12-2011

Engineering Model to Calculate Mass Flow Rate of a Two-Phase Saturated Fluid Through An Injector Orifice

Brian J. Solomon
Utah State University

ENGINEERING MODEL TO CALCULATE MASS FLOW RATE OF A TWO-PHASE SATURATED FLUID THROUGH AN INJECTOR ORIFICE

by

Brian J. Solomon

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Aerospace Engineering

Approved:

Dr. Stephen A. Whitmore Major Professor	Dr. R. Rees Fullmer Committee Member

Dr. Donald Cripps
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011

Abstract

ENGINEERING MODEL TO CALCULATE MASS FLOW RATE OF A TWO-PHASE SATURATED FLUID THROUGH AN INJECTOR ORIFICE

by

Brian J. Solomon, Master of Science

Utah State University, 2011

Major Professor: Dr. Stephen A. Whitmore
Department: Mechanical and Aerospace Engineering

An engineering model is developed to calculate mass flow rate of nitrous oxide, a self-pressurizing saturated oxidizer commonly used in hybrid rocket motors. While use of N_2O in a self-pressurizing oxidizer system has its advantages, there also exists some disadvantages. N_2O cannot be accurately modelled using traditional ideal gas, compressible, or incompressible flow assumptions. To obtain accurate mass flow rate this one-dimensional analysis includes both incompressible fluid and homogeneous equilibrium mass flow rate models. Massflow calculations from the two models are independently weighted and summed to obtain representative two-phase mass flow rate. Fluid properties are iterated in time by keeping track of fluid enthalpy and are propagated across the injector using either isentropic or adiabatic assumptions. The model excellently predicts mass flow rates as verified by comparison to experimental cold flow data. The experimental conditions resulting from the test apparatus set-up produces fluid stratification and mixing effects that cannot be modelled by the algorithm as developed. Thus the run tank and temperature drops as predicted by the model are significantly larger than measured.

(84 pages)

Acknowledgments

I would like to thank my major professor, Dr. Stephen A. Whitmore for his guidance in the fields of rocketry and fluid dynamics. I learned a lot from him over the past several years. Shannon Eilers helped guide my project with his willingness to help me when I got lost, he has been very helpful in fielding my many questions. He was also instrumental in providing the experimental data used for model validation.

I would additionally like to thank my employer, ATK, for providing me with the financial means to complete this degree.

Finally, I would like to thank my wife, Megan, and two children, Scarlett and Benson. They have been very patient with me during the research and work involved in completing this project. I look forward to getting this behind me so that I can spent more time with them and less time with my computer.

Brian J. Solomon

Contents

List of Tables

List of Figures

Acronyms

CO_2	carbon dioxide
HEM	homogeneous equilibrium model
N_2O	nitrous oxide
NHNE	non-homogeneous non-equilibrium model

Nomenclature

χ_1	Orifice upstream quality
χ_o	Initial fluid quality
δ	Reduced density
$\dot{H}$	Total enthalpy flow rate
$\dot{m}_{HEM}$	Mass flow rate, Homogeneous Equilibrium Model (HEM)
$\dot{m}_{inc}$	Mass flow rate, incompressible fluid
$\dot{m}_{NHNE}$	Mass flow rate, Non-Homogeneous Non-Equilibrium Model (NHNE)
$\dot{m}_{venturi}$	Mass flow rate, venturi
κ	Non-equilibrium parameter
ϕ	Dimensionless Helmholtz energy
ϕ^o	Dimensionless Helmholtz energy, ideal gas part
ϕ^r	Dimensionless Helmholtz energy, residual part
ρ	Density
ρ^L	Liquid density
ρ^V	Vapor density
ρ_1	Orifice upstream density
ρ_2	Orifice downstream density
ρ_c	Critical density
ρ_o	Initial fluid density
τ	Inverse reduced temperature
τ_b	Bubble growth time
τ_r	Residence time of fluid in an injector
A_t	Venturi throat cross sectional area
A_c	Cross sectional area
A_{in}	Venturi inlet cross sectional area
C_d	Discharge coefficient

h^L	Liquid specific enthalpy
h^V	Vapor specific enthalpy
h_1	Orifice upstream specific enthalpy
h_2	Orifice downstream specific enthalpy
H_o	Initial fluid total enthalpy
h_o	Initial fluid specific enthalpy
L	Injector length
M_o,	Initial fluid mass
M_o^L	Initial liquid mass
M_o^V	Initial vapor mass
P	Pressure
P_t	Venturi throat fluid pressure
P_1	Orifice upstream pressure
P_2	Orifice downstream pressure
P_{ν_1}	Upstream fluid vapor pressure
P_{in}	Venturi inlet fluid pressure
R	Gas constant
s^L	Liquid specific entropy
s^V	Vapor specific entropy
s_1	Orifice upstream specific entropy
s_2	Orifice downstream specific entropy
s_o	Initial fluid specific entropy
T_1	Orifice upstream temperature
T_c	Critical temperature
T_o	Initial fluid temperature
V_{tank}	Tank volume
Z	Compressibility factor

Chapter 1

Introduction

The use of nitrous oxide (N_2O) as a rocket fuel dates back to one of modern rocketry's founding fathers, Robert H. Goddard. In one of his earliest patents, U.S. Patent 1,103,503 [1], Goddard described a rocket that would use N_2O and gasoline as fuels. In recent times N_2O has been used as an oxidizer in non-military suborbital hybrid rockets. Most notably, N_2O was used by Scaled Composites' experimental spacecraft named SpaceShipOne [2]. In 2004, SpaceShipOne won the Ansari X Prize after sending the first human to space via a privately funded spacecraft [3]. Going forward the Scaled Composites team has partnered with Virgin GALACTIC becoming the world's first space tourism company with initial flights planned for 2012. Like SpaceShipOne, SpaceShipTwo will use a hybrid rocket using N_2O as its oxidizer [4].

While N_2O does not offer as high as performance as cryogenic oxidizers like liquid oxygen (LOX) or non-saturated propellants like nitrogen tetraoxide (N_2O_4) or hydrogen peroxide (H_2O_2), nitrous oxide has several advantages that make it very competitive for hybrid applications. First, in contrast to N_2O_4 and H_2O_2, N_2O is non-toxic and can be handled without special precautions. Second, Unlike LOX, Nitrous Oxide is highly storable, and allows rocket systems to be loaded well far in advance of launch. Store-ability also presents significant advantages for in-space propulsion systems. Finally, N_2O is a self-pressurizing propellant, and this property can be used to reduce the complexity of the propellant delivery systems. Unlike other more widely used rocket propellants, N_2O exists as a saturated liquid at room temperature, and has a relatively high vapor pressure of 5.729 MPa at 293.15 K (830.9 psi at 68 °F). Using N_2O as a self-pressurizing propellant eliminates or reduces the need for complex, costly, and heavy pumps or other types of pressurization systems. In addition to simplification of a rocket propellant system design, N_2O also offers

safety improvements over other more highly energetic propellants. Though not without a history of lethal disasters, N_2O is non-toxic and relatively more forgiving than other more energetic rocket propellants. For these reasons, N_2O is preferred by commercial space flight operators and amateur rocket system designers.

While use of N_2O in a self-pressurizing oxidizer system has its advantages, there also exists some disadvantages. Unlike more energetic propellants, N_2O cannot be accurately modelled using traditional ideal gas, compressible, or incompressible flow assumptions. The model developed here will build upon earlier studies attempting to predict flow characteristics of self-pressurizing fluids for use in hybrid rockets. The two-phase flow model to be presented is developed for both nitrous oxide and carbon dioxide (CO_2). Clearly, due to its non-energetic nature CO_2 cannot be used as a rocket propellant; however, its saturation properties including density, vapor pressure, and viscosity are very similar to N_2O. Thus CO_2 makes a convenient test analog, and can be used to evaluate the accuracy of the developed two-phase flow models with reduced risk and cost. A detailed comparison of CO_2 and N_2O properties is presented in Table A.1.

Chapter 2

Literature Search and Review of Previous Work

In this section two methods of determining fluid properties of N_2O and CO_2 are presented from the literature. Several mass flow models are also presented and discussed. A combination of two of the presented models is critiqued and redefined.

2.1 Fluid Properties

Two methods are investigated for calculating the saturated fluid properties. The first method uses NIST Chemistry WebBook database [5]. NIST Webbook contains thermophysical properties for multiple fluids, including both N_2O and CO_2. The second method used to calculate CO_2 properties, is by Span and Wagner [6]. This model formulates the equation of state explicitly in terms of the Helmholtz free energy. A similar method developed by Dyer [7] based upon the model of Span [8] is used for N_2O fluid properties.

2.1.1 NIST CO_2 Fluid Properties

A table of saturated CO_2 fluid properties can be downloaded from NIST Webbook website in ASCII format. Data may be tabulated in increments of temperature or pressure between the triple point and critical point. Both liquid and vapor properties for either temperature or pressure are provided:

$$P, \rho^L, \rho^V, s^L, s^V, h^L, h^V = NIST_{WEBBOOK}(216.592K \cdots 304.1282K) \tag{2.1}$$

$$T, \rho^L, \rho^V, s^L, s^V, h^L, h^V = NIST_{WEBBOOK}(0.51795MPa \cdots 7.3773MPa) \tag{2.2}$$

Eqs. 2.1 and 2.2 are functions coded to return select saturated CO_2 properties for given temperatures or pressures between the triple and critical points respectively. In both Eqs.

2.1 and 2.2 ρ is the density, s is the specific entropy, and h is the specific enthalpy where the superscripts ^{L} and ^{V} represent the liquid and vapor properties respectively. In Eq. 2.1, P is pressure and in Eq. 2.2 T is the temperature. Identical functions are created for N_2O.

2.1.2 Helmholtz Energy Method

Span and Wagner present an equation of state that predicts CO_2 thermodynamic properties to within uncertainties of available experimental data. The fundamental equation given in Span is the dimensionless Helmholtz energy ϕ, which is split into two parts. The first part ϕ^o depends upon the ideal-gas behaviour and the second part ϕ^r takes into account the residual fluid behaviour.

$$\phi(\delta,\tau) = \phi^o(\delta,\tau) + \phi^r(\delta,\tau) \tag{2.3}$$

In Eq. 2.3, τ is the inverse reduced temperature, and δ is the reduced density, and are defined as:

$$\tau = T_c/T \tag{2.4}$$

$$\delta = \rho/\rho_c \tag{2.5}$$

where T_c and ρ_c are the critical temperature and density respectively.

There exists a relationship between the Helmholtz energy defined in Eq. 2.3 and the thermodynamic properties needed in this analysis. Specifically the fluid pressure, specific entropy, and specific enthalpy are related to the Helmholtz energy by:

$$\frac{P(\delta,\tau)}{\rho RT} = 1 + \delta\phi^r_\delta \tag{2.6}$$

$$\frac{s(\delta,\tau)}{R} = \tau(\phi^o_\tau + \phi^r_\tau) - \phi^o - \phi^r \tag{2.7}$$

$$\frac{h(\delta,\tau)}{RT} = 1 + \tau(\phi^o_\tau + \phi^r_\tau) + \delta\phi^o_\delta \tag{2.8}$$

where R is the gas constant and the subscripted ideal gas and residual Helmholtz energies are the following derivatives:

$$\phi^r_\delta = \frac{d\phi^r}{d\delta}, \quad \phi^o_\tau = \frac{d\phi^o}{d\tau}, \quad \phi^r_\tau = \frac{d\phi^r}{d\tau}, \quad \phi^o_\delta = \frac{d\phi^o}{d\delta} \tag{2.9}$$

Span further defines the ideal gas and residual parts of the Helmholtz energy and their derivatives. The equations become long summations with hundreds of coefficients and exponents. The final equations are all presented in Span and can also be found as coded for this paper in the CO_2 properties source code in Appendix B.2.

Equations presented by Span and Wagner [6] are compiled into an algorithm that returns the fluid properties of CO_2.

$$P, s^L, h^L = Helmholtz(\tau, \delta^L) \tag{2.10}$$

$$P, s^V, h^V = Helmholtz(\tau, \delta^V) \tag{2.11}$$

CO_2 pressure, specific entropy, and specific enthalpy as returned from the NIST Webbook, and Span-Wagner method are compared in Figures 2.1, 2.2, 2.3. The comparisons are essentially identical. Advantages to using Span-Wagner equations directly are: no need to interpolate between tabulated data sets, and properties outside of the saturation region are possible. Once again, the same is true for N_2O. The Webbbook comparisons are used primarily to validate the Span-Wagner equations as programmed by the author.

2.2 Mass Flow Models

The model developed here to predict mass flow rate through and orifice style injector will use a weighted average of two different mass flow rate models. The first being the traditional incompressible viscous fluid model and the second being the Homogeneous Equilibrium Model (HEM). Figure 2.4 depicts a cross section of an orifice with the upstream and downstream mass flow parameter locations defined. The dual model method is necessary for a saturated fluid like N_2O or CO_2 because they do not behave like an incompressible liquid

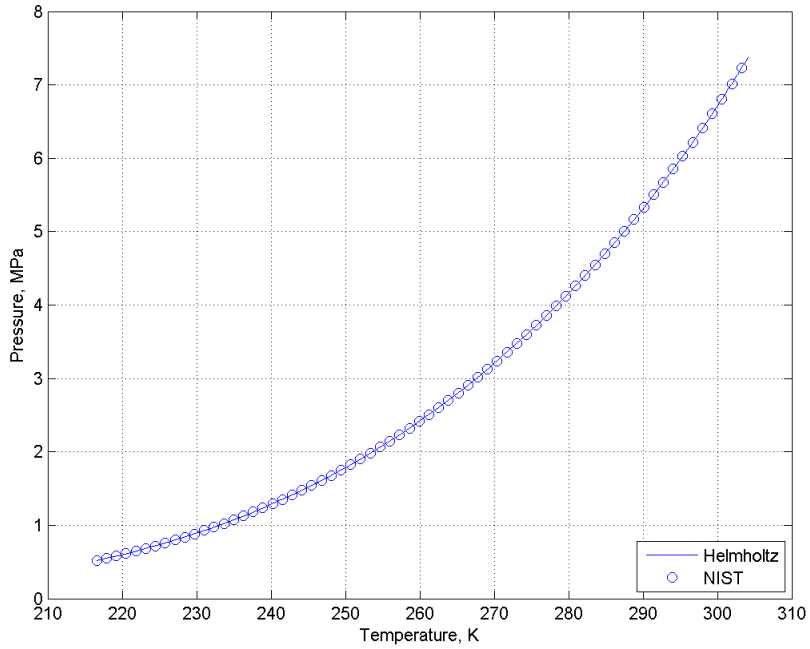

Fig. 2.1: Pressure of CO_2 as calculated by NIST Webbook and Helmholtz Models.

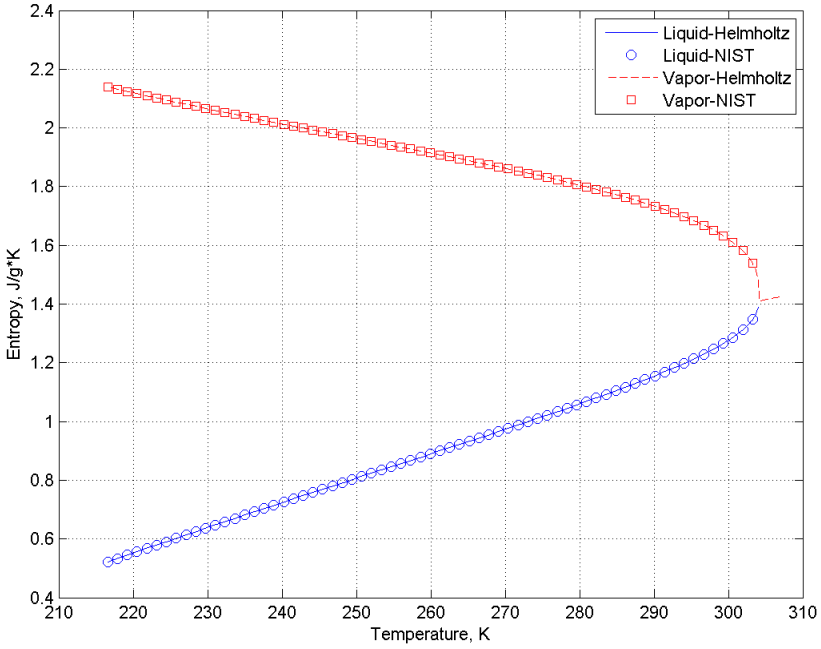

Fig. 2.2: Specific Entropy of CO_2 as calculated by NIST Webbook and Helmholtz Models.

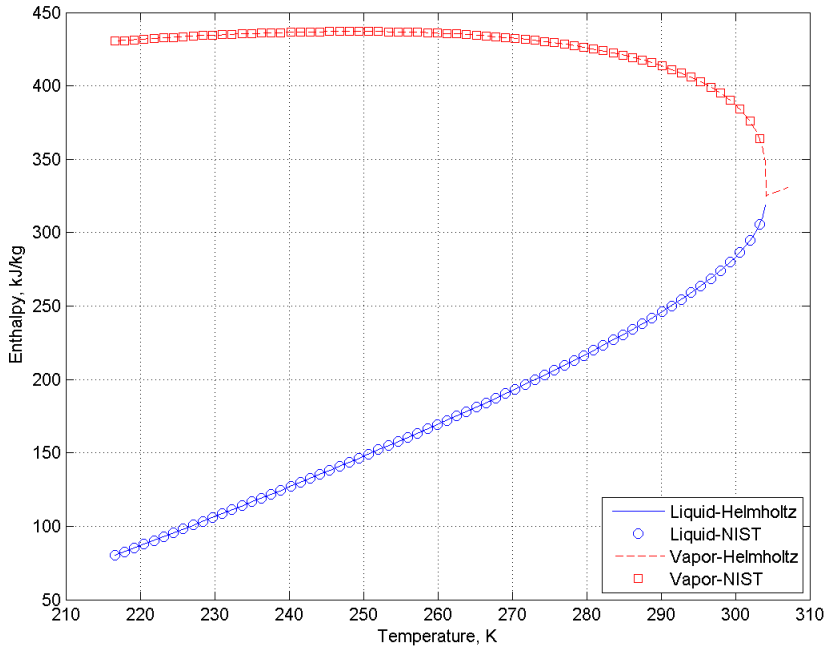

Fig. 2.3: Specific Enthalpy of CO_2 as calculated by NIST Webbook and Helmholtz Models.

or an ideal gas even at the saturated liquid and saturated vapor states. The compressibility factor, Z, for N_2O and CO_2 as calculated by [9]:

$$Z = \frac{P}{\rho RT} \tag{2.12}$$

are shown in Table 2.1. A fluid with a compressibility factor of either zero or one could use an incompressible liquid or ideal gas assumption. It is clearly seen by the compressibility factors of either N_2O or CO_2 that either of these assumptions would yield considerable errors in predicting mass flow rates.

