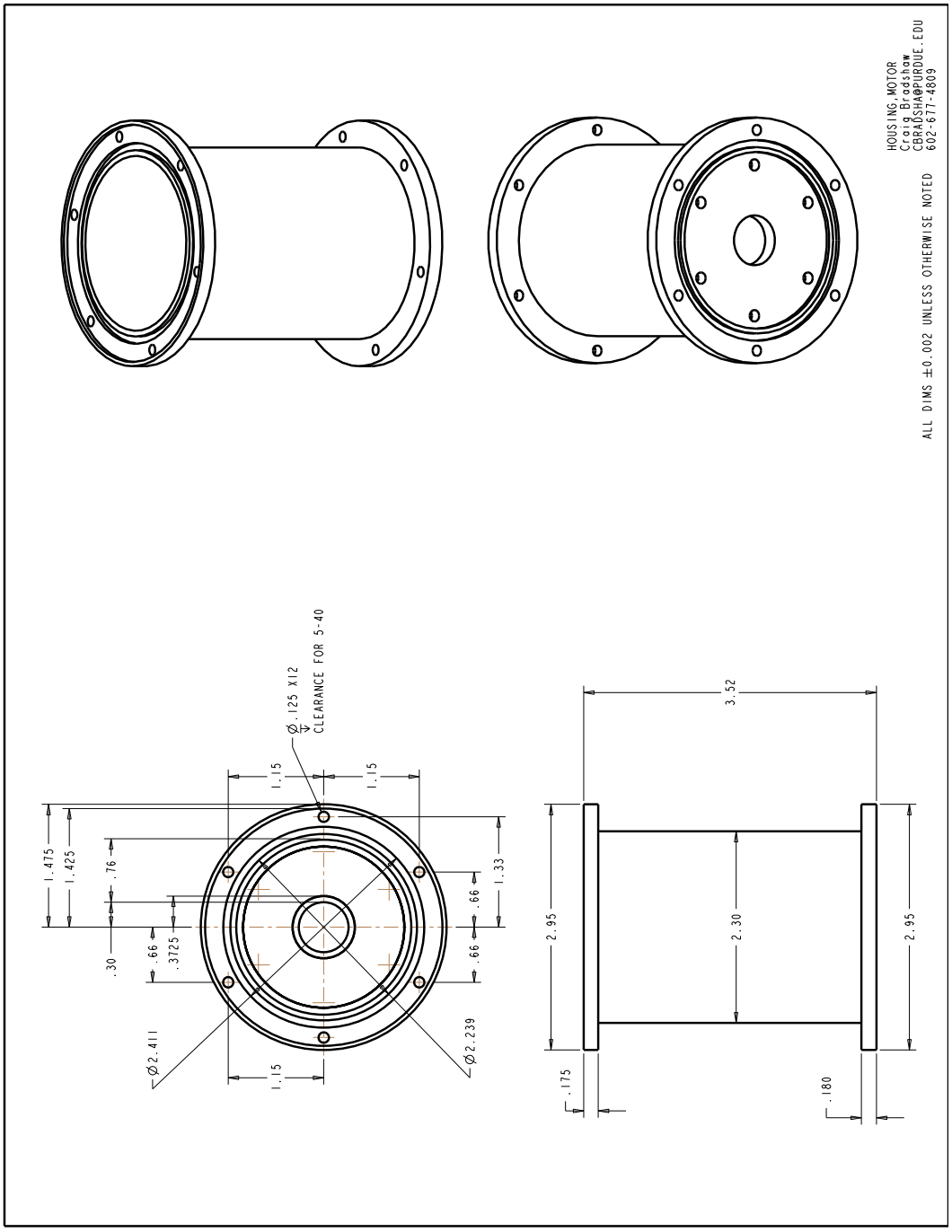


Figure B.1. Prototype linear compressor motor housing page 1.

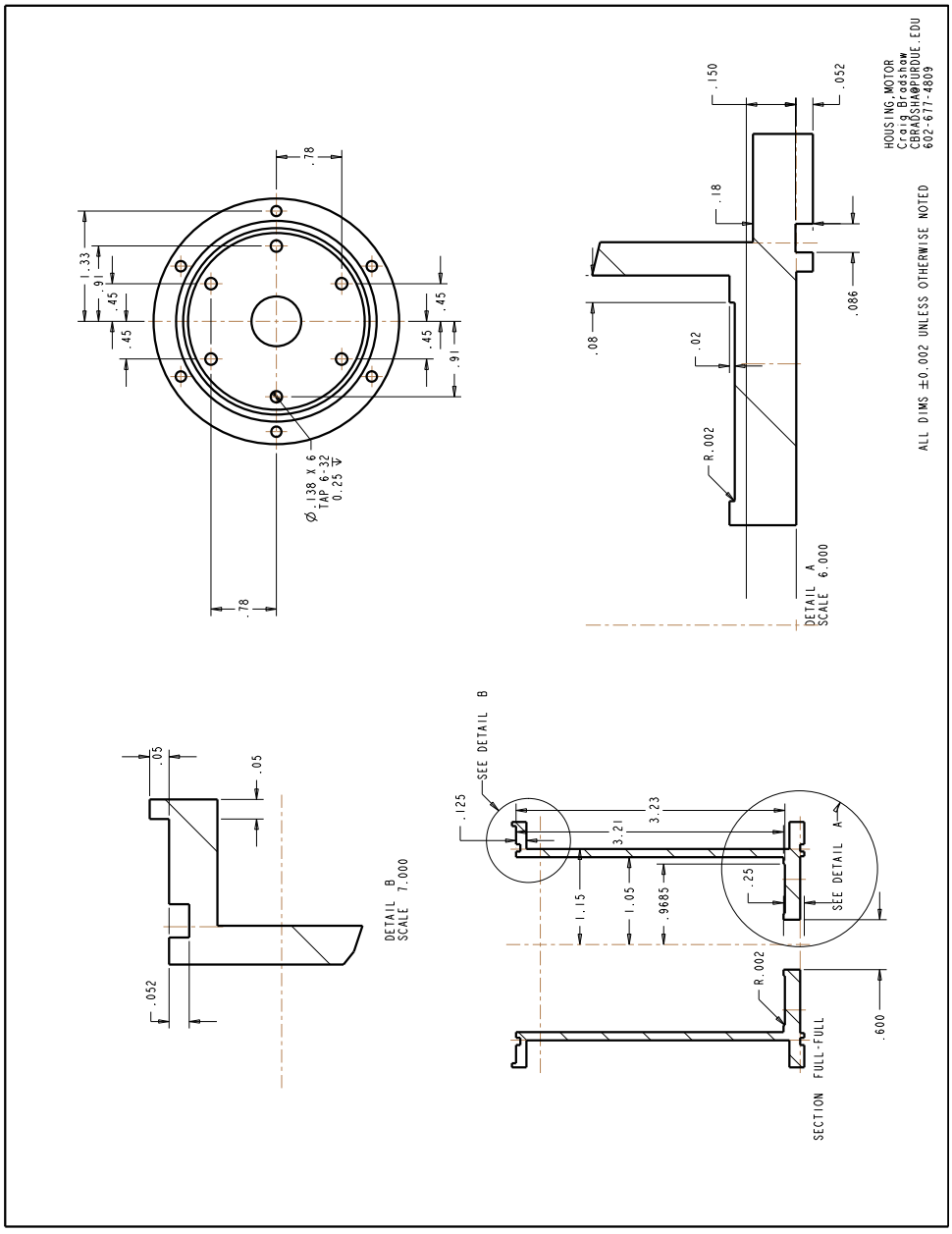


Figure B.2. Prototype linear compressor motor housing page 2.

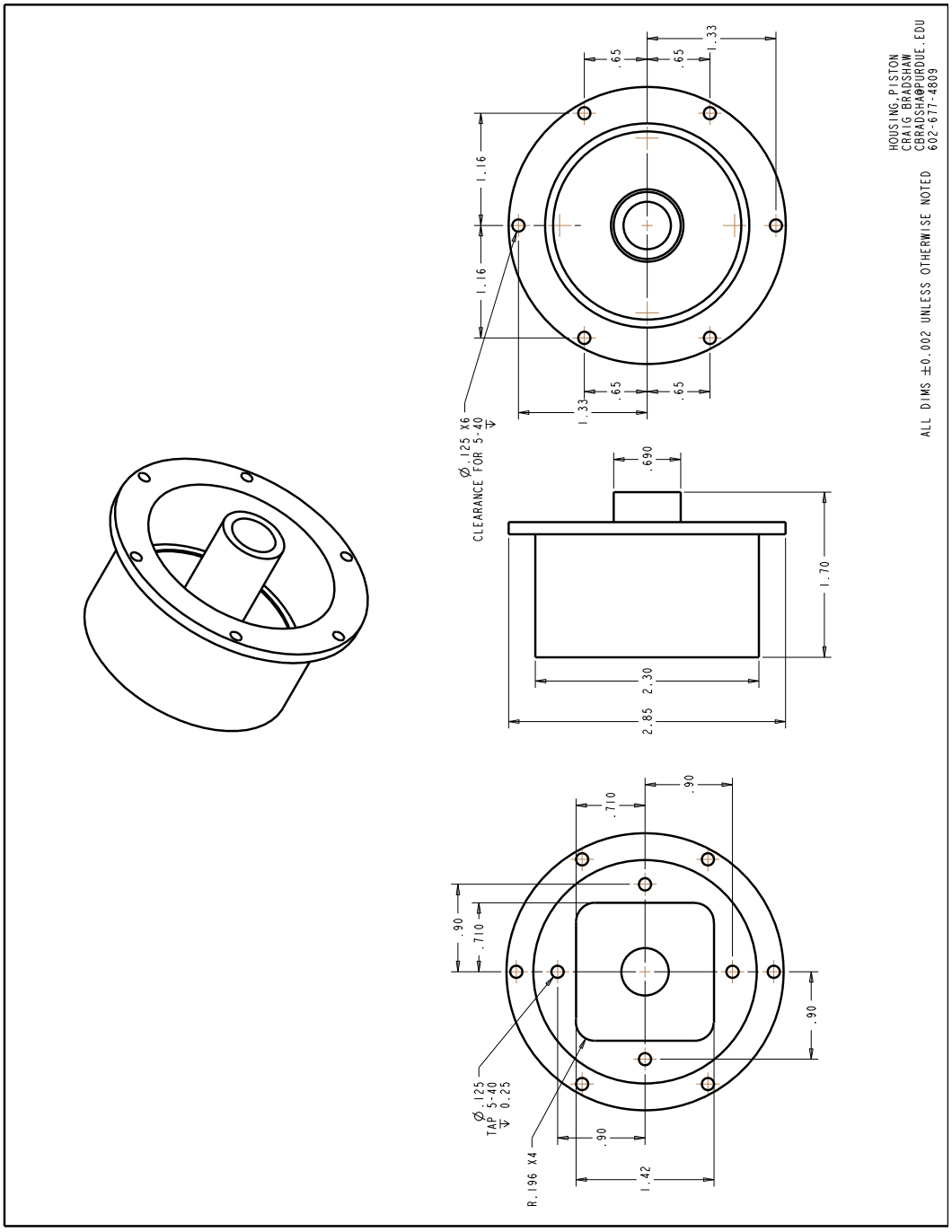


Figure B.3. Prototype linear compressor piston housing page 1.

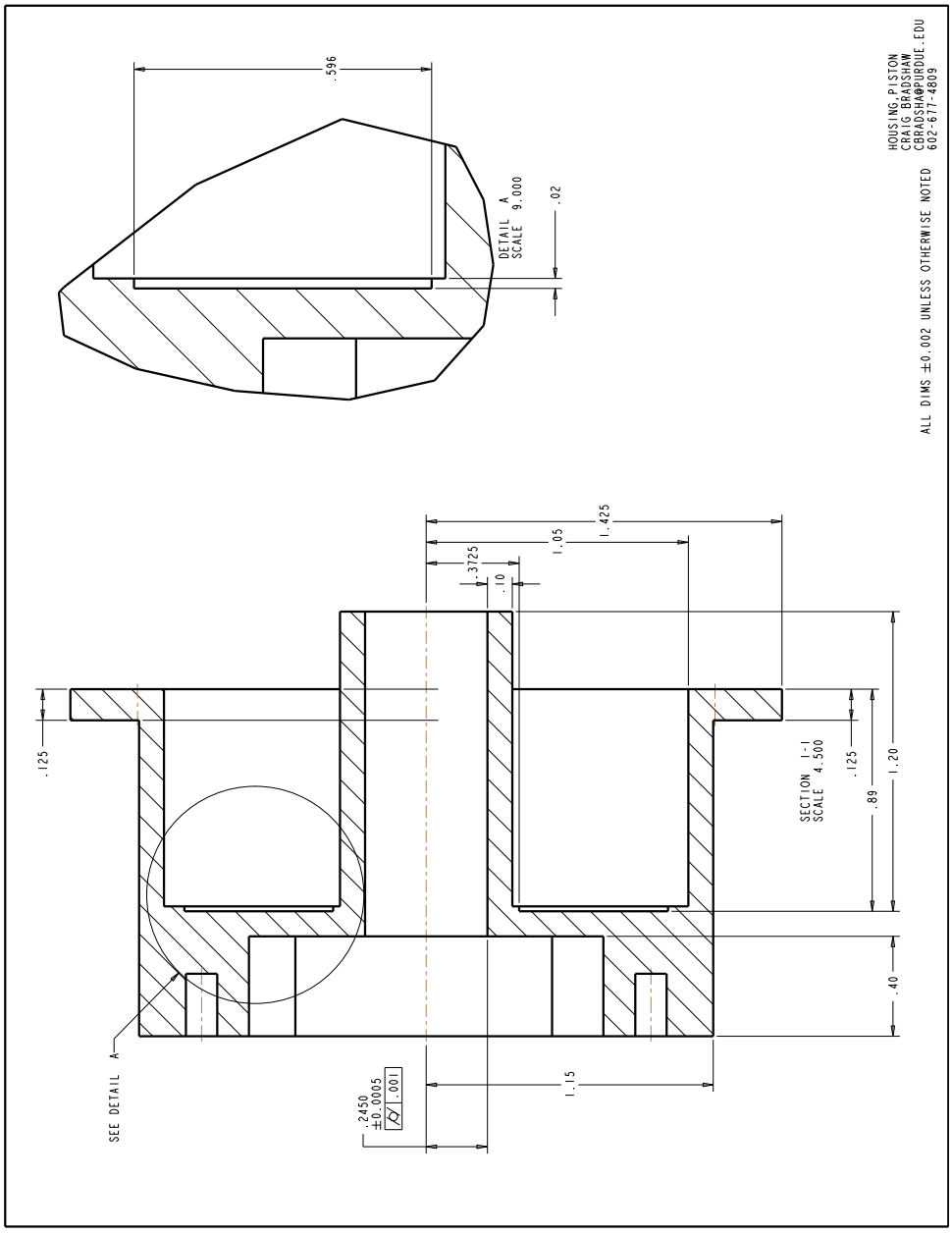


Figure B.4. Prototype linear compressor piston housing page 2.

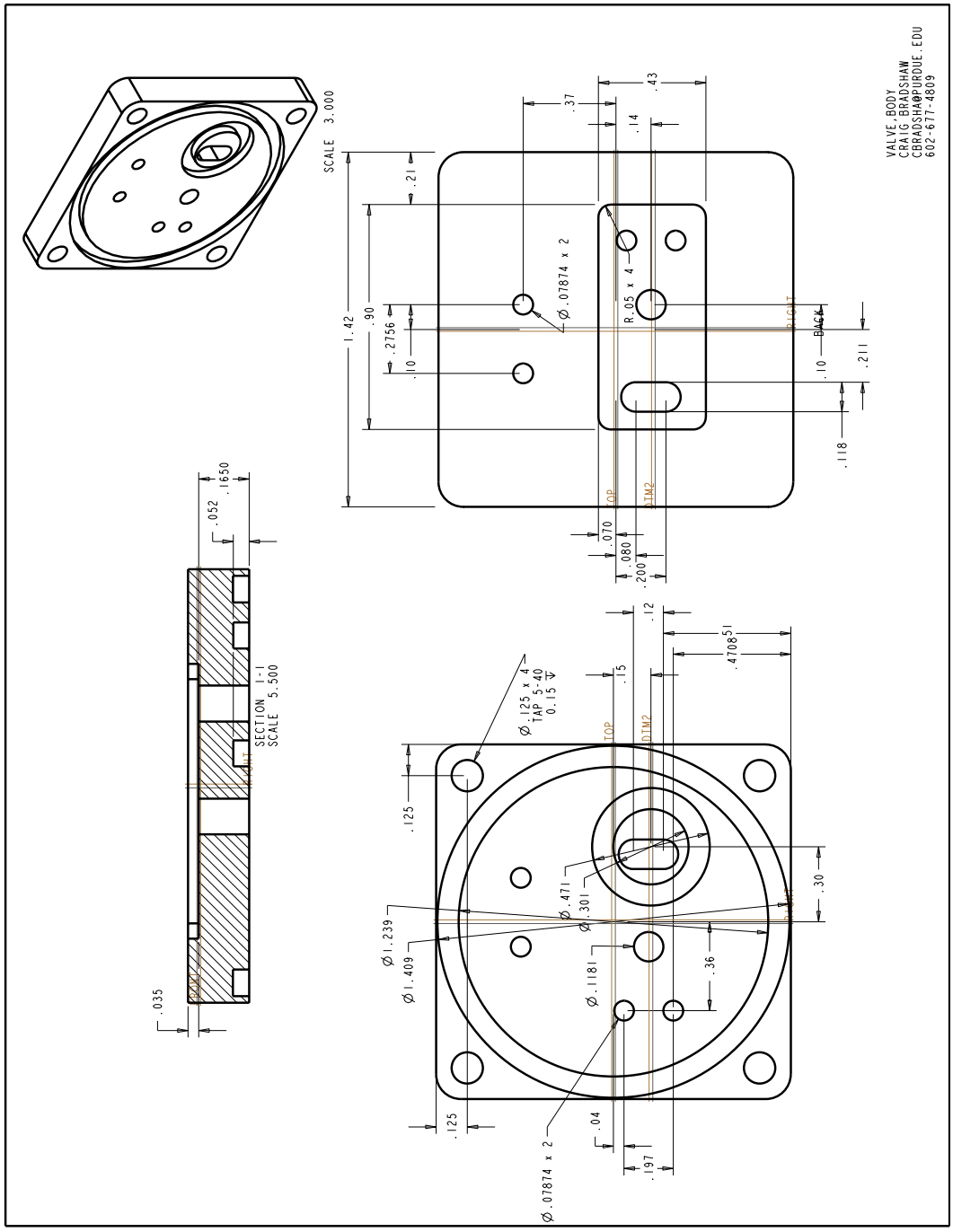


Figure B.5. Prototype linear compressor valve body.

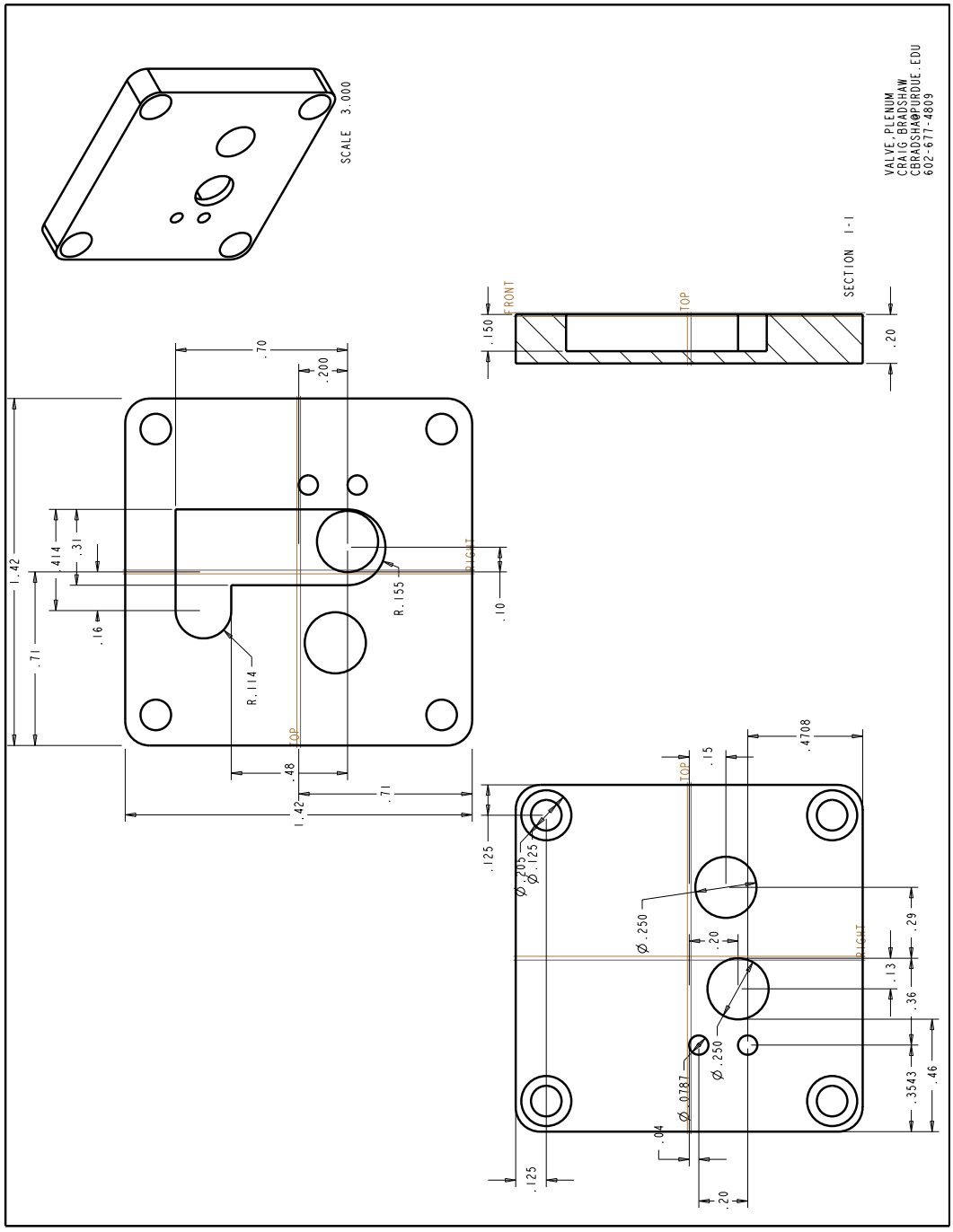


Figure B.6. Prototype linear compressor valve plenum.

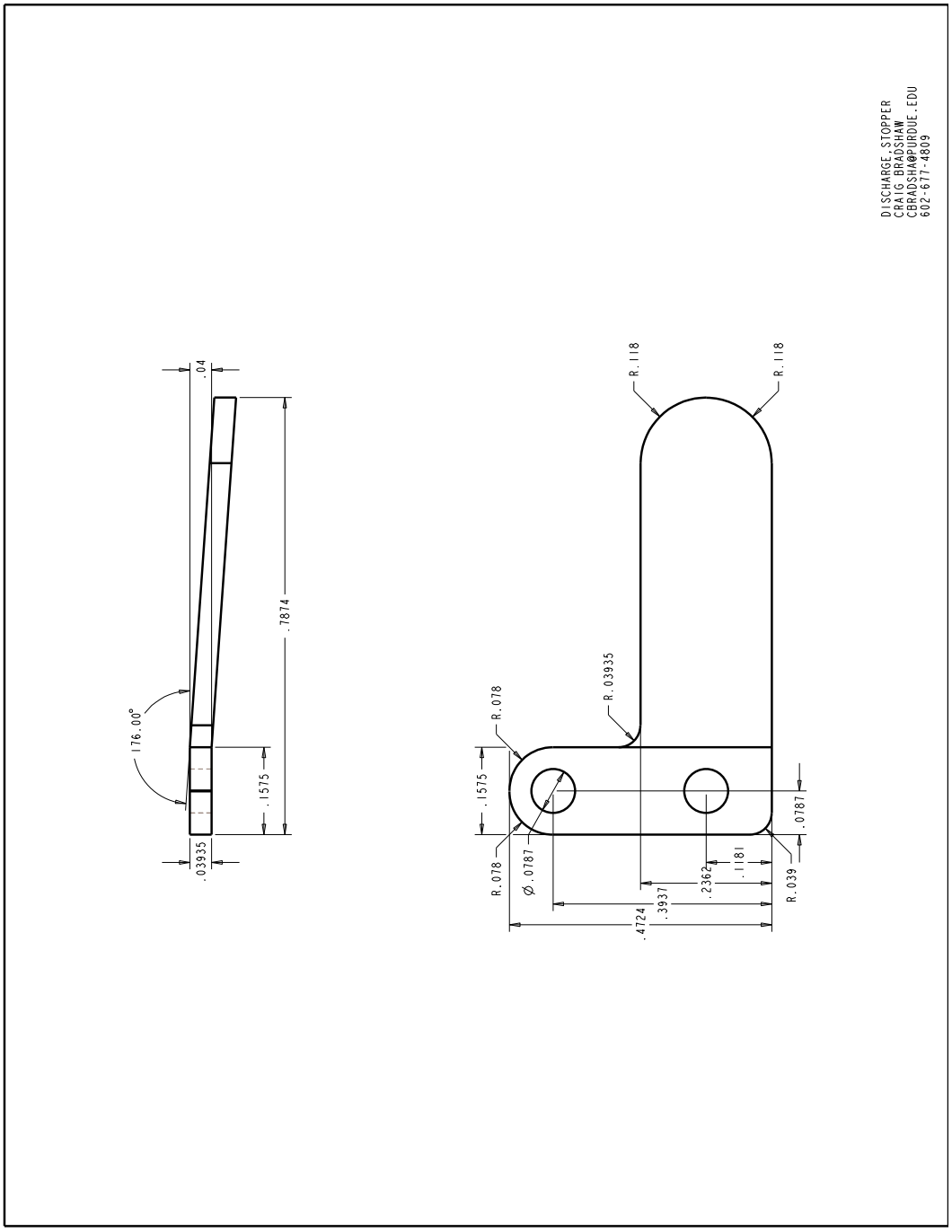


Figure B.7. Prototype linear compressor valve discharge stopper.

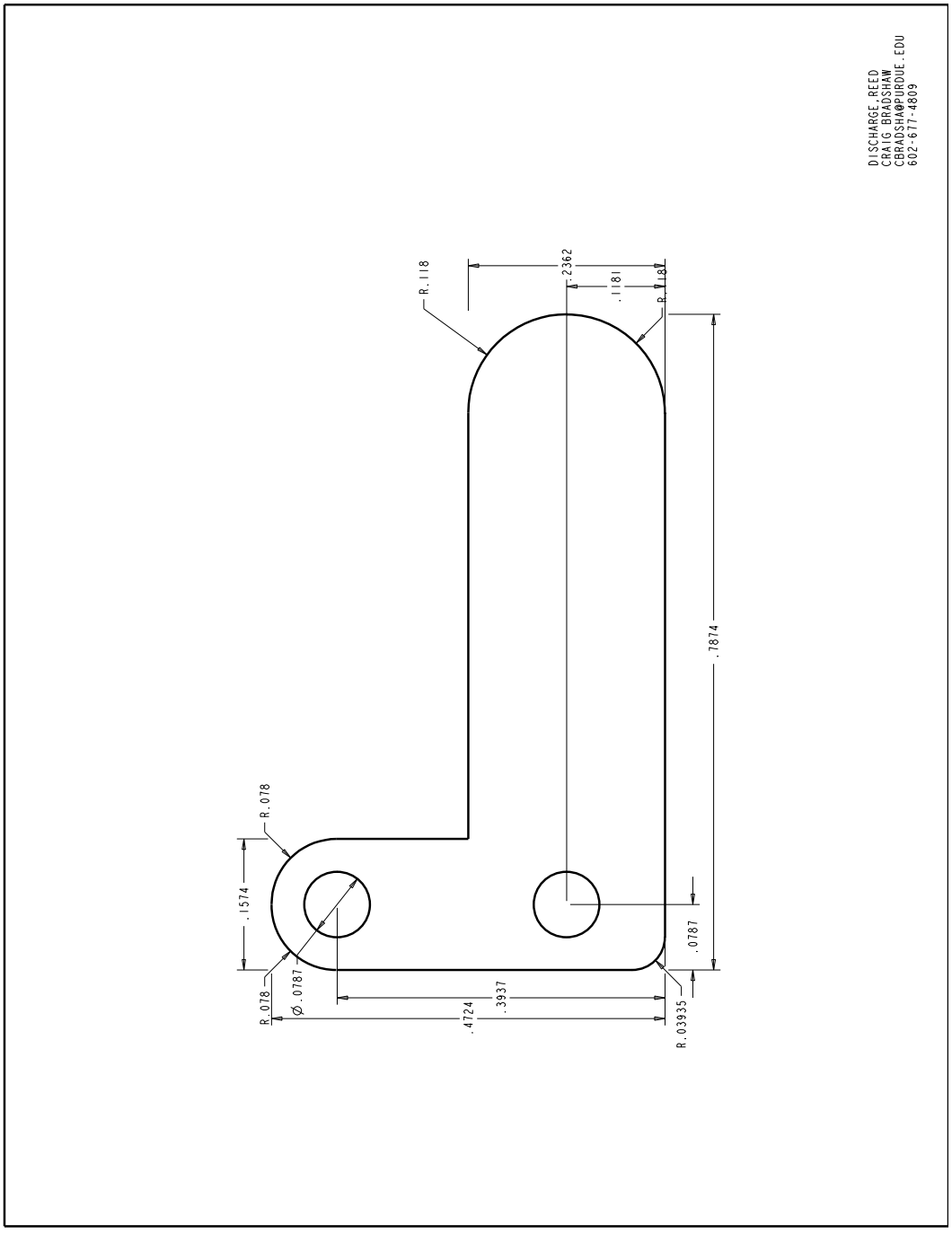


Figure B.8. Prototype linear compressor valve discharge reed.

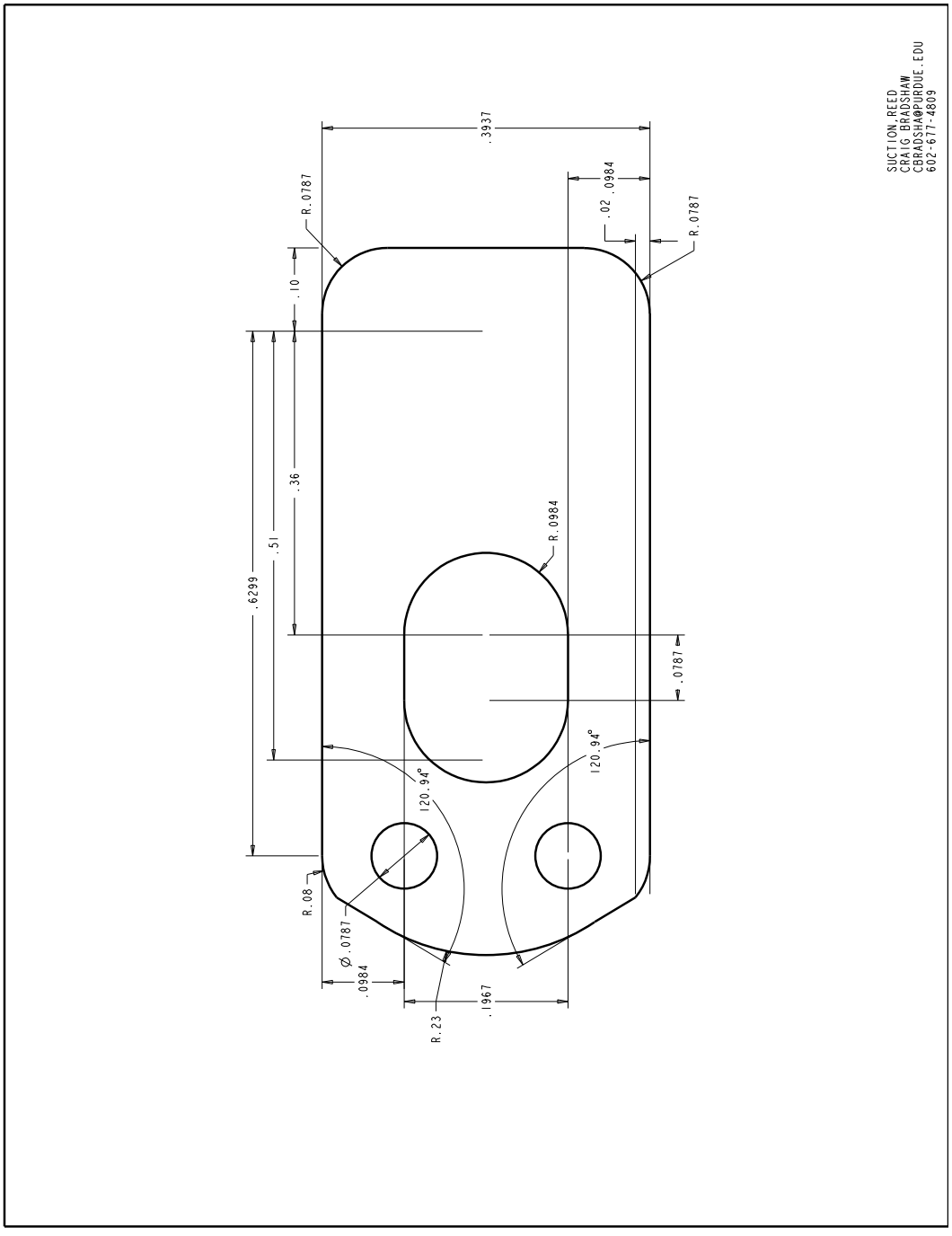


Figure B.9. Prototype linear compressor valve suction reed.

