Investigation of Aluminum Equation of State Generation

Aaron Ward
Marquette University

INVESTIGATION OF ALUMINUM EQUATION OF STATE GENERATION

by

Aaron J. Ward

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

December 2011

ABSTRACT
INVESTIGATION OF ALUMINUM EQUATION OF STATE GENERATION

Aaron J. Ward

Marquette University, 2011

There are many forms and methods to construct equations of state, EOSs. These methods are usually tailored for the particular problem of interest. Here, the EOSs of interest are those used in modeling shock responses. These EOSs cover a wide range of physical characteristics such as detonation and explosions, armor and anti-armor materials, and space structures protection. Aluminum will be the primary focus of this work. Aluminum was chosen because it has been studied in great length in the shock regime and is a common component in shock experiments and space type vehicles.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this study is to examine how different equations of state (EOSs) perform in the shock regime. The EOSs will be evaluated against available experimental data to determine their performance. The experimental data includes shock Hugoniots, release isentropes, reshock experiments and phase diagrams. This study includes investigations into complete and incomplete EOSs. Here incomplete EOSs refer to an EOS that requires an additional relationship to fully describe the thermodynamic state. The simulated EOS results were generated with a Fortran code written specifically for this project. This study will focus on aluminum, but methods presented throughout can be applied to most metallic materials.

1.1 Shock Physics

An understanding of the behavior of materials under high-strain rates is necessary to grasp the EOS concepts as they relate to shock physics. While not the focus of this research, a short review of shock physics follows. For a more in-depth overview of

shock physics see references [1–4].

Shock waves represent a rapid change between the undisturbed state and the shock excited state. Conservation of mass, momentum, and energy are performed across the shock wave. The relationships derived from the conservation equations are commonly referred to as the Rankine-Hugoniot or jump equations shown below.

$$\frac{U_s - u_{p_0}}{V_1} = \frac{U_s - u_{p_1}}{V_0} \tag{1.1}$$

$$P_1 - P_0 = \frac{U_s u_{p_1}}{V_0} \tag{1.2}$$

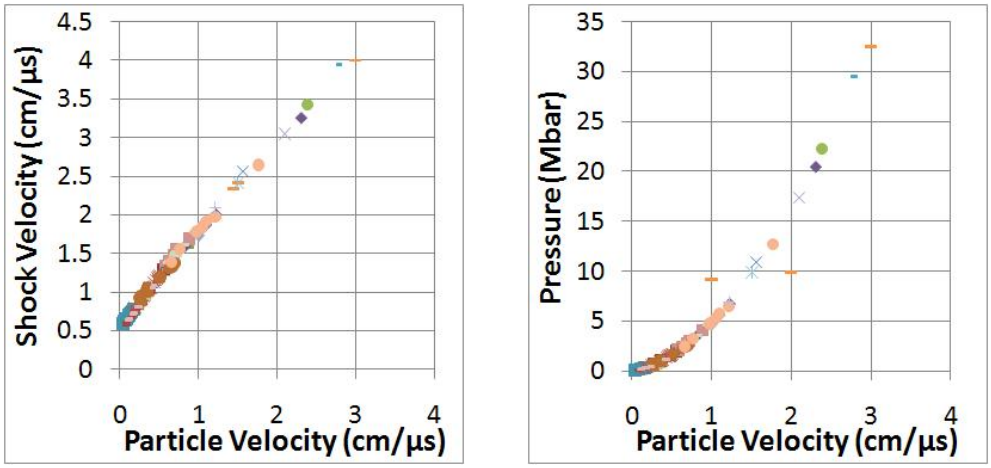
$$E_1 - E_0 = \frac{1}{2}\left(P_1 + P_0\right)\left(V_0 - V_1\right) \tag{1.3}$$

Along the principle Hugoniot P_0, E_0, and u_{p_0} are zero and V_0 is the initial known density. This leaves three equations with four unknowns (u_p, U_s, P, V). A fourth relationship is necessary to solve for the shock state. A common assumption is $U_s = f(u_p)$. This relationship represents the locus of possible shock states, it is not a path-dependent process. The U_s-u_p relationship is commonly determined through extensive experimental studies. A common linear fit to the U_s-u_p is shown in Equation 1.4

$$U_s = C_0 + S_1 u_p \tag{1.4}$$

Only the three conservation equations and the particle-shock velocity relationship are needed to begin solving problems. This is the foundation to solve shock physics problems. At this point it is helpful to manipulate the above equations to form a series of curves. The three common Hugoniot curves are Shock Velocity (U_s) - Particle Velocity (u_p), Pressure (P) - Particle Velocity (u_p), and Pressure (P) - Specific Volume (V). Experimental shock Hugoniots from references [5–28] are shown in Figure

1.1.