Table 2.1: Saturated liquid and vapor compressibility factors for CO_2 and N_2O

Fluid	CO_2	N_2O
Saturated Liquid	0.16	0.135
Saturated Vapor	0.473	0.533

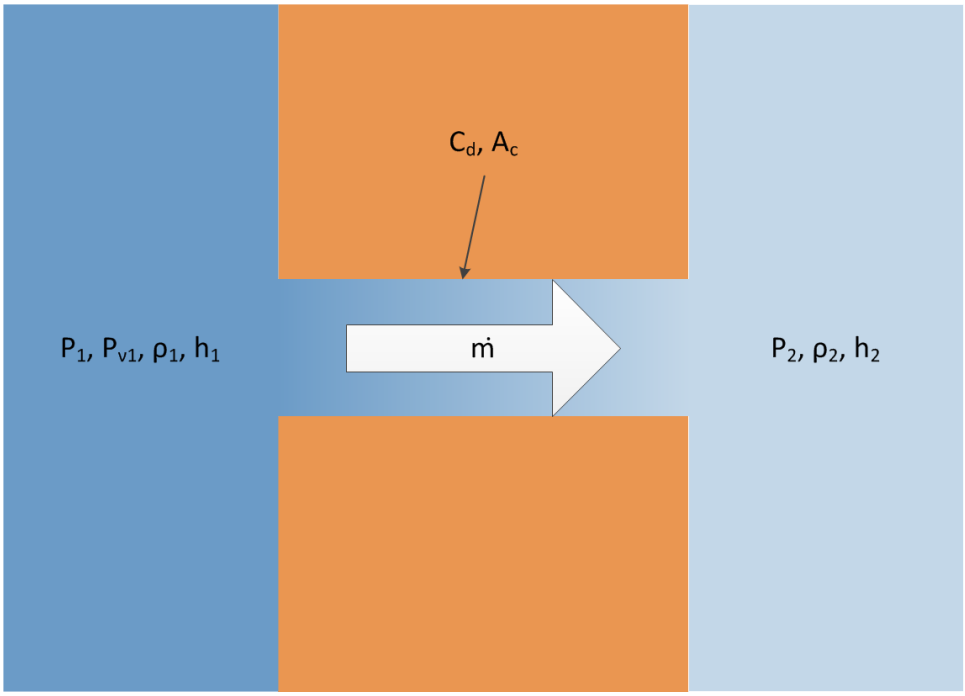

Fig. 2.4: Injector orifice cross section defining mass flow parameters.

2.2.1 Incompressible Viscous Fluid Model

For incompressible flows where fluid resides in the compressed liquid phase, mass flow rate, $\dot{m}_{inc}$, through an orifice or injector is defined as [10]:

$$\dot{m}_{inc} = C_d \cdot A_c \sqrt{2\rho_1(P_1 - P_2)} \tag{2.13}$$

where C_d is the orifice or injector discharge coefficient, A_c is the cross sectional area of the orifice, ρ_1 is the fluid density upstream of the orifice, and P_1 and P_2 are the fluid pressures upstream and downstream of the orifice respectively. This model accurately predicts the mass flow rate of a fluid with a compressibility factor near zero.

2.2.2 Homogeneous Equilibrium Model

For the flow cases of interest here, the static pressure of the fluid flowing into the injector is very close to the saturation state. As the flow begins to accelerate into the entrance of the injector, the pressure drops below the vapor pressure and the fluid begins to

flash from liquid to vapor. This results in a two-phase flow of vapor pockets or bubbles and liquid. One method proposed to predict mass flow of a two-phase flow through an injector is referred to as the Homogeneous Equilibrium Model (HEM) [11]. The HEM model assumes that the liquid-to-vapor phase change is isentropic, that the liquid and vapor portions of the flow are in thermodynamic equilibrium, and that there is no velocity difference between the fluid phases. The HEM mass flow rate $\dot{m}_{HEM}$, is a function of the injector discharge coefficient C_d, the injector cross sectional area A_c, the downstream fluid density ρ_2, and the upstream and downstream fluid specific enthalpies h_1 and h_2:

$$\dot{m}_{HEM} = C_d \cdot A_c \cdot \rho_2 \sqrt{2(h_1 - h_2)} \tag{2.14}$$

2.2.3 Non-Homogeneous Non-Equilibrium Model

Dyer et al. [7] have developed a model which combines the incompressible fluid model [Eq. 2.13] and the homogeneous equilibrium model (HEM) [Eq. 2.14] referred to here as non-homogeneous non-equilibrium (NHNE) model. The NHNE model weights incompressible fluid and HEM models using a "non-equilibrium" parameter. The non-equilibrium parameter, κ [Eq. 2.17], is defined in Dyer as the ratio of the bubble growth time, τ_b [Eq. 2.15], to residence time of the fluid in the injector element, τ_r [Eq. 2.16].

$$\tau_b \equiv \sqrt{\frac{3}{2}\frac{\rho^L}{P_{\nu_1} - P_2}} \tag{2.15}$$

$$\tau_r \equiv L\sqrt{\frac{\rho^L}{2 \cdot (P_1 - P_2)}} \tag{2.16}$$

$$k = \frac{\tau_b}{\tau_r} = \sqrt{\frac{P_1 - P_2}{P_{\nu_1} - P_2}} \tag{2.17}$$

Note the injector length L, a constants in τ_b and τ_r have been neglected by Dyer. Dropping the injector length and other constants has little effect on end results because κ is used to

weight the incompressible and HEM mass flow rates in an inversely proportional manner. The upstream fluid vapor pressure P_{ν_1}, is simply the total pressure for a saturated fluid. Dyer et al. have also noted κ is similar to the inverse of the cavitation number, except the denominator in κ is the difference between vapor pressure and downstream pressure rather than vapor pressure and upstream pressure, as in the cavitation number. The NHNE mass flow rate $\dot{m}_{NHNE}$, is defined here as:

$$\dot{m}_{NHNE} = C_d \cdot A_c \cdot \left(\frac{1}{1+\kappa} \dot{m}_{inc} + \left(1 - \frac{1}{1+\kappa} \right) \dot{m}_{HEM} \right) \tag{2.18}$$

2.2.4 NHNE Model Correction

An inconsistency between description of Dyer et al. and the mass flow rate equation 2.18 exists. In the text Dyer et al. states:

> "If the bubble growth time, τ_b, is large compared with the liquid residence time, τ_r, the flowrate should be well predicted by the classic incompressible CdA equation (Eq. 25). If, on the other hand, τ_r is much larger than τ_b the flowrate should approach the critical flowrate as predicted by the HEM."

In the flowrate equation in Dyer, the weighting coefficients are presented as:

$$\text{Incompressible Coefficient} = \frac{1}{1+\kappa} \tag{2.19}$$

$$\text{HEM Coefficient} = 1 - \left(\frac{1}{1+\kappa} \right) \tag{2.20}$$

Figure 2.5 shows coefficients in Eq. 2.19 and Eq. 2.20 plotted against the non-equilibrium parameter, κ. It is clearly seen that for large κ and therefore large bubble growth time, τ_b, compared to liquid residence time, τ_r, HEM dominates and not the incompressible portion, as was stated in Dyer.

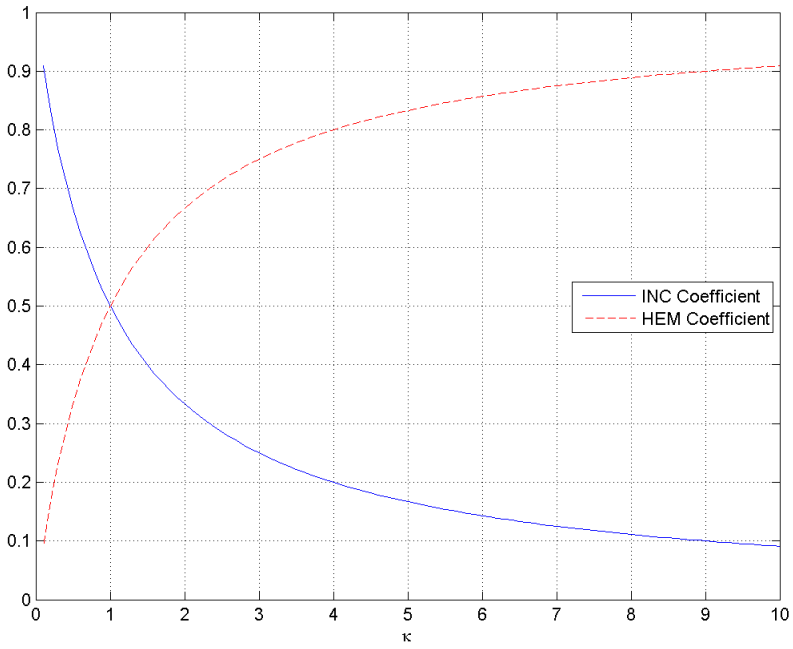

Fig. 2.5: Weighting coefficients vs non-equilibrium parameter κ, as presented in Dyer.

The description used by Dyer is physically consistent, in contrast to Dyer's original presentation of the NHNE massflow equation. Stating that liquid residence time, τ_r, is "large," is consistent with stating a specific mass of fluid takes a relatively long time to get through an injector. Stating that bubble growth time, τ_b, is "small," is consistent with stating a bubble develops quickly. When determining mass flowrate it is physically more consistent for a flow that develops bubbles quickly relative to the amount of time taken for a mass of fluid to get through an injector (early bubbly flow,) that flowrate is best predicted by the HEM model. Alternatively, it is physically more consistent for a flow that takes a long time for bubbles to develop relative to the amount of time it takes for a mass of fluid to get through the injector (late bubbly flow), that flowrate would be best predicted by the incompressible model. Based on this, it will be assumed Dyer's description was intended, and coefficients in the NHNE equation will be redefined as:

$$\dot{m}_{NHNE} = C_d \cdot A_c \cdot \left(\left(1 - \frac{1}{1+\kappa} \right) \dot{m}_{inc} + \frac{1}{1+\kappa} \dot{m}_{HEM} \right) \tag{2.21}$$

Chapter 3

Two-Phase Enthalpy Algorithm

A two-phase enthalpy algorithm is described in this section. The algorithm is designed to predict the mass flow rate of N_2O or CO_2 from an oxidizer storage tank through an injector. This algorithm is similar to that developed by Whitmore and Chandler [12]. The Whitmore and Chandler algorithm assumes an isentropic fluid expansion in order to propagate the fluid properties forward in time, giving it the "Two-Phase Entropy Model" name. The algorithm presented here will differ in that it will propagate the fluid properties forward in time by keeping track of the total fluid enthalpy rather than the entropy. The isentropic expansion across the injector is known to be physically incorrect. The assumption of an injector discharge coefficient already violates the isentropic flow assumption. The enthalpy model represents a more physically plausible extension of the entropy model. Figure 3.1 shows a top level flow chart of the major steps of the developed algorithm described here.

3.1 Initial Conditions

The algorithm initial conditions are derived by assuming an initial oxidizer tank volume V_{tank}, mass of oxidizer loaded into the tank M_o, and a temperature T_o at which the oxidizer is loaded. The fluid initial density ρ_o is calculated by assuming that the fluid in the tank is a uniform mixture of liquid (M_o^L) and vapor (M_o^V).

$$\rho_o = \frac{M_o^L + M_o^V}{V_{tank}} = \frac{M_o}{V_{tank}} \tag{3.1}$$

Since the initial saturated fluid temperature, and density, are known, the initial saturated pressure, P_o, the initial saturated liquid and vapor enthalpys, h_o^L and h_o^V, and

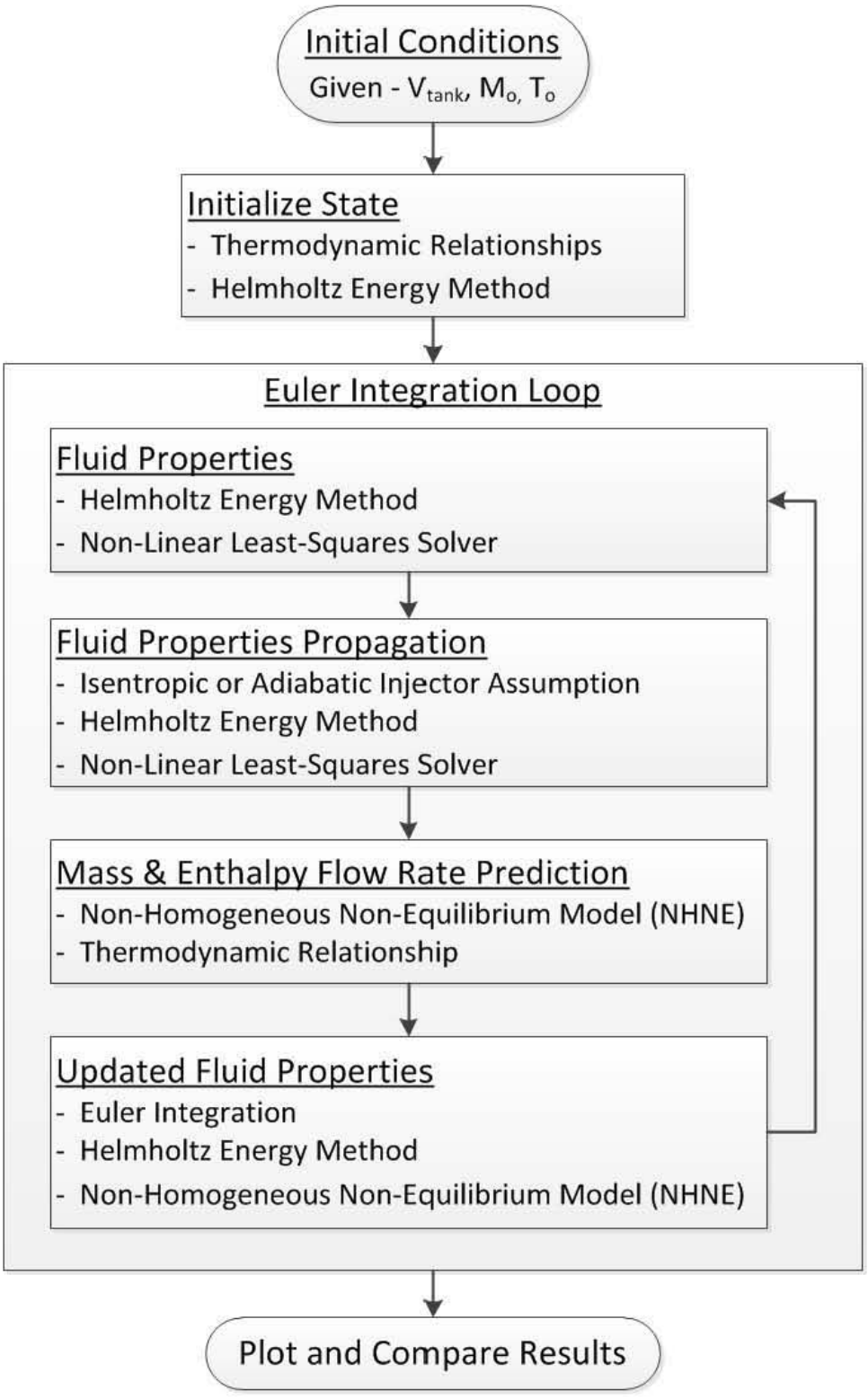

Fig. 3.1: Two-Phase Enthalpy algorithm flow chart.

entropies, s_o^L and s_o^V, are calculated using Eqs. 2.10 and 2.11. The initial fluid quality χ_o, and therefore initial specific enthalpy h_o, and specific entropy s_o, is calculated by:

$$\chi_o = \frac{\rho^V}{\rho_o} \cdot \frac{\rho^L - \rho_o}{\rho^L - \rho^V} \tag{3.2}$$

$$h_o = h_o^V \cdot \chi_o + h_o^L \cdot (1 - \chi_o) \tag{3.3}$$

$$s_o = s_o^V \cdot \chi_o + s_o^L \cdot (1 - \chi_o) \tag{3.4}$$

The total initial fluid enthalpy H_o, is calculated by multiplying the total initial fluid mass in the tank M_o, by the initial specific enthalpy h_o:

$$H_o = M_o \cdot h_o \tag{3.5}$$

3.2 Fluid Properties Calculation

Since the fluid state is propagated forward in time by tracking the total fluid enthalpy H, the new state must be calculated from the updated density ρ_{i+1}, and specific enthalpy h_{i+1}, after each time step. This calculation is problematic because the Span and Wagner Helmoltz energy equations [Eqs. 2.10 and 2.11] require temperature and density as inputs. To resolve this problem Eqs. 2.10, 2.11, 3.2, 3.3, and 3.4 were combined into a single function that returns pressure, quality, specific enthalpy, and specific entropy:

$$[P, \chi, h, s] = f(T, \rho) \tag{3.6}$$

Since the state of a fluid can be calculated by any two independent properties, a non-linear solver is used with Eq. 3.6 to obtain all fluid properties from any two. There are several instances where this technique is implemented in this algorithm. In the first case, both of the known properties are outputs of Eq. 3.6 with neither of the two inputs being known.