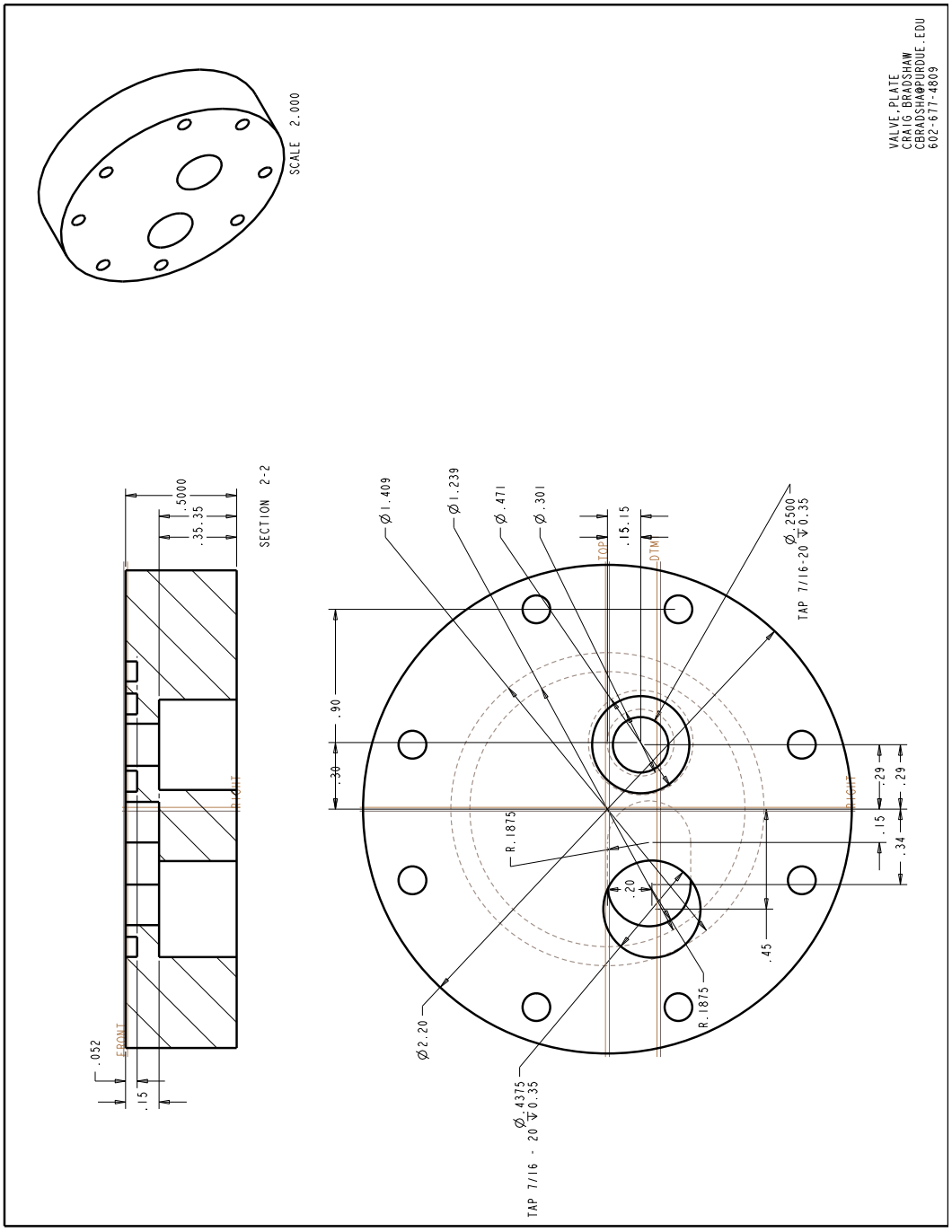


Figure B.10. Prototype linear compressor valve plate.

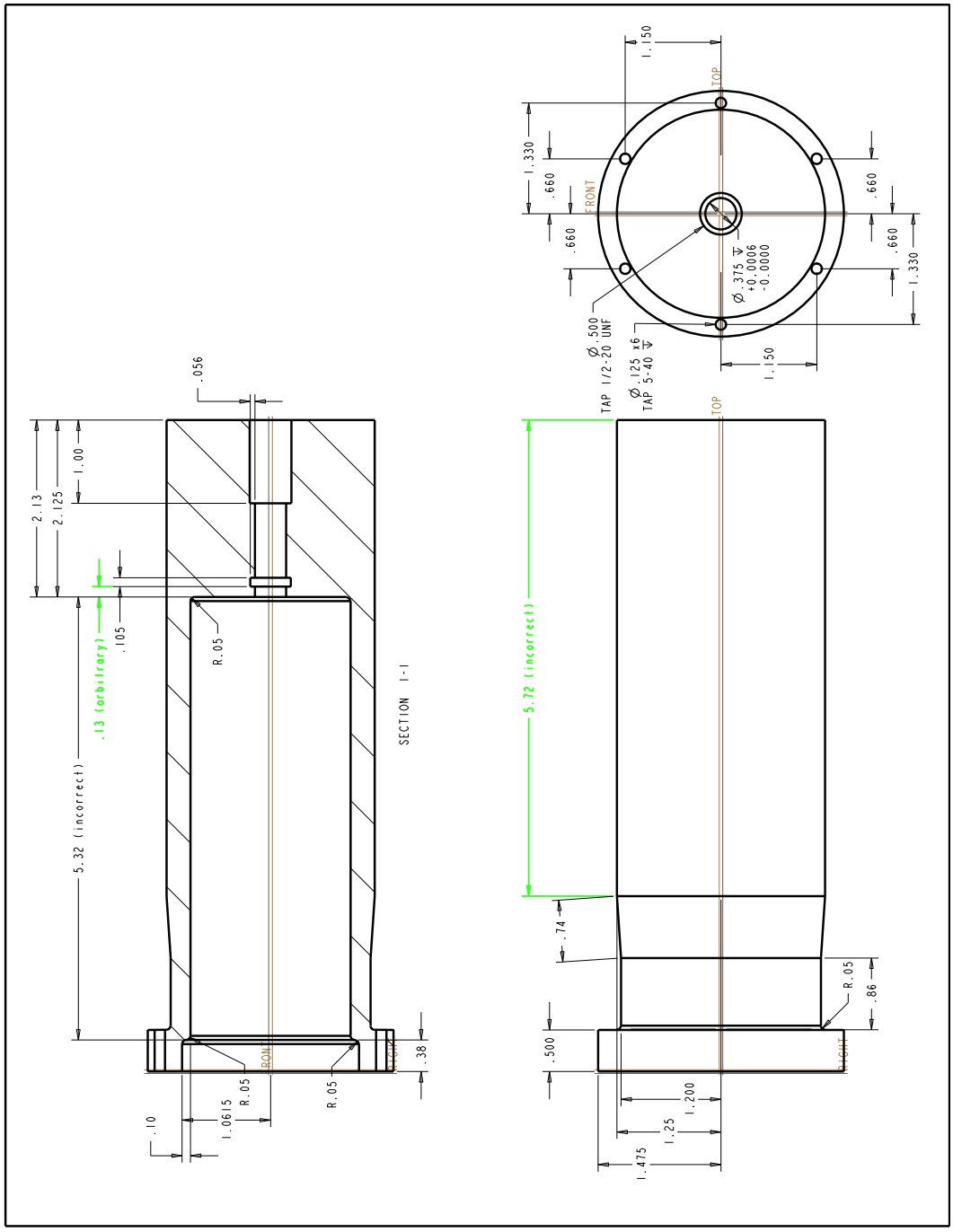


Figure B.11. Prototype linear compressor back cover.

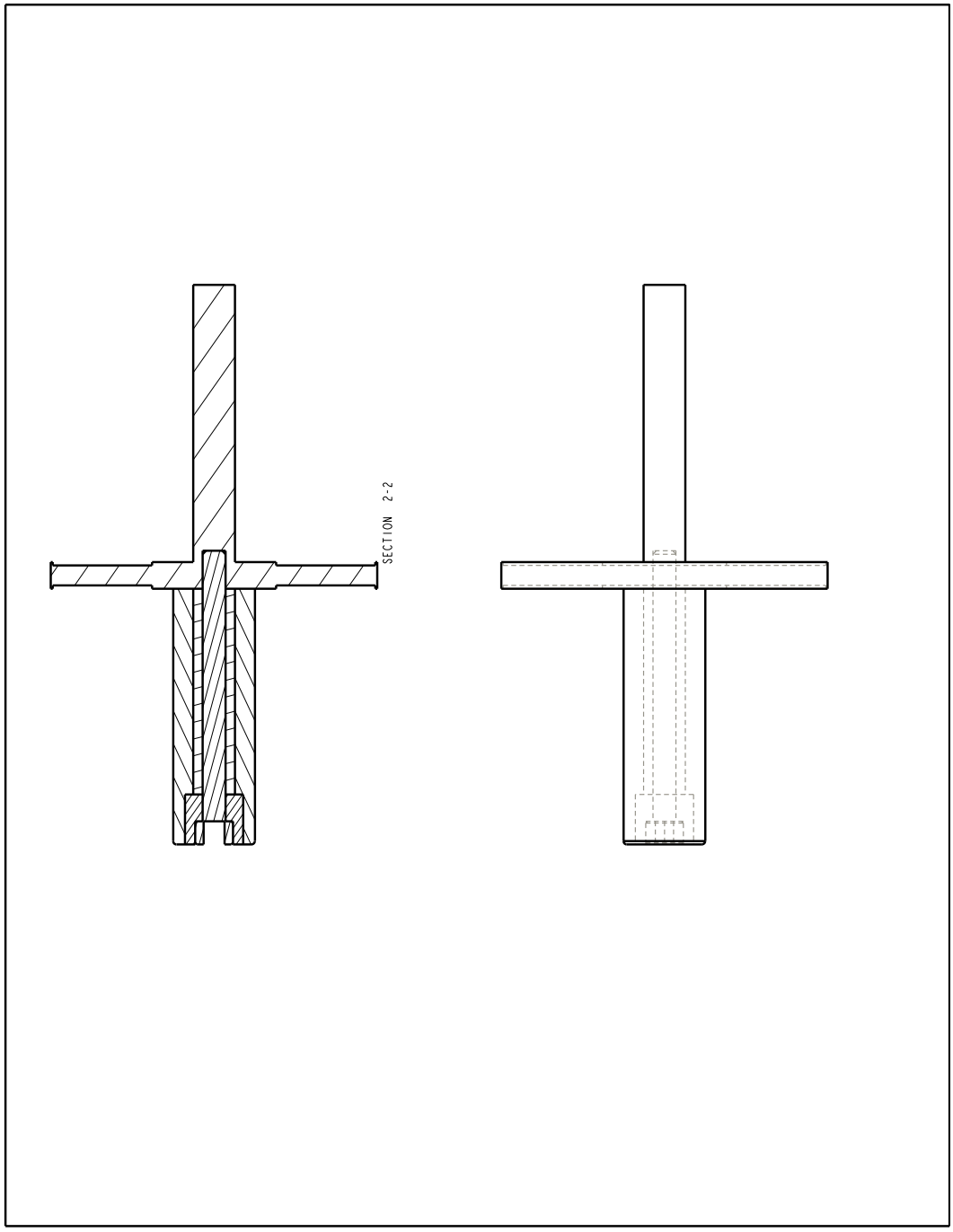


Figure B.12. Prototype linear compressor final piston assembly.

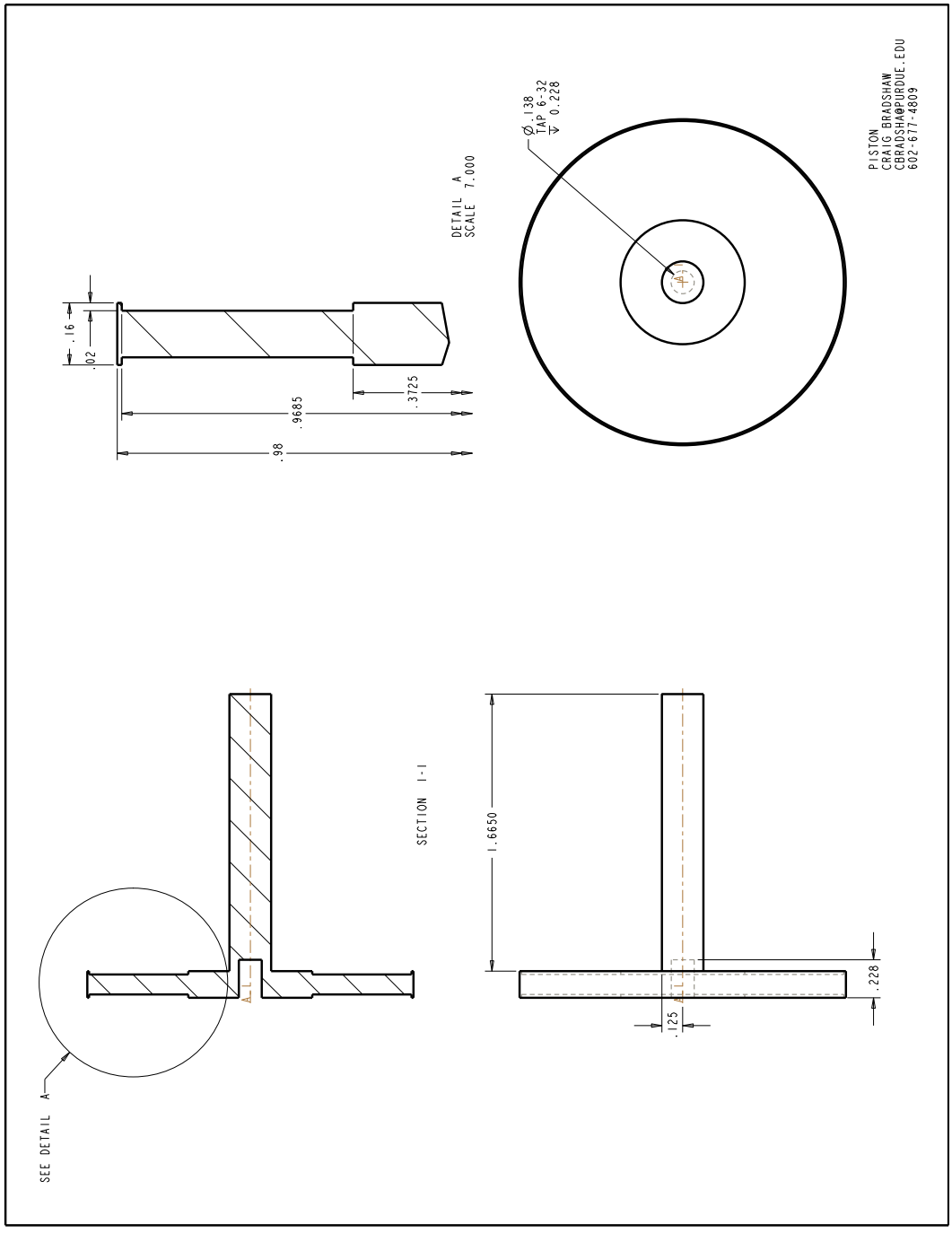


Figure B.13. Prototype linear compressor piston plate.

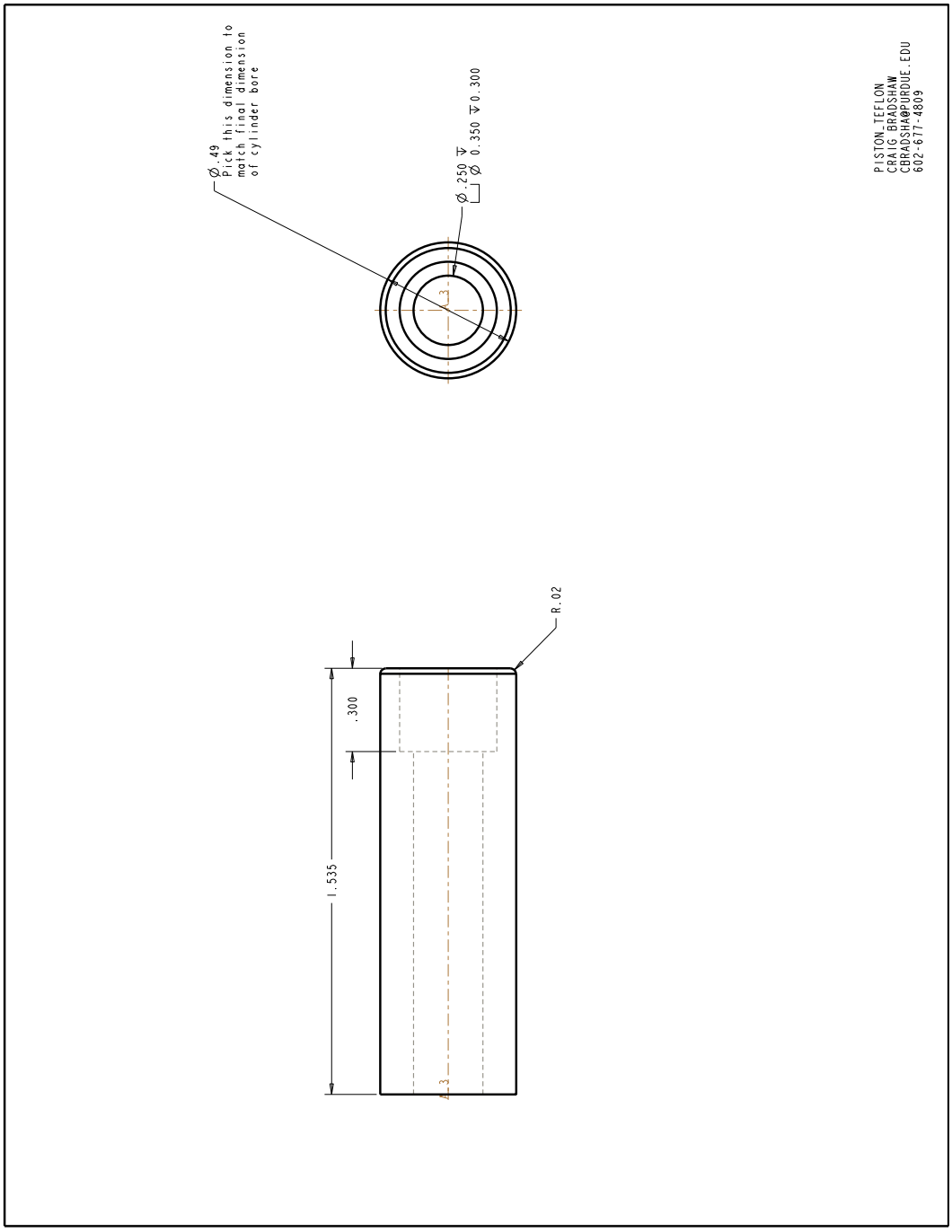


Figure B.14. Prototype linear compressor PTFE piston attachment.

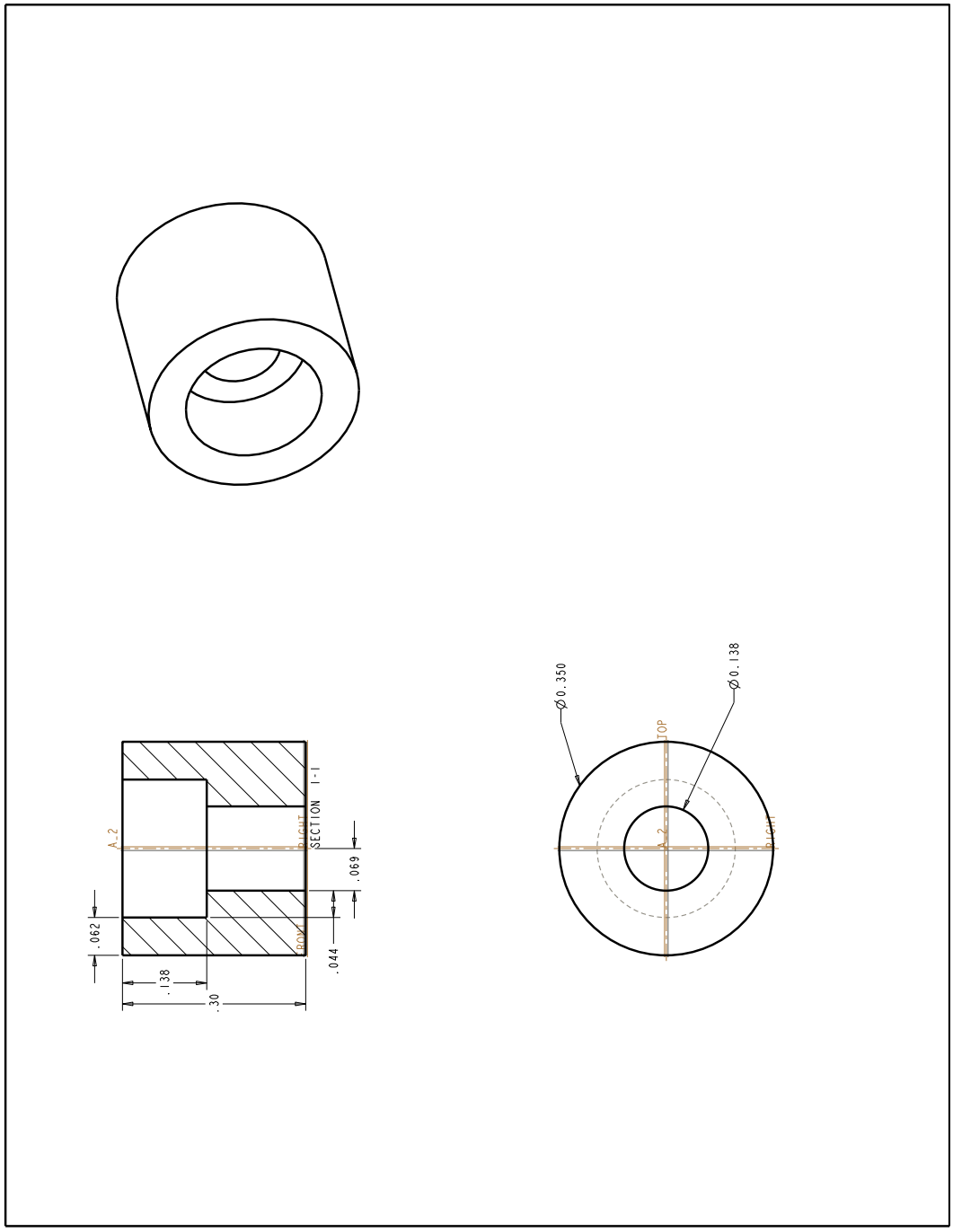


Figure B.15. Prototype linear compressor piston washer.

Appendix C: Comprehensive Model Code

The main linear compressor simulation code is presented here. The program initiates with a call to the main program function called centralprogram. This script is given below with details for all user written functions given in the subsequent sections in the chapter. Figure C.1 shows the logic of the main script file and correlates the submodels presented in Chapters 2 and 3 with the subfunction files presented in this appendix.

The model initiates with the user places the required input values into an Excel spreadsheet titled *inputs.xls*. This spreadsheet is also used to calculate the resonant frequency as detailed in Chapter 3. These input values are then passed into *centralprogram.m* where the simulation is initiated. Each function developed for this work is presented here. However, two functions used in this work are not standard MATLAB® functions but originated from other MATLAB® users. These functions are called *crossing.m* and *find_cross.m*. While they are not detailed in this chapter they are available free of charge with a BSD license on the MATLAB® file exchange. Additionally, built-in MATLAB® functions are not detailed in this chapter.

The equation of state used in the model calculations for R134a is given by Tillner-Roth and Baehr (1994). This equation is written into a series of subfunctions that are called when the function *EOS.m* is called from within the main model script *centralprogram.m*. Figure C.2 shows the logic built into *EOS.m*. The user is required to input temperature and either pressure or density as the second independent intensive property. The routine then returns the requested intensive property (C_v, C_p, u, h, s, ρ or P). The equation of state given by Tillner-Roth and Baehr (1994) is written in both MATLAB® and C and the user is presented with an option to use either the C or MATLAB® version. The advantage to the version written in C is computational speed. While the syntax is slightly different between C and MATLAB the equation of state is identical, therefore, only the MATLAB® subfunctions are presented. Additionally, the user can decide to use ideal gas relationships to calculate properties by

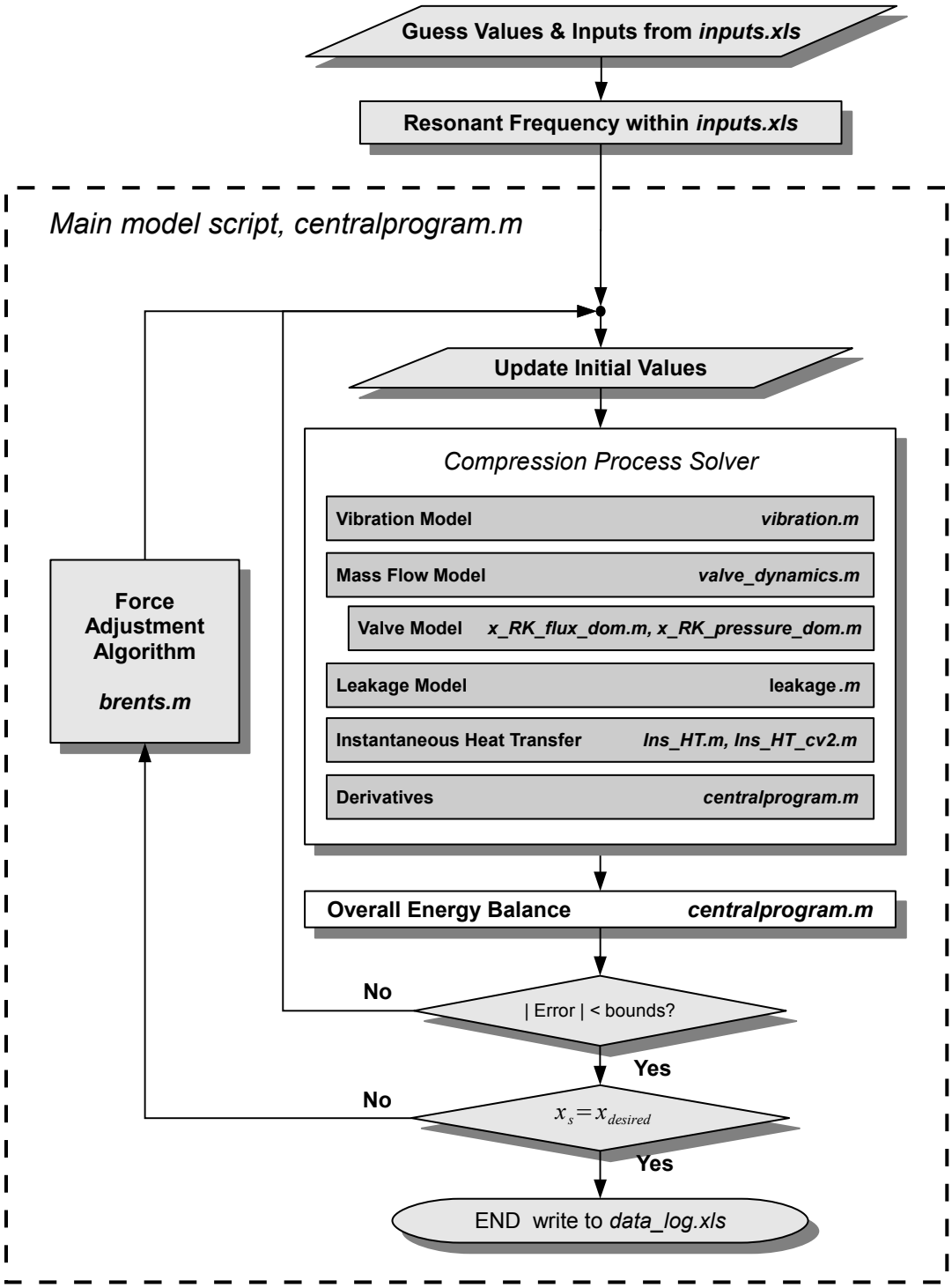


Figure C.1. Linear compressor model flowchart showing which function corresponds to each sub-model.

assuming constant specific heats. The advantage to this method is also speed as it represents the least computationally intensive option and is useful during debugging.

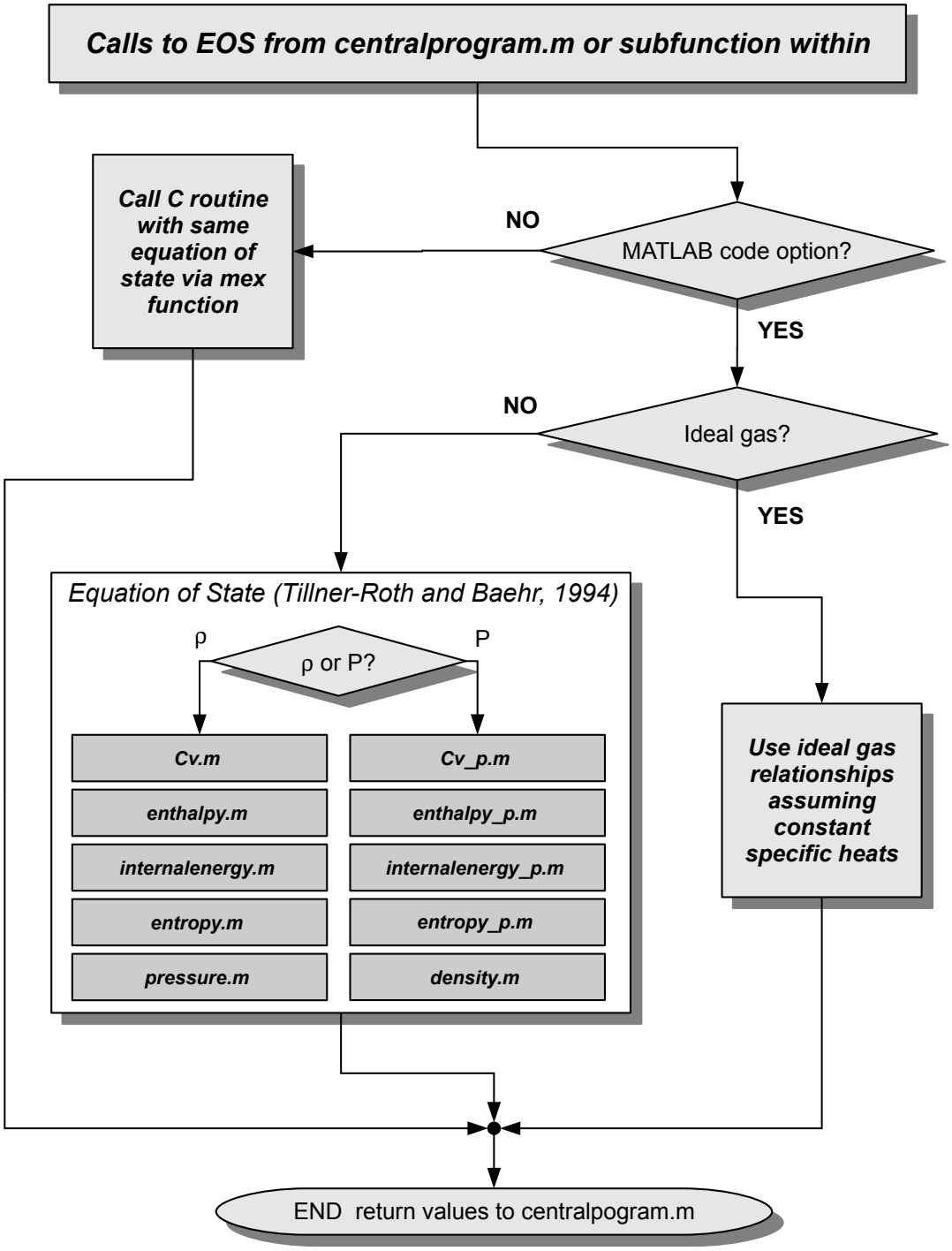


Figure C.2. Equation of state flowchart showing which function corresponds to each property subfunction as well as the different methods available for calculating fluid properties.