(a) U_s - u_p **(b)** P - u_p

(c) P - V

Figure 1.1 Experimental Hugoniot Data from References [5-28]

Analytical solutions to P-u_p and P-V Hugoniots can also be found by inserting the U_s - u_p relationship into the conservation equations. P-u_p and$P$-V solutions are shown in Equations 1.5 and 1.6

$$P = \rho_0 c_0 u_{p1} + \rho_0 s u_{p1}^2 \tag{1.5}$$

$$P = \frac{c_0^2 (V_0 - V)}{[V_0 - s_1 (V_0 - V)]^2} \tag{1.6}$$

Another important matrix in shock studies are rarefaction waves or release waves. Once a shock wave reaches a free surface, pressure returns to atmospheric. This sends a release wave back through the shocked material. A common experimental technique is to measure the velocity response of the free-surface, gaining insight to the release behavior. Also under sufficient loading, the material may release to a different material state. It is not uncommon to observe vaporization or melting. Experimental release data is shown in Figure 1.2 [21, 29]. The lines in Figure 1.2 do not represent the release path but merely connect the shocked and released states.

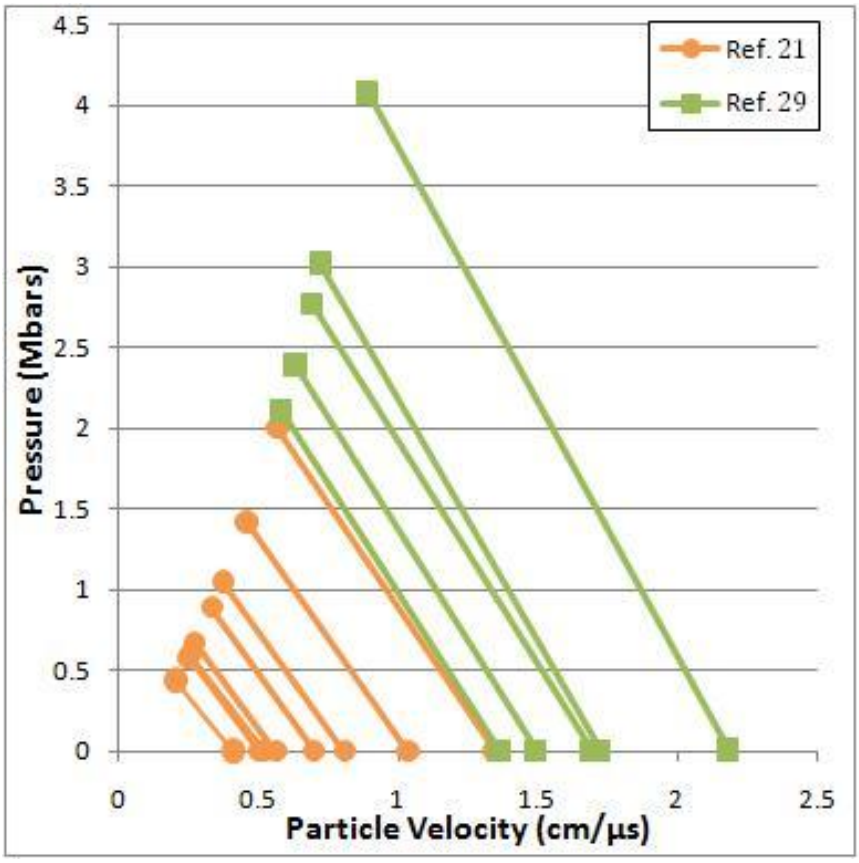
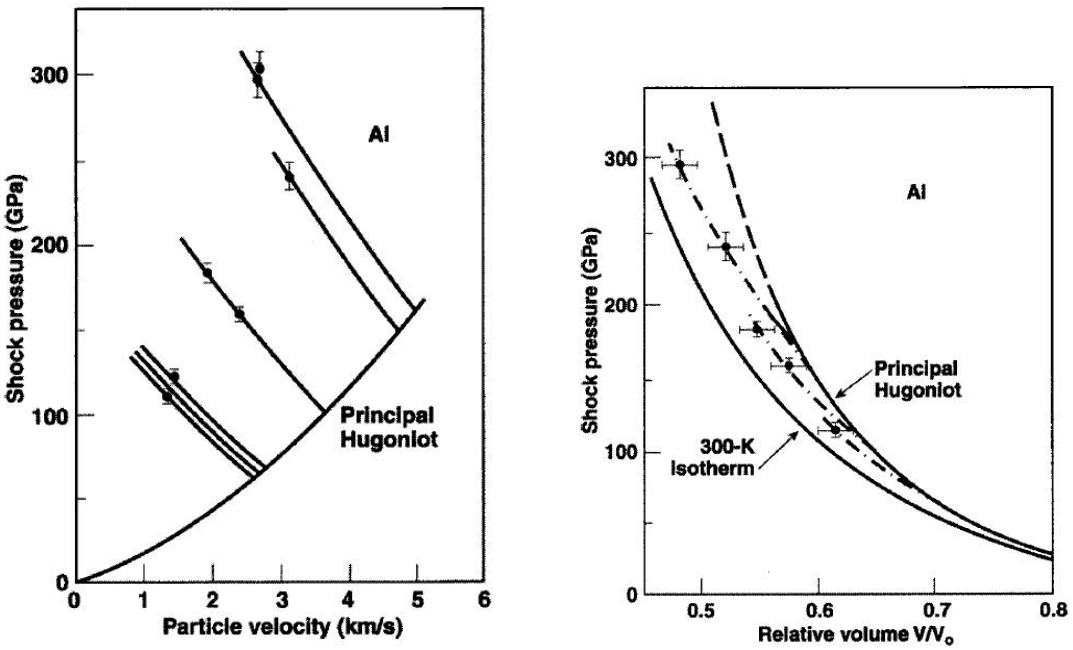