For this case Eq. 3.7 is solved with P_{known} and χ_{known} as knowns and everything else unknown:

$$\begin{bmatrix} P_{known}, \chi, h, s \\ P, \chi_{known}, h, s \end{bmatrix} = \begin{bmatrix} f(T, \rho) \\ f(T, \rho) \end{bmatrix} \tag{3.7}$$

To solve the problem presented in Eq. 3.7 a least-squares method is used. The MATLAB optimization toolbox contains a function called lsqnonlin designed to solve non-linear equations using a least-squares approach. In this case, the function defined in Eq. 3.6 minus the known tank pressure P_1, and quality χ_1, are input into the lsqnonlin function as shown here:

$$[T_1, \rho_1] = lsqnonlin\left(\begin{bmatrix} P(T, \rho) - P_1 \\ \chi(T, \rho) - \chi_1 \end{bmatrix}\right) \tag{3.8}$$

The lsqnonlin function returns the tank temperature T_1, and density ρ_1, that equate to the known pressure and quality. With the temperature and density known, the remaining properties are calculated using Eq. 3.6.

3.3 Fluid Property Propagation

The fluid properties are propagated across the injector to the downstream state using either an isentropic or adiabatic assumption. The isentropic case assumes that the flow across the injector is reversible, and therefore the downstream entropy s_2, is equal to the upstream entropy s_1, [Eq. 3.9]. The adiabatic case assumes no heat is lost or added to the fluid as it flows through the injector, and therefore the downstream enthalpy h_2, is equal to the upstream enthalpy h_1, [Eq. 3.10].

$$s_1 = s_2 \tag{3.9}$$

$$h_1 = h_2 \tag{3.10}$$

For the case where the injector is venting to ambient conditions, the downstream pressure is taken to be atmospheric. As was done for the upstream fluid properties in Eqs. 3.7 and

3.8, with the atmospheric pressure being known and the isentropic or adiabatic injector assumption, the downstream injector fluid properties are calculated by:

$$[T_2, \rho_2] = lsqnonlin\left(\begin{bmatrix} P(T,\rho) - P_2 \\ s(T,\rho) - s_2 \end{bmatrix}\right) \tag{3.11}$$

or

$$[T_2, \rho_2] = lsqnonlin\left(\begin{bmatrix} P(T,\rho) - P_2 \\ h(T,\rho) - h_2 \end{bmatrix}\right) \tag{3.12}$$

where the remaining downstream fluid properties are calculated by Eq. 3.6.

It should be noted that CO_2, due to it's high triple point pressure cannot exist in liquid phase at pressures below that of it's triple point pressure of 517.95 kPa. At any pressure below this value, the CO_2 flow would consist of solid-vapor mixture. It is for this reason that CO_2 exists as a "Dry Ice" at atmospheric pressure and temperature. For this analysis, the downstream injector pressure used is 85.9 kPa; the atmospheric pressure at the approximate altitude of the test facility located in Logan, Utah (4500 feet above sea level). Since the downstream pressure used is below the triple point temperature of CO_2, the fluid near the exit of the injector orifice is likely a solid gas mixture.

3.4 Mass Flow Rate Prediction

The injector mass flow rate is calculated using the NHNE model discussed earlier and defined in Eq. 2.21. From the mass flow rate, the total enthalpy flow rate is found to be:

$$\dot{H} = h_1 \cdot \dot{m}_{NHNE} \tag{3.13}$$

where $\dot{H}$, is the total enthalpy flow rate.

3.5 Update Fluid Properties

To update the total tank fluid mass M_{i+1}, and total tank fluid enthalpy H_{i+1}, an Euler integration method is used. The flow rates calculated in Eqs. 2.21 and 3.13, are multiplied

by the time step Δt and subtracted from the current total mass M_i, and total enthalpy H_i:

$$M_{i+1} = M_i - \dot{m}_{NHNE} \cdot \Delta t \tag{3.14}$$

$$H_{i+1} = H_i - \dot{H} \cdot \Delta t \tag{3.15}$$

The tank fluid total density and total specific enthalpy are updated from the new total fluid mass and total tank enthalpy:

$$\rho_{i+1} = M_{i+1}/V_{tank} \tag{3.16}$$

$$h_{i+1} = H_{i+1}/M_{i+1} \tag{3.17}$$

Once again the MATLAB lsqnonlin function is used to calculated the remaining fluid properties. In this case, one of each of the input and output properties of Eq. 3.6 are known. Equation 3.18 is solved using lsqnonlin with h_1 and ρ_1 as knowns with everything else unknown.

$$[P, \chi, h_{known}, s] = f(T, \rho_{known}) \tag{3.18}$$

$$[T_1] = lsqnonlin(\ \ h(T, \rho_1) - h_1\ \ ) \tag{3.19}$$

In this case, the lsqnonlin function in Eq. 3.19 returns the upstream temperature T_1, that equates to the known density and enthalpy. Example results are shown in Figure 3.2. Here the unknown temperature is plotted against the known total density for different specific enthalpies. All indices are moved forward in time, and the loop is iterated forward in time.

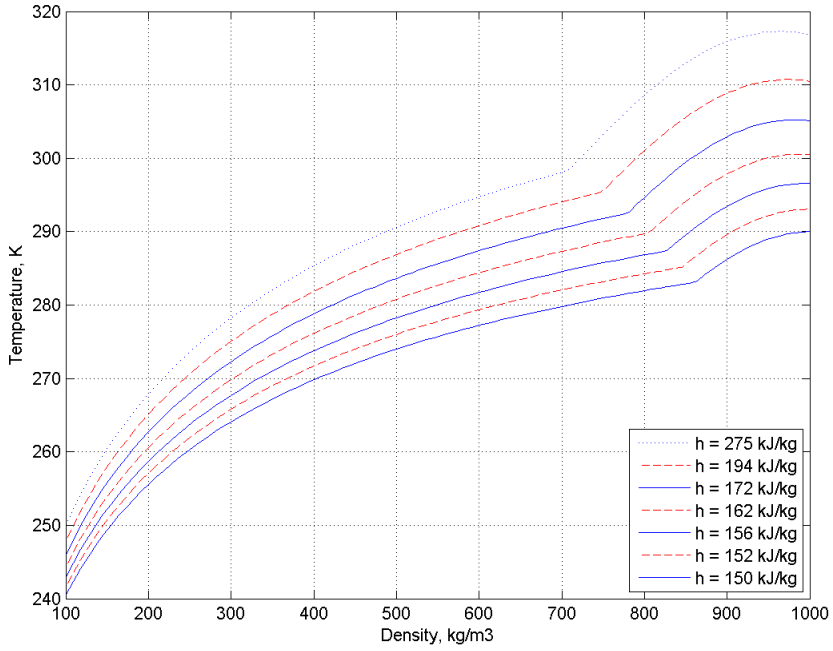

Fig. 3.2: CO_2 temperature as a function of density for different specific enthalpies.

Chapter 4

Experimental Validation

To validate the model developed here, it is compared against cold flow data derived from laboratory tests performed at Utah State University. Figure 4.1 shows a piping and instrumentation digram (P&ID) of the oxidizer flow test stand designed and built to facilitate hybrid rocket motor static firings. The stand has more capability than is shown here, including the ability to pressurize the run tank from the top with gaseous nitrogen or any other desired inert gas.

4.1 Experimental Apparatus

The run tank is constructed from a flat bottomed aluminium cylinder of size K with a total internal volume of 49.9 L (1.76 ft^3). The flat bottom is ported to accommodate a pressure transducer, vent valve, and relief valve. In the test configuration, the bottle has been turned upside down, fitted with clamp rings and hung by two s-beam load cells. Insulation has been wrapped around the bottle to help reduce heat transfer during blowdown. The main port on the tank is fitted with a tee so that a 0.125 inch diameter thermocouple probe can be inserted into the tank. The second port on the tee has a flex line attached to help obtain the best possible tank weight measurement from the load cells during blow down. The other end of the flex line is equipped with a pneumatically operated fill valve, used when filling the tank with fluid. After the fill valve there is a diverging-converging venturi fitted with two pressure transducers to measure mass flow rate. The first pressure transducer port leads to the inlet of the venturi prior to the divergent section, the second transducer port leads to the throat area of the venturi between the divergent and convergent sections. The outlet of the venturi flows into an additional pneumatically operated run valve, and finally into an orifice style injector.

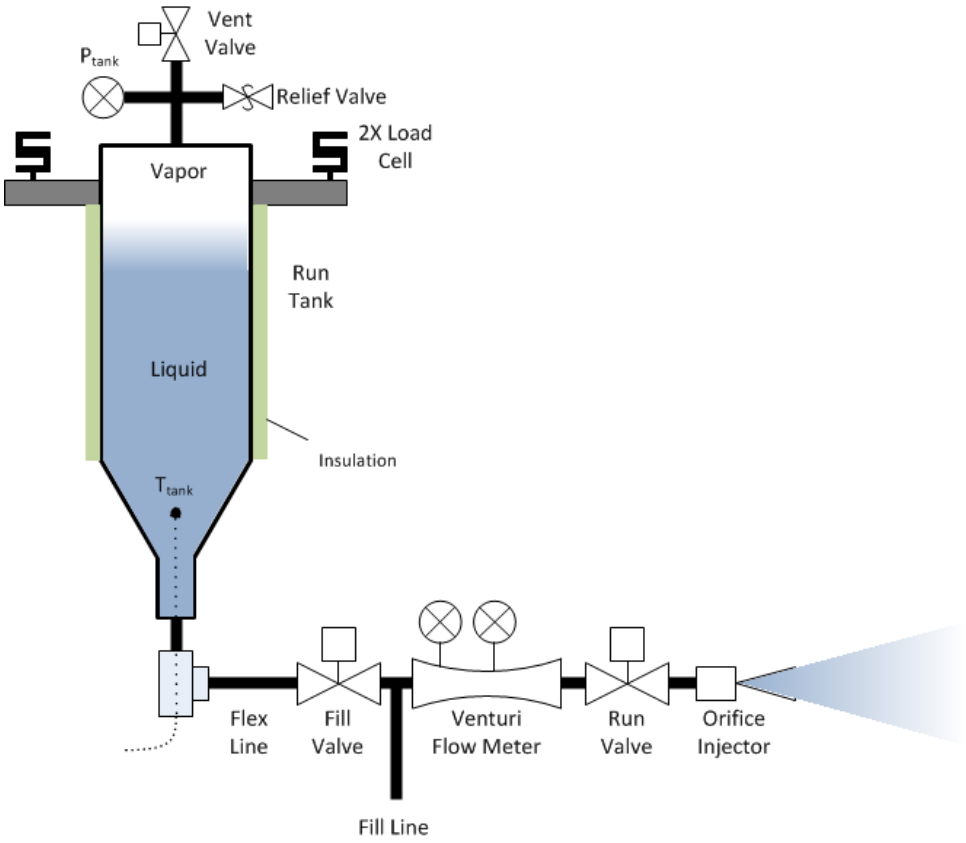

Fig. 4.1: Cold flow blowdown physical setup.

4.2 Experimental Procedures

To perform a cold flow blowdown test where the self-pressurized fluid is vented to ambient conditions, saturated CO_2 or N_2O is let into the tank through the fill valve. To prevent the saturated CO_2 or N_2O from flashing to vapor when first introduced into the tank, the tank and plumbing are pressurized with gaseous nitrogen to slightly above the saturation pressure of the CO_2 or N_2O. Additionally, the CO_2 or N_2O source tanks are pressurized with gaseous nitrogen prior to beginning the fill process. To achieve the highest total oxidizer fluid mass possible in the run tank, the tank should be filled with mostly liquid. To begin the filling process, the vent and fill valves are opened and the run tank is filled with saturated liquid from the bottom while gaseous nitrogen and CO_2 or N_2O vapor is vented from the top. If the pressure in the tank drops too quickly due to the venting, it is sometimes necessary to pulse the vent valve to minimize the amount of saturated CO_2 or N_2O that flashes to vapor. Once the run tank has been filled with the desired amount of fluid, the blowdown test is performed. With the vent valve closed, the run valve is opened and the fluid in the run tank is vented to ambient conditions through the single injector. As saturated liquid exits the run tank at the bottom, there is also saturated liquid flashing to saturated vapor in the upper portion of the tank. There reaches a point during the blowdown when all of the liquid has been blown out and the remainder of the blowdown vents vapor only.

The data gathered during a blow down that is applicable to this model is the run tank pressure, the fluid temperature in the bottom portion of the tank, the tank mass, and the venturi pressures. Since the pressure transducer is located on the port in the top of the tank, the pressure it measures is that of the warmest fluid within the tank. Neglecting the hydrostatic portion of the pressure, the pressure is constant throughout the tank. The temperature measurement however is likely not as reliable throughout the tank as the pressure measurement is. In fact, placing thermocouples at different locations along the length of the outside of the run tank shows that temperature stratification does occur after loading and during blowdown. Since the temperature is a measurement of the cooler liquid

in the tank bottom, and the pressure is a measurement of the warmer vapor in the tank top, picking one over the other to initialize the fluid state with, yields slightly different results. For convenience, the model presented here is initialized from the temperature measurement, for cases in which a pressure measurement initialization is desired, the initial temperature is increased to match that corresponding to the initial pressure. The temperature stratification could be minimized by allowing enough time for the fluid temperature to equalize to the ambient conditions prior to performing a blowdown test.

The venturi pressures are used to calculate the mass flow rate of fluid exiting the tank. From the load cell data, the mass of fluid in the tank is known throughout the blowdown process. The numerical derivative of the fluid mass is calculated to also obtain mass flow rate from the tank. This massflow estimate, although somewhat noisy, is used to support the venturi massflow measurements.

4.3 Model Comparisons

Figures 4.2, 4.3, 4.4, and 4.5 show typical blowdown test results and compare the sensed tank pressure, temperature and exit massflow to the NHNE model predictions. The model was initialized using the initial temperature measured in the tank as well as the initial fluid mass. From examination of figures 4.3 and 4.4 the initial conditions are apparent as the temperatures and fluid mass agree at time zero. At time zero the model calculates a fluid pressure from the initial density and temperature using equations 3.1 and 3.6. Examining figure 4.2 shows an almost 0.5 MPa difference between the calculated initial pressure and the measured initial pressure. From this, it is apparent that there is temperature stratification in the tank even before the blowdown has begun. The initial measured pressure is a measurement of the warmest vapor in the top of the tank, while the initial measured temperature is a measurement of the cooler liquid in the bottom of the tank. Further evidence of temperature stratification is seen by looking at the cold flow temperature data in figure 4.3. For the first few seconds of the blowdown, the measured temperature increases before it starts to decrease. This temperature rise suggests that the fluid in the top of the tank is warmer than in the fluid in the bottom. The opening of the run valve has a mixing effect

in the tank, pulling warmer fluid from the top of the tank towards the bottom, causing the measured temperature to rise. The NHNE model as developed cannot account for this temperature stratification and subsequent mixing.

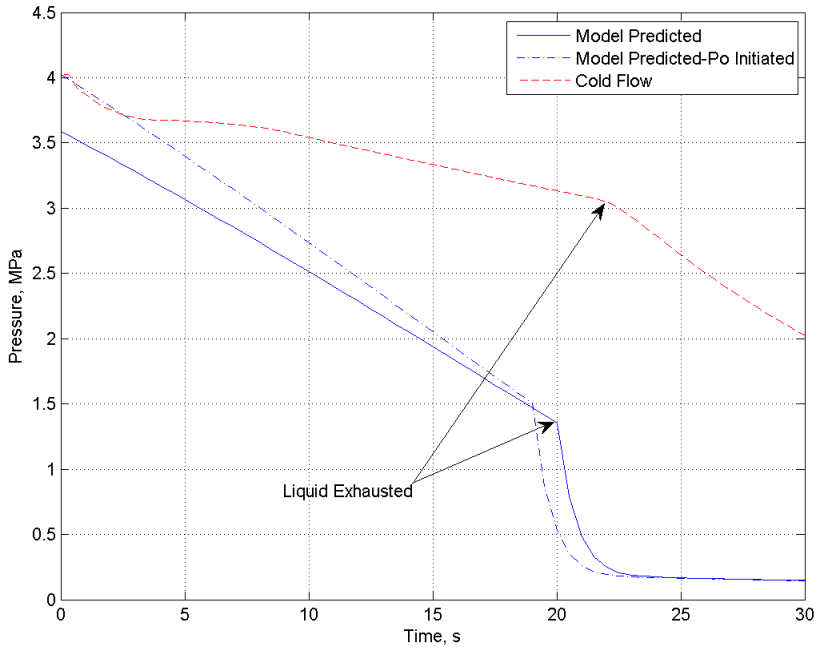

Fig. 4.2: Comparison, Model vs cold flow, run tank fluid pressure.

Examination of figures 4.4 and 4.5 show that the NHNE model does a good job of predicting the steady-state mass flow rate when compared to the incompressible and HEM models alone. The incompressible fluid model greatly over predicts the mass flow rate while the HEM model greatly under predicts it. The inflection points on each of the plots at approximately 20 seconds, represent the point during the blowdown where all of the liquid in the tank is exhausted and the flow transitions to purely vapor. Measured mass flow rates at the very beginning of blowdown is not predicted well by the model. This is because the plumbing between the run valve and injector is initially full of air. At time zero when the run valve is opened, air in the plumbing is forced out through the injector and fluid is flashing to vapor in the plumbing. Until fluid fills the plumbing between the run valve and injector, the venturi returned mass flow rate will be higher than predicted by the model.