C.1 Function *centralprogram*

```
%Main Model Program
%Craig Bradshaw
%Initial Revision: June 2009
%Final Revision: March 2012
%Herrick Labs, cbradsha@purdue.edu

try

    clc
    clear all
    close all

    %% Initializations

    pc_flag=getflag();

    if pc_flag == 0

        %How long do I need to run this simulation for? Obtained from
            first two
        %numbers on the inputs.xls file.
        num_size=xlsread('inputs.xls','A3:A4');
        analy_length=(num_size(2)-num_size(1))+1;

        %Open Excel Activex Server to use xlsread/write for prelim stuff
        Excel = actxserver ('Excel.Application');
        File=strcat(pwd,'\data_log.xls');
        if ~exist(File,'file')
            ExcelWorkbook = Excel.workbooks.Add;
            ExcelWorkbook.SaveAs(File,1);
            ExcelWorkbook.Close(false);
        end
        invoke(Excel.Workbooks,'Open',File);

        %Now using modified xlsread/write to work on data_log.xls
        [num,txt,raw]=xlsread1('data_log.xls');
        [rows,col]=size(raw);

        %Adding a note to the batch log so I know what I was doing
        batch_on=input('Is this a batch file? 1 for yes, 0 for no ');
        if batch_on==1
            batch_note=input('Please input a string to describe why you
                are running this batch.');
            batch_note=cellstr(batch_note);
            batch_range=strcat('A',num2str(rows+2),':A',num2str(rows+2))
                ;
            xlswrite1('data_log.xls',batch_note,batch_range);
        end

        %Shutting down activex server to excel
```

```
48          invoke(Excel.ActiveWorkbook,'Save');
49          Excel.Quit
50          Excel.delete
51          clear Excel
52
53      elseif pc_flag == 1
54
55          input_data = xlsread('inputs.xls');
56
57          %How long do I need to run this simulation for? Obtained from
                first two
58          %numbers on the inputs.xls file.
59          analy_length = (input_data(2,1) - input_data(1,1))+1;
60
61      else
62          error('Invalid pc_flag.  Please use either 0 or 1.');
63          clear all
64
65      end
66
67      %turn annoying warning off
68      warning('off','MATLAB:Print:SavingToDifferentName')
69
70      %%%%%%%%%%%%
71      % For loop used for batch operation, line by line from input file, z
72      % loop
73      %%%%%%%%%%%%
74
75      for z=1:1:analy_length
76
77          if pc_flag == 0
78
79              %pulling input data from inputs.xls
80              range=strcat('B',num2str(z+2),':','BV',num2str(z+2));
81              [num_inputs,txt_inputs]=xlsread('inputs.xls',range);
82
83              %creating my variable structure
84              [p]=create_struct(num_inputs,txt_inputs);
85
86              %name that I use to generate a save folder
87              p.save_name=strcat(p.save_name,'_',num2str(z));
88
89              %Open Excel Activex Server to use xlsread/write for the
                    remainder of
90              %the analysis
91              Excel = actxserver ('Excel.Application');
92              File=strcat(pwd,'\data_log.xls');
93              if ~exist(File,'file')
94                  ExcelWorkbook = Excel.workbooks.Add;
95                  ExcelWorkbook.SaveAs(File,1);
96                  ExcelWorkbook.Close(false);
97              end
```

```
 98              invoke(Excel.Workbooks,'Open',File);
 99
100              %Open data log
101              [num,txt,raw]=xlsread1('data_log.xls');
102              [rows,col]=size(raw);
103
104          elseif pc_flag == 1
105
106              if z == 1
107                  txt_inputs{8} = 'Euler';
108                  txt_inputs{56} = input('Name for Study:  ');
109                  d_name = strcat(txt_inputs{56},'.txt');
110                  diary(d_name)
111              end
112
113              %num_inputs = input_data(z+2,2:61);
114              num_inputs = input_data(z,2:74);
115
116              %Create structure of constant variables
117              [p]=create_struct(num_inputs,txt_inputs);
118
119              %name that I use to generate a save folder
120              p.save_name=strcat(p.save_name,'_',num2str(z));
121
122          end
123
124      %Time/Volume Calculations
125      error=0.00001;      %1e-4
126      f_list=p.f_begin:p.f_increment:p.f_end;
127      w_d_list=f_list*2*pi;
128      p.x_stroke_initial = p.x_stroke;
129
130      %%%%%%%%
131      % Loop used for frequency sweeps, I loop
132      %%%%%%%%
133
134      p.resonant_loop = 0;        %initialize variable that tells
             frequency sweep loop if we have reached the maximum frequency.
135      I=1;                        %loop variable for the following loop.
136
137      while p.resonant_loop<1 && I<=length(w_d_list)
138      %for I=1:1:length(w_d_list)
139
140          close all
141          p.w_d=w_d_list(I);
142          p.Period=1/f_list(I);     %Time of one cycle
143          p.t_step=1/(f_list(I)*p.n);       %time step in seconds
144          p.transient=0;
145          t_adjust_force=0;
146
147          %%%%%%%%%%%%%%%%%%%%%%%%%%
148          %Brents Method estimations
```

```
149         %%%%%%%%%%%%%%%%%%%%%%%%%%
150
151         a = 1;              %loop iteration variable
152         if isnan(num_inputs(72)) && isnan(num_inputs(73))
153             p.P_brent(1) = 10;   %lower power guess
154             p.P_brent(2) = 15; %second lower power guess
155         elseif (isnan(num_inputs(72)) || isnan(num_inputs(73)))
156             error('One Brents method input but not the other, please
                    input two guess values');
157         else
158             p.P_brent(1) = num_inputs(72);
159             p.P_brent(2) = num_inputs(73);
160         end
161
162         error_brents = 0.00000005;    %0.05 microns
163         error_stroke = 1;
164
165         while error_stroke > error_brents   %a loop, brents method to
                back out power for full stroke
166
167             %Initialze residuals and convergence loop iteration counter
168             k=1;
169             dT_loop=1;
170             drho_loop=1;
171             dT_cv2_loop=1;
172             drho_cv2_loop=1;
173             dT_loop_k=1;
174             drho_loop_k=1;
175             dT_cv2_loop_k=1;
176             drho_cv2_loop_k=1;
177             dx_piston=1;
178             if p.brent == 1
179                 p.P_electric = p.P_brent(a);
180             end
181
182             %% Initalize Vectors
183             %These are the vectors that are iterated on and are thus not
                    included in
184             %the parameters structure, i loop variables
185             P=zeros(1,p.n_period*p.n);
186             dT=zeros(1,p.n_period*p.n);
187             h=zeros(1,p.n_period*p.n-1);
188             V=zeros(1,p.n_period*p.n-1);
189             dV=zeros(1,p.n_period*p.n-1);
190             dm=zeros(1,p.n_period*p.n-1);
191             dm_in=zeros(1,p.n_period*p.n-1);
192             dm_out=zeros(1,p.n_period*p.n-1);
193             drho=zeros(1,p.n_period*p.n-1);
194             m=zeros(1,p.n_period*p.n-1);
195             rho=zeros(1,p.n_period*p.n);
196             x_piston=zeros(1,p.n_period*p.n-1);
197             x_dot_piston=zeros(1,p.n_period*p.n-1);
```

```
198             Cv=zeros(1,p.n_period*p.n-1);
199             iter=zeros(1,p.n_period*p.n);
200             dP_dT_v=zeros(1,p.n_period*p.n-1);
201             Ma=zeros(1,p.n_period*p.n-1);
202             dT_check=zeros(1,p.n_period*p.n-1);
203             T=zeros(1,p.n_period*p.n);
204             x_valve=zeros(1,p.n_period*p.n);
205             x_dot_valve=zeros(1,p.n_period*p.n);
206             dm_leak_in=zeros(1,p.n_period*p.n-1);
207             dm_leak_out=zeros(1,p.n_period*p.n-1);
208             dm_out_valve=zeros(1,p.n_period*p.n-1);
209             dV_cv2=zeros(1,p.n_period*p.n-1);
210             V_cv2=zeros(1,p.n_period*p.n-1);
211             Ma_cv2=zeros(1,p.n_period*p.n-1);
212             h_cv2=zeros(1,p.n_period*p.n-1);
213             Cv_cv2=zeros(1,p.n_period*p.n-1);
214             dP_dT_v_cv2=zeros(1,p.n_period*p.n);
215             P_cv2=zeros(1,p.n_period*p.n);
216             drho_cv2=zeros(1,p.n_period*p.n-1);
217             rho_cv2=zeros(1,p.n_period*p.n-1);
218             dT_cv2=zeros(1,p.n_period*p.n-1);
219             T_cv2=zeros(1,p.n_period*p.n-1);
220             F_drive=zeros(1,p.n_period*p.n);
221             Q_motor=zeros(1,p.n_period*p.n);
222             theta=zeros(1,p.n_period*p.n);
223             theta_dot=zeros(1,p.n_period*p.n);
224             dP=zeros(1,p.n_period*p.n);
225             dx=zeros(1,p.n_period*p.n);
226             theta_dot_dot=zeros(1,p.n_period*p.n);
227             Q=zeros(1,p.n_period*p.n);
228             Q_cv2=zeros(1,p.n_period*p.n);
229             m_cv2=zeros(1,p.n_period*p.n);
230             c_eff = zeros(1,p.n_period*p.n);
231             c_friction = zeros(1,p.n_period*p.n);
232             c_gas = zeros(1,p.n_period*p.n);
233             x_dot_dot_piston = zeros(1,p.n_period*p.n);
234             F_gas = zeros(1,p.n_period*p.n);
235             % Initialize convergence loop variables, k loop
236             x_piston_m=0;
237             T_w = zeros(1,p.loop_error);
238             x_stroke_save = zeros(1,p.loop_error);
239             loop = zeros(1,p.loop_error);
240             m_dot = zeros(1,p.loop_error);
241             m_dot_leak_in = zeros(1,p.loop_error);
242             m_dot_leak_out = zeros(1,p.loop_error);
243             W_dot = zeros(1,p.loop_error);
244             eta_o = zeros(1,p.loop_error);
245             eta_vol = zeros(1,p.loop_error);
246             Q_dot = zeros(1,p.loop_error);
247             Q_dot_cv2 = zeros(1,p.loop_error);
248             m_dot_in = zeros(1,p.loop_error);
249             loop=1;     %loop counter variable, re-initialize
```

```
250             p.dP_max=p.P_d-p.P_i;
251             P_electric_temp = p.P_electric;         %save input value
252             error_res = 1;          %initialize error_res
253             error_counter = 0;      %count number of times get below
                    tolerance
254              %%%%%%%%%%%
255              %%  while loop to enforce convergence, k loop
256              %%%%%%%%%%%
257
258             while error_counter < 2
259
260                 W_dot_in(k) = p.P_electric;
261
262                 %% Inital Cylinder Conditions
263                 if k==1
264                     i=1;   %initialize i loop counter
265                     T(i)=p.T_i;   %initial temperature
266                     rho(i)=p.rho_i;  %initial density
267                     P(i)=EOS(T(i),rho(i),'pressure','rho');  %calculate
                            initial pressure
268                     T_cv2(1)=p.T_i;    %initial temperature, CV2
269                     rho_cv2(1)=p.rho_i;
                                                          %initial
                            density, CV2
270                     %P_cv2(1)=P(1);
271                     P_cv2(1)=(p.P_d + p.P_i)/2;  %initial pressure, CV2
272                     %rho_cv2(1) = EOS(T_cv2(1),P_cv2(1),'rho','P');
273                     x_piston(1)=p.x_piston_i; %initial piston position,
                            default is (max stroke/2)
274                     x_dot_piston(1)=p.x_dot_piston_i;   %initial piston
                            velocity, default is 0
275                     V(1)=(p.x_piston_i)*p.Ap+p.V_dead_valve;  %initial
                            volume
276                     dV(1)=-p.x_dot_piston_i*p.Ap;  %initial change in
                            volume
277                     theta(1)=p.theta_i;   %initial piston rotation
278                     theta_dot(1)=p.theta_dot_i;  %initial change in
                            piston rotation
279                     p.x_dot_ave=1; %%%%%%IMPORTANT, AVERAGE SPEED
280                     dP(1)=0;   %Change in pressure
281                     dx(1)=1;    %Change in stroke
282                     T_w(k)=p.T_w_i;    %initial compressor lump
                            temperature (wall temp)
283                     x_stroke_save(k)=p.x_stroke_initial; %saved value of
                             stroke
284                     p.x_stroke = p.x_stroke_initial;  %guess stroke
285                     p.dP_max = p.P_d-p.P_i;
286                     t=0;
287                     p.dV = 0;   %initialize dV
288                     p.P_current = P(i);
289                 else
290
```

```
291                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
292                 %Update values for iterations beyond first
293                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
294                 T_error = T;
295                 rho_error = rho;
296                 T_error_cv2 = T;
297                 rho_error_cv2 = rho;
298                 x_stroke_error = p.x_stroke;
299                 T(1)=X_n_1(1); %Temperatures and pressures equal to
                        the end values of the previous iteration
300                 rho(1)=X_n_1(2);
301                 P(1)=EOS( T(1),rho(1),'pressure','rho' );
302                 T_cv2(1)=X_n_1(3);
303                 rho_cv2(1)=X_n_1(4);
304                 P_cv2(1)=EOS( T_cv2(1),rho_cv2(1),'pressure','rho' )
                        ;
305                 p.x_stroke=max(x_piston_m)-min(x_piston_m);
                                %max stroke, current value
306                 p.x_dot_ave=0.707*max(x_dot_piston);
                                %%%%%%IMPORTANT, AVERAGE
                        SPEED, RMS value of speed
307                 x_stroke_save(k)=p.x_stroke;
                                                % stroke variable
                        saved to see progression
308                 x_piston(1)=x_piston(i);
309                 x_dot_piston(1)=x_dot_piston(i);
310                 V(1)=V(i);
311                 dV(1)=dV(i);
312                 theta(1)=theta(i);
313                 theta_dot(1)=theta_dot(i);
314                 t=t(i);
315                 dP(1)=dP(i);
316                 dx(1)=dx(i);      %resets remaining variables to the
                        values of the previous k loop
317                 i=1;                %resets inner loop variable
318             end
319
320             %% Initalize inner loop variables
321             t_cross=0;                  %number of times piston
                    crosses zero
322             p.t_force_adjust=0;  %adjustment in the time vector to
                    account for variable frequency response
323             x_piston_m=0;          %initialize x_piston
324             F_drive=0;                 %initalize driving force
                    vector
325             F_drive_clean=0;
326
327             %%%%%%%%%%
328             %% Main Loop for calculations, i loop
329             %%%%%%%%%%
330             while size(t_cross)<p.n_period*2-1
331
```

```
332                     %% Motor Model
333                     if isempty(p.t_force_adjust) == 1
334                         p.t_force_adjust=0;
335                     end
336
337                     if i == 1    %only want to perform these
                            calculations on first time through
338                         p.P_motor=p.eta_motor*p.P_electric;
                                                %power actually delivered to
                                 piston
339                         p.Q_motor=p.P_electric-p.P_motor;
                                                 %Heat transfer from motor
340
341                         %Converts power input to force output by motor,
342                         %based on motor literature
343                         p.k_m = 4.6831;
344
345                         p.F_drive_max=sqrt(p.P_motor)*(p.k_m);
                                               %Maximum Motor force, in
                                    Newtons.
346                     end
347
348                     F_drive_clean(i)=p.F_drive_max*(cos(p.w_d*(t(i)-p.
                            t_force_adjust)+pi));                 %Driving
                            force v. Time, Newtons
349                     p.dP_piston = P(i) - P_cv2(i);
350                     F_drive(i)=F_drive_clean(i) - p.dP_piston*p.Ap*1000;
351
352                     %% Vibration Model and Volume and Piston Position
353                     p.dP_piston = P(i) - P_cv2(i);
354                     [ dV(i+1),V(i+1),x_piston(i+1),x_dot_piston(i+1),
                            theta(i+1),theta_dot(i+1),x_piston_m(i+1),
                            theta_dot_dot(i+1),p ] = vibration( t(i),dP(i),
                            dx(i),x_piston(i),x_dot_piston(i),theta(i),
                            theta_dot(i),x_piston_m(i),theta_dot_dot(i),p );
355
356                     c_eff(i) = p.c_eff;
357                     c_friction(i) = p.c_friction;
358                     c_gas(i) = p.c_gas;
359                     x_dot_dot_piston(i) = p.x_dot_dot_piston;
360                     k_eff(i) = p.k_eff;
361                     F_gas(i) = (P(i) - P_cv2(i))*p.Ap;
362                     F_wall(i) = p.F_wall;
363                     p.dV = V(i+1) - V(i);
364                     p.P_current = P(i);
365
366                     %% Volume for Second Control Volume
367                     dV_cv2(i)=dV(i);
368                     V_cv2(i)=p.V_cv2_i-V(i);
369
370                     %% Massflow (Valve Model)
```

```
371                     [ x_valve(i+1),x_dot_valve(i+1),dm_valve, Ma(i) ] =
                            valve_dynamics(P(i),rho(i),T(i),x_valve(i),
                            x_dot_valve(i),p);
372
373                     %% Leakage Flowrates (Leakage Model)
374                     [dm_leak_in(i),dm_leak_out(i),Ma_cv2(i)] = leakage(
                            P(i),P_cv2(i),x_dot_piston(i),T(i),rho(i),T_cv2(
                            i),x_piston(i),p );
375
376                     %% Total flowrates (Massflow Summations)
377                     if dm_valve<0     %flow IN
378                         dm_in(i)=abs(dm_valve)+dm_leak_in(i);
379                         dm_out(i)=dm_leak_out(i);
380                         dm(i)=dm_in(i)-dm_out(i);
381                         dm_out_valve(i)=0;
382                     elseif dm_valve>0 %flow OUT
383                         dm_in(i)=dm_leak_in(i);
384                         dm_out(i)=dm_valve+dm_leak_out(i);
385                         dm(i)=dm_in(i)-dm_out(i);
386                         dm_out_valve(i)=dm_valve;
387                     else
388                         dm_in(i)=dm_leak_in(i);
389                         dm(i)=dm_leak_in(i)-dm_leak_out(i);
390                         dm_out(i)=dm_leak_out(i);
391                         dm_out_valve(i)=0;
392                     end
393
394                     %% Instantaneous Heat Transfer in control volumes
395                     [ Q(i) ] = Ins_HT( T(i),rho(i),T_w(k),V(i),dV(i),
                            x_piston(i),x_dot_piston(i),p);
396                     [ Q_cv2(i) ] = Ins_HT_cv2( T_cv2(i),rho_cv2(i),T_w(k
                            ),V_cv2(i),dV_cv2(i),x_dot_piston(i),p.Q_motor,p
                            );
397
398                     %% Mass of the gas in each control volume
399                     m(i)=rho(i)*V(i);
400                     m_cv2(i)=rho_cv2(i)*V_cv2(i);
401
402                     %% Fluid Properties for approximation
403                     if k==1     %only need to compute this value once
                            for each batch line
404                         p.h_in=EOS(p.T_i,p.rho_i,'enthalpy','rho');
405                     end
406                     h_out=EOS(T(i),rho(i),'enthalpy','rho');
407                     h(i)=h_out;
408                     dP_dT_v(i)= dP_dT(T(i),rho(i));
409                     Cv(i)=EOS(T(i),rho(i),'Cv','rho');
410                     h_cv2(i)=EOS(T_cv2(i),rho_cv2(i),'enthalpy','rho');
411                     dP_dT_v_cv2(i)=dP_dT(T_cv2(i),rho_cv2(i));
412                     Cv_cv2(i)=EOS(T_cv2(i),rho_cv2(i),'Cv','rho');
413
414                     if strcmp(p.method,'RK4')==1
```

```
415                         %% RK4 Approximations
416
417                         error('RK4 not available in this version');
418
419                     elseif strcmp(p.method,'Euler')==1
420                         %% Simple Euler Approximation
421                         %%%%%
422                         %1st Control Volume, Piston Cylinder
423                         %%%%%
424                         dT(i)=(Q(i)+T(i)*dP_dT_v(i)*(1/1000)*((1/rho(i))
                                *dm(i)-dV(i))-h(i)*dm(i)+dm_in(i)*p.h_in-
                                dm_out(i)*h(i))/(m(i)*Cv(i));
425                         drho(i)=(dm(i)-rho(i)*dV(i))/V(i);
426
427                         T(i+1)=T(i)+p.t_step*dT(i);
428                         rho(i+1)=rho(i)+p.t_step*drho(i);
429                         %%%%%
430                         %2nd Control Volume
431                         %%%%%
432                         dm_cv2=dm_leak_out(i)-dm_leak_in(i);
433                         dm_in_cv2=dm_leak_out(i);
434                         dm_out_cv2=dm_leak_in(i);
435
436                         dT_cv2(i)=(Q_cv2(i)+T_cv2(i)*dP_dT_v_cv2(i)
                                *(1/1000)*((1/rho_cv2(i))*dm_cv2-dV_cv2(i))-
                                h_cv2(i)*dm_cv2+dm_in_cv2*h(i)-dm_out_cv2*
                                h_cv2(i))/(m_cv2(i)*Cv_cv2(i));
437                         drho_cv2(i)=(dm_cv2-rho_cv2(i)*dV_cv2(i))/V_cv2(
                                i);
438
439                         T_cv2(i+1)=T_cv2(i)+p.t_step*dT_cv2(i);
440                         rho_cv2(i+1)=rho_cv2(i)+p.t_step*drho_cv2(i);
441
442                     else
443                         i=floor(p.Period/p.t_step)-1;
444                         disp('Method error, please select a correct ODE
                                Solution Method')
445                     end
446
447                     %%%%%%%%%%%%%%%%%%
448                     %% Calculate next step for pressure
449                     %%%%%%%%%%%%%%%%%%
450                     P(i+1)=EOS(T(i+1),rho(i+1),'pressure','rho');
451                     P_cv2(i+1)=EOS(T_cv2(i+1),rho_cv2(i+1),'pressure','
                            rho');
452
453                     %%%%%%%%%%%%
454                     %% Misc other properties
455                     %%%%%%%%%%%%
456                     dP(i+1)=P(i+1)-P(i);                    %Change in
                            pressure
```

```
457                     dx(i+1)=x_piston(i+1)-x_piston(i);      %change in
                            piston position
458
459                     %%%%%%%%%%%
460                     %% Loop Adjustments
461                     %%%%%%%%%%%
462                     %Section determines when the piston crosses back
463                     %through the midpoint, signifying one cycle.
464                     if k>1
465                         if i>1
466                             t_cross=find_cross(t',x_piston_m(1:end-1)
                                    ',0);
467                         end
468                     else
469                         if i>1
470                             t_cross=find_cross(t',x_piston_m(1:end-1)
                                    ',0);
471                         end
472                     end
473                     t(i+1)=t(i)+p.t_step;
                                                            %initially
                            frequency has to float, transient frequency is
                            different than driving frequency.
474                     i=i+1;
475
476                     if length(t)>=p.n*p.n_period*1.5
477                         t_cross=ones(2*p.n_period);
478                         disp('Convergence Error, piston did not cross
                                zero twice')
479                     end
480                 end     %End of While loop checking t_cross, i loop
481
482                 %%%%%%%%%%%%%%
483                 %% Adjustments for Time
484                 %%%%%%%%%%%%%%
485                 i=i-1;                                  %step i back one for
                        sanity
486                 t=t(1:end-1);                       %step time vector back one
487                 temp(k)=max(t_cross)-max(find_cross(t',F_drive_clean',0)
                        );
488                 p.t_force_adjust=p.t_force_adjust+temp(k);
                                        %when t_force_adjust is greater
                        than 3, stop i loop
489                 t_change(k)=p.t_force_adjust;
490                 p.Period=t(end)-t(1);
                                                            %floating
                        period
491
492                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
493                 %% Estimate Massflow, Power Input,Heat Transfer, and
                        Efficiencies
494                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
495                 %Mass flows calculated using numerical integration
496                 m_dot(k)=(trapz(dm_out_valve)*(t(2)-t(1)))/p.Period;
497                 m_dot_leak_out(k)=(trapz(dm_leak_out)*(t(2)-t(1)))/p.
                        Period;
498                 m_dot_leak_in(k)=(trapz(dm_leak_in)*(t(2)-t(1)))/p.
                        Period;
499                 m_dot_in(k) = (trapz(dm_in)*(t(2) - t(1)))/p.Period;
500
501                 %W_dot calculated assuming max(h) is similar to h_avg
502                 %(proper calculation of h_avg done after the loop is
                        over,
503                 %this value is just for diagnostic purposes)
504                 W_dot(k)=m_dot(k)*(max(h)-p.h_in);
505
506                 %Efficiency estimations, diagnostic values
507                 eta_o(k)=(m_dot(k)*(p.h_2_s-p.h_in)*1000)/(p.P_electric
                        *2);       %W_dot is in kW
508                 eta_vol(k)=(m_dot(k))/(p.rho_i*f_list(l)*(max(x_piston_m
                        )-min(x_piston_m))*p.Ap);
509
510                 %Power consumed by friction
511                 W_dot_friction = p.f_friction*F_wall*2*p.x_stroke*f_list
                        (l);
512                 W_dot_friction_ave=mean(W_dot_friction)/1000;
513
514                 %Heat Transfer estimations, numerically integrated.
515                 Q_dot(k)=((trapz(Q)*(t(2)-t(1)))/p.Period);
516                 Q_dot_cv2(k)=(trapz(Q_cv2)*(t(2)-t(1)))/p.Period;
517
518                 %equation is under-relaxed , alpha = 1/100;
519                 T_w(k+1)=(-Q_dot(k)-Q_dot_cv2(k)+W_dot_friction_ave)
                        *1000*(p.R_shell)+p.T_amb;
520                 T_w(k+1)=315;
521
522                 Q_dot_in=(-Q_dot(k)-Q_dot_cv2(k)+W_dot_friction_ave)
                        *1000;
523                 T_w_test(k)=Q_dot_in*p.R_shell+p.T_amb;
524
525                 %Diagnostic values
526                 m_change(k) = m(end) - m(1);
527                 m_change_cv2(k) = m_cv2(end) - m_cv2(1);
528                 dE(k) = -Q_dot(k) + m_dot(k)*max(h) - (2*p.P_electric
                        /1000) - m_dot_in(k)*p.h_in;
529                 resonant_freq(k) = sqrt(p.k_eff/p.M_mov)/2/pi;
530
531                 %% Iteration Counter
532                 count1=num2str(k);
533                 count2='Iteration Number ';
534                 count=[count2 count1];
535                 disp(' ')
536                 disp(count)
537                 loop(k)=k;
```

```
538
539                 %%%%%%%%%%%%%
540                 %% Residuals
541                 %%%%%%%%%%%%%
542                 % Error calculations
543                 %%%%%%%%%%%%%%%%
544                 if k>1
545                     res_T(k) = 1-abs(mean(T(1:length(t)-1)./T_error(1:
                            length(t)-1)));
546                     res_rho(k) = 1-abs(mean(rho(1:length(t)-1)./
                            rho_error(1:length(t)-1)));
547                     res_T_cv2(k) = 1-abs(mean(T_cv2(1:length(t)-1)./
                            T_error_cv2(1:length(t)-1)));
548                     res_rho_cv2(k) = 1-abs(mean(rho_cv2(1:length(t)-1)./
                            rho_error_cv2(1:length(t)-1)));
549                     res_x(k) = 1-abs(x_stroke_save(k)/x_stroke_error);
550                     error_res = [abs(res_T(k)), abs(res_rho(k)),abs(
                            res_x(k))];
551                     error_res = max(error_res);
552
553                     %adjust error counter...looking for two below, then
                            end
554                     if error_res < error
555                         error_counter = error_counter + 1;
556                     end
557
558                     CV1_res=strcat('CV1 Res.:| ',' DT| ',num2str(res_T(k
                            )),' Drho| ',num2str(res_rho(k)),' dx| ',num2str
                            (res_x(k)));
559                     CV2_res=strcat('CV2 Res.:| ',' DT| ',num2str(
                            res_T_cv2(k)),' Drho| ',num2str(res_rho_cv2(k)))
                            ;
560                     add_info=strcat('Line: ',num2str(z),' | x_stroke: ',
                            num2str(p.x_stroke),'| a: ',num2str(a),'|
                            P_electric: ',num2str(p.P_electric));
561                     disp(CV1_res)
562                     disp(CV2_res)
563                     disp(add_info)
564                 end
565
566                 %Error Loop Counter
567                 if k>=p.loop_error
568                     error_res = 0;
569                     error_counter = 2;
570                     count='Convergence Error, too many iterations';
571                     disp(count)
572                 end
573
574                 if k>p.loop_error+5
575                 else
576                     X_n=[T(i),rho(i),T_cv2(i),rho_cv2(i),x_stroke_save(k
                            )];
```

```
577                     F_n=[dT(i),drho(i),dT_cv2(i),drho_cv2(i),
                            x_dot_piston(i)];
578                     J=0;
579                     DELTAX_n=0;
580                     X_n_1=X_n+DELTAX_n;
581                 end
582
583                 iter(k)=k;
584                 k=k+1;
585                 p.dP_max=max(P)-p.P_i;        %Max change in pressure
                        over cycle relative to cylinder
586
587                 if k == 150
588                     error = error + 0.00005;
589                 elseif k == 200
590                     error = error + 0.00005;
591                 end
592
593                 %% Adjustments to power to try and maintain a correct
                        stroke
594                 %If stroke is outside of the range of 0.001 and 0.0254 m
                         either
595                 %add or substract 1W of power input to try and get it
                        into the
596                 %appropriate range
597                 if k > 6
598                     if p.x_stroke < 0.0001
599                         p.P_electric = p.P_electric +1;
600                         disp('Increase P_electric =')
601                         disp(p.P_electric)
602                     end
603                 end
604             end                                 %End of k while loop
605
606             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
607             %% Brents Method: Attempting to find the correct power input
                     to
608             %% obtain full stroke
609             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
610             [p,error_stroke,a] = brents(p,a,error_brents);
611             if p.brent == 1
612                 P_electric_save(l) = p.P_brent(a);
613                 p.P_electric = p.P_brent(a);
614                 if a == 35
615                     error_brents = 0.0000005; %0.5 micron
616                 elseif a == 40
617                     error_brents = 0.000001; % 1 micron
618                 elseif a == 50
619                     error_brents = 0.000005; % 5 microns
620                 elseif a==59
621                     error_brents = 0.00001; %10 microns
622                    %Find location of minimum error, re-run using that
```

```
623                     er=abs(p.x_stroke_brent-p.x_stroke_ref);
624                     er_min=min(er);
625                     ind_er=find(er==er_min);
626                     p.P_electric=p.P_brent(ind_er);
627                     P_electric_save(l)=p.P_brent(ind_er);
628                 elseif a==60
629                     error_brents = 1;
630                     %overwrite what brent's method calculates
631                     p.P_electric = p.P_brent(a-1);
632                     P_electric_save(l)=p.P_brent(a-1);
633                 elseif a>100
634                     error_brents= 1;
635                 end
636             end
637         end     %end a loop
638         %%%%%%%%%%%%%%
639         %% Post-Allocation, adjust for proper length
640         %%%%%%%%%%%%%%
641         P=P(1:length(t));
642         dT=dT(1:length(t));
643         h=h(1:length(t));
644         V=V(1:length(t));
645         dV=dV(1:length(t));
646         dm=dm(1:length(t));
647         dm_in=dm_in(1:length(t));
648         dm_out=dm_out(1:length(t));
649         drho=drho(1:length(t));
650         m=m(1:length(t));
651         rho=rho(1:length(t));
652         x_piston=x_piston(1:length(t));
653         x_dot_piston=x_dot_piston(1:length(t));
654         Cv=Cv(1:length(t));
655         dP_dT_v=dP_dT_v(1:length(t));
656         Ma=Ma(1:length(t));
657         T=T(1:length(t));
658         x_valve=x_valve(1:length(t));
659         x_dot_valve=x_dot_valve(1:length(t));
660         dm_leak_in=dm_leak_in(1:length(t));
661         dm_leak_out=dm_leak_out(1:length(t));
662         dm_out_valve=dm_out_valve(1:length(t));
663         dV_cv2=dV_cv2(1:length(t));
664         V_cv2=V_cv2(1:length(t));
665         Ma_cv2=Ma_cv2(1:length(t));
666         h_cv2=h_cv2(1:length(t));
667         Cv_cv2=Cv_cv2(1:length(t));
668         dP_dT_v_cv2=dP_dT_v_cv2(1:length(t));
669         P_cv2=P_cv2(1:length(t));
670         drho_cv2=drho_cv2(1:length(t));
671         rho_cv2=rho_cv2(1:length(t));
672         dT_cv2=dT_cv2(1:length(t));
673         T_cv2=T_cv2(1:length(t));
674         F_drive=F_drive(1:length(t));
```

```
675             Q_motor=Q_motor(1:length(t));
676             theta=theta(1:length(t));
677             theta_dot=theta_dot(1:length(t));
678             x_piston_m=x_piston_m(1:length(t));
679             dP=dP(1:length(t));
680             dx=dx(1:length(t));
681             theta_dot_dot=theta_dot_dot(1:length(t));
682             Q=Q(1:length(t));
683             Q_cv2=Q_cv2(1:length(t));
684             c_eff = c_eff(1:length(t));
685             c_friction = c_friction(1:length(t));
686             c_gas = c_gas(1:length(t));
687             x_dot_dot_piston = x_dot_dot_piston(1:length(t));
688             F_gas = F_gas(1:length(t));
689             F_wall=F_wall(1:length(t));
690             T_w = T_w(1:k-1);
691             x_stroke_save = x_stroke_save(1:k-1);
692             loop = loop(1:k-1);
693             m_dot = m_dot(1:k-1);
694             m_dot_leak_in = m_dot_leak_in(1:k-1);
695             m_dot_leak_out = m_dot_leak_out(1:k-1);
696             W_dot = W_dot(1:k-1);
697             eta_o = eta_o(1:k-1);
698             eta_vol = eta_vol(1:k-1);
699             Q_dot = Q_dot(1:k-1);
700             Q_dot_cv2 = Q_dot_cv2(1:k-1);
701             m_dot_in = m_dot_in(1:k-1);
702
703             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
704             %% Estimate Massflow, Power Input,Heat Transfer, and
                    Efficiencies
705             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
706             % Estimating exit temperature and enthalpy
707             if m_dot(k-1) > 0
708                 %finding key volumes
709                 V_1=max(V);                             %BDC
710                 V_3=min(V);                             %TDC
711                 V_4=min(find_cross(V',P',p.P_i));   %Suction valve open
712                 V_2=max(find_cross(V',P',p.P_d));   %discharge valve
                        open
713
714                 %Find indexes of important volumes
715                 i_v1=find(V == V_1);
716                 i_v3=find(V == V_3);
717                 i_v2_test = crossing(V,[],V_2);  %a vector of length 2
718                 i_v4_test = crossing(V,[],V_4);  %a vector of length 2
719
720                 % test above to see which values correspond to the
                        correct
721                 % pressure
722                 for i=1:1:length(i_v2_test)
723                     diff_P2(i) = abs(P(i_v2_test(i))-p.P_d);
```

```
724                     diff_P4(i) = abs(P(i_v4_test(i))-p.P_i);
725                 end
726
727                 %indicies of the point where the difference between the
728                 %crossover pressure and actual pressure is at a minimum
729                 i_v2 = i_v2_test(find(diff_P2 == min(diff_P2)));
730                 i_v4 = i_v4_test(find(diff_P4 == min(diff_P4)));
731
732                 if i_v2 < i_v3  %does discharge cross the boundary?
733                     T_d = mean(T(i_v2:i_v3));
734                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
735                 else %yes
736                     T_d = mean(cat(2,T(i_v2:end),T(1:i_v3)));
737                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
738                 end
739
740                 %estimate exit temperature and enthalpy
741                 if T_d == 0
742                     T_d = mean(cat(2,find_cross(T',P',p.P_d),max(T)));
743                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
744                 end
745
746                 %Boundary work calculations
747                 W_2_3 = mean(P(i_v2:i_v3))*(V_2 - V_3);
748                 W_4_1=mean(cat(2,P(i_v4:end),P(1:i_v1)))*(V_1-V_4);
749                 W_1_2 = mean(mean(P(i_v1:end)))*(V_1 - V_2);
750                 W_3_4 = mean(P(i_v3:i_v4))*(V_4 - V_3);
751
752                 %Actual boundary work of real compression process
753                 W_boundary = W_1_2 + W_2_3 - W_3_4 - W_4_1;
754                 W_dot_boundary = W_boundary*f_list(l);
755
756                 %Idealized suction and discharge processes
757                 W_gas_dis_ideal = p.P_d*(V_2 - V_3);
758                 W_gas_suc_ideal = p.P_i*(V_1 - V_4);
759
760                 %Hybrid idealized boundary work (compression and
                        expansion
761                 %are real)
762                 W_gas_net_ideal = W_gas_dis_ideal + W_1_2 - W_3_4 -
                        W_gas_suc_ideal;
763                 W_dot_net_ideal = W_gas_net_ideal*f_list(l);
764
765                 %Valve Losses, based on differences between ideal valves
                        and
766                 %real valves, in W
767                 W_dot_suc_loss = 1000*(-W_4_1 + W_gas_suc_ideal)*f_list(
                        l);
768                 W_dot_dis_loss = 1000*(W_2_3 - W_gas_dis_ideal)*f_list(l
                        );
769
770             else
```

```
771                 T_d = 0;
772                 h_2 = 0;
773                 h_2_calc = 0;
774                 W_dot_suc_loss = 0;
775                 W_dot_dis_loss = 0;
776             end
777
778             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
779             %% Losses and power calculations
780             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
781             %Power in the piston shaft, energy integrated over one
                    period
782             W_dot_shaft = 0.5*p.M_mov*x_dot_piston.^2*f_list(l);
783             W_dot_shaft_ave = sum(W_dot_shaft)/length(W_dot_shaft);
784
785             %Power in the rotation of piston shaft, energy integrated
                    over
786             %one period
787             W_dot_shaft_rotation = 0.5*p.J*theta_dot.^2*f_list(l);
788             W_dot_shaft_rotation_ave = sum(W_dot_shaft_rotation)/length(
                    W_dot_shaft_rotation);
789
790             %Power consumed by friction
791             %W_dot_friction = (c_eff - c_gas)*p.x_max*f_list(l)^2;
792             %W_dot_friction_ave = sum(W_dot_friction)/length(
                    W_dot_friction);
793             W_dot_friction = p.f_friction*F_wall*2*p.x_stroke*f_list(l);
794             W_dot_friction_ave=mean(W_dot_friction);
795
796             %Total power consumed by compressor, moving from compression
797             %chamber to motor
798             W_dot_total_ave = W_dot_shaft_ave + W_dot_shaft_rotation_ave
                     + W_dot_friction_ave + p.Q_motor;
799
800             %Maximum power stored in the mechanical springs
801             W_dot_stored_max = 0.5*p.k_mech*p.x_stroke^2*f_list(l);
802
803             %Net compressor power into the gas
804             W_dot(k-1)=1000*m_dot(k-1)*(h_2-p.h_in); %W_dot is in W
805
806             %Efficiency estimations
807             eta_o(k-1)=(m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/(p.P_electric
                    *2);
808             eta_o_test = (m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/
                    W_dot_total_ave;
809             eta_is = (p.h_2_s - p.h_in)/(h_2 - p.h_in);
810
811             %Net leakage losses
812             W_dot_leakage_loss = W_dot_shaft_ave - W_dot(k-1) - Q_dot(k
                    -1) - W_dot_suc_loss - W_dot_dis_loss;
813             W_dot_leak = 1000*mean(m_dot_leak_out(end)*h - m_dot_leak_in
                    (end)*h_cv2);
```

```
814
815             %Linear additions
816             W_dot_in_linear = mean(F_drive_clean.*x_dot_piston);
817             eta_o_linear = (m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/
                    W_dot_in_linear;
818
819             %% Other Estimations
820             x_mag(l)=p.x_stroke/p.x_max; %#ok<SAGROW>
821             x_stroke_resonant_save(l) = (max(x_piston_m)-min(x_piston_m)
                    ); %#ok<SAGROW>
822             f_list_resonant_save(l) = f_list(l); %#ok<SAGROW>
823             W_gas=((p.gamma*p.P_i*(max(V) - p.V_dead_valve))/(p.gamma-1)
                    )*(((p.P_d/p.P_i)^((p.gamma-1)/p.gamma))-1); %kJ
824             W_dot_gas = W_gas*f_list(l)*1000;
825
826         %% Calculating resonant frequency
827         if l>1
828             change_in_stroke = x_stroke_resonant_save(l) -
                    x_stroke_resonant_save(l-1);
829             if change_in_stroke < 0
830                 number_change_stroke = number_change_stroke +1;
831             end
832             if number_change_stroke < 3
833                 p.resonant_loop = 0;
834                 f_resonant_calculated = 0;
835                 w_d_resonant_calculated = 0;
836             elseif number_change_stroke >= 3      % is stroke decreasing
837                 p.resonant_loop = 1;
838                 if length(x_stroke_resonant_save) < 5
839                     f_resonant_calculated = 99;
840                     w_d_resonant_calculated = 99;
841                 elseif length(x_stroke_resonant_save) <= 15 && length(
                        x_stroke_resonant_save) >=5
842                     coefficients = polyfit(f_list_resonant_save,
                            x_stroke_resonant_save,2);
843                     f_resonant_calculated = -coefficients(2)/(2*
                            coefficients(1));
844                     w_d_resonant_calculated = f_resonant_calculated*2*pi
                            ;
845                 else
846                     x_stroke_resonant_save = x_stroke_resonant_save(end
                            -14:end);
847                     f_list_resonant_save = f_list_resonant_save(end-14:
                            end);
848                     coefficients = polyfit(f_list_resonant_save,
                            x_stroke_resonant_save,2);
849                     f_resonant_calculated = -coefficients(2)/(2*
                            coefficients(1));
850                     w_d_resonant_calculated = f_resonant_calculated*2*pi
                            ;
851                 end
852             end
```

```
853         else
854             f_resonant_calculated = 0;
855             w_d_resonant_calculated = 0;
856             number_change_stroke = 0;           %initialize variable
857             p.resonant_loop = 0;
858         end
859
860         %% Saving Important parameters
861         if pc_flag == 0 && p.resonant_loop == 1
862             d=date;
863             save_data={d,char(p.save_name),z,p.w_d,f_list(l),(max(
                    x_piston_m)-min(x_piston_m)),W_dot(k-1),...
864                 m_dot(k-1),m_dot_leak_out(k-1),p.P_i,p.P_d,p.T_i,length(
                        t),char(p.method),2*p.P_electric,...
865                 p.eta_motor,p.k_mech,p.f_friction,p.ecc_1,x_mag(l),p.g,
                        eta_o(k-1),eta_vol(k-1),T_d,...
866                 T_w(k-1),k-1,Q_dot(k-1),Q_dot_cv2(k-1),p.P_d/p.P_i,res_T
                        (k-1),res_rho(k-1),res_T_cv2(k-1),res_rho_cv2(k-1)
                        ,...
867                 res_x(k-1),0,0,p.M_mov,(max(x_piston_m)-min(x_piston_m))
                        ,p.P_electric,f_resonant_calculated,
                        w_d_resonant_calculated,p.k_eff,eta_is,...
868                 eta_o_test,W_dot_shaft_ave,W_dot_shaft_rotation_ave,
                        W_dot_friction_ave,W_dot_total_ave,W_dot_stored_max,
                        W_dot_suc_loss,W_dot_dis_loss,...
869                 W_dot_leakage_loss,W_dot_leak,mean(x_piston_m),p.
                        V_dead_valve,a,error_stroke,eta_o_linear,
                        W_dot_in_linear};
870
871             range_2=strcat('A',num2str(l+rows),':','BG',num2str(l+rows))
                    ;
872             xlswrite1('data_log.xls',save_data,range_2)
873             save_name = strcat('wksp_',p.save_name);
874             save_name = char(save_name);
875             save(save_name)
876
877         elseif pc_flag == 1
878
879             save_data_tmp = {z,p.w_d,f_list(l),(max(x_piston_m)-min(
                    x_piston_m)),W_dot(k-1),...
880                 m_dot(k-1),m_dot_leak_out(k-1),p.P_i,p.P_d,p.T_i,length(
                        t),2*p.P_electric,...
881                 p.eta_motor,p.k_mech,p.f_friction,p.ecc_1,x_mag(l),p.g,
                        eta_o(k-1),eta_vol(k-1),T_d,...
882                 T_w(k-1),k-1,Q_dot(k-1),Q_dot_cv2(k-1),p.P_d/p.P_i,res_T
                        (k-1),res_rho(k-1),res_T_cv2(k-1),res_rho_cv2(k-1)
                        ,...
883                 res_x(k-1),0,0,p.M_mov,(max(x_piston_m)-min(x_piston_m))
                        ,p.P_electric,f_resonant_calculated,
                        w_d_resonant_calculated,p.k_eff,eta_is,...
```

```
884                 eta_o_test,W_dot_shaft_ave,W_dot_shaft_rotation_ave,
                        W_dot_friction_ave,W_dot_total_ave,W_dot_stored_max,
                        W_dot_suc_loss,W_dot_dis_loss,...
885                 W_dot_leakage_loss,W_dot_leak,mean(x_piston_m),p.
                        V_dead_valve,a,error_stroke,eta_o_linear,
                        W_dot_in_linear};
886
887             save_data(z,:) = cell2mat(save_data_tmp);
888             save_name = strcat('data_log_',p.save_name);
889             wksp_name = strcat('wksp_',p.save_name);
890             save_name = char(save_name);
891             xlswrite(save_name,save_data);
892             save(char(wksp_name))
893         end
894         k=1;
895         dT_loop=1;
896         drho_loop=1;
897         dT_cv2_loop=1;
898         drho_cv2_loop=1;
899         l=l+1;              %Resonant loop counter update
900
901     end     %End of freq sweep while loop, l variable
902
903         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
904         %% Diagrams
905         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
906         if pc_flag == 0
907             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
908             % P-V Plot
909             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
910             figure(21)
911             %set(gcf,'Visible','off')
912             handles.plot(21)=plot(V,P); title('Pressure v. Volume');
913             ylabel('Pressure (kPa)');
914             xlabel('Volume (m^3)');
915             handles.figure(21)=gcf;
916             handles.axis(21)=gca;
917
918             %%%%%%%%%%%%%%%%%%%%%%%%%%%%
919             % Diagnostic Plots
920             %%%%%%%%%%%%%%%%%%%%%%%%%%%%
921             figure(22)
922             set(gcf,'Visible','off')
923             handles.plot(22)=plot(Q_dot); title('Heat Transfer v.
                    Iteration');
924             ylabel('Heat Transfer (kW)');
925             xlabel('Iteration');
926             handles.figure(22)=gcf;
927             handles.axis(22)=gca;
928
929             figure(23)
930             set(gcf,'Visible','off')
```

```
931             handles.plot(23)=plot(Q_dot_cv2); title('Heat Transfer v.
                    Iteration - CV2');
932             ylabel('Heat Transfer (kW)');
933             xlabel('Iteration');
934             handles.figure(23)=gcf;
935             handles.axis(23)=gca;
936
937             figure(24)
938             set(gcf,'Visible','off')
939             handles.plot(24)=plot(T_w); title('Shell Wall Temp v.
                    Iteration - CV2');
940             ylabel('Wall Temp (K)');
941             xlabel('Iteration');
942             handles.figure(24)=gcf;
943             handles.axis(24)=gca;
944
945            %Main subplot diagram
946             figure(25)
947            %set(gcf,'Visible','off')
948             subplot(3,3,1),plot(t,T);title('Temperature v. Time');
949             subplot(3,3,2),plot(t,P);title('Pressure v. Time');
950             subplot(3,3,3),plot(t,dm);title('Massflow v. Time');
951             subplot(3,3,4),plot(t,V);title('Volume v. Time');
952             subplot(3,3,5),plot(t,V);title('Volume v. Time');
953             subplot(3,3,6),plot(t,rho);title('Density v. Time');
954             subplot(3,3,7),plot(t,dV);title('Change in Volume v. Time');
955             subplot(3,3,8),plot(t,x_piston);title('Displacement v. Time'
                    );
956             subplot(3,3,9),plot(t,drho);title('Change in Rho v. Time');
957
958             figure(26)
959            %set(gcf,'Visible','off')
960            %plot(t,dT_check);hold on;plot(t,dT,'r');legend('
                    dT_c_h_e_c_k','dm')
961             subplot(2,2,1),plot(t,Ma);title('Mach Number v. Time')
962             subplot(2,2,2),plot(t,x_valve);title('Valve Lift v. Time')
963             subplot(2,2,3),plot(m_dot_leak_out);title('Leakage Outflow v
                    . Time')
964             subplot(2,2,4),plot(m_dot);title('Massflow v. Iteration')
965
966            % CV2 Plots
967             figure(27)
968            %set(gcf,'Visible','off')
969             subplot(3,3,1),plot(t,T_cv2);title('Temperature v. Time for
                    CV2');
970             subplot(3,3,2),plot(t,P_cv2);title('Pressure v. Time for CV2
                    ');
971             subplot(3,3,3),plot(t,dm_leak_out);title('Leakage Massflow
                    In v. Time for CV2');
972             subplot(3,3,4),plot(t,V_cv2);title('Volume v. Time for CV2')
                    ;
973             subplot(3,3,5),plot(t,rho_cv2);title('Density v. Time');
```

```
974             subplot(3,3,6),plot(t,h_cv2);title('Enthalpy v. Time');
975             subplot(3,3,7),plot(t,dT_cv2);title('Change in Temperature v
                    . Time for CV2');
976             subplot(3,3,8),plot(t,Ma_cv2);title('Mach Number v. Time');
977             subplot(3,3,9),plot(t,drho_cv2);title('Change in Rho v. Time
                     for CV2');
978
979                 %% Shutting down activex server to excel
980                 invoke(Excel.ActiveWorkbook,'Save');
981                 Excel.Quit
982                 Excel.delete
983                 clear Excel
984
985         end
986
987         %% other stuff
988         disp(strcat('Batch Line No: ',num2str(z)))
989
990     end         %end of z loop (number of batch lines)
991
992     %turn annoying warning back on
993     warning('on','MATLAB:Print:SavingToDifferentName')
994     disp('finished')
995     diary
996
997 catch ME
998     disp('finished w/ error')
999     disp(ME.message)
1000    disp(ME.cause)
1001    disp(ME.stack)
1002    disp(ME)
1003    diary
1004
1005    if pc_flag == 0
1006        %% Shutting down activex server to excel
1007        invoke(Excel.ActiveWorkbook,'Save');
1008        Excel.Quit
1009        Excel.delete
1010        clear Excel
1011        crash_name = strcat('crash_',save_name);
1012        save(crash_name)
1013    end
1014
1015    if pc_flag == 1
1016        save_name = strcat('data_log_',p.save_name);
1017        save_name = char(save_name);
1018        crash_name = strcat('crash_',save_name);
1019        save(crash_name)
1020        if l>1
1021            xlswrite(save_name,save_data);
1022        end
1023    end
```

```
1024 %turn annoying warning back on
1025 warning('on','MATLAB:Print:SavingToDifferentName')
1026
1027 end
```