Figure 1.2 Experimental Release Data from References [21,29]

In addition to shock Hugoniot and release isentropes, reshock experimental data is compared. Experimental data from Nellis et. al [30] is shown in Figure 1.3. A reshock experiment is a common shock experiment where a second compression wave occurs after the material undergoes an initial shock. The reshock is commonly a result of impedance increase at a material interface.



(a) P - u_p **(b)** P - V

Figure 1.3 Nellis Mie-Grüneisen Reshock Results from Reference [30]

1.1.1 Hugoniot Calculations

In this work the Hugoniot relationships are calculated using the EOS and the conservation equations. The EOS results are then compared to the available experimental data. First it is assumed that the material is initially at normal conditions, i.e. $P_0 = 0$, $E_0 = 0$, and $u_p = 0$, and these conditions correspond to the principle

Hugoniot. Through some manipulation the conservation of mass and momentum, Equations 1.1 and 1.2 take the form below:

$$U_s = \frac{V_0}{V_0 - V} u_p \tag{1.7}$$

$$P = \frac{u_p^2}{V_0 - V} \tag{1.8}$$

$$u_p^2 = P\left(V_0 - V\right) \tag{1.9}$$

Using Equation 1.9 and Equation 1.3, energy can be rewritten in terms of particle velocity shown in Equation 1.10.

$$E = \frac{u_p^2}{2} \tag{1.10}$$

Now pressure and energy are expressed as a function of particle velocity and specific volume. The Hugoniot states are found by substituting Equation 1.10 into the EOS and finding where P_{EOS} and P from the Hugoniot relationships are the same, shown in Equation 1.11.

$$\begin{aligned} P_{EOS} - P &= 0 \\ P_{EOS} - \frac{u_p^2}{V_0 - V} &= 0 \end{aligned} \tag{1.11}$$

1.1.2 Release Isentrope Calculations

The release process is commonly modeled isentropic. Unlike Hugoniots this process is path specific. The isentrope is calculated starting with the 2nd law of thermodynamics, Equation 1.12.