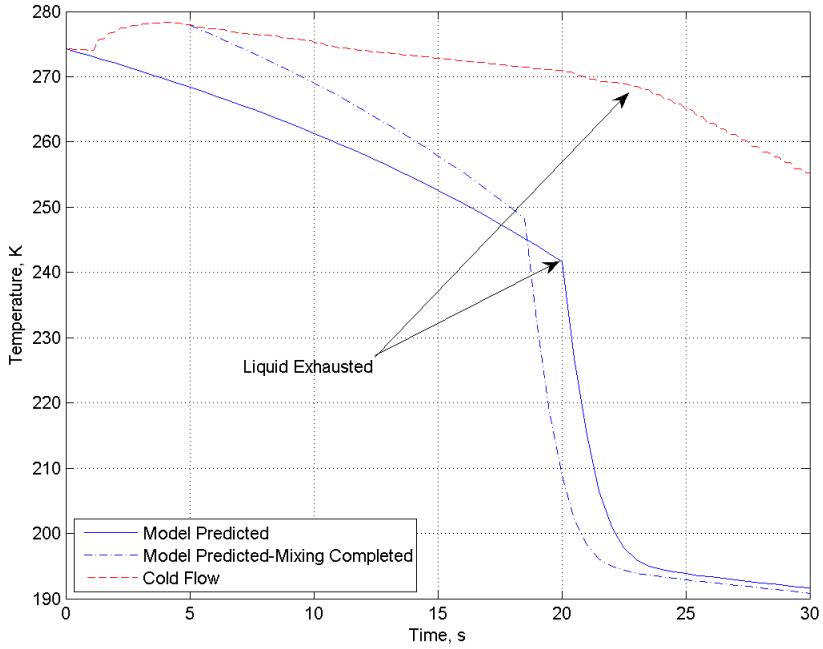

Fig. 4.3: Comparison, Model vs cold flow, run tank fluid temperature.

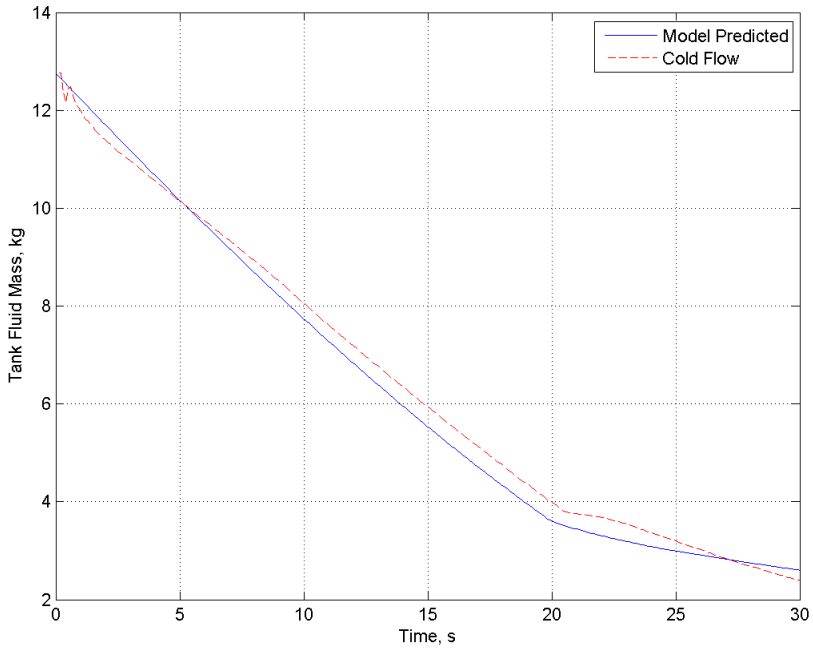

Fig. 4.4: Comparison, Model vs cold flow, run tank total fluid mass from load cells.

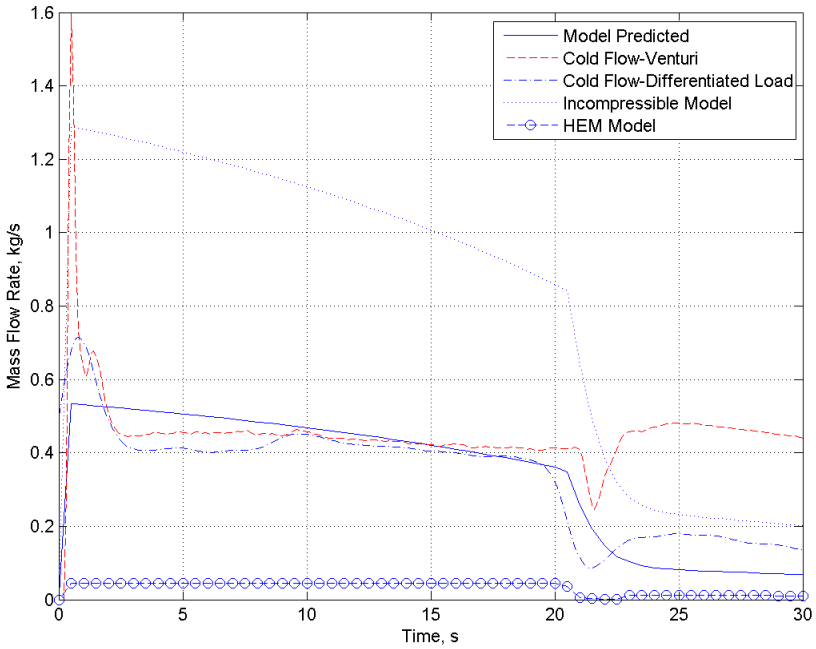

Fig. 4.5: Comparison, Model vs cold flow, fluid mass flow rate.

Additionally, the venturi measured mass flow rate as plotted, is not valid once the flow transitions from liquid to vapor. It can be seen in figure 4.5 that this is the case as the venturi mass flow rate starts to drop at the transition point, but then quickly starts giving erroneous values. This inaccuracy results from the equation used to calculate the venturi flow rate $\dot{m}_{venturi}$, from the venturi inlet pressure P_{in}, and venturi throat pressure P_t, is:

$$\dot{m}_{venturi} = A_t \cdot C_d \cdot \rho \cdot \sqrt{\frac{2(P_{in} - P_t)}{\rho(1 - (A_t/A_{in})^2)}} \tag{4.1}$$

In equation 4.1, P_{in} and P_t are the measured inlet and throat pressures, A_{in} and A_t are the inlet and throat cross sectional areas, C_d is the experimentally determined discharge coefficient, and ρ is the density of the fluid flowing through the venturi. For the cold flow data, ρ is calculated from a second order polynomial fit that is a function of the measured tank temperature:

$$\rho = -0.136929 \cdot T^2 + 68.960274 \cdot T - 7677.810286 \tag{4.2}$$

The curve fit method of Eq. 4.2 is used to significantly reduce the run time of the code

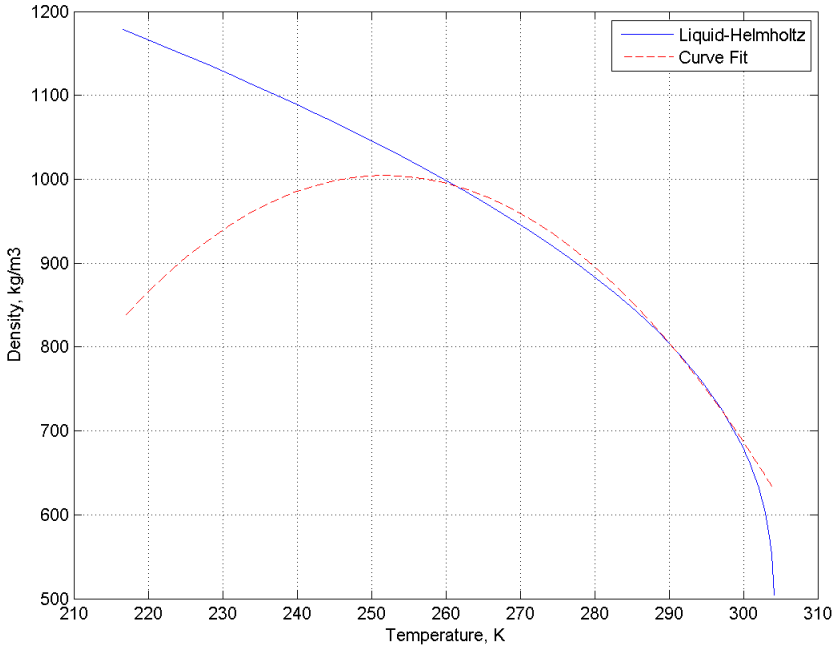

Fig. 4.6: CO_2 liquid density comparison.

developed to parse the cold flow data during repetitive analysis. Equation 4.2 is plotted against the previously shown Helmholtz liquid density in figure 4.6. Figure 4.6 shows that the curve fit polynomial does a good job in predicting the liquid density between 260 and 300 K, but deviates significantly once the temperature drops below 260 K. Thus as the liquid is depleted from the tank and the temperature drops, the venturi massflow measurements become increasingly inaccurate. For typical blowdown tests, where the liquid portion of the blowdown is what is of interest, the fluid temperature during this time stays between 260 and 300 K. Figure 4.7 depicts the venturi mass flow rate of the cold flow blowdown case presented here as calculated by Eq. 4.1, where the liquid density is calculated by both the Helmholtz energy method presented in Eq. 2.10 and the curve fit method of Eq. 4.2. The plot shows that for a typical blowdown test where the fluid temperature stays in the range of accuracy, that the curve fit polynomial does a good job of calculating the fluid liquid density.

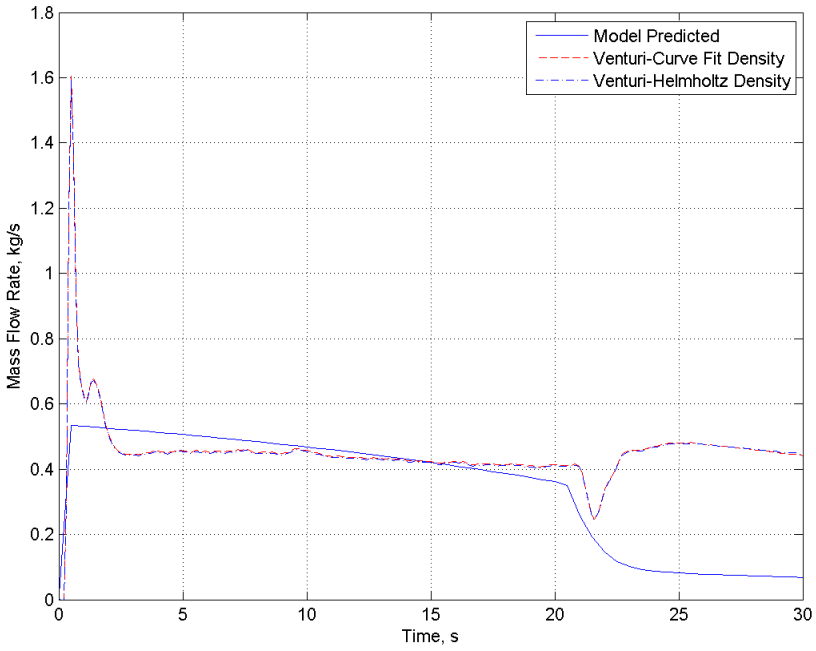

Fig. 4.7: Comparison, curve fit vs Helmholtz energy density, fluid mass flow rate.

In addition to venturi derived mass flow rate, the load cell data was numerically differentiated to obtain mass flow rate. Figure 4.8 depicts the tank fluid mass zoomed in time to the first five seconds of the blowdown. Within the first second, the load cell data decreases as expected, but then makes a sudden leap before continuing to decline. This unexpected mass increase is due to the tank slightly jumping on its load cells from the sudden rush of fluid from the bottom of the tank when the run valve is opened. This "noise" causes problems for the numerical differentiation. To minimise this, a differentiation method based on discrete wavelet transformations presented by Luo [13] is used. When a wavelet function is the derivative of a smoothing function, the wavelet transformation has the combined properties of smoothing and differentiation [14]. Figure 4.9 shows the improvement gained by using the wavelet transformation method as opposed to a simple difference approach. The wavelet method significantly improves the derived mass flow rate, and likely best captures the initially high mass flow rate caused by the plumbing volume filling with fluid.

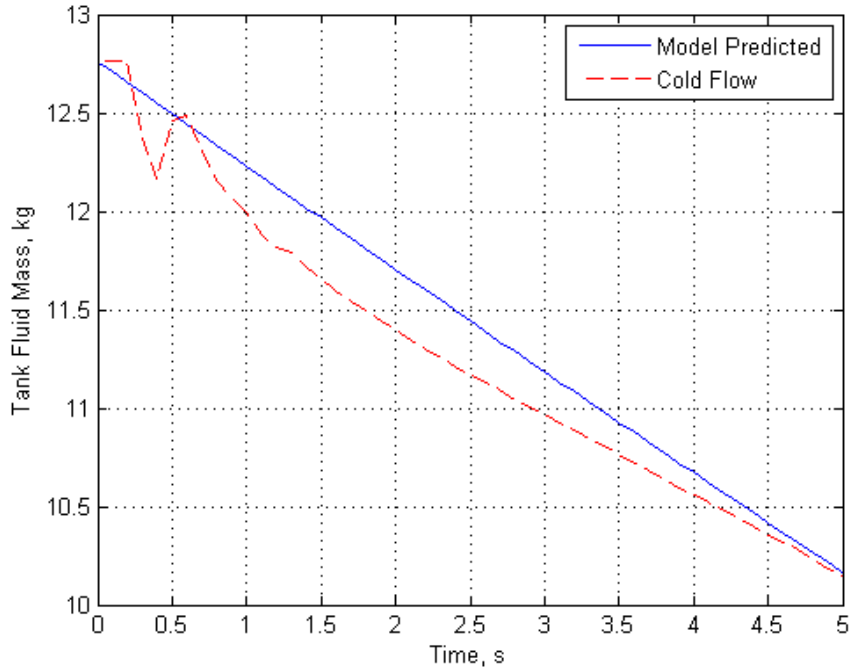

Fig. 4.8: Model vs cold flow, fluid mass flow rate comparison, detailed initial time.

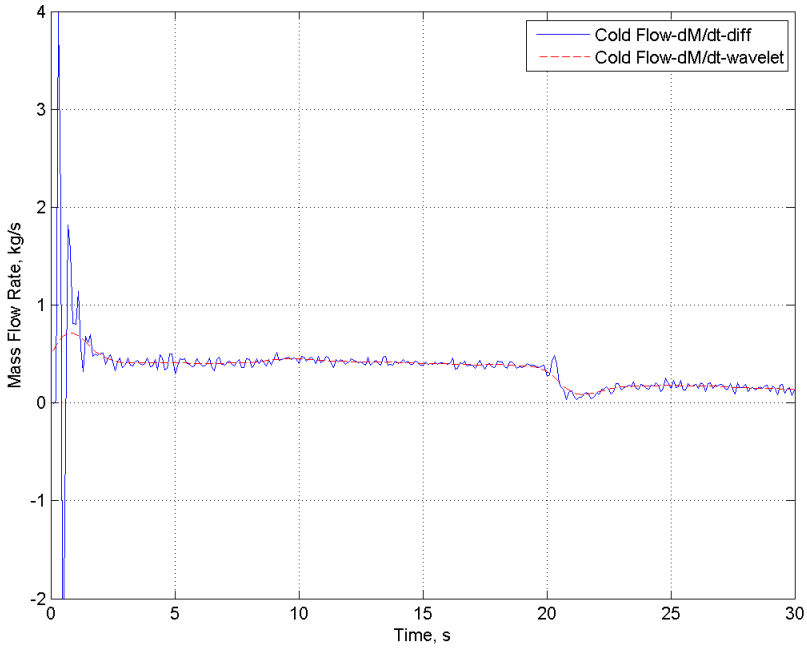

Fig. 4.9: Numerical differentiation methods comparison, cold flow mass flow rates.

Chapter 5

Conclusions

Nitrous oxide is an attractive oxidizer due to its high vapor pressure at room temperature. Because of this property, nitrous oxide can be used to self pressurize eliminating the need for more complex and costly propellant delivery systems. Nitrous oxide is also relatively safe to work with as it is non toxic, non corrosive, and relatively non reactive. N_2O is however not a simple fluid to model because of its non ideal fluid properties in its saturated state. These non-ideal properties make incompressible liquid and ideal gas assumptions invalid and requires a more complex two-phase model to predict performance. N_2O has an additional advantage in that its performance and properties are closely mirrored by a more widely available and even safer fluid. CO_2 exhibits similar behaviour to N_2O and is therefore used as a non energetic checkout fluid for N_2O systems. In order to develop a model to predict the performance of N_2O or CO_2 in an oxidizer system, some key concepts are introduced and an algorithm to calculate mass flow rate is assembled.

The fluid properties of N_2O and CO_2 are fetched in two ways yielding the same results. The first method evaluated was a simple table lookup of values published in the on-line NIST Webbook database. The second method involved building a function around a helmholtz free energy based set of equations.

Two independent methods of mass flow rate prediction have been discussed and another authors assembly of these models was presented in the form of the weighted non-homogeneous non-equilibrium (NHNE) model. An analysis of the NHNE model was performed and a correction was proposed. The proposed correction involved swapping the weighting coefficients so that the intended outcome of the model was realized.

An enthalpy based blowdown algorithm was presented based around the corrected NHNE model. The purpose of the blowdown algorithm was to predict the mass flow rate of

saturated N_2O or CO_2 from a storage tank through an orifice style injector. The algorithm loops through time updating the tank fluid properties by recalculating them each time step from the total enthalpy of the fluid in the tank. The injector downstream fluid properties are calculated using either an isentropic or adiabatic injector assumption.

The enthalpy based blowdown algorithm is compared against experimentally obtained data from a cold flow blowdown of CO_2. The NHNE enthalpy algorithm does an excellent job of predicting the mass flow rate of the fluid through the injector as compared to the alternatives. However, experimental conditions resulting from the test apparatus set-up produces fluid stratification and mixing effects that cannot be modelled by the algorithm as developed. Thus the run tank and temperature drops as predicted by the model are significantly larger than measured. The experimental data could be improved by allowing the fluid in the run tank to reach a homogeneous temperature, prior to performing blow down. To improve the models ability to predict the run tank temperature and pressure, the run tank could be conceptually split into vertical sections with the fluid properties tracked from top to bottom. Additionally, energy could be added to the tank fluid over time to model the heat transfer that occurs from the tank mass to the fluid mass throughout the blow down.

Naturally the next step would be to predict the mass flow of N_2O into a hybrid rocket motor combustion environment. It is expected that the model developed here would perform well in this application, because there is a lower pressure differential across the injector than was accurately modelled here. Overall, the model developed here is a valuable addition to the rocket system designers tool set. It does an excellent job of performing its primary function by accurately predicting the mass flow rate of a self-pressuring fluid.