C.2 Function *getflag*

```
1 function [pc_flag]=getflag()
2 %Get the pc_flag
3
4 pc_flag = 0; %This file is for the pc
```

C.3 Function *create_struct*

```
1  function [p]=create_struct(num_inputs,txt_inputs)
2  %This function creates a structure of input parameters that will be used
3  %throughout the program. The title of this structure will be 'p' for
4  %parameter.
5
6  [T_i,rho_i,P_i,R,P_d,f,x_o,t_step,gamma,x_d,method,n,V_cv2_i,
       L_piston_max,g,D_piston,x_stroke,V_max,w_d,V_dead_valve,eta_motor,Ap
       ,P_electric,x_max,leakage_on,mass_on,vibration_order,vibration_on,
       valve_dynamics_on,D_cv2,L_cv2,heat_transfer_on,h_2_s,brent,T_s_2] =
       givens(num_inputs,txt_inputs);
7  [d_valve_suction,d_valve_discharge,C_d,A_suction_valve,A_discharge_valve
       ,I_discharge,I_suction,k_discharge,k_suction,m_eff_discharge,
       m_eff_suction,A_suction,A_discharge,x_tr_suction,x_tr_discharge,
       x_stop,a_suction,a_discharge,d_suction,d_discharge] = valve_inputs(
       num_inputs,txt_inputs);
8  [ M_mov,J,L_1,L_2,k_mech,f_friction,x_piston_i,x_dot_piston_i,theta_i,
       theta_dot_i,x_piston_m_i,J_a,ecc_1,ecc_2,L_load_1,L_load_2 ] =
       vibration_givens(x_o,vibration_order,num_inputs,txt_inputs);
9  [alpha,k_r,A,B,C,T_w_i,R_shell,T_amb,t_shell,k_alum] =
       heat_transfer_givens(T_i,num_inputs,txt_inputs);
10
11 %%%%%%
12 %From General Givens
13 %%%%%%
14
15 p.T_i=T_i;
16 p.rho_i=rho_i;
17 p.P_i=P_i;
18 p.R=R;
19 p.P_d=P_d;
20 p.f=f;
21 p.x_o=x_o;
22 p.t_step=t_step;
23 p.gamma=gamma;
24 p.x_d=x_d;
25 p.method=method;
26 p.n=n;
27 p.V_cv2_i=V_cv2_i;
28 p.L_piston_max=L_piston_max;
29 p.g=g;
30 p.D_piston=D_piston;
31 p.x_stroke=x_stroke;
32 p.x_stroke_ref=x_stroke;
33 p.V_max=V_max;
34 p.w_d=w_d;
35 p.V_dead_valve=V_dead_valve;
36 p.eta_motor=eta_motor;
37 p.Ap=Ap;
38 p.P_electric=P_electric;
39 p.x_max=x_max;
```

```
40 p.leakage_on=leakage_on;
41 p.mass_on=mass_on;
42 p.vibration_order=vibration_order;
43 p.vibration_on=vibration_on;
44 p.valve_dynamics_on=valve_dynamics_on;
45 p.heat_transfer_on=heat_transfer_on;
46 p.D_cv2=D_cv2;
47 p.L_cv2=L_cv2;
48 p.h_2_s=h_2_s;
49 p.brent = brent;
50 p.T_s_2 = T_s_2;
51
52
53 %%%%%%%%
54 %From valve givens
55 %%%%%%%%
56
57 p.d_valve_suction=d_valve_suction;
58 p.d_valve_discharge=d_valve_discharge;
59 p.C_d=C_d;
60 p.A_suction_valve=A_suction_valve;
61 p.A_discharge_valve=A_discharge_valve;
62 p.l_discharge=l_discharge;
63 p.l_suction=l_suction;
64 p.k_discharge=k_discharge;
65 p.k_suction=k_suction;
66 p.m_eff_discharge=m_eff_discharge;
67 p.m_eff_suction=m_eff_suction;
68 p.A_suction=A_suction;
69 p.A_discharge=A_discharge;
70 p.x_tr_suction=x_tr_suction;
71 p.x_tr_discharge=x_tr_discharge;
72 p.x_stop=x_stop;
73 p.a_suction=a_suction;
74 p.a_discharge=a_discharge;
75 p.d_suction=d_suction;
76 p.d_discharge=d_discharge;
77
78
79 %%%%%%%
80 %From Vibration Givens
81 %%%%%%%
82
83 p.M_mov=M_mov;
84 p.J=J;
85 p.L_1=L_1;
86 p.L_2=L_2;
87 p.k_mech=k_mech;
88 p.f_friction=f_friction;
89 p.x_piston_i=x_piston_i;
90 p.x_dot_piston_i=x_dot_piston_i;
91 p.theta_i=theta_i;
```

```
92  p.theta_dot_i=theta_dot_i;
93  p.x_piston_m_i=x_piston_m_i;
94  p.J_a=J_a;
95  p.ecc_1=ecc_1;
96  p.ecc_2=ecc_2;
97  p.L_load_1=L_load_1;
98  p.L_load_2=L_load_2;
99
100 %%%%%%%%
101 %Heat Transfer Inputs
102 %%%%%%%%
103
104 p.alpha=alpha;
105 p.k_r=k_r;
106 p.A=A;
107 p.B=B;
108 p.C=C;
109 p.T_w_i=T_w_i;
110 p.T_w_cv2_i=T_w_i;
111 p.R_shell=R_shell;
112 p.T_amb=T_amb;
113 p.t_shell=t_shell;
114 p.k_alum=k_alum;
115
116
117 %%%%%%
118 % Miscellaneous inputs
119 %%%%%%
120 p.n_period=num_inputs(54);
121 p.loop_error=num_inputs(55);
122 p.save_name=txt_inputs(56);
123 p.save_all=num_inputs(57);
124
125 %%%%%%%%%%
126 % Frequency Sweep Parameters
127 %%%%%%%%%%
128
129 p.f_begin=num_inputs(58);
130 p.f_increment=num_inputs(59);
131 p.f_end=num_inputs(60);
```

C.4 Function *givens*

```
1  function [T_i,rho_i,P_i,R,P_d,f,x_o,t_step,gamma,x_d,method,n,V_cv2_i,
       L_piston_max,g,D_piston,x_stroke,V_max,w_d,V_dead_valve,eta_motor,Ap
       ,P_electric,x_max,leakage_on,mass_on,vibration_order,vibration_on,
       valve_dynamics_on,D_cv2,L_cv2,heat_transfer_on,h_2_s,brent,T_s_2] =
       givens(num_inputs,txt_inputs)
2  %A function that intializes all given parameters
3
4  method=txt_inputs(8);
5
6  %Turn model options on and off
7  leakage_on=num_inputs(19);                %Determines if leakage model is
       on or not. 1 it is open, 0 is no leakage
8  mass_on=num_inputs(20);                   %Determines if valves open or
       not, 0 valves stay closed, 1 they open normally
9  vibration_order=num_inputs(21);           %Order of Vibration model, 1=1
       st order, 2=2nd order, all else is first order
10 vibration_on=num_inputs(22);              %Determines if vibration model
       is used or not (0 is off, 1 is on)  If not the piston is assumed to
       move in a sine wave.
11 valve_dynamics_on=num_inputs(23);         %Determines if valve dynamics
       are on or off (1 is on, 0 is a digital valve)
12 heat_transfer_on=1;                       %Heat transfer in cylinder on
       (1) or off (2)
13 brent = 1;                                %Turn Brent's method on or off
14
15 %Inlet Properties
16 T_i=num_inputs(1);
17 T_i=T_i+273.15;                  %K
18 P_i=num_inputs(2);
19 rho_i=EOS(T_i,P_i,'rho','P');   %kg/m^3
20 R=num_inputs(3);
21
22 %Discharge Pressure
23 P_d=num_inputs(4);
24
25 %Input Parameters
26 f=num_inputs(5);
27 n=num_inputs(9);
28 w_d=f*2*pi;                  %Damped natural frequency, assumed to be the
       driving frequency to drive resonance.
29 t_step=1/(f*n);       %time step in seconds
30 gamma=num_inputs(6);
31
32 %Piston Parameters
33 D_piston=num_inputs(13);
34 x_d=num_inputs(7);
35 x_max=num_inputs(18);
36 x_stroke=num_inputs(14);
37 Ap=(pi*D_piston^2)/4;                    %Piston Area in m^2
38 V_max=Ap*x_max;                  %Maximum Cylinder volume, m^3
```

```
39 x_o=(x_max)/2;                    %Piston Starting Position
40 V_dead_valve=num_inputs(15);
41
42 %Control Volume 2 Givens
43 V_cv2_i=num_inputs(10);
44 D_cv2=2*(0.0254);   %Internal Diameter of CV2
45 L_cv2=10*(0.0254);  %Length of CV2
46 L_piston_max=num_inputs(11);
47 g=num_inputs(12);
48
49 %Motor Givens
50 eta_motor=num_inputs(16);
51 P_electric=num_inputs(17);
52
53 %Isentropic Relations, using EOS
54 s_1=EOS(T_i,rho_i,'entropy','rho')
55
56 %Secant Method to Find Exit Temp for Isentropic Compression
57 dT = 1;     %Initalize change in Temperature
58 T(1) = 320; %Guess Values
59 T(2) = 325;
60 i=2;
61
62 %Secant Loop
63 while abs(dT)>1e-6
64     T(i+1)=T(i)-(EOS(T(i),P_d,'entropy','P')-s_1)*(T(i)-T(i-1))/(EOS(T(i
       ),P_d,'entropy','P')-EOS(T(i-1),P_d,'entropy','P'))
65     dT=T(i+1)-T(i);
66     i=i+1;
67     if i>500
68         error('stuck in interations')
69     end
70 end
71
72 %Last Temperature in Iteration is Temperature at Isentropic Input
73 T_s_2 = T(i);
74 %Using Temperature and Discharge Pressure, enthalpy is Calcualted
75 h_2_s = EOS(T_s_2,P_d,'enthalpy','P');
```

C.5 Function *valve_inputs*

```
function [d_valve_suction,d_valve_discharge,C_d,A_suction_valve,
    A_discharge_valve,I_discharge,I_suction,k_discharge,k_suction,
    m_eff_discharge,m_eff_suction,A_suction,A_discharge,x_tr_suction,
    x_tr_discharge,x_stop,a_suction,a_discharge,d_suction,d_discharge] =
     valve_inputs(num_inputs,txt_inputs)
%inputs to valve model

%Valve Flapper Geometry Information
d_valve_suction= num_inputs(24); %Suction valve flapper diameter in
    meters
d_valve_discharge=num_inputs(25); %Discharge valve flapper diameter in
    meters
E=num_inputs(26); % (Pa) Young's Modulus of Spring Steel for valves,
    from Marks Std. Handbook
h_valve=num_inputs(27);
rho_valve=num_inputs(28); % kg/m^3, density of reed valve material
C_d=num_inputs(29); %Drag coefficient for the valve

l_suction_valve=num_inputs(30); %length of suction valve, meters
l_discharge_valve=num_inputs(31); %length of discharge valve, meters

A_suction_valve=num_inputs(32); % Suction Valve area,m^2 (From Pro-E
    Model)
A_discharge_valve=num_inputs(33); %Discharge Valve area,m^2 (From Pro-E
     Model)

a_discharge=num_inputs(34); % Distance from anchor to force,meters (
    from PRO_E)
a_suction=num_inputs(35);

%Valve flapper strength of materials values
I_discharge=(d_valve_suction*h_valve^3)/12; %Moment of Intertia for
    discharge valve,m^4
I_suction=(d_valve_discharge*h_valve^3)/12; % Moment of Intertia for
    suction valve,m^4

k_discharge=(6*E*I_discharge)/(a_discharge^2*(3*l_discharge_valve-
    a_discharge));
k_suction=(6*E*I_suction)/(a_suction^2*(3*l_suction_valve-a_suction));

m_eff_discharge=(1/3)*rho_valve*A_discharge_valve*h_valve;
m_eff_suction=(1/3)*rho_valve*A_suction_valve*h_valve;

w_n_discharge=sqrt(k_discharge/m_eff_discharge);
w_n_suction=sqrt(k_suction/m_eff_suction);

%Suction Valve Port Geometry Information
d_suction=num_inputs(36); %suction oval diameter, meters
A_suction=((pi*((d_suction^2)/4))+0.12*0.0254*d_suction); %Suction Port
     Area, m^2
```

```
37  d_h_suction=(4*A_suction)/(pi*d_suction+2*(0.12*0.0254));
38
39 %Discharge Valve Port Geometry Information
40  d_discharge=num_inputs(37);  %discharge diameter in meters
41  A_discharge=(pi*((d_discharge^2)/4));  %Discharge Port area,m^2
42
43 %Transitional valve lift
44  x_tr_suction=0.25*(d_h_suction^2/d_valve_suction);  % Transitional Valve
        Lift,meters
45  x_tr_discharge=0.25*(d_discharge^2/d_valve_discharge);
46 %Discharge Valve stopper distance
47  x_stop=num_inputs(38);      %Height of discharge stopper in meters
```