$$TdS = dE - PdV \tag{1.12}$$

By definition on an isentrope the change in entropy is zero; i.e. $TdS = 0$. Equation 1.12 is numerically differentiated and rearranged to Equation 1.13.

$$P = \frac{dE}{dV} = \frac{\Delta E}{\Delta V} \tag{1.13}$$

$$\frac{E_{i+1} - E_i}{V_{i+1} - V_i} = P_i \tag{1.14}$$

$$E_{i+1} = E_i + P_i \left(V_{i+1} - V_i\right) \tag{1.15}$$

In Equation 1.15 specific volume and pressure are assumed known and energy is calculated. If the path is reversed, this method can be used to calculate isentropic compression results. The free-surface velocity is calculated using Equation 1.16

$$u_{FS} = u_H - \int_{V_o}^{V} \left(-\frac{\partial P}{\partial V}\right)_S^{1/2} dV \tag{1.16}$$

1.1.3 Reshock Calculations

The reshock calculations are broken into two parts, the initial shock and the reshock. The initial shock is calculated using the same methods described in the principle Hugoniot calculation section. From the initial shock the reshock is calculated using the u_p, P_1, and E_1 from the initial shock as the starting state for the reshock. The reshock state is found using a method similar to the principle Hugoniot calculation but where the initial pressure, energy, and particle velocity are not equal to zero. Equations 1.17-1.20 are the full conservation equations used to calculate the reshock state.

$$\left(U_s - u_0\right) = \left(u_1 - u_0\right) \frac{\rho_1}{\rho_1 - \rho_0} \tag{1.17}$$

$$P_1 - P_0 = \rho \left(U - u_0\right) \left(u_1 - u_0\right) \tag{1.18}$$

$$P_1 = P_0 + \frac{\left(u_1 - u_0\right)^2}{\left(V_0 - V_1\right)} \tag{1.19}$$

$$E_1 - E_0 = \frac{1}{2} \left(P_0 + P_1\right) \left(V_0 - V_1\right) \tag{1.20}$$

1.1.4 Multiphase Calculations

Some models include the necessary physics to model the phase change in the material. There are several different methods to accomplish this. First there are empirical relationships fitted to the multiphase boundaries. A common example is a Clausius-Clapeyron fit. This fit represents the saturation dome. The fit is commonly done independent of the EOS. The Linderman law is also an example of an empirical relationship. The Linderman law is a pressure-volume relationship used in calculating the melting transitions.

A more accurate approach to calculate the saturation region are Maxwell constructs [31]. Along a given isotherm the saturation pressure is found at the location where the area under and above the line are the same, shown in Figure 1.4 for the 5000 K isotherm. This method is used commonly in conjunction with a Van der Waals EOS.