References

[1] Goddard, R. H., "Rocket Apparatus," 14 July 1914, US Patent No. 1,103,503.

[2] COMPOSITES, S., "SPACESHIPONE & WHITE KNIGHT," `http://www.scaled.com/projects/tierone/`, [retrieved 16 Nov 2011].

[3] PRIZE, X., "Ansari X PRIZE," `http://space.xprize.org/ansari-x-prize`, [retrieved 16 Nov 2011].

[4] Galactic, V., "Virgin Galactic," `http://www.virgingalactic.com/`, [retrieved 16 Nov 2011].

[5] National Institute of Standards and Technology, "Thermophysical Properties of Fluid Systems," `http://webbook.nist.gov/chemistry/fluid/`, [retrieved 17 Feb 2011].

[6] Span, R. and Wagner, W., "A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa," *Journal of Physical and Chemical Reference Data*, Vol. 25, No. 4, 1996, pp. 1509–1596.

[7] Dyer, J., Doran, E., Dunn, Z., and Lohner, K., "Modeling Feed System Flow Physics for Self-Pressurizing Propellants," *43rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*.

[8] Span, R. and Wagner, W., "Equations of State for Technical Applications. I. Simultaneously Optimized Functional Forms for Nonpolar and Polar Fluids," *International Journal of Thermophysics*, Vol. 24, No. 1, 2003, pp. 1–36.

[9] Cengel, Y. A. and Boles, M. A., *Thermodynamics: An Engineering Approach*, The McGraw-Hill Companies, Inc., 4th ed., 2002.

[10] Fox, R. W., McDonald, A. T., and Pritchard, P. J., *Introduction to Fluid Mechanics*, John Wiley and Sons, Inc., 6th ed., 2004.

[11] Darby, R., "Evaluation of Two-Phase Flow Models for Flashing Flow in Nozzles," *Process Safety Progress*, Vol. 19, No. 1, 2000, pp. 32–39.

[12] Whitmore, S. A. and Chandler, S. N., "Engineering Model for Self-Pressurizing Saturated-N_2O-Propellant Feed Systems," *Journal of Propulsion and Power*, Vol. 26, No. 4, 2010, pp. 706–714.

[13] Luo J, Bai J, S. J., "Application of the wavelet transforms on axial strain calculation in ultrasound elastography." *Prog. Nat. Sci.*, Vol. 16, No. 9, 2006, pp. 942–947.

[14] Luo, J., "Numerical differentiation based on wavelet transforms," `http://www.mathworks.com/matlabcentral/fileexchange/13948-numerical-differentiation-based-on-wavelet-transforms`, [retrieved 16 Nov 2011].

Appendices

Appendix A

Fluid Constants

A.1 Comparison of CO_2 and N_2O Properties

Table A.1: Comparison of CO_2 and N_2O Properties

Fluid	CO_2		N_2O	
Molar Mass	44.01 kg/kmol		44.013 kg/kmol	
Gas Constant	0.1889241 kJ/(kg*K)		0.1889 kJ/(kg*K)	
T triple	216.592 K	-69.804 °F	182.340 K	-131.458 °F
T critical	304.128 K	87.761 °F	309.520 K	97.500 °F
P triple	0.51795 MPa	75.122 psi	0.08785 MPa	12.742 psi
P critical	7.3773 MPa	1070.0 psi	7.245 MPa	1050.8 psi
ρ critical	467.5 kg/m3	29.185 lb/ft3	452 kg/m3	28.22 lb/ft3
Compressibility Factor	Saturated Liquid - 0.16 @ 298 K		Saturated Liquid - 0.14 @ 298 K	
Compressibility Factor	Saturated Vapor - 0.47 @ 298 K		Saturated Vapor - 0.53 @ 298 K	
No Liquid If	P $< P_{triple}$ or T $< T_{triple}$		P $< P_{triple}$ or T $< T_{triple}$	
Melting Point	216.59 K	-69.81 °F	182.29 K	-131.50 °F
Sublimation Point	194.75 K	-109.12 °F		
Boiling Point	194.75 K	-109.12 °F	184.67 K	-127.3 °F
P vapor	5.72 MPa @ 293.15 K	829.626 psi @ 68 °F	5.15 MPa @ 293.15 K	746.94 psi @ 68 °F
Ref. Int. Energy	U = 0 at 273.16 K for saturated liquid		U = 0 at 273.16 K for saturated liquid	
Ref. Entorpy	S = 0 at 273.16 K for saturated liquid		S = 0 at 273.16 K for saturated liquid	

Appendix B

Matlab Code

B.1 Blow Down Simulation

```
clear all; clc;
close all;
% Initial Conditions
%=========================================================================
% Tank Properties
%-------------------------------------------------------------------------
V       = .0498;                        % Tank volume, m3
M_o     = 12.76;                         % Initial tank fluid mass, kg
T1_o     = 274.25;                      % Initial tank fluid temperature, K
%M_o     = 10.15;
%T1_o     = 277.9;
%P_o     = 4.025;                       % Initial tank fluid pressure, MPa
%X_o     = 0.01;                        % Initial tank fluid quality,
R = 0.1889241;                          % Gas constant, kJ/(kg*K)
% Injector Properties
%-------------------------------------------------------------------------
d_inj   = 0.178;                         % Injector diameter, in
Ac      = (pi/4)*(d_inj*.0254)^2;    % Injector cross sectional area; m3
Cd      = 0.8;                          % Injector discharge coefficient
Pamb    = 85.9e-3;                      % Atmospheric pressure, MPa. (4500ft)
Inj_Switch = 1;                         % Isentropic = 1, Adiabatic = 2
% Time Iteration
%-------------------------------------------------------------------------
tstop   = 30;                         % Stop time, s
dt      = .5;                        % Step time, s
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize state
%-----------------------------------------------------------------------------
rho1_o   = M_o/V
Props1_o = CO2Props(T1_o,rho1_o);
%Props1_o = CO2PropsNIST(T1_o,rho1_o);
P1_o     = Props1_o.P;
rhol_o   = Props1_o.rho_l;
rhov_o   = Props1_o.rho_v;
X1_o     = (rhov_o/rho1_o)*(rhol_o-rho1_o)/(rhol_o-rhov_o);
h1_o     = Props1_o.h;
H1_o     = M_o*Props1_o.h;
st_o     = Props1_o.state;

t    = 0;        % Column 1: Time (sec)
M    = M_o;      % Column 2: Tank fluid mass, M (kg)
rho1 = rho1_o;   % Column 3: Tank fluid density, rho1 (kg/m^3)
T1   = T1_o;     % Column 4: Tank temperature, T1 (K)
P1   = P1_o;     % Column 5: Tank pressure, P1 (MPa)
X1   = X1_o;     % Column 6: Tank quality, X1 ()
h1   = h1_o;     % Column 7: Tank specific enthalpy, h1 (kJ/kg)
H1   = H1_o;     % Column 8: Tank total enthalpy, H1 (J) ??? Units
mdot = 0;        % Column 9: Tank mass flow rate, mdot (kg/s)
P2   = Pamb;     % Column 10: Injector outlet pressure, P2 (Pa)
st   = st_o      % Column 11: Fluid state, -1=- Input, 0=Liq, 1=Sat, 2=Gas


State = [t, M, rho1, T1, P1, X1, h1, H1, mdot, P2, st];
State2 = [0, 0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
i = 0;
while t < tstop
```

```
61      i = i + 1
62
63      % Exit the loop when the tank is out of fluid
64      %-------------------------------------------------------------------------
65      if M < 0; break; end
66
67      % Initial Tank Fluid Properties
68      %=========================================================================
69      %Note: this guess MUST yeild a quality less than 1 and >0
70      guess=[300 300]; %[T,rho]
71  %
72      %Set up function (need to match pressure and quality)
73      pFunc = @(v) [getfield(CO2Props(v(1),v(2)),'P')-P1; ...
74                    getfield(CO2Props(v(1),v(2)),'X')-X1];
75  %{
76      %Set up function (need to match pressure and quality)
77      pFunc = @(v) [getfield(CO2PropsNIST(v(1),v(2)),'P')-P1 ...
78                    getfield(CO2PropsNIST(v(1),v(2)),'X')-X1];
79  %}
80      % Solve for [T,rho] of saturated but pure liquid at P1
81      % lsqnonlin tries to find a T and rho that makes pFunc = 0
82      v1 = ...
            lsqnonlin(pFunc,guess,0,inf,optimset('Display','off','TolFun',1e-14));
83      T1 = v1(1); rho1 = v1(2);
84      Props1 = CO2Props(T1,rho1);
85  %     Props1 = CO2PropsNIST(T1,rho1);
86      Pv1    = Props1.P;              % Fluid Vapor Pressure, MPa
87      rhoL1  = Props1.rho_l;         % Fluid liquid density, kg/m3
88      h1     = Props1.h;             % Fluid specific enthalpy, kJ/kg
89      H1     = M*h1;                % Fluid total enthalpy, kJ
90      s1     = Props1.s;             % Fluid entropy, kJ/kg*K
91
92      % Propegate properties across the injector
93      %=========================================================================
```

```
94      %Assume an isentropic (s1=s2) or adiabatic (h1=h2) injector , solve ...
            for properties at P2
95      if Inj_Switch == 1
96          %Set up function to match pressure and entropy
97  %
98          pFunc = @(v) [getfield(CO2Props(v(1),v(2)),'P')-P2 ...
99                        getfield(CO2Props(v(1),v(2)),'s')-s1];
100 %{
101         pFunc = @(v) [getfield(CO2PropsNIST(v(1),v(2)),'P')-P2 ...
102                       getfield(CO2PropsNIST(v(1),v(2)),'s')-s1];
103 %}
104     elseif Inj_Switch == 2
105         %Set up function to match pressure and enthalpy
106         pFunc = @(v) [getfield(CO2Props(v(1),v(2)),'P')-P2 ...
107                       getfield(CO2Props(v(1),v(2)),'h')-h1];
108     end
109     % Solve for T2 & rho2 downstream of the injector
110     %-------------------------------------------------------------------------
111     guess=[300 300]; %[T,rho]
112     v2 = ...
            lsqnonlin(pFunc,guess,0,inf,optimset('Display','off','TolFun',1e-14));
113     T2 = v2(1); rho2 = v2(2);
114     % Solve for the remaining properties downstream of the injector
115     %-------------------------------------------------------------------------
116     Props2=CO2Props(T2,rho2);       % Downstream fluid properties
117 %    Props2=CO2PropsNIST(T2,rho2);
118     Pv2 = Props2.P;                 % Downstream fluid vapor pressure
119     h2 = Props2.h;                  % Downstream fluid enthalpy
120
121     % Calculate the Mass Flow
122     %=========================================================================
123     % Non-Equalibrium Parameter
124     %k = sqrt(abs(P1-P2)/abs(Pv2-P1));    % Whitmores Equation
125     k = sqrt((P1-P2)/(Pv1-P2));          % Spencer and Stanfords Equation
126     % Weighting Coefficient
```

```
127     W = (1/(k+1));
128     % Incompressible fluid mass flow rate
129     mdot_inc = Ac*sqrt(2*rhoL1*(P1-P2)*1e6);
130     % Homogeneous Equilibrium mass flow rate
131     mdot_HEM = rho2*Ac*sqrt(2*(h1-h2));
132     % Weighted Non-Homogeneous Equilibrium (modified Stanford) mass flow rate
133     mdot = Cd*((1-W) * mdot_inc + W * mdot_HEM);  % Shannon's Theory
134
135     % Update the upstream fluid properties for the next step
136     %=========================================================================
137     M = M - mdot*dt;     % Update tank fluid mass
138     Hdot = h1*mdot;      % Enthalpy flow rate
139     H1 = H1 - Hdot*dt;   % Update tank total enthalpy
140
141     % Calculate the new tank enthalpy and density
142     %-------------------------------------------------------------------------
143     rho1 = M/V;          % Update tank specific density
144     h1 = H1/M;           % Update tank specific enthalpy
145
146     % Calculate the new tank temperature
147     %-------------------------------------------------------------------------
148     % Create a function for lsqnonlin to solve T(rho,h)
149     pFunc = @(T_Unknown) getfield(CO2Props(T_Unknown,rho1),'h')-h1;
150 %    pFunc = @(T_Unknown) getfield(CO2PropsNIST(T_Unknown,rho1),'h')-h1;
151     % Sinse T_Unknown is not pre defined in pFunc, lsqnonlin will find a ...
          T for rho_Known and h_Known
152     T1 = lsqnonlin(pFunc,300,0,inf,optimset('Display','off','TolFun',1e-14));
153 %    [T1] = CO2_rho_h_2T(rho1,h1,300);
154
155     % Calulate the new tank pressure and quality
156     %-------------------------------------------------------------------------
157     Props1 = CO2Props(T1,rho1);
158 %    Props1 = CO2PropsNIST(T1,rho1);
159     P1  = Props1.P;
160     X1  = Props1.X;
```

```
161      st1 = Props1.state;
162
163      % Update the state
164      %-------------------------------------------------------------------------
165      t = t + dt;
166
167      State = [State; t, M, rho1, T1, P1, X1, h1, H1, mdot, Pamb, st1];
168      State2 = [State2; mdot_inc, mdot_HEM];
169  end
170  %
171  % Open cold flow data for comparison
172  %=========================================================================
173  CF = open('Cold Flow Data\FillTest1_29_largeNoTop.mat');
174  %CF = open('Cold Flow Data\FillTest1_29_smallNoTop.mat');
175  %CF = open('Cold Flow Data\FillTest1_29_large.mat');
176  %CF = open('Cold Flow Data\FillTest1_29_small.mat');
177
178  % Find portion of cold flow data where the run valve is open
179  %-------------------------------------------------------------------------
180  iter = 0;
181  for loop = 1:length(CF.Valvedat)
182      if CF.Valvedat(loop,5) == 1
183          iter = iter + 1;
184          CF_t(iter)            = CF.t(loop);
185          CF_Weight(iter)       = CF.Weight(loop);
186          CF_PTank(iter)        = CF.PRunTank(loop);
187          CF_Mdot(iter)         = CF.mdotVentIn(loop);
188          CF_DeltaWeight(iter)  = CF.DeltaWeightKg(loop);
189          CF_T_Tank(iter)       = CF.Tank_Temp(loop);
190          CF_dP_Vent(iter)      = CF.dpVenturi(loop);
191          CF_rhol_H(iter)      = CF.rhol_H(loop);
192      end
193  end
194  CF_t = CF_t - CF_t(1);
195
```

```
196 % Calculate the venturi mass flow rate the helmholtz density
197 %-------------------------------------------------------------------------
198 d1=0.0254762;
199 d2=0.0078486;
200 A1=pi*d1^2/4;
201 A2=pi*d2^2/4;
202 C=0.980243097172394;
203 CF_Mdot_H=A2.*C.*CF_rhol_H.*sqrt(2.*(CF_dP_Vent)./(CF_rhol_H.*(1-(A2/A1)^2)));
204
205 % Calculate the mass flow rate by differntiating the tank mass
206 %-------------------------------------------------------------------------
207 CF_Mdot_diff = -diff(CF_Weight)./diff(CF_t);
208 CF_dt = CF_t(2) - CF_t(1);
209 %CF_Mdot_wave = derivative_cwt(CF_Weight,'gaus1',16,CF_dt,1);
210 CF_Mdot_wave = derivative_dwt(CF_Weight,'spl',4,CF_dt,1);
211
212 % Open cold flow data for comparison
213 %=========================================================================
214 Whitmore = open('Whitmores Output\FillTest1_29_largeNoTop.mat');
215 W_t      = Whitmore.t;
216 W_M      = Whitmore.M;
217 W_rho1   = Whitmore.rho1;
218 W_T1     = Whitmore.T1;
219 W_P1     = Whitmore.P1;
220 W_X1     = Whitmore.X1;
221 W_H1     = Whitmore.H1;
222 W_mdot   = Whitmore.mdot;
223 W_2      = Whitmore.P2;
224
225 % Up-pack the state vector for ploting
226 %=========================================================================
227 t    = State(:,1);      % Column 1: Time (sec)
228 M    = State(:,2);      % Column 2: Tank fluid mass, M (kg)
229 rho1 = State(:,3);      % Column 3: Tank fluid density, rho1 (kg/m^3)
230 T1   = State(:,4);      % Column 4: Tank temperature, T1 (K)
```

```
231 P1   = State(:,5);       % Column 5: Tank pressure, P1 (Pa)
232 X1   = State(:,6);       % Column 6: Tank quality, X1 ()
233 h1   = State(:,7);       % Column 7: Tank specific enthalpy, h1 (kJ/kg)
234 H1   = State(:,8);       % Column 8: Tank total enthalpy, H1 (J) ??? Units
235 mdot = State(:,9);       % Column 9: Tank mass flow rate, mdot (kg/s)
236 P2   = State(:,10);      % Column 10: Injector outlet pressure, P2 (Pa)
237 st   = State(:,11);      % Column 11: Fluid state, -1=- Input, 0=Liq, ...
        1=Sat, 2=Gas
238
239 mdot_inc = State2(:,1); % Mdot Incompressible
240 mdot_HEM = State2(:,2); % Mdot HEM
241
242 %% Plot
243 %=========================================================================
244 % Column 2 - Tank fluid mass
245 %-------------------------------------------------------------------------
246 figure(2)
247 plot(t,M); hold on; grid on;
248 %plot(W_t,W_M,'--g');
249 plot(CF_t,CF_Weight,'--r');
250 %plot(CF_t,CF_DeltaWeight,'g');
251 %title('Tank Mass');
252 xlim([0 tstop]);
253 xlabel('Time, s'); ylabel('Tank Fluid Mass, kg');
254 %legend('Model Predicted','Whitmore','Cold Flow');
255 legend('Model Predicted','Cold Flow');
256 saveas(2,'Tank_Mass.png')
257
258 % Column 3 - Tank fluid specific density
259 %-------------------------------------------------------------------------
260 figure(3);
261 plot(t,rho1); hold on; grid on;
262 plot(W_t,W_rho1,'--g');
263 plot(CF_t,CF_Weight/V,'r');
264 title('rho1');
```

```
265 xlabel('Time, s'); ylabel('Density, kg/m3');
266 legend('Model Predicted','Whitmore','Cold Flow');
267 saveas(3,'Tank_Density.png')
268
269 % Column 4 - Tank temperature
270 %-------------------------------------------------------------------------
271 figure(4);
272 plot(t,T1); hold on; grid on;
273 %plot(W_t,W_T1,'--g');
274 plot(CF_t,CF_T_Tank,'--r');
275 %title('Temperature');
276 xlim([0 tstop]);
277 xlabel('Time, s'); ylabel('Temperature, K');
278 %legend('Model Predicted','Whitmore','Cold Flow');
279 legend('Model Predicted','Cold Flow');
280 saveas(4,'Tank_Temperature.png')
281
282 % Column 5 - Tank pressure
283 %-------------------------------------------------------------------------
284 figure (5)
285 plot(t,P1); hold on; grid on;
286 %plot(W_t,W_P1/1e3,'--g');
287 plot(CF_t,CF_PTank/1e6,'--r');
288 %title('P1');
289 xlim([0 tstop]);
290 xlabel('Time, s'); ylabel('Pressure, MPa');
291 legend('Model Predicted','Cold Flow');
292 %legend('Model Predicted','Whitmore','Cold Flow');
293 saveas(5,'Tank_Pressure.png')
294
295 % Column 6 - Tank quality
296 %-------------------------------------------------------------------------
297 figure (6)
298 plot(t,X1); hold on; grid on;
299 plot(W_t,W_X1,'--g');
```

```
ylim([0 1]);
title('X1');
xlabel('Time, s'); ylabel('Quality');
legend('Model Predicted','Whitmore');
saveas(6,'Tank_Quality.png')

% Coumn 7 - Tank specific enthalpy
%-------------------------------------------------------------------------
figure (7)
plot(t,h1); hold on; grid on;
title('h1');
xlabel('Time, s'); ylabel('Specific Enthalpy, kJ/kg-K');
legend('Model Predicted');
saveas(7,'Tank_Specific_Enthalpy.png')

% Coumn 8 - Tank total enthalpy
%-------------------------------------------------------------------------
figure (8)
plot(t,H1); hold on; grid on;
plot(W_t,W_H1,'--g');
title('H1'); xlabel('time, s'); ylabel('Enthalpy');
legend('Model Predicted','Whitmore');
saveas(8,'Tank_Total_Enthalpy.png')

% Column 9 - Mass flow rate
%-------------------------------------------------------------------------
figure(9);
plot(t,mdot,'b'); hold on; grid on;
plot(CF_t,CF_Mdot,'--r')
plot(CF_t,CF_Mdot_H,'-.b')
xlabel('Time, s'); ylabel('Mass Flow Rate, kg/s');
xlim([0 tstop]); %ylim([0 3]);
legend('Model Predicted','Venturi-Curve Fit Density','Venturi-Helmholtz ...
    Density');
saveas(9,'Tank_Mass_Flow_Rate_Venturi.png')
```

```
334
335 figure(90);
336 plot(t,mdot,'b'); hold on; grid on;
337 %plot(W_t,W_mdot,'--g');
338 plot(CF_t,CF_Mdot,'--r')
339 plot(CF_t,-CF_Mdot_wave,'-.b')
340 plot(t,mdot_inc,':b');
341 plot(t,mdot_HEM,'--bo');
342 xlabel('Time, s'); ylabel('Mass Flow Rate, kg/s');
343 xlim([0 tstop]); ylim([0 1.6]);
344 legend('Model Predicted','Cold Flow-Venturi','Cold Flow-Differentiated ...
        Load','Incompressible Model','HEM Model');
345 saveas(90,'Tank_Mass_Flow_Rate.png')
346
347 figure(900);
348 plot(CF_t(2:end),CF_Mdot_diff);  hold on; grid on;
349 plot(CF_t,-CF_Mdot_wave,'--r')
350 xlabel('Time, s'); ylabel('Mass Flow Rate, kg/s');
351 xlim([0 tstop]); ylim([-2 4]);
352 legend('Cold Flow-dM/dt-diff','Cold Flow-dM/dt-wavelet');
353 saveas(900,'Tank_Mass_Flow_Rate_Diff.png')
354
355 % Column 11 - Tank fluid state
356 %-------------------------------------------------------------------------
357 figure(11);
358 plot(t,st);  grid on;
359 title('Fluid State');
360 xlabel('Time, s'); ylabel('Fluid State');
361 ylim([-2 3]);
362 legend('Model Predicted');
363 saveas(11,'Tank_Fluid_State.png')
364 %
```