C.6 Function *vibration_givens*

```
function [ M_mov,J,L_1,L_2,k_mech,f_friction,x_piston_i,x_dot_piston_i,
    theta_i,theta_dot_i,x_piston_m_i,J_a,ecc_1,ecc_2,L_load_1,L_load_2 ]
     = vibration_givens(x_o,vibration_order,num_inputs,txt_inputs)
%Inputs to vibration model

L=num_inputs(40);        %Total length of piston
L_1=num_inputs(41);              %Distance from CG to front of piston
L_2=L-L_1;               %Distance from CG to back of piston
R=num_inputs(42);        %Radius of Piston Spring Seats
M_mov=num_inputs(39);          %Piston Equivalent moving mass, in kg
M_piston=num_inputs(43);         %Mass of just the piston
J=((1/12)*0.8*M_piston*L^2)+((1/12)*0.2*M_piston*(0.98*0.0254)^2);
    %Moment of Inertia from CG (kg-m^2)
J_a=((1/3)*0.8*M_piston*L^2)+((0.25)*0.2*M_piston*R^2)+(0.2*M_piston*L_2
    ^2);    %Moment of Inertia from back of Piston (kg-m^2)

if vibration_order==2
    ecc_1=num_inputs(50);
    ecc_2=num_inputs(51);
elseif vibration_order==1
    ecc_1=0;
    ecc_2=0;
else
    ecc_1=0;
    ecc_2=0;
end

L_load_1=num_inputs(52);            %Distance between spring seats, 1
    is forward spring, 2 is toward motor.
L_load_2=num_inputs(53);
k_mech=num_inputs(44);

model_exist = exist('friction_model','file');
if model_exist == 2
    %new friction model
    f_friction=friction_model(num_inputs);
else
    f_friction=num_inputs(45);
end

%% Initial Conditions for vibration model
x_piston_i=x_o;
x_dot_piston_i=num_inputs(46);
theta_i=num_inputs(47);
theta_dot_i=num_inputs(48);
x_piston_m_i=num_inputs(49);

end
```

C.7 Function *heat_transfer_givens*

```
1
2  function [alpha,k_r,A,B,C,T_w_i,R_shell,T_amb,t_shell,k_alum] =
       heat_transfer_givens(T_i,num_inputs,txt_inputs)
3  %Inputs to the heat transfer model
4
5  alpha=0.00000055;          %Thermal diffusivity in m^2
6  k_alum=160;                %Thermal conductivity of aluminum, W/m-K
7  A=0.25;
8  B=0.65;
9  C=0.25;                    %Coefficients from correlation
10 T_w_i=22.2+273.15;                %Inital wall temperature, in K
11 T_amb=22+273.15;                 %Ambient Temperature
12 k_r=15*10^-3;               %Thermal conductivity of refrigerant, W/m-K
13 rho_air=1.2;                 %Density of air, kg/m^3
14 Vel_air=1;                   %Velocity of airflow across compressor, m/s
15 D_shell=2*0.0254;            %Outer Diameter of Piston Cylinder, m
16 L_shell=10*0.0254;           %Length of Compressor, m
17 A_shell=pi*D_shell*L_shell;       %Surface area of compressor shell, m^2
18 k_air=30e-3;                 %W/m-K, conductivity of air
19 mu_air=2*10^-5;              %Dynamic Viscosity of air, kg/m-s
20 nu_air=mu_air/rho_air;       %Kinematic Viscosity of air, m^2/s
21 alpha_air=2.2160*10^-5;      %Thermal diffusivity of air, m^2/s
22 Pr=nu_air/alpha_air;         %Prandtl Number of air flow
23 Re_air=(rho_air*Vel_air*D_shell)/mu_air;    %Reynolds number of air
24 C_2=0.683;                   %Constants from Table 7.2 in Incropera
       DeWitt
25 m=0.466;
26 Nu_D=C_2*Re_air^m*Pr^(1/3);
27 h_shell=(Nu_D*k_air)/D_shell;      %Heat transfer coefficient for
       outside heat transfer
28 R_shell=1/(h_shell*A_shell);       %Overall thermal resistance from
       shell to air
29 t_shell = 0.0254;      %Thickness of compressor shell
```

C.8 Function *vibration*

```
1  function [ dV,V,x_piston,x_dot_piston,theta,theta_dot,x_piston_m,
       theta_dot_dot,p ] = vibration( t,dP,dx,x_piston_1,x_dot_piston_1,
       theta_1,theta_dot_1,x_piston_m_1,theta_dot_dot_1,p )
2  %Vibration Model Component of Linear Compressor Model, added ver. 1.4.
3
4  if p.vibration_on==1
5
6      V=x_piston_1*p.Ap+p.V_dead_valve;           %Piston volume, when
           x_piston=0 volume is dead volume in valves.
7      dV=-x_dot_piston_1*p.Ap;        %x_piston and x_piston_m are
           opposites, x_dot_piston follows x_piston_m but I want it to
           follow x_piston, hence the negative.
8      p.theta=theta_1;
9
10     %Linearized vibration model flag
11     linear = 0;
12
13     if linear == 1
14         theta_tmp = atan(p.g/p.L_1);
15         p.F_wall=(1/p.L_1)*(p.k_mech*(p.x_stroke_ref-p.ecc_1*theta_tmp)*
               p.ecc_1);
16         %k_gas adjusted for max stroke
17         k_gas=((p.P_d-p.P_i)*p.Ap*1000)/p.x_stroke_ref;  %Gas spring
               rate, linear estimate
18         p.V_max=p.Ap*p.x_stroke_ref;
19         W_gas=((p.gamma*p.P_i*p.V_max)/(p.gamma-1))*((p.P_d/p.P_i)^((p.
               gamma-1)/p.gamma)-1);          %Work done on gas in one cycle,
                kJ
20         c_friction=(4*p.f_friction*p.F_wall)/(p.w_d*p.x_stroke_ref*pi);
                %effective damping due to dry friction in cylinder
21         c_gas=(W_gas*1000)/(p.w_d*p.x_stroke_ref^2*pi);   %effecitve
               damping due to work done on gas, 1000 to convert work to J,
               (N-s)/m
22     else
23         if abs(theta_1) >= atan(p.g/p.L_1)  %if piston is in contact
               with wall, there is friction.
24             p.F_wall=(1/p.L_1)*(p.k_mech*abs((abs(x_piston_m_1)-p.ecc_1*
                   abs(theta_1)))*p.ecc_1); %Newtons
25         else
26             p.F_wall=0;
27             %keyboard
28             p.F_wall=(1/p.L_1)*(p.k_mech*abs((abs(x_piston_m_1)-p.ecc_1*
                   abs(theta_1)))*p.ecc_1); %Newtons
29         end
30         p.V_max = p.Ap*p.x_stroke;
31         if p.dP_max>=(p.P_d-p.P_i)
32             k_gas=((p.P_d-p.P_i)*p.Ap*1000)/p.x_stroke;  %Gas spring
                   rate, linear estimate
33             W_gas = (-p.P_current*p.dV + 0.5*p.dV*dP)*1000;
34         elseif p.dP_max<(p.P_d-p.P_i)
```

```
35              k_gas=(p.dP_max*p.Ap*1000)/p.x_stroke;  %Gas spring rate,
                    linear estimate
36              W_gas = (-p.P_current*p.dV + 0.5*p.dV*dP)*1000;
37          end
38
39          %Testing F_gas instead of k_gas
40          k_gas = 0;
41          c_friction=(4*p.f_friction*p.F_wall)/(p.w_d*p.x_stroke*pi);
42          %%effective damping due to dry friction in cylinder
43          c_gas=(W_gas)/(p.w_d*p.x_stroke^2*pi);
44          %%effecitve damping due to work done on gas, 1000 to convert
                work to J, (N-s)/m
45      end
46
47      k_eff=k_gas+p.k_mech;   %Effective Spring Rate, gas + mechanical
            springs
48      p.k_eff=k_eff;
49      c_eff=c_gas+c_friction;  %Total effective damping, (N-s)/m
50      p.c_eff = c_eff;
51      p.c_friction = c_friction;
52      p.c_gas = c_gas;
53
54      %%%%%%%%%%%%%%%%%%
55      %% Start - dx_vibration replacement
56      %%%%%%%%%%%%%%%%%%
57
58      %Vector of Timesteps
59      t_combined = [t,t+0.5*p.t_step,t+0.5*p.t_step,t+p.t_step];
60      k_m = 4.6831;
61      F_drive_max=sqrt(p.eta_motor*p.P_electric)*(k_m);        %Maximum
            Motor force, in Newtons.
62      F_drive=F_drive_max*(cos(p.w_d*(t-p.t_force_adjust)+pi));
                        %Driving force v. Time
63
64      if linear == 0
65          F_drive = F_drive - p.dP_piston*p.Ap*1000;
66      end
67
68      %Renaming displacements and angles for Runge-Kutta Calculations
69      x_1 = x_piston_m_1;
70      x_2 = x_dot_piston_1;
71      theta_1_calc = theta_1;
72      theta_2 = theta_dot_1;
73
74      %Replacing dx_vibration%
75      %[ F_drive,Q_motor ] = motor( p.w_d,t_combined(1),p.eta_motor,p.
            P_electric,p.t_force_adjust );
76      F_drive=F_drive_max*(cos(p.w_d*(t_combined(1)-p.t_force_adjust)+pi))
            ;                   %Driving force v. Time
77
78      if linear == 0
79          F_drive = F_drive - p.dP_piston*p.Ap*1000;
```

```
80      end
81
82      num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
83      num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
84      dx=[x_2,num2,theta_2,num4];
85      k1 = p.t_step*dx;
86
87      %Step 2, update displacements and angles
88      x_1 = x_1+0.5*k1(1);
89      x_2 = x_2+0.5*k1(2);
90      theta_1_calc = theta_1_calc+0.5*k1(3);
91      theta_2 = theta_2+0.5*k1(4);
92
93      %Replacing dx_vibration%
94      %[ F_drive,Q_motor ] = motor( p.w_d,t_combined(2),p.eta_motor,p.
            P_electric,p.t_force_adjust );
95      F_drive=F_drive_max*(cos(p.w_d*(t_combined(2)-p.t_force_adjust)+pi))
            ;
96
97      if linear == 0
98          F_drive = F_drive - p.dP_piston*p.Ap*1000;
99      end
100
101     num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
102     num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
103     dx=[x_2,num2,theta_2,num4];
104     k2 = p.t_step*dx;
105
106     %Step 3, update displacements and angles
107     x_1 = x_1+0.5*k2(1);
108     x_2 = x_2+0.5*k2(2);
109     theta_1_calc = theta_1_calc+0.5*k2(3);
110     theta_2 = theta_2+0.5*k2(4);
111
112     %Replacing dx_vibration%
113     num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
114     num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
115     dx=[x_2,num2,theta_2,num4];
116     k3 = p.t_step*dx;
117
118     %Step 4, update displacements and angles
119     x_1 = x_1+k2(1);
120     x_2 = x_2+k2(2);
121     theta_1_calc = theta_1_calc+k2(3);
122     theta_2 = theta_2+k2(4);
123
124     %Replacing dx_vibration%
125     %[ F_drive,Q_motor ] = motor( p.w_d,t_combined(4),p.eta_motor,p.
            P_electric,p.t_force_adjust );
```

```
126       F_drive=F_drive_max*(cos(p.w_d*(t_combined(4)-p.t_force_adjust)+pi))
              ;
127
128       if linear == 0
129           F_drive = F_drive - p.dP_piston*p.Ap*1000;
130       end
131
132       num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
              k_gas+p.k_mech)*x_1);
133       num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
134       dx=[x_2,num2,theta_2,num4];
135       k4 = p.t_step*dx;
136
137       %%%%%%%%%%%%%%%%%%%%%%%%%
138       %% End, dx_vibration replacement
139       %%%%%%%%%%%%%%%%%%%%%%%%%
140
141       dx_total=[x_piston_m_1,x_dot_piston_1,theta_1,theta_dot_1]+(1/6)*(k1
              +2*k2+2*k3+k4);
142
143       x_piston_m=dx_total(1);
144       x_dot_piston=dx_total(2);
145       theta=dx_total(3);
146       theta_dot=dx_total(4);
147       theta_dot_dot=k1(4);
148       p.x_dot_dot_piston=k1(2);
149       x_piston=-x_piston_m+p.x_o;
150
151       if x_piston < 0   %This second condition does not allow the piston
              to travel beyond the top of the cylinder
152           x_piston = 0;
153           x_piston_m=p.x_o;
154           x_dot_piston=0;
155       end
156
157       if abs(theta) > atan(p.g/p.L_1)%Restricts the rotation to approx 5
              deg.
158           if theta < 0
159               theta=-atan(p.g/p.L_1);
160               theta_dot=0;
161           elseif theta > 0
162               theta=atan(p.g/p.L_1);
163               theta_dot=0;
164           end
165       end
166
167   elseif p.vibration_on==0
168
169       x_piston=p.x_o*(cos(p.w_d*t+pi)+1);
170       V=p.Ap*x_piston+p.V_dead_valve;
171       dV=-p.Ap*p.x_o*p.w_d*sin(p.w_d*t+pi);
172       x_dot_piston=-p.x_o*p.w_d*(sin(p.w_d*t+pi));
```

```
173         theta=0;
174         theta_dot=0;
175         theta_dot_dot=0;
176         x_piston_m=x_piston-p.x_o;
177         p.F_wall=0;
178         p.theta=0;
179         p.k_eff=0;
180         p.c_eff = 0;
181         p.c_friction = 0;
182         p.c_gas = 0;
183         p.x_dot_dot_piston=0;
184     end
185 
186     end
```

C.9 Function *valve_dynamics*

```
1  function [ x_valve,x_dot_valve,dm,Ma ] = valve_dynamics(P,rho,T,
       x_valve_1,x_dot_valve_1,p)
2  %Valve flow model
3
4  Pr_limit=0.556999;          %Sonic limit to the pressure ratio, for R
       -134a
5
6  if p.mass_on==1
7      if p.valve_dynamics_on==1
8
9          %If there is flowrate coming in...
10         if p.P_i>P
11             Pr=P/p.P_i;
12
13             if Pr > Pr_limit
14                 Ma=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
15                 if Ma>1
16                     Ma=1;
17                 end
18                 V=sqrt(p.gamma*p.R*p.T_i*1000)*Ma;   %T_i because this
                       flow comes from suction valve
19             else
20                 Ma=1;
21                 V=sqrt(p.gamma*p.R*p.T_i*1000)*Ma;   %T_i because this
                       flow comes from suction valve
22             end
23
24             x_1=x_valve_1;
25             x_2=x_dot_valve_1;
26
27             if x_valve_1>=p.x_tr_suction
28
29                 k1 = p.t_step*x_RK_flux_dom(x_1,x_2,p.rho_i,p.
                       A_suction_valve,p.k_suction,V,p.m_eff_suction,p.C_d,
                       p.A_suction);
30                 k2 = p.t_step*x_RK_flux_dom(x_1+0.5*k1(1),x_2+0.5*k1(2),
                       p.rho_i,p.A_suction_valve,p.k_suction,V,p.
                       m_eff_suction,p.C_d,p.A_suction);
31                 k3 = p.t_step*x_RK_flux_dom(x_1+0.5*k2(1),x_2+0.5*k2(2),
                       p.rho_i,p.A_suction_valve,p.k_suction,V,p.
                       m_eff_suction,p.C_d,p.A_suction);
32                 k4 = p.t_step*x_RK_flux_dom(x_1+k3(1),x_2+k3(2),p.rho_i,
                       p.A_suction_valve,p.k_suction,V,p.m_eff_suction,p.
                       C_d,p.A_suction);
33
34                 x_total=[x_1;x_2]+(1/6)*(k1+2*k2+2*k3+k4);
35
36             else
```

```
37                  k1 = p.t_step*x_RK_pressure_dom(x_1,x_2,p.rho_i,p.
                        A_suction_valve,p.k_suction,V,p.m_eff_suction,p.C_d,
                        p.P_i,P);
38                  k2 = p.t_step*x_RK_pressure_dom(x_1+0.5*k1(1),x_2+0.5*k1
                        (2),p.rho_i,p.A_suction_valve,p.k_suction,V,p.
                        m_eff_suction,p.C_d,p.P_i,P);
39                  k3 = p.t_step*x_RK_pressure_dom(x_1+0.5*k1(1),x_2+0.5*k1
                        (2),p.rho_i,p.A_suction_valve,p.k_suction,V,p.
                        m_eff_suction,p.C_d,p.P_i,P);
40                  k4 = p.t_step*x_RK_pressure_dom(x_1+k1(1),x_2+k1(2),p.
                        rho_i,p.A_suction_valve,p.k_suction,V,p.
                        m_eff_suction,p.C_d,p.P_i,P);
41
42                  x_total=[x_1;x_2]+(1/6)*(k1+2*k2+2*k3+k4);
43
44              end
45
46              if x_total(1)<0
47                  x_total(1)=0;   %This is the hard stop limit, valve
                        cannot have a negative position
48              end
49
50              if x_total(1)==0 %|| x_total(1)==1.4*x_stop
51                  x_total(2)=0;
52              end
53              x_valve=x_total(1);
54              x_dot_valve=x_total(2);
55
56              x_right=0.5*p.d_suction+p.a_suction;
57              x_left=p.a_suction-0.5*p.d_suction;
58
59              x_ave_suction=(1/(x_right-x_left))*(((3*x_valve)/p.a_suction
                    ^2)*(p.a_suction^3-x_left^3)-((3*x_valve)/(4*p.a_suction
                    ^3))*(p.a_suction^4-x_left^4)...
60                  +((9*x_valve)/(2*p.a_suction))*(x_right^2-p.a_suction^2)
                        -3*x_valve*(x_right-p.a_suction));
61
62              A_port=p.A_suction;
63
64              if pi*p.d_suction*x_ave_suction < A_port
65                  dm=-p.rho_i*V*pi*p.d_suction*x_ave_suction;
66              else
67                  dm=-p.rho_i*V*A_port;
68              end
69
70          %if there is flowrate coming out...
71          elseif P>=p.P_d
72             Pr=p.P_d/P;
73             if Pr > Pr_limit
74                 Ma=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
75                  if Ma>1
76                      Ma=1;
```

```
77                  end
78                  V=sqrt(p.gamma*p.R*T*1000)*Ma;
79              else
80                  Ma=1;
81                  V=sqrt(p.gamma*p.R*T*1000)*Ma;
82              end
83              x_1=x_valve_1;
84              x_2=x_dot_valve_1;
85
86               if x_valve_1>=p.x_tr_discharge
87
88                   k1 = p.t_step*x_RK_flux_dom(x_1,x_2,rho,p.
                         A_discharge_valve,p.k_discharge,V,p.m_eff_discharge,
                         p.C_d,p.A_discharge);
89                   k2 = p.t_step*x_RK_flux_dom(x_1+0.5*k1(1),x_2+0.5*k1(2),
                         rho,p.A_discharge_valve,p.k_discharge,V,p.
                         m_eff_discharge,p.C_d,p.A_discharge);
90                   k3 = p.t_step*x_RK_flux_dom(x_1+0.5*k2(1),x_2+0.5*k2(2),
                         rho,p.A_discharge_valve,p.k_discharge,V,p.
                         m_eff_discharge,p.C_d,p.A_discharge);
91                   k4 = p.t_step*x_RK_flux_dom(x_1+k3(1),x_2+k3(2),rho,p.
                         A_discharge_valve,p.k_discharge,V,p.m_eff_discharge,
                         p.C_d,p.A_discharge);
92
93                   x_total=[x_1;x_2]+(1/6)*(k1+2*k2+2*k3+k4);
94
95               else
96
97                   k1 = p.t_step*x_RK_pressure_dom(x_1,x_2,rho,p.
                         A_discharge_valve,p.k_discharge,V,p.m_eff_discharge,
                         p.C_d,P,p.P_d);
98                   k2 = p.t_step*x_RK_pressure_dom(x_1+0.5*k1(1),x_2+0.5*k1
                         (2),rho,p.A_discharge_valve,p.k_discharge,V,p.
                         m_eff_discharge,p.C_d,P,p.P_d);
99                   k3 = p.t_step*x_RK_pressure_dom(x_1+0.5*k1(1),x_2+0.5*k1
                         (2),rho,p.A_discharge_valve,p.k_discharge,V,p.
                         m_eff_discharge,p.C_d,P,p.P_d);
100                  k4 = p.t_step*x_RK_pressure_dom(x_1+k1(1),x_2+k1(2),rho,
                         p.A_discharge_valve,p.k_discharge,V,p.
                         m_eff_discharge,p.C_d,P,p.P_d);
101
102                  x_total=[x_1;x_2]+(1/6)*(k1+2*k2+2*k3+k4);
103
104              end
105
106              if x_total(1)<0
107                  x_total(1)=0;   %This is the hard stop limit, valve
                         cannot have a negative position
108              end
109              if x_total(1)>p.x_stop
110                  x_total(1)=p.x_stop;   %Hard stop limit for discharge
                         valve
```

```
111             end
112
113             if x_total(1)==0 || x_total(1)==p.x_stop
114                 x_total(2)=0;
115             end
116             x_valve=x_total(1);
117             x_dot_valve=x_total(2);
118
119             x_right=0.5*p.d_discharge+p.a_discharge;
120             x_left=p.a_discharge-0.5*p.d_discharge;
121
122             x_ave_discharge=(1/(x_right-x_left))*(((3*x_valve)/p.
                    a_discharge^2)*(p.a_discharge^3-x_left^3)-((3*x_valve)
                    /(4*p.a_discharge^3))*(p.a_discharge^4-x_left^4)...
123                 +((9*x_valve)/(2*p.a_discharge))*(x_right^2-p.
                        a_discharge^2)-3*x_valve*(x_right-p.a_discharge));
124
125             A_port=(pi*p.d_discharge^2)/4;
126
127             if pi*p.d_valve_discharge*x_ave_discharge < A_port
128                 dm=rho*V*pi*p.d_valve_discharge*x_ave_discharge;
129             else
130                 dm=rho*V*A_port;
131             end
132
133         else
134            dm=0;
135            Ma=0;
136            x_valve=0;
137            x_dot_valve=0;
138         end
139
140     elseif p.valve_dynamics_on==0
141
142         %If there is flowrate coming in...
143         if p.P_i>P
144             Pr=P/p.P_i;
145             Ma=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
146             V=sqrt(p.gamma*p.R*(p.T_i)*1000)*Ma;
147             dm=-p.rho_i*V*p.A_suction;  %negative means flow in!!
148             x_valve=0;
149             x_dot_valve=0;
150
151         %if there is flowrate coming out...
152         elseif P>=p.P_d
153            Pr=p.P_d/P;
154            Ma=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
155            V=sqrt(p.gamma*p.R*(T)*1000)*Ma;
156            dm=rho*V*p.A_discharge;  %positive means flow out!!
157            x_valve=0;
158            x_dot_valve=0;
159
```

```
160             else
161                 dm=0;
162                 Ma=0;
163                 x_valve=0;
164                 x_dot_valve=0;
165             end
166         end
167     elseif p.mass_on==0
168         x_valve=0;
169         dm=0;
170         x_dot_valve=0;
171         Ma=0;
172     else
173         x_valve=0;
174         dm=0;
175         x_dot_valve=0;
176         Ma=0;
177     end
178
179     end
```

C.10 Function *x_RK_flux_dom*

```
1  function [a] = x_RK_flux_dom(x_1,x_2,rho,A_valve,k_valve,V,m_eff,C_d,
       A_port)
2  %mass flux dominant valve position derivatives
3
4  a=[x_2;(rho*A_port*(V-x_2)^2-k_valve*x_1+0.5*C_d*rho*V^2*A_valve-0.5*C_d
       *rho*x_2^2*A_valve)/m_eff];
5
6  end
```

C.11 Function *x_RK_pressure_dom*

```
1 function [a] = x_RK_pressure_dom(x_1,x_2,rho,A_valve,k_valve,V,m_eff,C_d
      ,P_high,P_low)
2 %pressure dominant valve position derivatives
3
4 a=[x_2;(-k_valve*x_1+0.5*C_d*rho*V^2*A_valve+A_valve*(P_high-P_low)
      *1000-0.5*C_d*rho*x_2^2*A_valve)/m_eff];
5
6 end
```

C.12 Function *leakage*

```
function [ dm_leak_in,dm_leak_out,Ma_cv2 ] = leakage( P,P_cv2,
    x_dot_piston,T,rho,T_cv2,x_piston,p )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

Pr_limit=0.556999;          %Sonic limit to the pressure ratio, for R
    -134a

if p.leakage_on==1

    %Couette-Posille Flow, for when Ma<0.3

    mu=5.87509602E-07+3.79308232E-08*T;  %approximation for mu between
        290 and 390K, in kg/m-s

    L_piston=p.L_piston_max-x_piston;

    dP_dx=(P_cv2-P)/L_piston;
    dP_dx = dP_dx*1000;  %convert to N/m^3

    u_bar=(-x_dot_piston/2)+(p.g^2/(4*mu))*(-dP_dx)+(p.g^3/(6*mu))*dP_dx
        ;

    A=(pi/4)*(p.D_piston+2*p.g)^2-(pi/4)*(p.D_piston^2);


    Ma_1=abs(u_bar)/sqrt(p.gamma*p.R*T);

    Ma_2=abs(u_bar)/sqrt(p.gamma*p.R*T_cv2);

    if Ma_1>Ma_2
        Ma_cv2=Ma_1;
    else
        Ma_cv2=Ma_2;
    end

    % Isentropic compressible flow when Ma>=0.3

    if Ma_cv2 >= 0.3    %Compressible flow

        Pr=min(P/P_cv2,P_cv2/P);  %grab the PR smaller than one


        if P >= P_cv2  %flow into CV2
            Ma_cv2=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
            if Ma_cv2>1
                Ma_cv2=1;
            end
            u_bar=sqrt(p.gamma*p.R*T*1000)*Ma_cv2;   %T because this
                flow comes from compression chamber
```

```
46
47
48          elseif P < P_cv2  %flow into compression chamber
49              Ma_cv2=sqrt((2/(p.gamma-1))*((Pr^((1-p.gamma)/p.gamma))-1));
50              if Ma_cv2>1
51                  Ma_cv2=1;
52              end
53              u_bar=-sqrt(p.gamma*p.R*T_cv2*1000)*Ma_cv2;    %T because
                    this flow comes from CV2
54
55          else
56              error('your logic is poor in the leakage sub-model')
57          end
58
59      end
60
61      dm_leak=rho*u_bar*A;
62
63      if dm_leak<0   %flow into compression chamber
64          dm_leak_out=0;
65          dm_leak_in=abs(dm_leak);
66      elseif dm_leak>0   %flow out of compression chamber
67          dm_leak_out=abs(dm_leak);
68          dm_leak_in=0;
69      else
70          dm_leak_in=0;
71          dm_leak_out=0;
72      end
73
74  elseif p.leakage_on==0
75
76      Ma_cv2=0;
77      dm_leak_in=0;
78      dm_leak_out=0;
79
80  else
81
82      Ma_cv2=0;
83      dm_leak_in=0;
84      dm_leak_out=0;
85  end
86
87  end
```

C.13 Function *Ins_HT*

```
function [ Q ] = Ins_HT( T,rho,T_w,V,dV,x_piston,x_dot_piston,p)
%Instanteneous heat transfer model for heat transfer between the
    cylinder
%wall and the compressor gas.  Used in all control volumes.

if p.heat_transfer_on==1

%%%%%%%
%Approximate Viscosity
%%%%%%%
mu=5.87509602E-07+3.79308232E-08*T;  %approximation for mu between 290
    and 390K, in kg/m-s

%%%%%%
%Fagotti Correlation, 1998 Purdue Compressor conference

%%%%%%
%Compressibility Number
%%%%%%
L=((p.gamma-1)/V)*dV*sqrt(p.D_piston^3/(p.alpha*abs(p.x_dot_ave)));

if isnan(L)
    L=0;
end
%%%%%%
%Reynolds Numbers for the process
%%%%%%
Re=(rho*abs(p.x_dot_ave)*p.D_piston)/mu;

%%%%%%
%Heat Transfer Area
%%%%%%
A_ht=p.D_piston*pi*x_piston;

%%%%%
%Calculate Heat Transfer
%%%%%
Q=((A_ht*p.k_r*(T-T_w))/p.D_piston)*(p.A*(Re^p.B)+p.C*L*(T_w/(T-T_w)));

if isnan(Q)
    Q=0;
end

Q=-Q/1000;
else
    Q=0;
end

end
```

C.14 Function *Ins_HT_cv2*

```
function [ Q_cv2 ] = Ins_HT_cv2( T_cv2,rho_cv2,T_w_cv2,V_cv2,dV_cv2,
    x_dot_piston,Q_motor,p)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

if p.heat_transfer_on==1 && p.leakage_on == 1
    %%%%%%%
    %Heat Transfer Area
    %%%%%%%
     A_ht_cv2=p.D_cv2*pi*p.L_cv2;
     Q_cv2=(1/1000)*(p.k_alum*A_ht_cv2*(T_w_cv2-T_cv2))/(p.t_shell);  %
         Heat Transfer in kW

else
     Q_cv2=0;
end

end
```

C.15 Function *brents*

```
function [p, error_stroke,a,below,above] = brents(p,a,error_brents)
%function help here

tol = 1e-5;

if p.brent == 1
    p.x_stroke_brent(a) = p.x_stroke;
    if a >= 2   %a greater than 1

        if p.x_stroke_brent(a)< p.x_stroke_ref
            p.below = a;
        elseif p.x_stroke_brent(a) > p.x_stroke_ref
            p.above = a;
        end

        %both previous pts are below desired stroke, brents if there
            exists
        %a point above
        if p.x_stroke_brent(a) < p.x_stroke_ref && p.x_stroke_brent(a-1)
            < p.x_stroke_ref
            if isfield(p,'above') == 1
                b_k = p.P_brent(a);
                a_k = p.P_brent(p.above);
                b_k_1 = p.P_brent(a-1);
                %bisection step
                m=(p.P_brent(a)+a_k)/2;
                %Secant step
                s=p.P_brent(a) - (p.x_stroke_brent(a) - p.x_stroke_ref)
                    *(p.P_brent(a) - p.P_brent(a-1))/(p.x_stroke_brent(a
                    ) - p.x_stroke_brent(a-1));
                if s < b_k && s > m
                    p.P_brent(a+1)=s;
                    p.method_n(a)=1;
                else
                    p.P_brent(a+1)=m;
                    p.method_n(a)=2;
                end
                if a>=3
                    if abs(p.P_brent(a)-p.P_brent(a-1)) < tol
                        if p.method_n(a-1) == 1  %used secant method
                            if abs(p.P_brent(a-1)-p.P_brent(a-2)) < tol
                                p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
                                p.method_n(a)=2;
                            end
                        else
                            p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
                            p.method_n(a)=2;
                        end
                    elseif abs(s-b_k) > 0.5*abs(b_k-b_k_1)
                        if p.method_n(a-1) == 1  %used secant method
```

```
47                              if abs(s-b_k) > 0.5*abs(b_k_1-p.P_brent(a-2)
                                    )
48                                  p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
49                                  p.method_n(a)=2;
50                              end
51                          else
52                              p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
53                              p.method_n(a)=2;
54                          end
55                      end
56                  end
57              else
58                  %Secant Step
59                  p.P_brent(a+1) = p.P_brent(a) - (p.x_stroke_brent(a) - p
                        .x_stroke_ref)*(p.P_brent(a) - p.P_brent(a-1))/(p.
                        x_stroke_brent(a) - p.x_stroke_brent(a-1));
60                  p.method_n(a)=1;  %1 is secant, 2 is bisection
61              end
62          %One below, one above (straddled), employ brents method
63          elseif (p.x_stroke_brent(a) < p.x_stroke_ref && p.x_stroke_brent
                (a-1) > p.x_stroke_ref) || (p.x_stroke_brent(a) > p.
                x_stroke_ref && p.x_stroke_brent(a-1) < p.x_stroke_ref)
64              %If there is only one point below, then...
65              if a==2
66                  %Bi-section Step
67                  p.P_brent(a+1) = (p.P_brent(a)+p.P_brent(a-1))/2;
68                  p.method_n(a)=2;
69              else   %Brents method start
70                  %if the current iterate is above the reference
71                  if p.x_stroke_brent(a) >= p.x_stroke_ref
72                      b_k = p.P_brent(a);
73                      a_k = p.P_brent(p.below);
74                      b_k_1 = p.P_brent(a-1);
75                      %bisection step
76                      m=(p.P_brent(a)+a_k)/2;
77                      %Secant step
78                      s=p.P_brent(a) - (p.x_stroke_brent(a) - p.
                            x_stroke_ref)*(p.P_brent(a) - p.P_brent(a-1))/(p
                            .x_stroke_brent(a) - p.x_stroke_brent(a-1));
79                      if s < b_k && s > m
80                          p.P_brent(a+1)=s;
81                          p.method_n(a)=1;
82                      else
83                          p.P_brent(a+1)=m;
84                          p.method_n(a)=2;
85                      end
86                  %if the current iterate is below the reference
87                  else
88                      b_k = p.P_brent(a);
89                      a_k = p.P_brent(p.above);
90                      b_k_1 = p.P_brent(a-1);
91                      %bisection step
```

```
92                      m=(p.P_brent(a)+a_k)/2;
93                      %Secant step
94                      s=p.P_brent(a) - (p.x_stroke_brent(a) - p.
                            x_stroke_ref)*(p.P_brent(a) - p.P_brent(a-1))/(p
                            .x_stroke_brent(a) - p.x_stroke_brent(a-1));
95                      if s < b_k && s > m
96                          p.P_brent(a+1)=s;
97                          p.method_n(a)=1;
98                      else
99                          p.P_brent(a+1)=m;
100                         p.method_n(a)=2;
101                     end
102                 end
103
104                 %Brent's addition to the algorithmn
105                 if a>=3
106                     if abs(p.P_brent(a)-p.P_brent(a-1)) < tol
107                         if p.method_n(a-1) == 1  %used secant method
108                             if abs(p.P_brent(a-1)-p.P_brent(a-2)) < tol
109                                 p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
110                                 p.method_n(a)=2;
111                             end
112                         else
113                             p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
114                             p.method_n(a)=2;
115                         end
116                     elseif abs(s-b_k) > 0.5*abs(b_k-b_k_1)
117                         if p.method_n(a-1) == 1  %used secant method
118                             if abs(s-b_k) > 0.5*abs(b_k_1-p.P_brent(a-2)
                                    )
119                                 p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
120                                 p.method_n(a)=2;
121                             end
122                         else
123                             p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
124                             p.method_n(a)=2;
125                         end
126                     end
127                 end
128             end %brents method end
129
130         %both above the desired stroke, if p.below does exist, employ
                brents method
131         elseif p.x_stroke_brent(a) > p.x_stroke_ref && p.x_stroke_brent(
                a-1) > p.x_stroke_ref
132             if isfield(p,'below')==0
133                 p.P_brent(a+1)=p.P_brent(a-1)*0.4;
134             else
135                 %if the current iterate is above the reference
136                 if p.x_stroke_brent(a) >= p.x_stroke_ref
137                     b_k = p.P_brent(a);
138                     a_k = p.P_brent(p.below);
```

```
139                     b_k_1 = p.P_brent(a-1);
140                     %bisection step
141                     m=(p.P_brent(a)+a_k)/2;
142                     %Secant step
143                     s=p.P_brent(a) - (p.x_stroke_brent(a) - p.
                            x_stroke_ref)*(p.P_brent(a) - p.P_brent(a-1))/(p
                            .x_stroke_brent(a) - p.x_stroke_brent(a-1));
144                     if s < b_k && s > m
145                         p.P_brent(a+1)=s;
146                         p.method_n(a)=1;
147                     else
148                         p.P_brent(a+1)=m;
149                         p.method_n(a)=2;
150                     end
151                 %if the current iterate is below the reference
152                 else
153                     b_k = p.P_brent(a);
154                     a_k = p.P_brent(p.above);
155                     b_k_1 = p.P_brent(a-1);
156                     %bisection step
157                     m=(p.P_brent(a)+a_k)/2;
158                     %Secant step
159                     s=p.P_brent(a) - (p.x_stroke_brent(a) - p.
                            x_stroke_ref)*(p.P_brent(a) - p.P_brent(a-1))/(p
                            .x_stroke_brent(a) - p.x_stroke_brent(a-1));
160                     if s < b_k && s > m
161                         p.P_brent(a+1)=s;
162                         p.method_n(a)=1;
163                     else
164                         p.P_brent(a+1)=m;
165                         p.method_n(a)=2;
166                     end
167                 end
168
169                 %Brent's addition to the algorithmn
170                 if a>=3
171                     if abs(p.P_brent(a)-p.P_brent(a-1)) < tol
172                         if p.method_n(a-1) == 1  %used secant method
173                             if abs(p.P_brent(a-1)-p.P_brent(a-2)) < tol
174                                 p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
175                                 p.method_n(a)=2;
176                             end
177                         else
178                             p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
179                             p.method_n(a)=2;
180                         end
181                     elseif abs(s-b_k) > 0.5*abs(b_k-b_k_1)
182                         if p.method_n(a-1) == 1  %used secant method
183                             if abs(s-b_k) > 0.5*abs(b_k_1-p.P_brent(a-2)
                                    )
184                                 p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
185                                 p.method_n(a)=2;
```

```
186                                 end
187                             else
188                                 p.P_brent(a+1)=(p.P_brent(a)+a_k)/2;
189                                 p.method_n(a)=2;
190                             end
191                         end
192                     end
193                 end %brents method end
194             end
195
196             error_stroke = abs(p.x_stroke_ref - p.x_stroke_brent(a));
197             a=a+1;
198
199         else  %a == 1
200             error_stroke = 1;
201             a = a+1;
202             p.method_n=0;
203         end  % end entry logic
204     else  %if Brent's method is turned off
205         error_stroke = 0;
206     end
207
208     end
```

C.16 Function *EOS*

```
function [ property ] = EOS( value1,value2,propertyname,name )
%EOS Mex function wrapper
%
%Equation of state for r-134a calculations
%   give the function two values, temperature is required.
%   the preferred second value is density, the alternative is pressure.
%  The name field is 'rho' if the second value is density.
%  The name field is 'P' if the second value is pressure.
%  Property name is the property you are looking for:
%       -rho
%       -pressure
%       -enthalpy
%       -internalenergy
%       -entropy
%       -Cp
%       -Cv
%       -conductivity
%       -viscosity
%  [ property ] = EOS( value1,value2,propertyname,name )

%either 'mex' or 'code'
code = 'mex';

switch code
    case 'mex'
        switch name
            case 'rho'

                switch propertyname
                 case 'rho'
                     property=EOS_mex(value1,value2,1,1);
                 case 'pressure'
                     property=EOS_mex(value1,value2,2,1);
                 case 'enthalpy'
                     property=EOS_mex(value1,value2,3,1);
                 case 'internalenergy'
                     property=EOS_mex(value1,value2,4,1);
                 case 'entropy'
                     property=EOS_mex(value1,value2,5,1);
                 case 'Cp'
                     property=EOS_mex(value1,value2,6,1);
                 case 'Cv'
                     property=EOS_mex(value1,value2,7,1);
                 case 'conductivity'
                     property=EOS_mex(value1,value2,8,1);
                 case 'viscosity'
                     property=EOS_mex(value1,value2,9,1);
                 case 'psat'
                     property=EOS_mex(value1,value2,10,1);
                 otherwise
```

```
51                      error('Incorrect property name.')
52                  end
53
54              case 'P'
55
56                  switch propertyname
57                   case 'rho'
58                      property=EOS_mex(value1,value2,1,2);
59                   case 'pressure'
60                      property=EOS_mex(value1,value2,2,2);
61                   case 'enthalpy'
62                      property=EOS_mex(value1,value2,3,2);
63                   case 'internalenergy'
64                      property=EOS_mex(value1,value2,4,2);
65                   case 'entropy'
66                      property=EOS_mex(value1,value2,5,2);
67                   case 'Cp'
68                      property=EOS_mex(value1,value2,6,2);
69                   case 'Cv'
70                      property=EOS_mex(value1,value2,7,2);
71                   case 'conductivity'
72                      property=EOS_mex(value1,value2,8,2);
73                   case 'viscosity'
74                      property=EOS_mex(value1,value2,9,2);
75                   case 'rhosat'
76                      property=EOS_mex(value1,value2,10,2);
77                   otherwise
78                      error('Incorrect property name.')
79                  end
80
81
82              otherwise
83                  error('Incorrect name input to EOS, either rho or P.')
84          end
85
86
87      case 'code'
88
89          fluid='R134A'; %Used to change the fluid desired, only
                applicable when using REFPROP to calculate properties.
90
91
92          method=1;  %Operator to determine which calcuation method to use
                for calculating fluid properties.
93                      % 1=ideal gas
94                      % 2=REFPROP.
95                      % 3=R134a EOS hard-coded
96
97          if method==1
98                  Cv_v=0.7633;   %All these values for r-134a
99                  R=0.08149;
100                 Cp_v=R+Cv_v;
```

```
101                 switch propertyname
102                 case 'pressure'
103                       property=value2*R*value1;
104                 case 'enthalpy'
105                     if name=='rho'
106
107                         property=Cp_v*value1;
108                     end
109
110                     if name=='P'
111                         property=Cp_v*value1;
112                     end
113                  case 'entropy'
114                     if name=='rho'
115                         property=refpropm('S','T',value1,'D',value2,
                                fluid);
116                     end
117
118                     if name=='P'
119                         property=refpropm('S','T',value1,'P',value2,
                                fluid);
120                     end
121                  case 'internalenergy'
122                     if name=='rho'
123                         property=Cv_v*value1;
124                     end
125
126                     if name=='P'
127                         property=Cv_v*value1;
128                     end
129                  case 'Cv'
130                     if name=='rho'
131                         property=Cv_v;
132                     end
133
134                     if name=='P'
135                         property=Cv_v;
136                     end
137                  case 'rho'
138                     property=value2/(R*value1);
139                 end
140         elseif method==2
141             switch propertyname
142                 case 'pressure'
143                     property=refpropm('P','T',value1,'D',value2,fluid);
144                 case 'enthalpy'
145                     if name=='rho'
146                         property=refpropm('H','T',value1,'D',value2,
                                fluid);
147                     end
148
149                     if name=='P'
```

```
150                         property=refpropm('H','T',value1,'P',value2,
                                fluid);
151                     end
152                 case 'entropy'
153                     if name=='rho'
154                         property=refpropm('S','T',value1,'D',value2,
                                fluid);
155                     end
156
157                     if name=='P'
158                         property=refpropm('S','T',value1,'P',value2,
                                fluid);
159                     end
160                 case 'internalenergy'
161                     if name=='rho'
162                         property=refpropm('U','T',value1,'D',value2,
                                fluid);
163                     end
164
165                     if name=='P'
166                         property=refpropm('U','T',value1,'P',value2,
                                fluid);
167                     end
168                 case 'Cv'
169                     if name=='rho'
170                         property=refpropm('O','T',value1,'D',value2,
                                fluid);
171                     end
172
173                     if name=='P'
174                         property=refpropm('O','T',value1,'P',value2,
                                fluid);
175                     end
176                 case 'rho'
177                     property=refpropm('D','T',value1,'P',value2,fluid);
178             end
179
180
181         elseif method==3
182             switch propertyname
183             case 'pressure'
184                 property=pressure(value1,value2);
185             case 'enthalpy'
186                 if name=='rho'
187                     property=enthalpy(value1,value2);
188                 end
189
190                 if name=='P'
191                     property=enthalpy_P(value1,value2);
192                 end
193              case 'entropy'
194                 if name=='rho'
```

```
195                     property=entropy(value1,value2);
196                 end
197
198                 if name=='P'
199                     property=entropy_P(value1,value2);
200                 end
201             case 'internalenergy'
202                 if name=='rho'
203                     property=internalenergy(value1,value2);
204                 end
205
206                 if name=='P'
207                     property=internalenergy_P(value1,value2);
208                 end
209             case 'Cv'
210                 if name=='rho'
211                     property=Cv(value1,value2);
212                 end
213
214                 if name=='P'
215                     property=Cv_P(value1,value2);
216                 end
217             case 'rho'
218                 property=rho(value1,value2);
219             end
220
221         else
222             disp('Error in EOS, property calculation selection invalid.'
                    )
223         end
224
225 end
```

C.17 Function *Cv*

```
function [Cv] = Cv( T,rho )
%This function calculates the specific heat (at constant volume)
%for R-134a with a reference state at -40C and saturated  liquid.



%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08149;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];

%%Initial Calculations
tau=T_star/T;
delta=rho/rho_star;

%Calculate phi_o_tau_tau
```

```
51
52  t2=0;
53  for i=4:1:5
54      t2=t2+a_o(i)*t_o(i)*(t_o(i)-1)*tau^(t_o(i)-2);
55  end
56
57  phi_o_tau_tau=-(a_o(3)/tau^2)+t2;
58
59  %Calculate phi_r_tau_tau
60
61  t1=0;
62
63  for i=1:1:N(1)
64      t1=t1+a(i)*t(i)*(t(i)-1)*(delta^(d(i)))*(tau^(t(i)-2));
65  end
66
67  t2_a=0;
68  for k=1:1:4
69      t2_b=0;
70      for i=(N(k)+1):1:N(k+1)
71          t2_b=t2_b+a(i)*t(i)*(t(i)-1)*(delta^(d(i)))*(tau^(t(i)-2));
72      end
73      t2_a=t2_a+exp(-delta^k)*t2_b;
74
75  end
76
77  phi_r_tau_tau=t1+t2_a;
78
79  Cv=-R*(tau^2)*(phi_o_tau_tau+phi_r_tau_tau);
80
81  end
```

C.18 Function *Cv_p*

```
function [ Cv ] = Cv_P( T,P )
%Cv given T and P
%   Detailed explanation goes here

%T=284;
%P=426.51;

clear rho
rho(2)=10;
rho(1)=9.5;
i=2;
dP=1;


while abs(dP)>0.00001
    rho(i+1)=rho(i)-(pressure(T,rho(i))-P)*(rho(i)-rho(i-1))/(pressure(T
        ,rho(i))-pressure(T,rho(i-1)));

    dP=pressure(T,rho(i+1))-P;
    i=i+1;
end

   rho_c=rho(i);

   Cv=Cv_P(T,rho_c);

end
```

C.19 Function *enthalpy*

```
function [h]=enthalpy(T,rho)
%%Function calculates the enthalpy of R-134a given the temperature and
%%density. Reference state is -40C, sat liquid.

%T=100+273.15;
%rho=1/0.50410;

%Reference value pre-allocated for speed.
h=(internalenergy(T,rho)+pressure(T,rho)/rho)-0.020168116568033;
%h=(internalenergy(T,rho)+pressure(T,rho)/rho)-enthalpy_ref;
%end
```

C.20 Function *enthalpy_P*

```
function [ h ] = enthalpy_P( T,P )
%enthalpy given T and P
%   Detailed explanation goes here

%T=284;
%P=426.51;

clear rho
rho(2)=10;
rho(1)=9.5;
i=2;
dP=1;


while abs(dP)>0.00001
    rho(i+1)=rho(i)-(pressure(T,rho(i))-P)*(rho(i)-rho(i-1))/(pressure(T
        ,rho(i))-pressure(T,rho(i-1)));

    dP=pressure(T,rho(i+1))-P;
    i=i+1;
end

   rho_c=rho(i);

   h=enthalpy(T,rho_c);
end
```

C.21 Function *enthalpy_ref*

```
function [h]=enthalpy_ref
%%Function calculates the enthalpy of R-134a given the temperature and
%%density.

T=-40+273.15;
rho=1/0.0007054;
h=internalenergy(T,rho)+pressure(T,rho)/rho;

end
```

C.22 Function *internalenergy*

```
function [u]=internalenergy(T,rho)
%%Function calculates the Internal energy of R-134a given the
    temperature and
%%density. Reference state is -40C, sat liquid.

%T=52+273.15;
%rho=1/0.0009150;
%T=284;
%rho=20.794;


%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];

```

```
50 %%Initial Calculations
51 tau=T_star/T;
52 delta=rho/rho_star;
53
54 %%Calculate phi_o_tau%%
55
56 t3=0;  %initialize t3
57 for i=4:1:5
58     t3=t3+a_o(i)*t_o(i)*tau^(t_o(i)-1);  %calculate the third term
59 end
60 phi_o_tau=a_o(2)+(a_o(3)/tau)+t3;  %add up all three terms in phi_o_tau
61
62 %%Calculate phi_r_tau%%
63
64
65 t1=0;
66
67 for i=1:1:N(1)
68     t1=t1+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
69 end
70
71 i=0;
72 t2_a=0;
73 c=0;
74 for k=1:1:4
75     t2_b=0;
76     for i=(N(k)+1):1:N(k+1)
77         t2_b=t2_b+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
78         c=c+1;
79     end
80     t2_a=t2_a+exp(-delta^k)*t2_b;
81
82 end
83
84 phi_r_tau=t1+t2_a;
85
86 %%Calculate internal energy%%
87
88 %Second number is the reference internal energy, pre-calculated for
      speed
89 u=(R*T*tau*(phi_r_tau+phi_o_tau))-1.481554668959972e+02;
90 %u=(R*T*tau*(phi_r_tau+phi_o_tau))-internalenergy_ref;
91
92 h=u+pressure(T,rho)/rho;
93
94
95 %end
```

C.23 Function *internalenergy_P*

```
function [ u ] = internalenergy_P( T,P )
%internal energy given T and P
%   Detailed explanation goes here

%T=284;
%P=426.51;

clear rho
rho(2)=10;
rho(1)=9.5;
i=2;
dP=1;


while abs(dP)>0.00001
    rho(i+1)=rho(i)-(pressure(T,rho(i))-P)*(rho(i)-rho(i-1))/(pressure(T
        ,rho(i))-pressure(T,rho(i-1)));

    dP=pressure(T,rho(i+1))-P;
    i=i+1;
end

   rho_c=rho(i);

   u=internalenergy(T,rho_c);

end
```

C.24 Function *internalenergy_ref*

```
function [u]=internalenergy_ref
%%Function calculates the Internal energy of R-134a given the
    temperature and
%%density.

T=-40+273.15;
rho=1/0.00070540;
%T=284;
%rho=20.794;


%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];

```

```
%%Initial Calculations
tau=T_star/T;
delta=rho/rho_star;

%%Calculate phi_o_tau%%

t3=0;  %initialize t3
for i=4:1:5
    t3=t3+a_o(i)*t_o(i)*tau^(t_o(i)-1);  %calculate the third term
end
phi_o_tau=a_o(2)+(a_o(3)/tau)+t3;  %add up all three terms in phi_o_tau

%%Calculate phi_r_tau%%


t1=0;

for i=1:1:N(1)
    t1=t1+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
end

i=0;
t2_a=0;
c=0;
for k=1:1:4
    t2_b=0;
    for i=(N(k)+1):1:N(k+1)
        t2_b=t2_b+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
        c=c+1;
    end
    t2_a=t2_a+exp(-delta^k)*t2_b;

end

phi_r_tau=t1+t2_a;

%%Calculate internal energy%%

u=R*T*tau*(phi_r_tau+phi_o_tau);

h=u+pressure(T,rho)/rho;


%end
```

C.25 Function *internalenergy*

```
function [u]=internalenergy(T,rho)
%%Function calculates the Internal energy of R-134a given the
    temperature and
%%density. Reference state is -40C, sat liquid.

%T=52+273.15;
%rho=1/0.0009150;
%T=284;
%rho=20.794;


%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];

```

```
50 %%Initial Calculations
51 tau=T_star/T;
52 delta=rho/rho_star;
53
54 %%Calculate phi_o_tau%%
55
56 t3=0;  %initialize t3
57 for i=4:1:5
58     t3=t3+a_o(i)*t_o(i)*tau^(t_o(i)-1);  %calculate the third term
59 end
60 phi_o_tau=a_o(2)+(a_o(3)/tau)+t3;  %add up all three terms in phi_o_tau
61
62 %%Calculate phi_r_tau%%
63
64
65 t1=0;
66
67 for i=1:1:N(1)
68     t1=t1+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
69 end
70
71 i=0;
72 t2_a=0;
73 c=0;
74 for k=1:1:4
75     t2_b=0;
76     for i=(N(k)+1):1:N(k+1)
77         t2_b=t2_b+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
78         c=c+1;
79     end
80     t2_a=t2_a+exp(-delta^k)*t2_b;
81
82 end
83
84 phi_r_tau=t1+t2_a;
85
86 %%Calculate internal energy%%
87
88 %Second number is the reference internal energy, pre-calculated for
      speed
89 u=(R*T*tau*(phi_r_tau+phi_o_tau))-1.481554668959972e+02;
90 %u=(R*T*tau*(phi_r_tau+phi_o_tau))-internalenergy_ref;
91
92 h=u+pressure(T,rho)/rho;
93
94
95 %end
```

C.26 Function *entropy*

```
function [s]=entropy(T,rho)
%%Function calculates the Internal energy of R-134a given the
    temperature and
%%density. Reference state is -40C, sat liquid.


%T=100+273.15;
%rho=1/0.50410;
%T=284;
%rho=20.794;


%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];
```

```
50
51 %%Initial Calculations
52 tau=T_star/T;
53 delta=rho/rho_star;
54
55 %%Calculate phi_o_tau%%
56
57 t3=0; %initialize t3
58 for i=4:1:5          %MADE A CHANGE HERE
59     t3=t3+a_o(i)*t_o(i)*tau^(t_o(i)-1); %calculate the third term
60 end
61 phi_o_tau=a_o(2)+(a_o(3)/tau)+t3; %add up all three terms in phi_o_tau
62
63 %%Calculate phi_r_tau%%
64
65
66 t1=0;
67
68 for i=1:1:N(1)
69     t1=t1+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
70 end
71
72 i=0;
73 t2_a=0;
74 c=0;
75 for k=1:1:4
76     t2_b=0;
77     for i=(N(k)+1):1:N(k+1)
78         t2_b=t2_b+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
79         c=c+1;
80     end
81     t2_a=t2_a+exp(-delta^k)*t2_b;
82
83 end
84
85 phi_r_tau=t1+t2_a;
86
87 %Calculate phi_o
88
89 phi_o=a_o(1)+a_o(2)*tau+a_o(3)*log(tau)+log(delta)+a_o(4)*(tau^(-1/2))+
     a_o(5)*(tau^(-3/4));
90
91 %Calculate phi_r
92
93 i=1;
94 t1=0;
95 for i=1:1:8
96     t1=t1+a(i)*(tau^t(i))*(delta^d(i));
97 end
98
99 t2=0;
100 for i=9:1:11
```

```
101         t2=t2+a(i)*(tau^t(i))*(delta^d(i));
102     end
103     t2=t2*exp(-delta);
104
105     t3_1=0;
106     t3_2=0;
107     for i=12:1:17
108         t3_1=t3_1+a(i)*(tau^t(i))*(delta^d(i));
109     end
110     t3_1=t3_1*exp(-delta^2);
111
112     for i=18:1:20
113         t3_2=t3_2+a(i)*(tau^t(i))*(delta^d(i));
114     end
115     t3_2=t3_2*exp(-delta^3);
116     t3=t3_1*t3_2;
117
118     t4=a(21)*exp(-delta^4)*(tau^t(21))*(delta^d(21));
119
120     phi_r=t1+t2+t3+t4;
121
122     %%Calculate entropy
123
124     s=(R*(tau*(phi_o_tau+phi_r_tau)-phi_o-phi_r))-entropy_ref;
```

C.27 Function *entropy_P*

```
function [ s ] = entropy_P( T,P )
%Cv given T and P
%   Detailed explanation goes here

%T=284;
%P=426.51;

clear rho
rho(2)=10;
rho(1)=9.5;
i=2;
dP=1;


while abs(dP)>0.00001
    rho(i+1)=rho(i)-(pressure(T,rho(i))-P)*(rho(i)-rho(i-1))/(pressure(T
        ,rho(i))-pressure(T,rho(i-1)));

    dP=pressure(T,rho(i+1))-P;
    i=i+1;
end

   rho_c=rho(i);

   s=entropy(T,rho_c);

end
```

C.28 Function *entropy_ref*

```
function [s]=entropy_ref
%%Function calculates the Internal energy of R-134a given the
    temperature and
%%density. Reference state is -40C, sat liquid.


T=-40+273.15;
rho=1/0.0007054;
%T=284;
%rho=20.794;


%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];

a_o=[-1.019535,
     9.047135,
     -1.629789,
     -9.723916,
     -3.927170];

t_o=[0,1,0,-0.5,-0.75];
```

```
50
51 %%Initial Calculations
52 tau=T_star/T;
53 delta=rho/rho_star;
54
55
56 %%Calculate phi_o_tau%%
57
58 t3=0; %initialize t3
59 for i=4:1:5     %MADE A CHANGE HERE
60     t3=t3+a_o(i)*t_o(i)*tau^(t_o(i)-1); %calculate the third term
61 end
62 phi_o_tau=a_o(2)+(a_o(3)/tau)+t3; %add up all three terms in phi_o_tau
63
64 %%Calculate phi_r_tau%%
65
66
67 t1=0;
68
69 for i=1:1:N(1)
70     t1=t1+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
71 end
72
73 i=0;
74 t2_a=0;
75 c=0;
76 for k=1:1:4
77     t2_b=0;
78     for i=(N(k)+1):1:N(k+1)
79         t2_b=t2_b+a(i)*t(i)*(delta^(d(i)))*(tau^(t(i)-1));
80         c=c+1;
81     end
82     t2_a=t2_a+exp(-delta^k)*t2_b;
83
84 end
85
86 phi_r_tau=t1+t2_a;
87
88 %Calculate phi_o
89
90 phi_o=a_o(1)+a_o(2)*tau+a_o(3)*log(tau)+log(delta)+a_o(4)*(tau^(-1/2))+
    a_o(5)*(tau^(-3/4));
91
92 %Calculate phi_r
93
94 i=1;
95 t1=0;
96 for i=1:1:8
97     t1=t1+a(i)*(tau^t(i))*(delta^d(i));
98 end
99
100 t2=0;
```

```
101 for i=9:1:11
102     t2=t2+a(i)*(tau^t(i))*(delta^d(i));
103 end
104 t2=t2*exp(-delta);
105
106 t3_1=0;
107 t3_2=0;
108 for i=12:1:17
109     t3_1=t3_1+a(i)*(tau^t(i))*(delta^d(i));
110 end
111 t3_1=t3_1*exp(-delta^2);
112
113 for i=18:1:20
114     t3_2=t3_2+a(i)*(tau^t(i))*(delta^d(i));
115 end
116 t3_2=t3_2*exp(-delta^3);
117 t3=t3_1*t3_2;
118
119 t4=a(21)*exp(-delta^4)*(tau^t(21))*(delta^d(21));
120
121 phi_r=t1+t2+t3+t4;
122
123 %%Calculate entropy
124
125 s=(R*(tau*(phi_o_tau+phi_r_tau)-phi_o-phi_r));
```

C.29 Function *pressure*

```
function [P]=pressure(T,rho)
%%Function calculates the pressure of R-134a given the temperature and
%%density.



%T=-40+273.15;
%rho=1/0.14729;

%%Define Constants
T_star=374.18;    %K
rho_star=508;     %kg/m^3
R=0.08148886;        %kJ/kg-K

a=[0.5586817e-1,
     0.4982230,
     0.2458698e-1,
     0.8570145e-3,
     0.4788584e-3,
     -0.1800808e1,
     0.2671641,
     -0.4781652e-1,
     0.1423987e-1,
     0.3324062,
     -0.7485907e-2,
     0.1017263e-3,
     -0.5184567,
     -0.8692288e-1,
     0.2057144,
     -0.5000457e-2,
     0.4603262e-3,
     -0.3497836e-2,
     0.6995038e-2,
     -0.1452184e-1,
     -0.1285458e-3];

t=[-0.5,0,0,0,1.5,1.5,2,2,1,3,5,1,5,5,6,10,10,10,18,22,50];
d=[2,1,3,6,6,1,1,2,5,2,2,4,1,4,1,2,4,1,5,3,10];
N=[8,11,17,20,21];



%%Initial Calculations
tau=T_star/T;
delta=rho/rho_star;


%%Partial Residuals
%%Initalize residual pieces

```

```
51 t2_a=0;
52 t2_b=0;
53 phi_r_delta=0;
54 t2=0;
55 t1=0;
56
57 %%Calculating first term in residual
58 for i=1:1:N(1)
59     t1=a(i)*d(i)*(delta^(d(i)-1))*tau^(t(i));    %temporary first term
60     phi_r_delta=t1+phi_r_delta;                  %summation of first
           terms
61 end
62
63
64 %%Calculating second term in residual
65 for k=1:1:4
66     t2_b_t=0;
67     t2_b=0;    %%%% THIS IS THE CRITICAL LINE! %%%%%
68     t2_a=exp(-delta^k);       %outer term of the second term
69     for i=N(k)+1:1:N(k+1)
70         t2_b_t=a(i)*(d(i)-k*delta^k)*(delta^(d(i)-1))*tau^(t(i));     %
               temporary inner term
71         t2_b=t2_b+t2_b_t;    % Keep sum of inner loop terms
72     end
73     t2_t=t2_a*t2_b;          % temporary multiple of outer and inner
           terms
74     t2=t2_t+t2;              % Keep sum of terms
75 end
76
77 phi_r_delta=phi_r_delta+t2;
78
79
80 %phi_r_delta=1;
81
82
83 %%Calculate Pressure
84
85 P=R*T*rho*(1+delta*phi_r_delta);
```

C.30 Function *rho*

```
function [ rho_c ] = rho( T,P)
%rho given T and P
%   Detailed explanation goes here

%T=284;
%P=426.51;

clear rho
rho=zeros(50);
rho(2)=20;
rho(1)=19.5;
i=2;
dP=1;
P_loop=zeros(50);



while abs(dP)>0.00001
    P_loop(i)=pressure(T,rho(i));
    rho(i+1)=rho(i)-(P_loop(i)-P)*(rho(i)-rho(i-1))/(P_loop(i)-P_loop(i
        -1));

    dP=pressure(T,rho(i+1))-P;
    i=i+1;
end

   rho_c=rho(i);



end
```

C.31 Function *dP_dT*

```
function [ dP_dT ] = dP_dT( T,rho )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

%T=400;
%rho=1/0.04;

P=EOS(T,rho,'pressure','rho');

dT=0.0001;

dP_dT=((EOS(T+dT,rho,'pressure','rho')-P)/dT)*1000;

end
```

Appendix D: Modified Comprehensive Model Code to Represent Commercial Linear Compressor

The comprehensive model code presented in Appendix C was modified to better represent the commercial linear compressor discussed in Chapter 6. The changes made to the solution algorithm are presented in a modified *centralprogram.m* and changes made to the vibration model are presented in *vibration.m*. The remaining functions that construct the model remain unchanged as presented in Appendix C, including the equation of state.