B.2 CO_2 Properties - Helmholtz Energy Method

```
function [Props] = CO2Props(T,rho)
%#eml
%% INTRODUCTION
%=========================================================================
% Purpose:
% Returns a sturcture containing the thermodynamic properties for carbon
% dioxide (CO2) for a given density (kg/m^3) and temperature (K). Valid
% temperature and pressure ranges are:
%    216 K ≤ T ≤ 1100K       0 MPa ≤ P ≤ 800 MPa
%-------------------------------------------------------------------------
% The properties in the structure are:
%-------------------------------------------------------------------------
%   P     X     s     u     cv    cp    h     c
%         rho_l s_l   u_l   cv_l  cp_l  h_l   c_l
%         rho_v s_v   u_v   cv_v  cp_v  h_v   c_v
%-------------------------------------------------------------------------
% Inputs:
%   T       -   Temperature, K
%   rho     -   Density, kg/m3
% Outputs:
%   P       -   Pressure, MPa
%   X       -   Quality, (Vapor Mass/Total Fluid Mass)
%   s       -   Specific Entropy, kJ/(kg*K)
%   u       -   Specific Internal Energy, kJ/kg
%   cv      -   Specific Heat at Constant Volume, kJ/(kg*K)
%   cp      -   Specific Heat at Constant Pressure, kJ/(kg*K)
%   h       -   Specific Enthalpy, kJ/kg
%   c       -   Speed of Sound, m/s
%   rho     -   Density, kg/m3
%   state   -   -1 = Negative Input, 0 = Liquid, 1 = Saturated, 2 = Gas
%   _l      -   Liquid designator
%   _v      -   Vapor designator
%-------------------------------------------------------------------------
% Revision History:
```

```
% Written for a CO2 Blowdown model developed at Utah State University by
%  Matthew Wilson
%  4130 Old Main Hill
%  Logan, UT 84322-4130
% Recommented and checked by:
% Brian Solomon
%-------------------------------------------------------------------------
% Based upon the Helmholtz Energy based equations of state described by
% Span, R. and Wagner, W. in
%  "A New Equation of State for Carbon Dioxide Covering the Fluid Region
%  from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa"
% Journal of Physical and Chemical Reference Data
% Vol 25, No. 6, 1996. Pp 1509-1596
%-------------------------------------------------------------------------

%% CALCULATE THE SATURATION PROPERTIES
%=========================================================================
% CO2 Constants
%-------------------------------------------------------------------------
R = 0.1889241;      % Gas constant, kJ/(kg*K)
Tt = 216.592;       % Triple point temperature, K   (Eq. 3.1)
Pt = 0.51795;       % Triple point pressure, MPa    (Eq. 3.2)
Tc = 304.1282;      % Critical temperature, K       (Eq. 3.3)
Pc = 7.3773;        % Critical pressure, MPa        (Eq. 3.4)
rhoc = 467.6;       % Critical density, kg/m3       (Eq. 3.5)
% d = rho/rhoc;     % Reduced density, Δ = phi/phi_critical, phi - mass ...
    density
t = Tc/T;           % Inverse reduced temperature, tau = T_critical/T, T ...
    - temperature

% Coefficients
%-------------------------------------------------------------------------
% Melting pressure  coefficients, Section 3.3
a_m = [1955.5390 2055.4593]';
% Sublimation pressure coefficients, Section 3.4
```

```
68  a_s = [-14.740846 2.4327015 -5.3061778]';
69  % Vapor pressure coefficients, Section 3.5
70  a_p = [-7.0602087 1.9391218 -1.6463597 -3.2995634]';
71  t_p = [1.0 1.5 2.0 4.0]';
72  % Saturated liquid density coeffiecients, Section 3.6
73  a_l = [1.9245108 -0.62385555 -0.32731127 0.39245142]';
74  t_l = [0.34 0.5 10/6 11/6]';
75  % Saturated vapor density coeffiecients, Section 3.7
76  a_v = [-1.7074879 -0.82274670 -4.6008549 -10.111178 -29.742252]';
77  t_v = [0.34 0.5 1.0 7/3 14/3]';
78
79  % Phase Property Calculations
80  %-------------------------------------------------------------------------
81  % P_melt = Pt* (1 + a_m(1)*(T/Tt-1) + a_m(2)*(T/Tt-1)^2); % Melting ...
        Pressure, Eq. 3.10
82  % P_sub = Pt*exp(Tt/T*(a_s(1)*(1-T/Tt) + ...              % Sublimation ...
        Presssure, Eq. 3.12
83  %                      a_s(2)*(1-T/Tt)^1.9 + ...
84  %                      a_s(3)*(1-T/Tt)^2.9));
85  P_sat = Pc * exp(Tc/T*sum(a_p.*(1-T/Tc).^t_p)); % Vapor Pressure, Eq. 3.13
86  rho_l = rhoc * exp(sum(a_l.*(1-T/Tc).^t_l));    % Saturated Liquid ...
        Density, Eq. 3.14
87  rho_v = rhoc * exp(sum(a_v.*(1-T/Tc).^t_v));    % Saturated Vapor ...
        Density, Eq. 3.15
88
89  %% CALCULATE THE PROPERTIES AT THE GIVEN TEMPERATURE AND DENSITY
90  %=========================================================================
91  % Account for negative density or temperature
92  %-------------------------------------------------------------------------
93  if rho < 0 || T < 0
94      X = NaN;
95      P = NaN;
96      s_v = NaN; u_v = NaN; cp_v = NaN;
97      cv_v = NaN; h_v = NaN; c_v = NaN;
98      s_l = NaN; u_l = NaN; cp_l = NaN;
```

```
99      cv_l = NaN; h_l = NaN; c_l = NaN;
100     s = s_v*X + s_l*(1-X);
101     u = u_v*X + u_l*(1-X);
102     h = h_v*X + h_l*(1-X);
103     cp = cp_v*X + cp_l*(1-X);
104     cv = cv_v*X + cv_l*(1-X);
105     c = NaN;
106     state = -1;
107 % GAS
108 %-------------------------------------------------------------------------
109 elseif rho < rho_v || imag(rho_l)
110     [P s u cp cv h c] = Helmholtz(t,rho/rhoc);
111     X = 1;
112     s_v = s; u_v = u; cp_v = cp;
113     cv_v = cv; h_v = h; c_v = c;
114     s_l = NaN; u_l = NaN; cp_l = NaN;
115     cv_l = NaN; h_l = NaN; c_l = NaN;
116     state = 2;
117     %P = P/1E6;
118 % LIQUID
119 %-------------------------------------------------------------------------
120 elseif rho > rho_l
121     [P s u cp cv h c] = Helmholtz(t,rho/rhoc);
122     X = 0;
123     s_l = s; u_l = u; cp_l = cp;
124     cv_l = cv; h_l = h; c_l = c;
125     s_v = NaN; u_v = NaN; cp_v = NaN;
126     cv_v = NaN; h_v = NaN; c_v = NaN;
127     state = 0;
128     %P = P/1E6;
129 % MELTING???
130 %-------------------------------------------------------------------------
131 %
132 % SATURATED
133 %-------------------------------------------------------------------------
```

```
134 else
135     X = rho_v*(rho_l - rho)/(rho*(rho_l-rho_v));
136     [P s_l u_l cp_l cv_l h_l c_l] = Helmholtz(t,rho_l/rhoc);
137     [P s_v u_v cp_v cv_v h_v c_v] = Helmholtz(t,rho_v/rhoc);
138     s = s_v*X + s_l*(1-X);
139     u = u_v*X + u_l*(1-X);
140     h = h_v*X + h_l*(1-X);
141     cp = cp_v*X + cp_l*(1-X);
142     cv = cv_v*X + cv_l*(1-X);
143
144     c = NaN;
145     P = P_sat;
146     state = 1;
147 end
148
149 %% CREATE THE OUTPUT STRUCTURE
150 %=========================================================================
151 Props.P     = P;                                          % Pressure
152 Props.X     = X;                                          % Quality
153 Props.s     = s;    Props.s_l   = s_l;  Props.s_v   = s_v;  % Entropy
154 Props.u     = u;    Props.u_l   = u_l;  Props.u_v   = u_v;  % Internal Energy
155 Props.cv    = cv;   Props.cv_l  = cv_l; Props.cv_v  = cv_v; % Specific ...
        Heat at Constant Volume
156 Props.cp    = cp;   Props.cp_l  = cp_l; Props.cp_v  = cp_v; % Specific ...
        Heat at Constant Pressure
157 Props.h     = h;    Props.h_l   = h_l;  Props.h_v   = h_v;  % Enthalpy
158 Props.c     = c;    Props.c_l   = c_l;  Props.c_v   = c_v;  % Speed of Sound
159 Props.rho_l = rho_l;Props.rho_v = rho_v;                    % Density
160 Props.state = state;                                       % State
161 %Props.gma_l = cp_l/cv_l;
162 %Props.gma_v = cp_v/cv_v;
163 end
164
165 %#########################################################################
166 function [P s u cp cv h c] = Helmholtz(t,d)
```

```
%#eml
%% INTORDUCTION
%=========================================================================
% Calculates CO2 properties using the residual and ideal gas portions of
% the Helmholtz Energy at a given temperature and density normalized to the
% critical point temperature and density.
%-------------------------------------------------------------------------
% Based upon the Helmholtz Energy based equations of state described by
% Span, R. and Wagner, W. in
%  "A New Equation of State for Carbon Dioxide Covering the Fluid Region
%  from the Triple—Point Temperature to 1100 K at Pressures up to 800 MPa"
% Journal of Physical and Chemical Reference Data
% Vol 25, No. 6, 1996. Pp 1509—1596
%-------------------------------------------------------------------------
% Input:
%   t   —   Inverse Reduced Temperature, tau = T_critical/T, T — Temperature
%   d   —   Reduced Density, Δ = phi/phi_critical, phi — Mass Density
%-------------------------------------------------------------------------
% Output:
%   P   —   Pressure, Pa
%   s   —   Entropy, kJ/(kg*K)
%   u   —   Internal energy, kJ/kg
%   cp  —   Isochoric heat capacity, kJ/(kg*K)
%   cv  —   Isobaric heat capacity, kJ/(kg*K)
%   h   —   Entropy, kJ/kg
%   c   —   Speed of sound, m/s
%=========================================================================

%%  RESIDUAL PART OF HELMHOLTZ
%=========================================================================
% CONSTANT PARAMETER DEFINITION —
% Page 1544, Table 31. Coefficients and exponents of Eq. (6.5)
%-------------------------------------------------------------------------
n7 = [ 0.38856823203161E0
       0.29385475942740E1
```

```
       -0.55867188534934E1
       -0.76753199592477E0
        0.31729005580416E0
        0.54803315897767E0
        0.12279411220335E0];
d7 = [1 1 1 1 2 2 3]';
t7 = [0.00 0.75 1.00 2.00 0.75 2.00 0.75]';
%-------------------------------------------------------------------------
n34 = [ 0.21658961543220E1
         0.15841735109724E1
        -0.23132705405503E0
         0.58116916431436E-1
        -0.55369137205382E0
         0.48946615909422E0
        -0.24275739843501E-1
         0.62494790501678E-1
        -0.12175860225246E0
        -0.37055685270086E0
        -0.16775879700426E-1
        -0.11960736637987E0
        -0.45619362508778E-1
         0.35612789270346E-1
        -0.74427727132052E-2
        -0.17395704902432E-2
        -0.21810121289527E-1
         0.24332166559236E-1
        -0.37440133423463E-1
         0.14338715756878E0
        -0.13491969083286E0
        -0.23151225053480E-1
         0.12363125492901E-1
         0.21058321972940E-2
        -0.33958519026368E-3
         0.55993651771592E-2
        -0.30335118055646E-3];
```

```
237 d34 = [1 2 4 5 5 5 6 6 6 1 1 4 4 4 7 8 2 3 3 5 5 6 7 8 10 4 8]';
238 t34 = [1.5000 1.5000 2.5000 0.0000 1.5000 2.0000 0.0000 1.0000...
239 2.0000 3.0000 6.0000 3.0000 6.0000 8.0000 6.0000 0.0000 7.0000 12.0000 ...
240 16.0000 22.0000 24.0000 16.0000 24.0000 8.0000 2.0000 28.0000 14.0000]';
241 c34 = [1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 4 4 4 4 4 4 5 6]';
242 %-------------------------------------------------------------------------
243 n39 = [-0.21365488688320E3
244          0.26641569149272E5
245         -0.24027212204557E5
246         -0.28341603423999E3
247          0.21247284400179E3];
248 d39 = [2 2 2 3 3]';
249 t39 = [1.00 0.00 1.00 3.00 3.00]';
250 a39 = [25 25 25 15 20]';
251 bt39 = [325 300 300 275 275]';
252 y39 = [1.16 1.19 1.19 1.25 1.22]';
253 e39 = [1.00 1.00 1.00 1.00 1.00]';
254 %-------------------------------------------------------------------------
255 n42 = [-0.66642276540751E0
256          0.72608632349897E0
257          0.55068668612842E-1];
258 a42 = [3.500 3.500 3.000]';
259 b42 = [0.875 0.925 0.875]';
260 bt42 = [0.300 0.300 0.300]';
261 A42 = [0.700 0.700 0.700]';
262 B42 = [0.3 0.3 1.0]';
263 C42 = [10.0 10.0 12.5]';
264 D42 = [275 275 275]';
265
266 R = 0.1889241;  % Gas Constant, kJ/(kg*K)
267 Tc = 304.1282;  % Critical Temperature, K
268 rhoc = 467.6;   % Critical Density, kg/m3
269 %-------------------------------------------------------------------------
270
271 % NONDIMENSIONALIZED HELMHOLTZ FREE ENERGY
```

```
272 %  RESIDUAL PORTION OF THE HELMHOLTZ ENERGY
273 %-------------------------------------------------------------------------
274 tta = (1-t) + A42 .* ((d-1).^2).^(0.5./bt42);
275 Del = tta.^2 + B42.*((d-1).^2).^a42;
276 Psi = exp(-C42.*(d-1).^2 - D42.*(t-1).^2);
277
278 Psi_d = -2 .* C42 .* (d-1) .* Psi;
279 Psi_t = -2 .* D42 .* (t-1) .* Psi;
280 Del_d = (d-1).*(A42 .* tta .* 2./bt42 .* ...
281           ((d-1).^2).^(.5./bt42-1) + ...
282           2 .* B42 .* a42 .* ((d-1).^2).^(a42-1));
283
284 Psi_dd = 2.*C42.*Psi.*(2.*C42.*(d-1).^2 - 1);
285 Psi_tt = 2.*D42.*Psi.*(2.*D42.*(t-1).^2 - 1);
286 Del_dd = 1./(d-1).*Del_d + (d-1).^2 ...
287        .* (4.*B42.*a42.*(a42-1).*((d-1).^2).^(a42-2) + ...
288              2.*A42.^2.*(1./bt42).^2.*(((d-1).^2).^(1./(2.*bt42)-1)).^2 + ...
289              A42.*tta.*4./bt42.*(.5./bt42-1).*((d-1).^2).^(.5./bt42-2));
290
291 Psi_dt = 4.*C42.*D42.*(d-1).*(t-1).*Psi;
292
293 % Residual part of the Helmholtz energy, Eq. 6.5
294 %-------------------------------------------------------------------------
295 phir = sum(n7 .* d.^d7 .* t.^t7) ...
296      + sum(n34 .* d.^d34 .* t.^t34 .* exp(-d.^c34)) ...
297      + sum(n39 .* d.^d39 .* t.^t39 .* ...
298          exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2)) ...
299      + sum(n42 .* Del.^b42 .* d .* Psi);
300
301 phir_d = sum(n7 .* d7 .* d.^(d7-1) .* t.^t7) ...
302        + sum(n34 .* d.^(d34-1) .* t.^t34 .* ...
303            exp(-d.^c34) .* (d34 - c34.* d.^c34)) ...
304        + sum(n39 .* d.^d39 .* t.^t39 .* ...
305            exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2) .* ...
306            (d39./d - 2.*a39.*(d-e39))) ...
```

```
307         + sum(n42 .* (Del.^b42 .* (Psi + d.*Psi_d) + ...
308             b42 .* Del.^(b42-1) .* Del_d .* d .* Psi));
309
310  phir_dd = sum(n7 .* d7 .* (d7 - 1) .* d.^(d7-2) .* t.^t7) ...
311          + sum(n34 .* exp(-d.^c34) .* d.^(d34-2) .* t.^t34 .* ...
312              ((d34 - c34.* d.^c34).*(d34 -1-c34.* d.^c34) - c34.^2 .* ...
                    d.^c34)) ...
313          + sum(n39  .* t.^t39 .* ...
314              exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2) .* ...
315              (-2.*a39.*d.^d39 + 4.*a39.^2.*d.^d39.*(d-e39).^2 - ...
316              4.*d39.*a39.*d.^(d39-1).*(d-e39) + d39.*(d39-1).*d.^(d39-2))) ...
317          + sum(n42 .* (Del.^b42 .* (2.*Psi_d + d*Psi_dd) + ...
318              2.*b42 .* Del.^(b42-1) .* Del_d .*(Psi + d.*Psi_d) + ...
319              b42.*d.*Psi.*...
320                (Del.^(b42-1).*Del_dd + (b42-1).*Del.^(b42-2).*Del_d.^2)));
321
322  phir_t = sum(n7 .* t7 .* d.^d7 .* t.^(t7-1)) ...
323         + sum(n34 .* d.^d34 .* t34 .* t.^(t34-1) .* exp(-d.^c34)) ...
324         + sum(n39 .* d.^d39 .* t.^t39 .* ...
325             exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2) .* ...
326             (t39./t - 2.*bt39.*(t-y39))) ...
327         + sum(n42 .*d .* (-2.*tta.*b42.*Del.^(b42-1).*Psi + ...
328             Del.^b42.*Psi_t));
329
330  phir_tt = sum(n7 .* t7 .* (t7-1) .* d.^d7 .* t.^(t7-2)) ...
331          + sum(n34 .* t34 .* (t34-1) .* d.^d34 .* t.^(t34-2) .* ...
332              exp(-d.^c34)) ...
333          + sum(n39 .* d.^d39 .* t.^t39 .* ...
334             exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2) .* ...
335             ((t39./t - 2.*bt39.*(t-y39)).^2 - t39./(t.^2)-2.*bt39)) ...
336          + sum(n42 .*d .* (Psi.*(2.*b42.*Del.^(b42-1) + ...
337               4.*tta.^2.*b42.*(b42-1).*Del.^(b42-2)) - ...
338              4.*tta.*b42.*Del.^(b42-1).*Psi_t + Del.^b42.*Psi_tt));
339
340  phir_dt = sum(n7 .* t7 .* d7 .* d.^(d7-1) .* t.^(t7-1)) ...
```

```
340         + sum(n34 .* exp(-d.^c34) .* t34 .* d.^(d34-1) .* t.^(t34-1) .* ...
341             (d34 - c34.* d.^c34)) ...
342         + sum(n39 .* d.^d39 .* t.^t39 .* ...
343             exp(-a39.*(d - e39).^2 - bt39.*(t - y39).^2) .* ...
344             (t39./t - 2.*bt39.*(t-y39)).*(d39./d - 2.*a39.*(d-e39))) ...
345         + sum(n42.* ( Del.^b42.*(Psi_t + d.*Psi_dt) + ...
346             d.*b42.*Del.^(b42-1).*Del_d.*Psi_t - ...
347             2.*tta.*b42.*Del.^(b42-1).*(Psi + d.*Psi_d) + ...
348             d.*Psi.*(-A42.*b42.*0.5./bt42.*Del.^(b42-1).*(d-1).*((d-1).^2).^(0.5./bt42-1) .
                    - ...
349               2.*tta.*b42.*(b42-1).*Del.^(b42-2).*Del_d)));
350
351 %%  IDEAL GAS HELMHOLTZ
352 %=========================================================================
353 % CONSTANT PARAMETER DEFINITION -
354 % Page 1540, Table 27. Coefficients of the correlation equations, Eq. (6.2)
355 % and Eq. (6.3)
356 %-------------------------------------------------------------------------
357 ao3 = [8.37304456 -3.70454304 2.50000000]';
358 ao8 = [1.99427042
359        0.62105248
360        0.41195293
361        1.04028922
362        0.08327678];
363 ttao8 = [3.15163
364           6.11190
365           6.77708
366           11.32384
367           27.08792];
368
369 % Ideal Gas part of the Helmholtz energy, Eq. 6.3
370 %-------------------------------------------------------------------------
371 phio = log(d) + ao3(1) + ao3(2)*t + ao3(3)*log(t)...
372      + sum(ao8.*log( 1-exp(-ttao8.*t) ));
373 % phio_d = 1/d;
```

```
374 % phio_dd = -1/d^2;
375 % phio_dt = 0;
376
377 phio_t = ao3(2) + ao3(3)/t + sum(ao8.*ttao8.*( (1-exp(-ttao8.*t)).^(-1) ...
        -1 ));
378
379 phio_tt = -ao3(3)/(t^2) - sum(ao8 .* ttao8.^2 .* exp(-ttao8.*t) .* ...
380               ( 1-exp(-ttao8.*t) ).^(-2) );
381
382 %% THERMODYNAMIC PROPERTY CALCULATIONS
383 %=========================================================================
384 % Page 1517, Table 3. Relations of thermodynamic properties to the
385 % dimensionless Helmholtz function phi consisting of phi_ideal_gas and
386 % phi_residual
387 %-------------------------------------------------------------------------
388 % Pressure
389 P = 1000*d*rhoc*R*Tc/t*(1+d*phir_d);
390 % Entropy
391 s = R*(t*(phio_t + phir_t) - phio - phir);
392 % Internal energy
393 u = R*Tc/t*t*(phio_t+phir_t);
394 % Isochoric heat capacity
395 cv = -R*t^2*(phio_tt+phir_tt);
396 % Isobaric heat capacity
397 cp = R*(-t^2*(phio_tt+phir_tt) + ...
398          (1+d*phir_d-d*t*phir_dt)^2/(1+2*d*phir_d+d^2*phir_dd));
399 % Enthalpy
400 h = R*Tc/t*(1+t*(phio_t+phir_t)+d*phir_d);
401 % Speed of sound
402 c = sqrt( R*Tc/t*1000*( 1 + 2*d*phir_d + d^2*phir_dd - ...
403           ((1 + d*phir_d - d*t*phir_dt)^2/(t^2*(phio_tt+phir_tt)))) );
404 % Joule-Thompson coefficient
405 mu = -1/(R*d*rhoc)*(d*phir_d + d^2*phir_dd + d*t*phir_dt) ...
406     / ((1+d*phir_d-d*t*phir_dt)^2 - ...
          t^2*(phio_tt+phir_tt)*(1+2*d*phir_d+d^2*phir_dd));
```

```
407 % Fugacity
408 %psi = exp(phir+d*phir_d-log(1+d*phir_d));
409 % 2nd Virial
410 %B = lim(phir_d)/rhoc as d->0
411 % 3rd Virial
412 %C = lim(phir_dd)/rhoc^2 as d->0
413
414 %% Convert the energy terms from J/gram to J/kg
415 %=========================================================================
416 %u = u*1000; s = s*1000; h = h*1000;
417 %
418 %% Reference the energy terms to the same reference as the NIST tables
419 %=========================================================================
420 u_NIST = 506.8006990081582;
421 s_NIST = 2.739101354918014;
422 h_NIST = 506.77604151182834;
423
424 % Shift the energy terms to match the NIST reference point
425 %-------------------------------------------------------------------------
426 u = u + u_NIST;
427 s = s + s_NIST;
428 h = h + h_NIST;
429 %
430 end
```