D.1 Function *centralprogram*

```
1
2  %Main Model Program
3  %Craig Bradshaw
4  %Feburary 2012
5  %Herrick Labs, cbradsha@purdue.edu
6  %
7
8  try
9
10     clc
11     clear all
12     close all
13
14     %% Initializations
15
16     pc_flag=getflag();  %1 if on Linux, 0 if on PC
17
18     if pc_flag == 0
19
20         %How long do I need to run this simulation for? Obtained from
               first two
21         %numbers on the inputs.xls file.
22         num_size=xlsread('inputs.xls','A3:A4');
23         analy_length=(num_size(2)-num_size(1))+1;
24
25         %Open Excel Activex Server to use xlsread/write for prelim stuff
26         Excel = actxserver ('Excel.Application');
27         File=strcat(pwd,'\data_log.xls');
28         if ~exist(File,'file')
29             ExcelWorkbook = Excel.workbooks.Add;
30             ExcelWorkbook.SaveAs(File,1);
31             ExcelWorkbook.Close(false);
```

```
32          end
33          invoke(Excel.Workbooks,'Open',File);
34
35          %Now using modified xlsread/write to work on data_log.xls
36          [num,txt,raw]=xlsread1('data_log.xls');
37          [rows,col]=size(raw);
38
39          %Adding a note to the batch log so I know what I was doing
40          batch_on=input('Is this a batch file? 1 for yes, 0 for no ');
41          if batch_on==1
42              batch_note=input('Please input a string to describe why you
                    are running this batch.');
43              batch_note=cellstr(batch_note);
44              batch_range=strcat('A',num2str(rows+2),':A',num2str(rows+2))
                    ;
45              xlswrite1('data_log.xls',batch_note,batch_range);
46          end
47
48          %Shutting down activex server to excel
49          invoke(Excel.ActiveWorkbook,'Save');
50          Excel.Quit
51          Excel.delete
52          clear Excel
53
54      elseif pc_flag == 1
55          row_start = input('Row to start with: ');
56          row_end = input('Row to end with: ');
57          analy_length = row_end - row_start + 1;
58          input_data = xlsread('inputs.xls');
59      else
60          error('Invalid pc_flag.  Please use either 0 or 1.');
61          clear all
62      end
63      %turn annoying warning off
64      warning('off','MATLAB:Print:SavingToDifferentName')
65
66      %%%%%%%%%%%%
67      % For loop used for batch operation, line by line from input file, z
68      % loop
69      %%%%%%%%%%%%
70
71      for z=1:1:analy_length
72
73          if pc_flag == 0 %on a PC
74
75              %pulling input data from inputs.xls
76              range=strcat('B',num2str(z+2),':','BV',num2str(z+2));
77              [num_inputs,txt_inputs]=xlsread('inputs.xls',range);
78
79              %creating my variable structure
80              [p]=create_struct(num_inputs,txt_inputs);
81
```

```
82              %name that I use to generate a save folder
83              p.save_name=strcat(p.save_name,'_',num2str(z));
84
85              %Open Excel Activex Server to use xlsread/write for the
                    remainder of
86              %the analysis
87              Excel = actxserver ('Excel.Application');
88              File=strcat(pwd,'\data_log.xls');
89              if ~exist(File,'file')
90                  ExcelWorkbook = Excel.workbooks.Add;
91                  ExcelWorkbook.SaveAs(File,1);
92                  ExcelWorkbook.Close(false);
93              end
94              invoke(Excel.Workbooks,'Open',File);
95
96              %Open data log
97              [num,txt,raw]=xlsread1('data_log.xls');
98              [rows,col]=size(raw);
99
100         elseif pc_flag == 1 %on Linux
101             if z == 1
102                 txt_inputs{8} = 'Euler';
103                 txt_inputs{56} = input('Name for Study:  ');
104                 d_name = strcat(txt_inputs{56},'.txt');
105                 diary(d_name)
106             end
107
108             %num_inputs = input_data(z+2,2:61);
109             num_inputs = input_data(z+row_start-1,2:74);
110
111             %Create structure of constant variables
112             [p]=create_struct(num_inputs,txt_inputs);
113
114             %name that I use to generate a save folder
115             p.save_name=strcat(p.save_name,'_',num2str(z+row_start-1));
116         end
117
118     %Time/Volume Calculations
119     error=0.000001;      %1e-5, total error boundaries
120     f_list=p.f_begin:p.f_increment:p.f_end;
121     w_d_list=f_list*2*pi;
122     p.x_stroke_initial = p.x_stroke;
123
124     %%%%%%%%
125     % Loop used for frequency sweeps, I loop
126     %%%%%%%%
127
128     p.resonant_loop = 0;        %initialize variable that tells
            frequency sweep loop if we have reached the maximum frequency.
129     I=1;                        %loop variable for the following loop.
130
131     while p.resonant_loop<1 && I<=length(w_d_list)
```

```
132     %for l=1:1:length(w_d_list)
133
134         close all
135         p.w_d=w_d_list(l);
136         p.Period=1/f_list(l);     %Time of one cycle
137         p.t_step=1/(f_list(l)*p.n);       %time step in seconds
138         p.transient=0;
139         t_adjust_force=0;
140
141         %Initialze residuals and convergence loop iteration counter
142         k=1;
143         dT_loop=1;
144         drho_loop=1;
145         dT_cv2_loop=1;
146         drho_cv2_loop=1;
147         dT_loop_k=1;
148         drho_loop_k=1;
149         dT_cv2_loop_k=1;
150         drho_cv2_loop_k=1;
151         dx_piston=1;
152
153         if p.brent == 1
154             %p.P_electric = p.P_brent(a);
155         end
156
157         %% Initalize Vectors
158         %These are the vectors that are iterated on and are thus not
                included in
159         %the parameters structure, i loop variables
160
161         P=zeros(1,p.n_period*p.n);
162         dT=zeros(1,p.n_period*p.n);
163         h=zeros(1,p.n_period*p.n-1);
164         V=zeros(1,p.n_period*p.n-1);
165         dV=zeros(1,p.n_period*p.n-1);
166         dm=zeros(1,p.n_period*p.n-1);
167         dm_in=zeros(1,p.n_period*p.n-1);
168         dm_out=zeros(1,p.n_period*p.n-1);
169         drho=zeros(1,p.n_period*p.n-1);
170         m=zeros(1,p.n_period*p.n-1);
171         rho=zeros(1,p.n_period*p.n);
172         x_piston=zeros(1,p.n_period*p.n-1);
173         x_dot_piston=zeros(1,p.n_period*p.n-1);
174         Cv=zeros(1,p.n_period*p.n-1);
175         iter=zeros(1,p.n_period*p.n);
176         dP_dT_v=zeros(1,p.n_period*p.n-1);
177         Ma=zeros(1,p.n_period*p.n-1);
178         dT_check=zeros(1,p.n_period*p.n-1);
179         T=zeros(1,p.n_period*p.n);
180         x_valve=zeros(1,p.n_period*p.n);
181         x_dot_valve=zeros(1,p.n_period*p.n);
182         dm_leak_in=zeros(1,p.n_period*p.n-1);
```

```
183         dm_leak_out=zeros(1,p.n_period*p.n-1);
184         dm_out_valve=zeros(1,p.n_period*p.n-1);
185         dV_cv2=zeros(1,p.n_period*p.n-1);
186         V_cv2=zeros(1,p.n_period*p.n-1);
187         Ma_cv2=zeros(1,p.n_period*p.n-1);
188         h_cv2=zeros(1,p.n_period*p.n-1);
189         Cv_cv2=zeros(1,p.n_period*p.n-1);
190         dP_dT_v_cv2=zeros(1,p.n_period*p.n);
191         P_cv2=p.P_i*ones(1,p.n_period*p.n);
192         drho_cv2=zeros(1,p.n_period*p.n-1);
193         rho_cv2=p.rho_i*ones(1,p.n_period*p.n-1);
194         dT_cv2=zeros(1,p.n_period*p.n-1);
195         T_cv2=p.T_i*ones(1,p.n_period*p.n-1);
196         F_drive=zeros(1,p.n_period*p.n);
197         Q_motor=zeros(1,p.n_period*p.n);
198         theta=zeros(1,p.n_period*p.n);
199         theta_dot=zeros(1,p.n_period*p.n);
200         dP=zeros(1,p.n_period*p.n);
201         dx=zeros(1,p.n_period*p.n);
202         theta_dot_dot=zeros(1,p.n_period*p.n);
203         Q=zeros(1,p.n_period*p.n);
204         Q_cv2=zeros(1,p.n_period*p.n);
205         m_cv2=zeros(1,p.n_period*p.n);
206         c_eff = zeros(1,p.n_period*p.n);
207         c_friction = zeros(1,p.n_period*p.n);
208         c_gas = zeros(1,p.n_period*p.n);
209         x_dot_dot_piston = zeros(1,p.n_period*p.n);
210         F_gas = zeros(1,p.n_period*p.n);
211         % Initialize convergence loop variables, k loop
212         x_piston_m=0;
213         T_w = zeros(1,p.loop_error);
214         x_stroke_save = zeros(1,p.loop_error);
215         loop = zeros(1,p.loop_error);
216         m_dot = zeros(1,p.loop_error);
217         m_dot_leak_in = zeros(1,p.loop_error);
218         m_dot_leak_out = zeros(1,p.loop_error);
219         W_dot = zeros(1,p.loop_error);
220         eta_o = zeros(1,p.loop_error);
221         eta_vol = zeros(1,p.loop_error);
222         Q_dot = zeros(1,p.loop_error);
223         Q_dot_cv2 = zeros(1,p.loop_error);
224         m_dot_in = zeros(1,p.loop_error);
225         loop=1;     %loop counter variable, re-initialize
226         p.dP_max=p.P_d-p.P_i;
227         P_electric_temp = p.P_electric;         %save input value
228         error_res = 1;          %initialize error_res
229         error_counter = 0;      %count number of times get below
              tolerance
230          %%%%%%%%%%%
231          %%  while loop to enforce convergence, k loop
232          %%%%%%%%%%%
233
```

```
234         while error_counter < 2
235
236             W_dot_in(k) = p.P_electric;
237             %% Inital Cylinder Conditions
238             if k==1
239                 i=1;
                        %initialize i loop counter
240                 T(i)=p.T_i;
                        %initial temperature
241                 rho(i)=p.rho_i;
                        %initial density
242                 P(i)=EOS(T(i),rho(i),'pressure','rho');
                        %calculate initial pressure
243                 T_cv2(1)=p.T_i;
                        %initial temperature, CV2
244                 rho_cv2(1)=p.rho_i;
                        %initial density, CV2
245                 %P_cv2(1)=P(1);
246                 P_cv2(1)=(p.P_d + p.P_i)/2;
                        %initial pressure, CV2
247                 %rho_cv2(1) = EOS(T_cv2(1),P_cv2(1),'rho','P');
248                 x_piston(1)=p.x_piston_i;
                        %initial piston position, default is (max stroke/2)
249                 x_dot_piston(1)=p.x_dot_piston_i;
                        %initial piston velocity, default is 0
250                 V(1)=(p.x_piston_i)*p.Ap+p.V_dead_valve;
                        %initial volume
251                 dV(1)=-p.x_dot_piston_i*p.Ap;
                        %initial change in volume
252                 theta(1)=p.theta_i;
                        %initial piston rotation
253                 theta_dot(1)=p.theta_dot_i;
                        %initial change in piston rotation
254                 p.x_dot_ave=1;
                        %%%%%%IMPORTANT, AVERAGE SPEED
255                 dP(1)=0;
                        %Change in pressure
256                 dx(1)=1;
                        %Change in stroke
257                 T_w(k)=p.T_w_i;
                        %initial compressor lump temperature (wall temp)
258                 x_stroke_save(k)=p.x_stroke_initial;
                        %saved value of stroke
259                 p.x_stroke = p.x_stroke_initial;
                        %guess stroke
260                 p.dP_max = p.P_d-p.P_i;
261                 t=0;
262                 p.dV = 0;
                        %initialize dV
263                 p.P_current = P(i);
264
265             else
```

```
266
267                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
268                 %Update values for iterations beyond first
269                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
270                 T_error = T;
271                 rho_error = rho;
272                 T_error_cv2 = T;
273                 rho_error_cv2 = rho;
274                 x_stroke_error = p.x_stroke;
275                 T(1)=X_n_1(1);
                        %Temperatures and pressures equal to the end values
                        of the previous iteration
276                 rho(1)=X_n_1(2);
277                 P(1)=EOS( T(1),rho(1),'pressure','rho' );
278                 T_cv2(1)=X_n_1(3);
279                 rho_cv2(1)=X_n_1(4);
280                 P_cv2(1)=EOS( T_cv2(1),rho_cv2(1),'pressure','rho' );
281                 p.x_stroke=max(x_piston_m)-min(x_piston_m);
                        %max stroke, current value
282                 p.x_dot_ave=0.707*max(x_dot_piston);
                        %%%%%%IMPORTANT, AVERAGE SPEED, RMS value of speed
283                 x_stroke_save(k)=p.x_stroke;
                        % stroke variable saved to see progression
284                 x_piston(1)=x_piston(i);
285                 x_dot_piston(1)=x_dot_piston(i);
286                 V(1)=V(i);
287                 dV(1)=dV(i);
288                 theta(1)=theta(i);
289                 theta_dot(1)=theta_dot(i);
290                 t=t(i);
291                 dP(1)=dP(i);
292                 dx(1)=dx(i);
                        %rest of above resets remaining variables to the
                        ending previous values of the previous k loop
293                 i=1;                %resets inner loop variable
294             end
295
296             %% Initalize inner loop variables
297             t_cross=0;
                    %number of times piston crosses zero
298             p.t_force_adjust=0;
                    %adjustment in the time vector to account for variable
                    frequency response
299             x_piston_m=0;
                    %initialize x_piston
300             F_drive=0;
                    %initalize driving force vector
301             F_drive_clean=0;
302             %%%%%%%%%
303             %% Main Loop for calculations, i loop
304             %%%%%%%%%
305             while size(t_cross)<p.n_period*2-1
```

```
306
307                 %% Motor Model
308                 if isempty(p.t_force_adjust) == 1
309                     p.t_force_adjust=0;
310                 end
311
312                 if i == 1
                        %only want to perform these calculations on first
                        time through
313                     p.P_motor=p.eta_motor*p.P_electric;
                            %power actually delivered to piston
314                     p.Q_motor=p.P_electric-p.P_motor;
                            %Heat transfer from motor
315                     p.k_m = 4.6831;
316                     p.F_drive_max=sqrt(p.P_motor)*(p.k_m);
                            %Maximum Motor force, in Newtons.
317                 end
318                 F_drive_clean(i)=p.F_drive_max*(cos(p.w_d*(t(i)-p.
                        t_force_adjust)+pi));                %Driving force
                        v. Time, Newtons
319                 p.dP_piston = P(i) - P_cv2(i);
320                 F_drive(i)=F_drive_clean(i) - p.dP_piston*p.Ap*1000;
321
322                 %% Vibration Model and Volume and Piston Position
323
324                 mu_oil = oilvisc( T_w(k),32 );  %oil viscosity, iso 32
325                 [ dV(i+1),V(i+1),x_piston(i+1),x_dot_piston(i+1),theta(i
                        +1),theta_dot(i+1),x_piston_m(i+1),theta_dot_dot(i
                        +1),p ] = vibration( t(i),dP(i),dx(i),x_piston(i),
                        x_dot_piston(i),theta(i),theta_dot(i),x_piston_m(i),
                        theta_dot_dot(i),mu_oil,p );
326                 c_eff(i) = p.c_eff;
327                 c_friction(i) = p.c_friction;
328                 c_gas(i) = p.c_gas;
329                 x_dot_dot_piston(i) = p.x_dot_dot_piston;
330                 k_eff(i) = p.k_eff;
331                 F_gas(i) = p.dP_piston*p.Ap*1000;
332                 F_wall(i) = p.F_wall;
333                 p.dV = V(i+1) - V(i);
334                 p.P_current = P(i);
335
336                 %% Volume for Second Control Volume
337                 dV_cv2(i)=dV(i);
338                 V_cv2(i)=p.V_cv2_i-V(i);
339
340                 %% Massflow (Valve Model)
341                 [ x_valve(i+1),x_dot_valve(i+1),dm_valve, Ma(i) ] =
                        valve_dynamics(P(i),rho(i),T(i),x_valve(i),
                        x_dot_valve(i),p);
342
343                 %% Leakage Flowrates (Leakage Model)
```

```
344                 [dm_leak_in(i),dm_leak_out(i),Ma_cv2(i)] = leakage( P(i)
                        ,P_cv2(i),x_dot_piston(i),T(i),rho(i),T_cv2(i),
                        x_piston(i),p );
345
346                 %% Total flowrates (Massflow Summations)
347                 if dm_valve<0     %flow IN
348                     dm_in(i)=abs(dm_valve)+dm_leak_in(i);
349                     dm_out(i)=dm_leak_out(i);
350                     dm(i)=dm_in(i)-dm_out(i);
351                     dm_out_valve(i)=0;
352                 elseif dm_valve>0 %flow OUT
353                     dm_in(i)=dm_leak_in(i);
354                     dm_out(i)=dm_valve+dm_leak_out(i);
355                     dm(i)=dm_in(i)-dm_out(i);
356                     dm_out_valve(i)=dm_valve;
357                 else
358                     dm_in(i)=dm_leak_in(i);
359                     dm(i)=dm_leak_in(i)-dm_leak_out(i);
360                     dm_out(i)=dm_leak_out(i);
361                     dm_out_valve(i)=0;
362                 end
363
364                 %% Instantaneous Heat Transfer in control volumes
365                 [ Q(i) ] = Ins_HT( T(i),rho(i),T_w(k),V(i),dV(i),
                        x_piston(i),x_dot_piston(i),p);
366                 [ Q_cv2(i) ] = Ins_HT_cv2( T_cv2(i),rho_cv2(i),T_w(k),
                        V_cv2(i),dV_cv2(i),x_dot_piston(i),p.Q_motor,p);
367
368                 %% Mass of the gas in each control volume
369                 m(i)=rho(i)*V(i);
370                 m_cv2(i)=rho_cv2(i)*V_cv2(i);
371
372                 %% Fluid Properties for approximation
373                 if k==1
                        %only need to compute this value once for each batch
                            line
374                     p.h_in=EOS(p.T_i,p.rho_i,'enthalpy','rho');
375                 end
376                 h_out=EOS(T(i),rho(i),'enthalpy','rho');
377                 h(i)=h_out;
378                 dP_dT_v(i)= dP_dT(T(i),rho(i));
379                 Cv(i)=EOS(T(i),rho(i),'Cv','rho');
380                 h_cv2(i)=EOS(T_cv2(i),rho_cv2(i),'enthalpy','rho');
381                 dP_dT_v_cv2(i)=dP_dT(T_cv2(i),rho_cv2(i));
382                 Cv_cv2(i)=EOS(T_cv2(i),rho_cv2(i),'Cv','rho');
383                 if strcmp(p.method,'RK4')==1
384                     %% RK4 Approximations
385
386                     error('RK4 not available in this version');
387
388                 elseif strcmp(p.method,'Euler')==1
389
```

```
390                     %% Simple Euler Approximation
391                     %%%%%%%%%%%%%%%%%
392                     %Euler Approximation, Temperature and Density
393                     %%%%%%%%%%%%%%%%%
394
395                     %%%%%
396                     %1st Control Volume, Piston Cylinder
397                     %%%%%
398                     dT(i)=(Q(i)+T(i)*dP_dT_v(i)*(1/1000)*((1/rho(i))*dm(
                            i)-dV(i))-h(i)*dm(i)+dm_in(i)*p.h_in-dm_out(i)*h
                            (i))/(m(i)*Cv(i));
399                     drho(i)=(dm(i)-rho(i)*dV(i))/V(i);
400
401                     T(i+1)=T(i)+p.t_step*dT(i);
402                     rho(i+1)=rho(i)+p.t_step*drho(i);
403
404                     %%%%%
405                     %2nd Control Volume
406                     %%%%%
407                     dm_cv2=dm_leak_out(i)-dm_leak_in(i);
408                     dm_in_cv2=dm_leak_out(i);
409                     dm_out_cv2=dm_leak_in(i);
410                 else
411                     i=floor(p.Period/p.t_step)-1;
412                     disp('Method error, please select a correct ODE
                            Solution Method')
413                 end
414
415                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
416                 %% Calculate next step for pressure
417                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
418                 P(i+1)=EOS(T(i+1),rho(i+1),'pressure','rho');
419
420                 %%%%%%%%%%%%
421                 %% Misc other properties
422                 %%%%%%%%%%%%
423                 dP(i+1)=P(i+1)-P(i);                    %Change in
                        pressure
424                 dx(i+1)=x_piston(i+1)-x_piston(i);      %change in
                        piston position
425
426                 %%%%%%%%%%%%
427                 %% Loop Adjustments
428                 %%%%%%%%%%%%
429                 %Section determines when the piston crosses back
430                 %through the midpoint, signifying one cycle.
431
432                 if k>1
433                     if i>1
434                         t_cross=find_cross(t',x_piston_m(1:end-1)',0);
435                     end
436                 else
```

```
                if i>1
                    t_cross=find_cross(t',x_piston_m(1:end-1)',0);
                end
            end
            t(i+1)=t(i)+p.t_step;   %initially frequency has to float, transient frequency is different than driving frequency.
            i=i+1;
            if length(t)>=p.n*p.n_period*1.5
                t_cross=ones(2*p.n_period);
                disp('Convergence Error, piston did not cross zero twice')
            end
        end     %End of While loop checking t_cross, i loop

        %%%%%%%%%%%%%%%
        %% Adjustments for Time
        %%%%%%%%%%%%%%%
        i=i-1;   %step i back one for sanity
        t=t(1:end-1);   %step time vector back one
        temp(k)=max(t_cross)-max(find_cross(t',F_drive_clean',0));
        p.t_force_adjust=p.t_force_adjust+temp(k);   %when t_force_adjust is greater than 3, stop i loop
        t_change(k)=p.t_force_adjust;
        p.Period=t(end)-t(1);   %floating period

        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Estimate Massflow, Power Input,Heat Transfer, and Efficiencies
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Mass flows calculated using numerical integration
        m_dot(k)=(trapz(dm_out_valve)*(t(2)-t(1)))/p.Period;
        m_dot_leak_out(k)=(trapz(dm_leak_out)*(t(2)-t(1)))/p.Period;
        m_dot_leak_in(k)=(trapz(dm_leak_in)*(t(2)-t(1)))/p.Period;
        m_dot_in(k) = (trapz(dm_in)*(t(2) - t(1)))/p.Period;

        %W_dot calculated assuming max(h) is similar to h_avg
        %(proper calculation of h_avg done after the loop is over,
        %this value is just for diagnostic purposes)
        W_dot(k)=m_dot(k)*(max(h)-p.h_in);

        %Efficiency estimations, diagnostic values
        eta_o(k)=(m_dot(k)*(p.h_2_s-p.h_in)*1000)/(p.P_electric*2);   %W_dot is in kW
```

```
475             eta_vol(k)=(m_dot(k))/(p.rho_i*f_list(l)*(max(x_piston_m)-
                    min(x_piston_m))*p.Ap);
476
477             %Power consumed by friction
478             W_dot_friction = p.f_friction*F_wall*2*p.x_stroke*f_list(l);
479             W_dot_friction_ave=mean(W_dot_friction)/1000;
480
481             %Heat Transfer estimations, numerically integrated.
482             Q_dot(k)=((trapz(Q)*(t(2)-t(1)))/p.Period);
483             Q_dot_cv2(k)=(trapz(Q_cv2)*(t(2)-t(1)))/p.Period;
484
485             %equation is under-relaxed , alpha = 1/100;
486             T_w(k+1)=(-Q_dot(k)-Q_dot_cv2(k)+W_dot_friction_ave)*1000*(p
                    .R_shell)+p.T_amb;
487             T_w(k+1)=315;
488
489             Q_dot_in=(-Q_dot(k)-Q_dot_cv2(k)+W_dot_friction_ave)*1000;
490             T_w_test(k)=Q_dot_in*p.R_shell+p.T_amb;
491
492             if pc_flag == 0
493                 figure(30)
494                 plot(t,x_piston_m(1:end-1))
495                 hold on
496             end
497             %Diagnostic values
498             m_change(k) = m(end) - m(1);
499             m_change_cv2(k) = m_cv2(end) - m_cv2(1);
500             dE(k) = -Q_dot(k) + m_dot(k)*max(h) - (2*p.P_electric/1000)
                    - m_dot_in(k)*p.h_in;
501             resonant_freq(k) = sqrt(p.k_eff/p.M_mov)/2/pi;
502
503             %% Iteration Counter
504             count1=num2str(k);
505             count2='Iteration Number ';
506             count=[count2 count1];
507             disp(' ')
508             disp(count)
509             loop(k)=k;
510
511             %%%%%%%%%%%%%
512             %% Residuals
513             %%%%%%%%%%%%%
514
515             %%%%%%%%%%%%%%%%%
516             % Error calculations
517             %%%%%%%%%%%%%%%%%
518             if k>1
519                 res_T(k) = 1-abs(mean(T(1:length(t)-1)./T_error(1:length
                        (t)-1)));
520                 res_rho(k) = 1-abs(mean(rho(1:length(t)-1)./rho_error(1:
                        length(t)-1)));
```

```
521                 res_T_cv2(k) = 1-abs(mean(T_cv2(1:length(t)-1)./
                        T_error_cv2(1:length(t)-1)));
522                 res_rho_cv2(k) = 1-abs(mean(rho_cv2(1:length(t)-1)./
                        rho_error_cv2(1:length(t)-1)));
523                 res_x(k) = 1-abs(x_stroke_save(k)/x_stroke_error);
524                 error_res = [abs(res_T(k)), abs(res_rho(k)),abs(res_x(k)
                        )];
525                 error_res = max(error_res);
526
527                 %adjust error counter...looking for two below, then end
528                 if error_res < error
529                     error_counter = error_counter + 1;
530                 end
531
532                 if pc_flag == 1
533                     CV1_res=strcat('CV1 Res.:| ',' DT| ',num2str(res_T(k
                            )),' Drho| ',num2str(res_rho(k)),' dx| ',num2str
                            (res_x(k)));
534                     CV2_res=strcat('CV2 Res.:| ',' DT| ',num2str(
                            res_T_cv2(k)),' Drho| ',num2str(res_rho_cv2(k)))
                            ;
535                     add_info=strcat('Line: ',num2str(z+row_start-1),' |
                            x_stroke: ',num2str(p.x_stroke),'| P_electric: '
                            ,num2str(p.P_electric));
536                     disp(CV1_res)
537                     disp(CV2_res)
538                     disp(add_info)
539                 else
540                     CV1_res=strcat('CV1 Res.:| ',' DT| ',num2str(res_T(k
                            )),' Drho| ',num2str(res_rho(k)),' dx| ',num2str
                            (res_x(k)));
541                     CV2_res=strcat('CV2 Res.:| ',' DT| ',num2str(
                            res_T_cv2(k)),' Drho| ',num2str(res_rho_cv2(k)))
                            ;
542                     add_info=strcat('Line: ',num2str(z),' | x_stroke: ',
                            num2str(p.x_stroke),'| P_electric: ',num2str(p.
                            P_electric));
543                     disp(CV1_res)
544                     disp(CV2_res)
545                     disp(add_info)
546                 end
547
548             end
549
550             %Error Loop Counter
551             if k>=p.loop_error
552                 error_res = 0;
553                 error_counter = 2;
554                 count='Convergence Error, too many iterations';
555                 disp(count)
556             end
557
```

```
558             if k>p.loop_error+5
559
560             else
561                 X_n=[T(i),rho(i),T_cv2(i),rho_cv2(i),x_stroke_save(k)];
562                 F_n=[dT(i),drho(i),dT_cv2(i),drho_cv2(i),x_dot_piston(i)
                        ];
563                 J=0;
564                 DELTAX_n=0;
565                 X_n_1=X_n+DELTAX_n;
566             end
567             %stroke is the mean value of the past 10 iterations
568             if k > 10
569                 p.x_stroke = mean(x_stroke_save(k-10:k));
570             end
571             iter(k)=k;
572             k=k+1;
573             p.dP_max=max(P)-p.P_i;        %Max change in pressure over
                    cycle relative to cylinder
574             if k == 150
575                 error = error + 0.00005;
576             elseif k == 200
577                 error = error + 0.00005;
578             end
579
580             %% Adjustments to power to try and maintain a correct stroke
581             %If stroke is outside of the range of 0.001 and 0.0254 m
                    either
582             %add or substract 1W of power input to try and get it into
                    the
583             %appropriate range
584             if k > 6
585                 if p.x_stroke < 0.0001
586                     p.P_electric = p.P_electric +1;
587                     disp('Increase P_electric =')
588                     disp(p.P_electric)
589                 end
590             end
591         end                                 %End of k while loop
592
593         if pc_flag == 0
594             close(30)  %close stroke plot
595         end
596         %%%%%%%%%%%%%%%%%%%%%%%%%%%%
597         %Brents Method estimations
598         %%%%%%%%%%%%%%%%%%%%%%%%%%%%
599         a = 1;                  %loop iteration variable
600         if isnan(num_inputs(72)) && isnan(num_inputs(73))
601             p.P_brent(1) = 10;   %lower power guess
602             p.P_brent(2) = 15; %second lower power guess
603         elseif (isnan(num_inputs(72)) || isnan(num_inputs(73)))
604             error('One Brents method input but not the other, please
                    input two guess values');
```

```
605         else
606             p.P_brent(1) = num_inputs(72);
607             p.P_brent(2) = num_inputs(73);
608         end
609         error_brents = 0.00000005;    %0.05 microns
610         error_stroke = 1;
611         p.vibration_on = 1;  %turn vibration model back on
612
613         while error_stroke > error_brents   %a loop, brents method to
                back out power for full stroke
614
615             if a == 1
616                 p.P_electric = p.P_brent(a);
617             end
618             x_dot_piston_avg = rms(x_dot_piston);
619             x_piston = zeros(1,2*p.n);
620             x_dot_piston = zeros(1,2*p.n);
621             theta = zeros(1,2*p.n);
622             theta_dot = zeros(1,2*p.n);
623             theta_dot_dot = zeros(1,2*p.n);
624             p.x_stroke = p.x_piston_i;
625             t = 0;
626             x_piston_err = 1;
627             x_piston_m = 0;
628             q=1;
629
630             %loop to ensure piston is at steady state, q loop
631             while x_piston_err > 0.00000001
632                 t_cross=0;
                                                                          %
                    number of times piston crosses zero
633                 p.t_force_adjust=0;
                                                                %adjustment
                    in the time vector to account for variable frequency
                     response
634                 x_piston_m=0;
                                                                        %
                    initialize x_piston
635                 F_drive=0;
                                                                          %
                    initalize driving force vector
636                 F_drive_clean=0;
637                 i=1;
638                 if q>1
639                     t = t(end);
640                 end
641                 %%%%%%%%%%
642                 %% Main Loop for calculations, i loop
643                 %%%%%%%%%%
644                 while size(t_cross)<p.n_period*2-1
645                     if i == 1
                                                                          %
```

```
                    only want to perform these calculations on first
                     time through
646                 p.P_motor=p.eta_motor*p.P_electric;
                                    %power actually delivered to
                         piston
647                 p.Q_motor=p.P_electric-p.P_motor;
                                     %Heat transfer from motor
648                 p.k_m = 4.6831;
649                 p.F_drive_max=sqrt(p.P_motor)*(p.k_m);
                                    %Maximum Motor force, in
                        Newtons.
650             end
651             F_drive_clean(i)=p.F_drive_max*(cos(p.w_d*(t(i)-p.
                    t_force_adjust)+pi));               %Driving
                    force v. Time, Newtons
652             p.dP_piston = P(i) - P_cv2(i);
653             F_drive(i)=F_drive_clean(i) - p.dP_piston*p.Ap*1000;
654             [ dV_tmp(i+1),V_tmp(i+1),x_piston(i+1),x_dot_piston(
                    i+1),theta(i+1),theta_dot(i+1),x_piston_m(i+1),
                    theta_dot_dot(i+1) ] = vibration( t(i),dP(i),dx(
                    i),x_piston(i),x_dot_piston(i),theta(i),
                    theta_dot(i),x_piston_m(i),theta_dot_dot(i),
                    mu_oil,p,x_dot_piston_avg );
655             %%%%%%%%%%%
656             %% Loop Adjustments
657             %%%%%%%%%%%
658             %Section determines when the piston crosses back
659             %through the midpoint, signifying one cycle.
660             if q>1
661                 if i>1
662                     t_cross=find_cross(t',x_piston_m(1:end-1)
                            ',0);
663                 end
664             else
665                 if i>1
666                     t_cross=find_cross(t',x_piston_m(1:end-1)
                            ',0);
667                 end
668             end
669             t(i+1)=t(i)+p.t_step;
670             i=i+1;
671              if length(t)>=p.n*p.n_period*1.5
672                 t_cross=ones(2*p.n_period);
673                 disp('Convergence Error, piston did not cross
                        zero twice')
674              end
675         end  %end cycle through loop one time
676         i=i-1;

            %step i back one for sanity
```

```
677                 t=t(1:end-1);
                                                                %step
                    time vector back one
678                 temp(q)=max(t_cross)-max(find_cross(t',F_drive_clean',0)
                    );
679                 p.t_force_adjust=p.t_force_adjust+temp(q);
                                %when t_force_adjust is greater
                    than 3, stop i loop
680                 t_change(q)=p.t_force_adjust;
681                 p.Period=t(end)-t(1);
                                                        %floating
                    period
682                 if q < 100
683                     x_piston_err = abs(p.x_stroke - (max(x_piston_m) -
                        min(x_piston_m)));  %stroke error
684                 else
685                     x_piston_err = 0;
686                 end
687                 p.x_stroke = max(x_piston_m) - min(x_piston_m);  %update
                     stroke
688                 x_dot_piston_avg = rms(x_dot_piston); %average piston
                    speed
689                 %inital conditions for next loop equal to the extra
                    index in
690                 %current
691                 x_piston(1) = x_piston(end);
692                 x_dot_piston(1) = x_dot_piston(end);
693                 x_piston_m(1) = x_piston_m(end);
694                 theta(1) = theta(end);
695                 theta_dot(1) = theta_dot(end);
696                 theta_dot_dot(1) = theta_dot_dot(end);
697                 %concate vectors to correct length
698                 n_new = length(t);
699                 x_piston = x_piston(1:n_new);
700                 x_dot_piston = x_dot_piston(1:n_new);
701                 x_piston_m = x_piston_m(1:n_new);
702                 theta = theta(1:n_new);
703                 theta_dot = theta_dot(1:n_new);
704                 theta_dot_dot = theta_dot_dot(1:n_new);
705                 if pc_flag == 0
706                     figure(30)
707                     plot(t,x_piston_m)
708                     hold on
709                 end
710                 x_res = strcat('x_res:  ',num2str(x_piston_err));
711                 disp(x_res)
712                 q=q+1;
713             end  %end steady state piston loop, q loop
714             if pc_flag == 0
715                 close(30)
716             end
717             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
718             %% Brents Method: Attempting to find the correct power input
                    to
719             %% obtain full stroke
720             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
721             [p,error_stroke,a] = brents(p,a,error_brents);
722             if p.brent == 1
723                 P_electric_save(l) = p.P_brent(a);
724                 p.P_electric = p.P_brent(a);
725                 p.x_stroke_brent
726                 if a == 20
727                     error_brents = 0.000005; %50 micron
728                 elseif a == 60
729                     error_brents = 0.000001; % 1 micron
730                 elseif a == 70
731                     error_brents = 0.000005; % 5 microns
732                 elseif a==79
733                     error_brents = 0.00001; %10 microns
734                     %Find location of minimum error, re-run using that
735                     er=abs(p.x_stroke_brent-p.x_stroke_ref);
736                     er_min=min(er);
737                     ind_er=find(er==er_min);
738                     p.P_electric=p.P_brent(max(ind_er));
739                     P_electric_save(l)=p.P_brent(max(ind_er));
740                 elseif a==80
741                     error_brents = 1;
742                     %overwrite what brent's method calculates
743                     p.P_electric = p.P_brent(a-1);
744                     P_electric_save(l)=p.P_brent(a-1);
745                 elseif a>100
746                     error_brents= 1;
747                 end
748             end
749         end     %end a loop, brents method
750             %%%%%%%%%%%%%%
751             %% Post-Allocation, adjust for proper length
752             %%%%%%%%%%%%%%
753             P=P(1:length(t));
754             dT=dT(1:length(t));
755             h=h(1:length(t));
756             V=V(1:length(t));
757             dV=dV(1:length(t));
758             dm=dm(1:length(t));
759             dm_in=dm_in(1:length(t));
760             dm_out=dm_out(1:length(t));
761             drho=drho(1:length(t));
762             m=m(1:length(t));
763             rho=rho(1:length(t));
764             x_piston=x_piston(1:length(t));
765             x_dot_piston=x_dot_piston(1:length(t));
766             Cv=Cv(1:length(t));
767             dP_dT_v=dP_dT_v(1:length(t));
768             Ma=Ma(1:length(t));
```

```
T=T(1:length(t));
x_valve=x_valve(1:length(t));
x_dot_valve=x_dot_valve(1:length(t));
dm_leak_in=dm_leak_in(1:length(t));
dm_leak_out=dm_leak_out(1:length(t));
dm_out_valve=dm_out_valve(1:length(t));
dV_cv2=dV_cv2(1:length(t));
V_cv2=V_cv2(1:length(t));
Ma_cv2=Ma_cv2(1:length(t));
h_cv2=h_cv2(1:length(t));
Cv_cv2=Cv_cv2(1:length(t));
dP_dT_v_cv2=dP_dT_v_cv2(1:length(t));
P_cv2=P_cv2(1:length(t));
drho_cv2=drho_cv2(1:length(t));
rho_cv2=rho_cv2(1:length(t));
dT_cv2=dT_cv2(1:length(t));
T_cv2=T_cv2(1:length(t));
F_drive=F_drive(1:length(t));
Q_motor=Q_motor(1:length(t));
theta=theta(1:length(t));
theta_dot=theta_dot(1:length(t));
x_piston_m=x_piston_m(1:length(t));
dP=dP(1:length(t));
dx=dx(1:length(t));
theta_dot_dot=theta_dot_dot(1:length(t));
Q=Q(1:length(t));
Q_cv2=Q_cv2(1:length(t));
c_eff = c_eff(1:length(t));
c_friction = c_friction(1:length(t));
c_gas = c_gas(1:length(t));
x_dot_dot_piston = x_dot_dot_piston(1:length(t));
F_gas = F_gas(1:length(t));
T_w = T_w(1:k-1);
x_stroke_save = x_stroke_save(1:k-1);
loop = loop(1:k-1);
m_dot = m_dot(1:k-1);
m_dot_leak_in = m_dot_leak_in(1:k-1);
m_dot_leak_out = m_dot_leak_out(1:k-1);
W_dot = W_dot(1:k-1);
eta_o = eta_o(1:k-1);
eta_vol = eta_vol(1:k-1);
Q_dot = Q_dot(1:k-1);
Q_dot_cv2 = Q_dot_cv2(1:k-1);
m_dot_in = m_dot_in(1:k-1);
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Estimate Massflow, Power Input,Heat Transfer, and
     Efficiencies
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
816             % Estimating exit temperature and enthalpy
817             if m_dot(k-1) > 0
818                 %finding key volumes
819                 V_1=max(V);                                 %BDC
820                 V_3=min(V);                                 %TDC
821                 V_4=min(find_cross(V',P',p.P_i));   %Suction valve open
822                 V_2=max(find_cross(V',P',p.P_d));   %discharge valve
                        open
823                 %Find indexes of important volumes
824                 i_v1=find(V == V_1);
825                 i_v3=find(V == V_3);
826                 i_v2_test = crossing(V,[],V_2);  %a vector of length 2
827                 i_v4_test = crossing(V,[],V_4);  %a vector of length 2
828                 % test above to see which values correspond to the
                        correct
829                 % pressure
830                 for i=1:1:length(i_v2_test)
831                     diff_P2(i) = abs(P(i_v2_test(i))-p.P_d);
832                     diff_P4(i) = abs(P(i_v4_test(i))-p.P_i);
833                 end
834                 %indicies of the point where the difference between the
835                 %crossover pressure and actual pressure is at a minimum
836                 i_v2 = i_v2_test(find(diff_P2 == min(diff_P2)));
837                 i_v4 = i_v4_test(find(diff_P4 == min(diff_P4)));
838                 if i_v2 < i_v3  %does discharge cross the boundary?
839                     T_d = mean(T(i_v2:i_v3));
840                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
841                 else %yes
842                     T_d = mean(cat(2,T(i_v2:end),T(1:i_v3)));
843                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
844                 end
845                 %estimate exit temperature and enthalpy
846                 if T_d == 0
847                     T_d = mean(cat(2,find_cross(T',P',p.P_d),max(T)));
848                     h_2 = EOS(T_d,p.P_d,'enthalpy','P');
849                 end
850                 %Boundary work calculations
851                 W_2_3 = mean(P(i_v2:i_v3))*(V_2 - V_3);
852                 W_4_1=mean(cat(2,P(i_v4:end),P(1:i_v1)))*(V_1-V_4);
853                 W_1_2 = mean(mean(P(i_v1:end)))*(V_1 - V_2);
854                 W_3_4 = mean(P(i_v3:i_v4))*(V_4 - V_3);
855                 %Actual boundary work of real compression process
856                 W_boundary = W_1_2 + W_2_3 - W_3_4 - W_4_1;
857                 W_dot_boundary = W_boundary*f_list(l);
858                 %Idealized suction and discharge processes
859                 W_gas_dis_ideal = p.P_d*(V_2 - V_3);
860                 W_gas_suc_ideal = p.P_i*(V_1 - V_4);
861                 %Hybrid idealized boundary work (compression and
                        expansion
862                 %are real)
863                 W_gas_net_ideal = W_gas_dis_ideal + W_1_2 - W_3_4 -
                        W_gas_suc_ideal;
```

```
864                 W_dot_net_ideal = W_gas_net_ideal*f_list(l);
865                 %Valve Losses, based on differences between ideal valves
                        and
866                 %real valves, in W
867                 W_dot_suc_loss = 1000*(-W_4_1 + W_gas_suc_ideal)*f_list(
                        l);
868                 W_dot_dis_loss = 1000*(W_2_3 - W_gas_dis_ideal)*f_list(l
                        );
869             else
870                 T_d = 0;
871                 h_2 = 0;
872                 h_2_calc = 0;
873                 W_dot_suc_loss = 0;
874                 W_dot_dis_loss = 0;
875             end
876             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
877             %% Losses and power calculations
878             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
879             %Power in the piston shaft, energy integrated over one
                    period
880             W_dot_shaft = 0.5*p.M_mov*x_dot_piston.^2*f_list(l);
881             W_dot_shaft_ave = sum(W_dot_shaft)/length(W_dot_shaft);
882             %Power in the rotation of piston shaft, energy integrated
                    over
883             %one period
884             W_dot_shaft_rotation = 0.5*p.J*theta_dot.^2*f_list(l);
885             W_dot_shaft_rotation_ave = sum(W_dot_shaft_rotation)/length(
                    W_dot_shaft_rotation);
886             %Power consumed by friction
887             W_dot_friction = p.f_friction*F_wall*2*p.x_stroke*f_list(l);
888             W_dot_friction_ave=mean(W_dot_friction);
889             %Total power consumed by compressor, moving from compression
890             %chamber to motor
891             W_dot_total_ave = W_dot_shaft_ave + W_dot_shaft_rotation_ave
                        + W_dot_friction_ave + p.Q_motor;
892             %Maximum power stored in the mechanical springs
893             W_dot_stored_max = 0.5*p.k_mech*p.x_stroke^2*f_list(l);
894             %Net compressor power into the gas
895             W_dot(k-1)=1000*m_dot(k-1)*(h_2-p.h_in); %W_dot is in W
896             %Efficiency estimations
897             eta_o(k-1)=(m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/(p.P_electric
                    *2);
898             eta_o_test = (m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/
                    W_dot_total_ave;
899             eta_is = (p.h_2_s - p.h_in)/(h_2 - p.h_in);
900             %Net leakage losses
901             W_dot_leakage_loss = W_dot_shaft_ave - W_dot(k-1) - Q_dot(k
                    -1) - W_dot_suc_loss - W_dot_dis_loss;
902             W_dot_leak = 1000*mean(m_dot_leak_out(end)*h - m_dot_leak_in
                    (end)*h_cv2);
903             %Linear additions
904             W_dot_in_linear = mean(F_drive_clean.*x_dot_piston);
```

```
905             eta_o_linear = (m_dot(k-1)*(p.h_2_s-p.h_in)*1000)/
                    W_dot_in_linear;
906             %% Other Estimations
907             x_mag(l)=p.x_stroke/p.x_max; %#ok<SAGROW>
908             x_stroke_resonant_save(l) = (max(x_piston_m)-min(x_piston_m
                    )); %#ok<SAGROW>
909             f_list_resonant_save(l) = f_list(l); %#ok<SAGROW>
910             W_gas=((p.gamma*p.P_i*(max(V) - p.V_dead_valve))/(p.gamma-1)
                    )*(((p.P_d/p.P_i)^((p.gamma-1)/p.gamma))-1); %kJ
911             W_dot_gas = W_gas*f_list(l)*1000;
912         %% Calculating resonant frequency
913         if l>1
914             change_in_stroke = x_stroke_resonant_save(l) -
                    x_stroke_resonant_save(l-1);
915             if change_in_stroke < 0
916                 number_change_stroke = number_change_stroke +1;
917             end
918             if number_change_stroke < 3
919                 p.resonant_loop = 0;
920                 f_resonant_calculated = 0;
921                 w_d_resonant_calculated = 0;
922             elseif number_change_stroke >= 3      % is stroke decreasing
923                 p.resonant_loop = 1;
924                 if length(x_stroke_resonant_save) < 5
925                     f_resonant_calculated = 99;
926                     w_d_resonant_calculated = 99;
927                 elseif length(x_stroke_resonant_save) <= 15 && length(
                        x_stroke_resonant_save) >=5
928                     coefficients = polyfit(f_list_resonant_save,
                        x_stroke_resonant_save,2);
929                     f_resonant_calculated = -coefficients(2)/(2*
                        coefficients(1));
930                     w_d_resonant_calculated = f_resonant_calculated*2*pi
                        ;
931                 else
932                     x_stroke_resonant_save = x_stroke_resonant_save(end
                        -14:end);
933                     f_list_resonant_save = f_list_resonant_save(end-14:
                        end);
934                     coefficients = polyfit(f_list_resonant_save,
                        x_stroke_resonant_save,2);
935                     f_resonant_calculated = -coefficients(2)/(2*
                        coefficients(1));
936                     w_d_resonant_calculated = f_resonant_calculated*2*pi
                        ;
937                 end
938             end
939         else
940             f_resonant_calculated = 0;
941             w_d_resonant_calculated = 0;
942             number_change_stroke = 0;           %initialize variable
943             p.resonant_loop = 0;
```

```
944         end
945         %% Saving Important parameters
946         if pc_flag == 0 %&& p.resonant_loop == 1
947             d=date;
948             save_data={d,char(p.save_name),z,p.w_d,f_list(l),(max(
                    x_piston_m)-min(x_piston_m)),W_dot(k-1),...
949                 m_dot(k-1),m_dot_leak_out(k-1),p.P_i,p.P_d,p.T_i,length(
                        t),char(p.method),2*p.P_electric,...
950                 p.eta_motor,p.k_mech,p.f_friction,p.ecc_1,x_mag(l),p.g,
                        eta_o(k-1),eta_vol(k-1),T_d,...
951                 T_w(k-1),k-1,Q_dot(k-1),Q_dot_cv2(k-1),p.P_d/p.P_i,res_T
                        (k-1),res_rho(k-1),res_T_cv2(k-1),res_rho_cv2(k-1)
                        ,...
952                 res_x(k-1),0,0,p.M_mov,(max(x_piston_m)-min(x_piston_m))
                        ,p.P_electric,f_resonant_calculated,
                        w_d_resonant_calculated,p.k_eff,eta_is,...
953                 eta_o_test,W_dot_shaft_ave,W_dot_shaft_rotation_ave,
                        W_dot_friction_ave,W_dot_total_ave,W_dot_stored_max,
                        W_dot_suc_loss,W_dot_dis_loss,...
954                 W_dot_leakage_loss,W_dot_leak,mean(x_piston_m),p.
                        V_dead_valve,a,error_stroke,eta_o_linear,
                        W_dot_in_linear};
955
956             range_2=strcat('A',num2str(l+rows),':','BG',num2str(l+rows))
                    ;
957             xlswrite1('data_log.xls',save_data,range_2)
958             save_name = strcat('wksp_',p.save_name);
959             save_name = char(save_name);
960             save(save_name)
961         elseif pc_flag == 1
962             save_data_tmp = {z+row_start-1,p.w_d,f_list(l),(max(
                    x_piston_m)-min(x_piston_m)),W_dot(k-1),...
963                 m_dot(k-1),m_dot_leak_out(k-1),p.P_i,p.P_d,p.T_i,length(
                        t),2*p.P_electric,...
964                 p.eta_motor,p.k_mech,p.f_friction,p.ecc_1,x_mag(l),p.g,
                        eta_o(k-1),eta_vol(k-1),T_d,...
965                 T_w(k-1),k-1,Q_dot(k-1),Q_dot_cv2(k-1),p.P_d/p.P_i,res_T
                        (k-1),res_rho(k-1),res_T_cv2(k-1),res_rho_cv2(k-1)
                        ,...
966                 res_x(k-1),0,0,p.M_mov,(max(x_piston_m)-min(x_piston_m))
                        ,p.P_electric,f_resonant_calculated,
                        w_d_resonant_calculated,p.k_eff,eta_is,...
967                 eta_o_test,W_dot_shaft_ave,W_dot_shaft_rotation_ave,
                        W_dot_friction_ave,W_dot_total_ave,W_dot_stored_max,
                        W_dot_suc_loss,W_dot_dis_loss,...
968                 W_dot_leakage_loss,W_dot_leak,mean(x_piston_m),p.
                        V_dead_valve,a,error_stroke,eta_o_linear,
                        W_dot_in_linear};
969
970             save_data(z,:) = cell2mat(save_data_tmp);
971             save_name = strcat('data_log_',p.save_name);
972             wksp_name = strcat('wksp_',p.save_name);
```

```
973             save_name = char(save_name);
974             xlswrite(save_name,save_data);
975             save(char(wksp_name))
976         end
977         k=1;
978         dT_loop=1;
979         drho_loop=1;
980         dT_cv2_loop=1;
981         drho_cv2_loop=1;
982         l=l+1;              %Resonant loop counter update
983     end     %End of freq sweep while loop, l variable
984         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
985         %% Diagrams
986         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
987         if pc_flag == 0
988             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
989             % P-V Plot
990             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
991             figure(21)
992             %set(gcf,'Visible','off')
993             handles.plot(21)=plot(V,P); title('Pressure v. Volume');
994             ylabel('Pressure (kPa)');
995             xlabel('Volume (m^3)');
996             handles.figure(21)=gcf;
997             handles.axis(21)=gca;
998             %%%%%%%%%%%%%%%%%%%%%%%%%%%%
999             % Diagnostic Plots
1000            %%%%%%%%%%%%%%%%%%%%%%%%%%%%
1001            figure(22)
1002            set(gcf,'Visible','off')
1003            handles.plot(22)=plot(Q_dot); title('Heat Transfer v.
                    Iteration');
1004            ylabel('Heat Transfer (kW)');
1005            xlabel('Iteration');
1006            handles.figure(22)=gcf;
1007            handles.axis(22)=gca;
1008
1009            figure(23)
1010            set(gcf,'Visible','off')
1011            handles.plot(23)=plot(Q_dot_cv2); title('Heat Transfer v.
                    Iteration - CV2');
1012            ylabel('Heat Transfer (kW)');
1013            xlabel('Iteration');
1014            handles.figure(23)=gcf;
1015            handles.axis(23)=gca;
1016
1017            figure(24)
1018            set(gcf,'Visible','off')
1019            handles.plot(24)=plot(T_w); title('Shell Wall Temp v.
                    Iteration - CV2');
1020            ylabel('Wall Temp (K)');
1021            xlabel('Iteration');
```

```
1022            handles.figure(24)=gcf;
1023            handles.axis(24)=gca;
1024
1025            %Main subplot diagram
1026            figure(25)
1027            %set(gcf,'Visible','off')
1028            subplot(3,3,1),plot(t,T);title('Temperature v. Time');
1029            subplot(3,3,2),plot(t,P);title('Pressure v. Time');
1030            subplot(3,3,3),plot(t,dm);title('Massflow v. Time');
1031            subplot(3,3,4),plot(t,V);title('Volume v. Time');
1032            subplot(3,3,5),plot(t,V);title('Volume v. Time');
1033            subplot(3,3,6),plot(t,rho);title('Density v. Time');
1034            subplot(3,3,7),plot(t,dV);title('Change in Volume v. Time');
1035            subplot(3,3,8),plot(t,x_piston);title('Displacement v. Time'
                    );
1036            subplot(3,3,9),plot(t,drho);title('Change in Rho v. Time');
1037
1038            figure(26)
1039            %set(gcf,'Visible','off')
1040            %plot(t,dT_check);hold on;plot(t,dT,'r');legend('
                    dT_c_h_e_c_k','dm')
1041            subplot(2,2,1),plot(t,Ma);title('Mach Number v. Time')
1042            subplot(2,2,2),plot(t,x_valve);title('Valve Lift v. Time')
1043            subplot(2,2,3),plot(m_dot_leak_out);title('Leakage Outflow v
                    . Time')
1044            subplot(2,2,4),plot(m_dot);title('Massflow v. Iteration')
1045
1046            % CV2 Plots
1047            figure(27)
1048            %set(gcf,'Visible','off')
1049            subplot(3,3,1),plot(t,T_cv2);title('Temperature v. Time for
                    CV2');
1050            subplot(3,3,2),plot(t,P_cv2);title('Pressure v. Time for CV2
                    ');
1051            subplot(3,3,3),plot(t,dm_leak_out);title('Leakage Massflow
                    In v. Time for CV2');
1052            subplot(3,3,4),plot(t,V_cv2);title('Volume v. Time for CV2')
                    ;
1053            subplot(3,3,5),plot(t,rho_cv2);title('Density v. Time');
1054            subplot(3,3,6),plot(t,h_cv2);title('Enthalpy v. Time');
1055            subplot(3,3,7),plot(t,dT_cv2);title('Change in Temperature v
                    . Time for CV2');
1056            subplot(3,3,8),plot(t,Ma_cv2);title('Mach Number v. Time');
1057            subplot(3,3,9),plot(t,drho_cv2);title('Change in Rho v. Time
                     for CV2');
1058
1059                %% Shutting down activex server to excel
1060                invoke(Excel.ActiveWorkbook,'Save');
1061                Excel.Quit
1062                Excel.delete
1063                clear Excel
1064        end
```

```
1065          %% other stuff
1066          if pc_flag == 1
1067              disp(strcat('Batch Line No: ',num2str(z+row_start-1)))
1068          else
1069              disp(strcat('Batch Line No: ',num2str(z)))
1070          end
1071      end          %end of z loop (number of batch lines)
1072
1073      %turn annoying warning back on
1074      warning('on','MATLAB:Print:SavingToDifferentName')
1075      disp('finished')
1076      diary
1077  catch ME
1078      disp('finished w/ error')
1079      disp(ME.message)
1080      disp(ME.cause)
1081      disp(ME.stack)
1082      disp(ME)
1083      diary
1084      if pc_flag == 0
1085          %% Shutting down activex server to excel
1086          invoke(Excel.ActiveWorkbook,'Save');
1087          Excel.Quit
1088          Excel.delete
1089          clear Excel
1090          crash_name = strcat('crash_',save_name);
1091          save(crash_name)
1092      end
1093      if pc_flag == 1
1094          save_name = strcat('data_log_',p.save_name);
1095          save_name = char(save_name);
1096          crash_name = strcat('crash_',save_name);
1097          save(crash_name)
1098          if l>1
1099              xlswrite(save_name,save_data);
1100          end
1101      end
1102  %turn annoying warning back on
1103  warning('on','MATLAB:Print:SavingToDifferentName')
1104
1105  end
```

D.2 Function *vibration*

```
1  function [ dV,V,x_piston,x_dot_piston,theta,theta_dot,x_piston_m,
       theta_dot_dot,p ] = vibration( t,dP,dx,x_piston_1,x_dot_piston_1,
       theta_1,theta_dot_1,x_piston_m_1,theta_dot_dot_1,mu_oil,p )
2  %Vibration Model Component of Linear Compressor Model, added ver. 1.4.
3
4  if p.vibration_on==1
5
6      V=x_piston_1*p.Ap+p.V_dead_valve;           %Piston volume, when
           x_piston=0 volume is dead volume in valves.
7      dV=-x_dot_piston_1*p.Ap;        %x_piston and x_piston_m are
           opposites, x_dot_piston follows x_piston_m but I want it to
           follow x_piston, hence the negative.
8      p.theta=theta_1;
9
10     %Linearized vibration model flag
11     linear = 0;
12
13     if linear == 1
14         theta_tmp = atan(p.g/p.L_1);
15         A_p = pi*p.D_piston*p.x_stroke;
16         p.F_wall = (mu_oil*A_p*x_dot_piston_avg)/p.g; %oil film damping
17         %k_gas adjusted for max stroke
18         k_gas=((p.P_d-p.P_i)*p.Ap*1000)/p.x_stroke_ref;  %Gas spring
               rate, linear estimate
19         p.V_max=p.Ap*p.x_stroke_ref;
20         W_gas=((p.gamma*p.P_i*p.V_max)/(p.gamma-1))*((p.P_d/p.P_i)^((p.
               gamma-1)/p.gamma)-1);          %Work done on gas in one cycle,
                kJ
21         c_friction=(4*p.f_friction*p.F_wall)/(p.w_d*p.x_stroke_ref*pi);
                %effective damping due to dry friction in cylinder
22         c_gas=(W_gas*1000)/(p.w_d*p.x_stroke_ref^2*pi);                %
               effecitve damping due to work done on gas, 1000 to convert
               work to J, (N-s)/m
23     else
24
25         if abs(theta_1) >= atan(p.g/p.L_1)  %if piston is in contact
               with wall, there is friction.
26             A_p = pi*p.D_piston*p.x_stroke;
27             p.F_wall = (mu_oil*A_p*x_dot_piston_1)/p.g; %oil film
                   damping
28         else
29             p.F_wall=0;
30             A_p = pi*p.D_piston*p.x_stroke;
31             p.F_wall = (mu_oil*A_p*x_dot_piston_1)/p.g;  %oil film
                   damping
32         end
33         p.V_max = p.Ap*p.x_stroke;
34         if p.dP_max>=(p.P_d-p.P_i)
35             k_gas=((p.P_d-p.P_i)*p.Ap*1000)/p.x_stroke;  %Gas spring
                   rate, linear estimate
```

```
36              W_gas = (-p.P_current*p.dV + 0.5*p.dV*dP)*1000;
37          elseif p.dP_max<(p.P_d-p.P_i)
38              k_gas=(p.dP_piston*p.Ap*1000)/p.x_stroke;  %Gas spring rate,
                    linear estimate
39              W_gas = (-p.P_current*p.dV + 0.5*p.dV*dP)*1000;
40          end
41          c_friction=(4*p.F_wall)/(p.w_d*p.x_stroke*pi);
42          %%effective damping due to dry friction in cylinder
43          c_gas=(W_gas)/(p.w_d*p.x_stroke^2*pi);
44          %%effecitve damping due to work done on gas, 1000 to convert
                work to J, (N-s)/m
45      end
46      k_eff=k_gas+p.k_mech;          %Effective Spring Rate, gas +
            mechanical springs
47      p.k_eff=k_eff;
48      c_eff=c_gas+c_friction;                              %Total effective
            damping, (N-s)/m
49      p.c_eff = c_eff;
50      p.c_friction = c_friction;
51      p.c_gas = c_gas;
52      %%%%%%%%%%%%%%%%%%%
53      %% Start - dx_vibration replacement
54      %%%%%%%%%%%%%%%%%%%
55      %Vector of Timesteps
56      t_combined = [t,t+0.5*p.t_step,t+0.5*p.t_step,t+p.t_step];
57      k_m = 4.6831;
58      F_drive_max=sqrt(p.eta_motor*p.P_electric)*(k_m);   %Maximum Motor
            force, in Newtons.
59      F_drive=F_drive_max*(cos(p.w_d*(t-p.t_force_adjust)+pi));   %Driving
             force v. Time
60
61      %Renaming displacements and angles for Runge-Kutta Calculations
62      x_1 = x_piston_m_1;
63      x_2 = x_dot_piston_1;
64      theta_1_calc = theta_1;
65      theta_2 = theta_dot_1;
66
67      %Replacing dx_vibration%
68      F_drive=F_drive_max*(cos(p.w_d*(t_combined(1)-p.t_force_adjust)+pi))
            ;                  %Driving force v. Time
69      if linear == 0
70          %F_drive = F_drive - p.dP_piston*p.Ap*1000;
71      end
72
73      num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
74      num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
75      dx=[x_2,num2,theta_2,num4];
76      k1 = p.t_step*dx;
77
78      %Step 2, update displacements and angles
79      x_1 = x_1+0.5*k1(1);
```

```
80      x_2 = x_2+0.5*k1(2);
81      theta_1_calc = theta_1_calc+0.5*k1(3);
82      theta_2 = theta_2+0.5*k1(4);
83      F_drive=F_drive_max*(cos(p.w_d*(t_combined(2)-p.t_force_adjust)+pi))
            ;
84
85      if linear == 0
86          %F_drive = F_drive - p.dP_piston*p.Ap*1000;
87      end
88
89      num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
90      num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
91      dx=[x_2,num2,theta_2,num4];
92      k2 = p.t_step*dx;
93
94      %Step 3, update displacements and angles
95      x_1 = x_1+0.5*k2(1);
96      x_2 = x_2+0.5*k2(2);
97      theta_1_calc = theta_1_calc+0.5*k2(3);
98      theta_2 = theta_2+0.5*k2(4);
99
100     %Replacing dx_vibration%
101     num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
102     num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
103     dx=[x_2,num2,theta_2,num4];
104     k3 = p.t_step*dx;
105
106     %Step 4, update displacements and angles
107     x_1 = x_1+k2(1);
108     x_2 = x_2+k2(2);
109     theta_1_calc = theta_1_calc+k2(3);
110     theta_2 = theta_2+k2(4);
111
112     F_drive=F_drive_max*(cos(p.w_d*(t_combined(4)-p.t_force_adjust)+pi))
            ;
113
114     if linear == 0
115         %F_drive = F_drive - p.dP_piston*p.Ap*1000;
116     end
117
118     num2=(1/p.M_mov)*(p.k_mech*p.ecc_1*theta_1_calc+F_drive-c_eff*x_2-(
            k_gas+p.k_mech)*x_1);
119     num4=(p.k_mech/p.J)*(x_1-p.ecc_1*theta_1)*p.ecc_1;
120     dx=[x_2,num2,theta_2,num4];
121     k4 = p.t_step*dx;
122
123     %%%%%%%%%%%%%%%%%%%%%%%%
124     %% End, dx_vibration replacement
125     %%%%%%%%%%%%%%%%%%%%%%%%
126
```

```
127       dx_total=[x_piston_m_1,x_dot_piston_1,theta_1,theta_dot_1]+(1/6)*(k1
              +2*k2+2*k3+k4);
128       x_piston_m=dx_total(1);
129       x_dot_piston=dx_total(2);
130       theta=dx_total(3);
131       theta_dot=dx_total(4);
132       theta_dot_dot=k1(4);
133       p.x_dot_dot_piston=k1(2);
134       x_piston=-x_piston_m+p.x_o;
135
136       if x_piston < 0                  %This second condition does not
              allow the piston to travel beyond the top of the cylinder
137           x_piston = 0;
138           x_piston_m=p.x_o;
139           x_dot_piston=0;
140       end
141
142       if abs(theta) > atan(p.g/p.L_1)          %Restricts the rotation to
              approx 5 deg.
143           if theta < 0
144               theta=-atan(p.g/p.L_1);
145               theta_dot=0;
146           elseif theta > 0
147               theta=atan(p.g/p.L_1);
148               theta_dot=0;
149           end
150       end
151
152   elseif p.vibration_on==0
153       x_piston=p.x_o*(cos(p.w_d*t+pi)+1);
154       V=p.Ap*x_piston+p.V_dead_valve;
155       dV=-p.Ap*p.x_o*p.w_d*sin(p.w_d*t+pi);
156       x_dot_piston=-p.x_o*p.w_d*(sin(p.w_d*t+pi));
157       theta=0;
158       theta_dot=0;
159       theta_dot_dot=0;
160       x_piston_m=x_piston-p.x_o;
161       p.F_wall=0;
162       p.theta=0;
163       p.k_eff=0;
164       p.c_eff = 0;
165       p.c_friction = 0;
166       p.c_gas = 0;
167       p.x_dot_dot_piston=0;
168   end
169
170   end
```

VITA

VITA

Mr. Craig R. Bradshaw received his Bachelor of Science in Mechanical Engineering from Purdue University in 2007. As part of the undergraduate honors option Craig studied the movement of fluids between two particles under the advisement of Prof. Carl Wassgren.

Craig began his PhD studies in January of 2008 at Herrick Labs and will finish in May 2012 with a dissertation titled 'A Miniature-Scale Linear Compressor for Electronics Cooling'. Craig's linear compressor research was funded by the Cooling Technology Research Center (CTRC). Craig has also completed a project for Hoosier Energy rural energy cooperative which examined a variety of waste heat recovery technologies. Craig was a 2011 recipient of the Ward A. Lambert Teaching Fellowship allowed Craig to teach Thermodynamics I during the Spring semester of 2012. Craig's research interests include compressor, expander, waste-heat recovery, and renewable energy conversion technologies. Craig also has a passion for educating and will pursue a career that generates knowledge and promotes education.