B.3 CO_2 Properties - NIST Webbook Method

```
1 function [Props] = CO2PropsNIST(T,rho)
2 %#eml
3 %% INTRODUCTION
4 %=========================================================================
5 % Purpose:
6 % Returns a sturcture containing the thermodynamic properties for saturated
```

```
% carbon  dioxide (CO2) for a given density (kg/m^3) and temperature (K).
% Valid temperature and pressure ranges are:
%    216.59 K ≤ T ≤ 303.69K       0 MPa ≤ P ≤ 800 MPa
%--------------------------------------------------------------------------
% The properties in the structure are:
%--------------------------------------------------------------------------
%   P     X     s     u     cv    cp    h     c
%         rho_l s_l   u_l   cv_l  cp_l  h_l   c_l
%         rho_v s_v   u_v   cv_v  cp_v  h_v   c_v
%--------------------------------------------------------------------------
% Inputs:
%   T       -   Temperature, K
%   rho     -   Density, kg/m3
% Outputs:
%   P       -   Pressure, MPa
%   X       -   Quality, (Vapor Mass/Total Fluid Mass)
%   s       -   Specific Entropy, kJ/(kg*K)
%   u       -   Specific Internal Energy, kJ/kg
%   cv      -   Specific Heat at Constant Volume, kJ/(kg*K)
%   cp      -   Specific Heat at Constant Pressure, kJ/(kg*K)
%   h       -   Specific Enthalpy, kJ/kg
%   c       -   Speed of Sound, m/s
%   rho     -   Density, kg/m3
%   state   -   -1 = Negative Input, 0 = Liquid, 1 = Saturated, 2 = Gas
%   _l      -   Liquid designator
%   _v      -   Vapor designator
%--------------------------------------------------------------------------
% Revision History:
% Written for a CO2 Blowdown model developed at Utah State University by
% Brian Solomon
%--------------------------------------------------------------------------
% Based upon the Helmholtz Energy based equations of state described by
% Span, R. and Wagner, W. in
%  "A New Equation of State for Carbon Dioxide Covering the Fluid Region
%  from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa"
```

```
% Journal of Physical and Chemical Reference Data
% Vol 25, No. 6, 1996. Pp 1509–1596
%-------------------------------------------------------------------------

%% CALCULATE THE SATURATION PROPERTIES
%=========================================================================
% CO2 Constants
%-------------------------------------------------------------------------
R = 0.1889241;      % Gas constant, kJ/(kg*K)
Tt = 216.592;       % Triple point temperature, K   (Eq. 3.1)
Pt = 0.51795;       % Triple point pressure, MPa    (Eq. 3.2)
Tc = 304.1282;      % Critical temperature, K       (Eq. 3.3)
Pc = 7.3773;        % Critical pressure, MPa        (Eq. 3.4)
rhoc = 467.6;       % Critical density, kg/m3       (Eq. 3.5)
%{
%% Load the CO2 properties created by the NIST Webbook
%=========================================================================
[labels,T_NIST,y] = readColData('SatCO2Props.txt',25);
P_NIST      = y(:,1);       % Pressure (MPa)
rho_l_NIST  = y(:,2);       % Density (l, kg/m3)
rho_v_NIST  = y(:,14);      % Density (v, kg/m3)
u_l_NIST    = y(:,4);       % Internal Energy (l, kJ/kg)
u_v_NIST    = y(:,16);      % Internal Energy (v, kJ/kg)
h_l_NIST    = y(:,5);       % Enthalpy (l, kJ/kg)
h_v_NIST    = y(:,17);      % Enthalpy (v, kJ/kg)
s_l_NIST    = y(:,6);       % Entropy (l, J/g*K)
s_v_NIST    = y(:,18);      % Entropy (v, J/g*K)
cv_l_NIST   = y(:,7);       % Cv (l, J/g*K)
cv_v_NIST   = y(:,19);      % Cv (v, J/g*K)
cp_l_NIST   = y(:,8);       % Cp (l, J/g*K)
cp_v_NIST   = y(:,20);      % Cp (v, J/g*K)
c_l_NIST    = y(:,9);       % Sound Speed (l, m/s)
c_v_NIST    = y(:,21);      % Sound Speed (v, m/s)
%}
SatCO2Props = open('SatCO2Props.mat');
```

```
77  T_NIST      = SatCO2Props.T_NIST;       % Temperature (K)
78  P_NIST      = SatCO2Props.P_NIST;       % Pressure (MPa)
79  rho_l_NIST  = SatCO2Props.rho_l_NIST;   % Density (l, kg/m3)
80  rho_v_NIST  = SatCO2Props.rho_v_NIST;   % Density (v, kg/m3)
81  u_l_NIST    = SatCO2Props.u_l_NIST;     % Internal Energy (l, kJ/kg)
82  u_v_NIST    = SatCO2Props.u_v_NIST;     % Internal Energy (v, kJ/kg)
83  h_l_NIST    = SatCO2Props.h_l_NIST;     % Enthalpy (l, kJ/kg)
84  h_v_NIST    = SatCO2Props.h_v_NIST;     % Enthalpy (v, kJ/kg)
85  s_l_NIST    = SatCO2Props.s_l_NIST;     % Entropy (l, J/g*K)
86  s_v_NIST    = SatCO2Props.s_v_NIST;     % Entropy (v, J/g*K)
87  cv_l_NIST   = SatCO2Props.cv_l_NIST;    % Cv (l, J/g*K)
88  cv_v_NIST   = SatCO2Props.cv_v_NIST;    % Cv (v, J/g*K)
89  cp_l_NIST   = SatCO2Props.cp_l_NIST;    % Cp (l, J/g*K)
90  cp_v_NIST   = SatCO2Props.cp_v_NIST;    % Cp (v, J/g*K)
91  c_l_NIST    = SatCO2Props.c_l_NIST;     % Sound Speed (l, m/s)
92  c_v_NIST    = SatCO2Props.c_v_NIST;     % Sound Speed (v, m/s)
93  %% Interpolate to find the liquid and vapor properties
94  %=========================================================================
95  rho_l = interp1(T_NIST,rho_l_NIST,T,'spline');
96  rho_v = interp1(T_NIST,rho_v_NIST,T,'spline');
97  P     = interp1(T_NIST,P_NIST,T,'spline');
98  s_l   = interp1(T_NIST,s_l_NIST,T,'spline');
99  s_v   = interp1(T_NIST,s_v_NIST,T,'spline');
100 u_l   = interp1(T_NIST,u_l_NIST,T,'spline');
101 u_v   = interp1(T_NIST,u_v_NIST,T,'spline');
102 cp_l  = interp1(T_NIST,cp_l_NIST,T,'spline');
103 cp_v  = interp1(T_NIST,cp_v_NIST,T,'spline');
104 cv_l  = interp1(T_NIST,cv_l_NIST,T,'spline');
105 cv_v  = interp1(T_NIST,cv_v_NIST,T,'spline');
106 h_l   = interp1(T_NIST,h_l_NIST,T,'spline');
107 h_v   = interp1(T_NIST,h_v_NIST,T,'spline');
108 c_l   = interp1(T_NIST,c_l_NIST,T,'spline');
109 c_v   = interp1(T_NIST,c_v_NIST,T,'spline');
110
111 %% CALCULATE THE PROPERTIES AT THE GIVEN TEMPERATURE AND DENSITY
```

```
112 %=========================================================================
113 % Account for negative density or temperature
114 %-------------------------------------------------------------------------
115 if rho < 0 || T < 0
116     X = NaN;
117     P = NaN;
118     s_v = NaN; u_v = NaN; cp_v = NaN;
119     cv_v = NaN; h_v = NaN; c_v = NaN;
120     s_l = NaN; u_l = NaN; cp_l = NaN;
121     cv_l = NaN; h_l = NaN; c_l = NaN;
122     s = s_v*X + s_l*(1-X);
123     u = u_v*X + u_l*(1-X);
124     h = h_v*X + h_l*(1-X);
125     cp = cp_v*X + cp_l*(1-X);
126     cv = cv_v*X + cv_l*(1-X);
127     c = NaN;
128     state = -1;
129 % SATURATED
130 %-------------------------------------------------------------------------
131 else
132     X = (rho_v/rho)*((rho_l-rho)/(rho_l-rho_v));
133     s = s_v*X + s_l*(1-X);
134     u = u_v*X + u_l*(1-X);
135     h = h_v*X + h_l*(1-X);
136     cp = cp_v*X + cp_l*(1-X);
137     cv = cv_v*X + cv_l*(1-X);
138     c = c_v*X + c_l*(1-X);
139     state = 1;
140 end
141
142 %% CREATE THE OUTPUT STRUCTURE
143 %=========================================================================
144 Props.P     = P;                                          % Pressure
145 Props.X     = X;                                          % Quality
146 Props.s     = s;    Props.s_l   = s_l;  Props.s_v   = s_v;  % Entropy
```

```
147 Props.u     = u;    Props.u_l   = u_l;  Props.u_v   = u_v;  % Internal Energy
148 Props.cv    = cv;   Props.cv_l  = cv_l; Props.cv_v  = cv_v; % Specific ...
        Heat at Constant Volume
149 Props.cp    = cp;   Props.cp_l  = cp_l; Props.cp_v  = cp_v; % Specific ...
        Heat at Constant Pressure
150 Props.h     = h;    Props.h_l   = h_l;  Props.h_v   = h_v;  % Enthalpy
151 Props.c     = c;    Props.c_l   = c_l;  Props.c_v   = c_v;  % Speed of Sound
152 Props.rho_l = rho_l;Props.rho_v = rho_v;                    % Density
153 Props.state = state;                                       % State
154 %Props.gma_l = cp_l/cv_l;
155 %Props.gma_v = cp_v/cv_v;
156 end
```

B.4 CO_2 Properties - Helmholtz vs. NIST Plotter

```
1  clc; clear all;
2
3  % Create a temperature and density vector to calculate CO2 properties at
4  %-------------------------------------------------------------------------
5  T = linspace(216.59,300,50);      % Temperature vector, K (-27.67 to 116.33F)
6  %T = [T linspace(300,304.1,15)];
7  T = [T linspace(300,307,50)];
8  %rho = linspace(200,1100);   % Density vector, kg/m3
9  rho = 500;
10
11 % Load the CO2 properties created by the NIST Webbook
12 %-------------------------------------------------------------------------
13 [labels,x,y] = readColData('SatCO2Props.txt',25);
14 n = 3;
15 T_NIST = downsample(x,n);
16 P_NIST      = downsample(y(:,1),n);
17 rho_l_NIST  = downsample(y(:,2),n);
18 rho_v_NIST  = downsample(y(:,14),n);
```

```
19 u_l_NIST    = downsample(y(:,4),n);
20 u_v_NIST    = downsample(y(:,16),n);
21 h_l_NIST    = downsample(y(:,5),n);
22 h_v_NIST    = downsample(y(:,17),n);
23 s_l_NIST    = downsample(y(:,6),n);
24 s_v_NIST    = downsample(y(:,18),n);
25 cv_l_NIST   = downsample(y(:,7),n);
26 cv_v_NIST   = downsample(y(:,19),n);
27 cp_l_NIST   = downsample(y(:,8),n);
28 cp_v_NIST   = downsample(y(:,20),n);
29 c_l_NIST    = downsample(y(:,9),n);
30 c_v_NIST    = downsample(y(:,21),n);
31
32 % Loop through temperature and density vectors to populate the CO2 properties
33 %-------------------------------------------------------------------------
34 for i = 1:length(T)
35     for j = 1:length(rho)
36         Props = CO2Props(T(i),rho(j));
37         P(i,j)      = Props.P;      % Pressure
38         X(i,j)      = Props.X;      % Quality
39         s(i,j)      = Props.s;      % Entropy
40         s_l(i,j)    = Props.s_l;
41         s_v(i,j)    = Props.s_v;
42         u(i,j)      = Props.u;      % Internal Energy
43         u_l(i,j)    = Props.u_l;
44         u_v(i,j)    = Props.u_v;
45         cv(i,j)     = Props.cv;     % Specific Heat at Constant Volume
46         cv_l(i,j)   = Props.cv_l;
47         cv_v(i,j)   = Props.cv_v;
48         cp(i,j)     = Props.cp;     % Specific Heat at Constant Pressure
49         cp_l(i,j)   = Props.cp_l;
50         cp_v(i,j)   = Props.cp_v;
51         h(i,j)      = Props.h;      % Enthalpy
52         h_l(i,j)    = Props.h_l;
53         h_v(i,j)    = Props.h_v;
```

```
54          c(i,j)      = Props.c;      % Speed of Sound
55          c_l(i,j)    = Props.c_l;
56          c_v(i,j)    = Props.c_v;
57          rho_l(i,j)  = Props.rho_l;  % Density
58          rho_v(i,j)  = Props.rho_v;
59          state(i,j)  = Props.state;  % State
60      end
61  end
62
63  %Shift energy terms so that the reference matches
64  %-------------------------------------------------------------------------
65  u_diff = u_l_NIST(1) - u_l(1)
66  s_diff = s_l_NIST(1) - s_l(1)
67  h_diff = h_l_NIST(1) - h_l(1)
68
69  % Change the independent variables to temperature and pressure.
70  %-------------------------------------------------------------------------
71  %[rhoArr gmaArr] = CO2tp(T_NIST, P_NIST);
72
73  % Parse mosster data CO2 density
74  %-------------------------------------------------------------------------
75  rho_parse=-0.136928567045648.*T.^2+68.960274348176327.*T-7677.810285569415;
76
77  % Plot the pressure vs temperature
78  %-------------------------------------------------------------------------
79  figure(1);
80  plot(T,P(:,1),'b'); hold on; grid on;
81  plot(T_NIST,P_NIST,'ob');
82  %title('CO2 Pressure');
83  legend('Helmholtz','NIST','Location','SE');
84  xlabel('Temperature, K'); ylabel('Pressure, MPa');
85  saveas(1,'CO2_Pressure.png')
86  %saveas(1,'LaTeX_Plots/CO2_Pressure.eps','epsc')
87
88  % Plot the density vs temperature
```

```
89 %-------------------------------------------------------------------------
90 figure(2);
91 plot(T,rho_l(:,1),'b'); hold on; grid on;
92 plot(T_NIST,rho_l_NIST,'ob');
93 plot(T,rho_v(:,1),'--r')
94 plot(T_NIST,rho_v_NIST,'sr')
95 %plot(T,rho_parse);
96 %title('CO2 Density');
97 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
98 %legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST','Monster');
99 xlabel('Temperature, K'); ylabel('Density, kg/m3');
100 saveas(2,'CO2_Density.png')
101 %saveas(2,'LaTeX_Plots/CO2_Density.eps','epsc')
102
103 % Plot the internal energy vs temperature
104 %-------------------------------------------------------------------------
105 figure(3);
106 plot(T,u_l(:,1),'b'); hold on; grid on;
107 plot(T_NIST,u_l_NIST,'ob');
108 plot(T,u_v(:,1),'--r')
109 plot(T_NIST,u_v_NIST,'sr')
110 title('CO2 Internal Energy');
111 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
112 xlabel('Temperature, K'); ylabel('Internal Energy, kJ/kg');
113 saveas(3,'CO2_Internal_Energy.png')
114 %saveas(3,'LaTeX_Plots/CO2_Internal_Energy.eps','epsc')
115
116 % Plot the enthalpy vs temperature
117 %-------------------------------------------------------------------------
118 figure(4);
119 plot(T,h_l(:,1),'b');hold on; grid on;
120 plot(T_NIST,h_l_NIST,'ob');
121 plot(T,h_v(:,1),'--r')
122 plot(T_NIST,h_v_NIST,'sr')
123 %title('CO2 Enthalpy');
```

```
124 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST','Location','SE');
125 %legend('Liquid-NIST','Vapor-NIST');
126 xlabel('Temperature, K'); ylabel('Enthalpy, kJ/kg');
127 saveas(4,'CO2_Enthalpy.png')
128 %saveas(4,'LaTeX_Plots/CO2_Enthalpy.eps','epsc')
129
130 % Plot the entropy vs temperature
131 %-------------------------------------------------------------------------
132 figure(5);
133 plot(T,s_l(:,1),'b'); hold on; grid on;
134 plot(T_NIST,s_l_NIST,'ob');
135 plot(T,s_v(:,1),'--r')
136 plot(T_NIST,s_v_NIST,'sr')
137 %title('CO2 Entropy');
138 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
139 xlabel('Temperature, K'); ylabel('Entropy, J/g*K');
140 saveas(5,'CO2_Entropy.png')
141 %saveas(5,'LaTex_Plots/CO2_Entropy.eps','epsc')
142
143 % Plot the specific heat at constant volume vs temperature
144 %-------------------------------------------------------------------------
145 figure(6);
146 plot(T,cv_l(:,1),'b'); hold on; grid on;
147 plot(T_NIST,cv_l_NIST,'ob');
148 plot(T,cv_v(:,1),'--r')
149 plot(T_NIST,cv_v_NIST,'sr')
150 title('CO2 Specific Heat at Constant Volume');
151 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
152 xlabel('Temperature, K'); ylabel('Specific Heat at Constant Volume, J/g*K');
153 saveas(6,'CO2_Specific_Heat_at_Constant_Volume.png')
154 %saveas(6,'LaTeX_Plots/CO2_Specific_Heat_at_Constant_Volume.eps','epsc')
155
156 % Plot the specific heat at constant pressure vs temperature
157 %-------------------------------------------------------------------------
158 figure(7);
```

```
159 plot(T,cp_l(:,1),'b'); hold on; grid on;
160 plot(T_NIST,cp_l_NIST,'ob');
161 plot(T,cp_v(:,1),'--r')
162 plot(T_NIST,cp_v_NIST,'sr')
163 title('CO2 Specific Heat at Constant Pressure');
164 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
165 xlabel('Temperature, K'); ylabel('Specific Heat at Constant Pressure, ...
        J/g*K');
166 saveas(7,'CO2_Specific_Heat_at_Constant_Pressure.png')
167 %saveas(7,'LaTeX_Plots/CO2_Specific_Heat_at_Constant_Pressure.eps','epsc')
168
169
170 % Plot the speed of sound vs temperature
171 %-------------------------------------------------------------------------
172 figure(8);
173 plot(T,c_l(:,1),'b'); hold on; grid on;
174 plot(T_NIST,c_l_NIST,'ob');
175 plot(T,c_v(:,1),'--r')
176 plot(T_NIST,c_v_NIST,'sr')
177 title('CO2 Speed of Sound');
178 legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
179 xlabel('Temperature, K'); ylabel('Speed of Sound, m/s');
180 saveas(8,'CO2_Speed_of_Sound.png')
181 %saveas(8,'LaTeX_Plots/CO2_Speed_of_Sound.eps','epsc')
182
183 % Plot the enthalpy vs density
184 %-------------------------------------------------------------------------
185 figure(9);
186 %plot(T,h_l(:,1),'b');
187 plot(rho_l,h_l,'--b');hold on; grid on;
188 %plot(T,h_v(:,1),'r')
189 plot(rho_v,h_v,'r')
190 %title('CO2 Enthalpy');
191 %legend('Liquid-Helmholtz','Liquid-NIST','Vapor-Helmholtz','Vapor-NIST');
192 legend('Liquid-Helmholtz','Vapor-Helmholtz');
```

```
193 xlabel('Denisty, kg/m3'); ylabel('Enthalpy, kJ/kg');
194 saveas(9,'CO2_Enthalpy_vs_Density.png')
195 %saveas(9,'LaTex_Plots/CO2_Enthalpy_vs_Density.eps','epsc')
```

B.5 Density and Enthalpy to Temperature

```
1  function [T] = CO2_rho_h_2T(rho, h, T_guess)
2
3  % Purpose:
4  %------------------------------------------------------------------------
5  % This function places a nonlinear solver around the CO2Props function in
6  % order to change the independant variable from T & rho to rho & h.
7  %========================================================================
8  % Inputs:
9  %------------------------------------------------------------------------
10 % rho   -   Density, kg/m3
11 % h     -   Enthalpy, kJ/kg
12 %========================================================================
13 % Outputs:
14 %------------------------------------------------------------------------
15 % T     -   Temperature, K
16 %========================================================================
17
18 rho_Known   = rho;  % Save rho to a more intuitive name
19 h_Known     = h;    % Save h to a more intuitive name
20 %T_guess     = 300;  % Guess value for T for lsqnonlin to use
21
22 % Create a function for lsqnonlin to solve T(rho,h)
23 pFunc = @(T_Unknown) getfield(CO2Props(T_Unknown,rho_Known),'h')-h_Known;
24 %pFunc = @(T_Unknown) ...
       getfield(CO2PropsNIST(T_Unknown,rho_Known),'h')-h_Known;
25 % Sinse T_Unknown is not pre defined in pFunc, lsqnonlin will find a T for
26 % rho_Known and h_Known
```

```
27 T = lsqnonlin(pFunc,T_guess,0,inf,optimset('Display','off'));
28
29 end
```

B.6 Enthalpy Plotter

```
1  clc; clear all;
2
3  % Creats a plot of C02 temperature vs density for different enthalpies
4
5  % Create a temperature and density vector to calculate CO2 properties at
6  %-------------------------------------------------------------------------
7  %Tarr = linspace(240,320);      % Temperature vector, K (-27.67 to 116.33F)
8  T = 250;
9  rho = linspace(100,1000);   % Density vector, kg/m3
10 %rho = 1000;
11
12 % Loop through temperature and density vectors to populate the CO2 properties
13 %-------------------------------------------------------------------------
14 for i = 1:length(T)
15     for j = 1:length(rho)
16         Props = CO2Props(T(i),rho(j));
17         h(i,j)      = Props.h;          % Save Enthalpy
18     end
19 end
20
21 % Create a vector of enthalpies to plot vs density and temperature
22 h_Known = h(1:15:length(h));
23 % Preallocate an array to save the calculated temperature to
24 Tsave = zeros(length(h_Known),length(rho));
25
26 % Calculate the temperature at the previously defined enthalpies and their
27 % corrosponding densities by looping through them and saving temperature
```

```
for iter = 1:length(h_Known)
    for k = 1:length(rho)
        Tsave(iter,k) = CO2_rho_h_2T(rho(k), h(iter), 300);
    end
end

% Plot temperature vs density at each of the enthalpies
figure(1);
plot(rho,Tsave(1,:),':b'); hold on; grid on;
plot(rho,Tsave(2,:),'--r');
plot(rho,Tsave(3,:),'b');
plot(rho,Tsave(4,:),'--r');
plot(rho,Tsave(5,:),'b');
plot(rho,Tsave(6,:),'--r');
plot(rho,Tsave(7,:),'b');
%title('CO2 Temperature vs Density at Specific Enthalpies');
legend(['h = ' num2str(h_Known(1),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(2),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(3),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(4),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(5),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(6),3) ' kJ/kg'], ...
       ['h = ' num2str(h_Known(7),3) ' kJ/kg'], ...
       'Location','SE');
xlabel('Density, kg/m3'); ylabel('Temperature, K');
saveas(1,'CO2 Temp vs Density for Diff Enthalpies.png')
```

B.7 Discrete Wavelet Transform Derivative

```
function dudx=derivative_dwt(u,wt_name,wt_level,dx,trt_flag)
%
% Differentiation (Derivative) of Sampled Data Based on Discrete Wavelet ...
    Transform
```

```
4  %
5  % dudx=derivative_dwt(u,wt_name,wt_level,dx)
6  %
7  % u:         uniformly-sampled data
8  % wt_name:   name of the wavelet function (haar or spl)
9  % wt_level:  level of wavelet decomposition
10 % dx:        sampling interval (default=1)
11 % trt_flag:  flag of translation-rotation transformation for boundary ...
       effect (default=1)
12 % dudx:      differentiations of data (u)
13 %
14 %See also
15 %            derivative_cwt
16 %
17 % Reference:
18 % J. W. Luo, J. Bai, and J. H. Shao,
19 % "Application of the wavelet transforms on axial strain calculation in ...
       ultrasound elastography,"
20 % Prog. Nat. Sci., vol. 16, no. 9, pp. 942-947, 2006.
21
22 if nargin<5
23     trt_flag=1;
24 end
25 if nargin<4
26     dx=1;
27 end
28
29 if trt_flag
30     x=(1:length(u))*dx;
31     a=(u(end)-u(1))/(x(end)-x(1));
32     b=u(1)-a*x(1);
33     u=u-a*x-b;
34 else
35     a=0;
36 end
```

```
37
38 wt_name=lower(wt_name);
39
40 if strcmp(wt_name,'haar')
41     h0=[sqrt(2)/2 sqrt(2)/2];  %the decomposition low-pass filter
42     h1=[-sqrt(2)/2 sqrt(2)/2]; %the decomposition high-pass filter
43 elseif strcmp(wt_name,'spl')
44     h0=[0.125 0.375 0.375 0.125]*sqrt(2);
45     h1=[-2 2]*sqrt(2);
46 else
47     error('wavelet name error');
48 end
49
50 y0=u;
51
52 % Algorithme a Trous
53 for n=1:wt_level
54     h0_atrous=[h0' zeros(length(h0),2^(n-1)-1)]';
55     h0_atrous=h0_atrous(1:(length(h0)-1)*(2^(n-1)-1)+length(h0));
56
57     h1_atrous=[h1' zeros(length(h1),2^(n-1)-1)]';
58     h1_atrous=h1_atrous(1:(length(h1)-1)*(2^(n-1)-1)+length(h1));
59
60     y1=conv(y0,h1_atrous);
61     y0=conv(y0,h0_atrous);
62 end
63
64 index=round(length(y1)/2-length(u)/2)+[1:length(u)];
65 dudx=y1(index);
66
67 wt_scale=2^wt_level;
68
69 if strcmp(wt_name,'haar')
70     dudx=-dudx/wt_scale^(3/2)*4;
71 elseif strcmp(wt_name,'spl')
```

```
72      dudx=-dudx/wt_scale^(3/2);
73  else
74      error('wavelet name error');
75  end
76
77  dudx=dudx/dx+a;
```