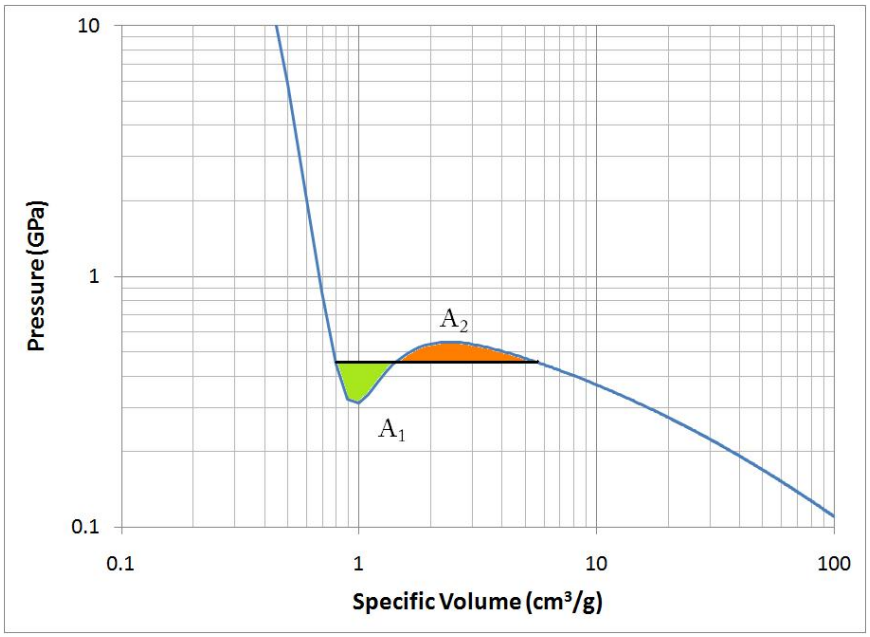


Figure 1.4 Maxwell Construct for 5000 K Isotherm

The final method for finding the stable state is using Gibb's free energy. For a detailed description on using Gibb's free energy for phase equilibrium see Wark [32]. In melting and polymorphic phase transitions, the state with the lowest Gibb's Free energy is the stable state. The saturation dome is found along an isotherm where the endpoints of the saturation pressure have the same Gibb's free energy.

Chapter 2

Equation of State

Equation of states have long been used to study material behaviors. A simple example of an EOS is the ideal gas equation, which relates pressure, specific volume and temperature. As the case with the ideal gas EOS, if volume and temperature are known, then pressure can be solved. If any two thermodynamic variables are known, the remaining thermodynamic variables can be found. It is important to note that most equations of state have limits. One would not want to apply the ideal gas equation in a region close to the saturation dome, where the ideal gas assumptions are not valid. So EOSs have evolved to problem-specific forms. EOS have been used to describe the material at the center of the earth and the behavior on suns that are galaxies away.

The primary focus of this work is on equations of state that describe materials in the shock regime. This is further broken down into two types, complete and incomplete EOS. A complete EOS, $F = f(T, V)$, is one that describes the thermodynamic state of a material. An incomplete EOS usually requires a second relationship to fully describe the thermodynamic state. Many commonly used shock physics EOSs are incomplete. These EOSs commonly take the form $P = f(E, V)$ and an addi-

tional relationship is needed to describe energy, temperature, and entropy. Typically $dE = Cp(dT)$ and the second law of thermodynamics, $TdS = dE$, are used to complete the EOS.

2.1 Mie-Grüneisen EOS

The Mie-Grüneisen EOS is a commonly used EOS in shock physics. It accurately represents the behavior of the Hugoniot relationships under modest compressions. Classically, the Mie-Grüneisen EOS considers only the effects of the lattice and zero-temperature contributions. Pressure and energy are determined from a given reference state; this could be room temperature, zero temperature, the Hugoniot, or other relationships. A detailed mathematical description of the Mie-Grüneisen EOS can be found in Gathers [3]. The Mie-Grüneisen EOS is commonly presented in the form shown in Equation 2.2. The Grüneisen parameter, γ_G, is assumed to be only a function of specific volume. The relationship for calculating γ_G is given in Equation 2.1. E_R and P_R are the given reference curves.

$$\gamma_G = V \left(\frac{\partial P}{\partial E} \right)_V \tag{2.1}$$

$$P - P_R = \frac{\gamma_G}{V} (E - E_R) \tag{2.2}$$

In many cases the Hugoniot is used as the reference curve. Under these circumstances Equation 2.2 can be rewritten as Equation 2.3. In Equation 2.3 γ_g/V is rewritten using the assumption given in Equation 2.4

$$P = \frac{\gamma_0}{V_0} E - P_H \left(1 - \frac{\gamma_0}{2} \frac{V_0 - V}{V}\right) \tag{2.3}$$

where

$$\frac{\gamma_G}{V} = \frac{\gamma_0}{V_0} = Constant \tag{2.4}$$

$$P_H = \frac{C_0^2 (V_0 - V)}{[V_0 - S (V_0 - V)]^2} \tag{2.5}$$

2.2 Tillotson EOS

The Tillotson EOS was developed to model hypervelocity impacts of metal materials [3, 33]. The Tillotson EOS divides the P-V phase space into four regions. The first region is the small compression (Elastic Compression) region. This region was not considered in the Tillotson EOS but merely mentioned. Region II is strong compression (Shock waves) where $V/V_0 < 1$ and $E > 0$. Region III is the release state from a strong compression but no phase change occurs, here $V/V_0 < V_s$ and $E < E'_s$. E'_s is defined as the energy required to have "gas-like behavior". In Region IV the material is released from a strong enough shock state to create phase change in the material where $1 < V/V_0 < V_s$ for $E > E'_S$ and V/V_0 for $E > 0$. The transition between the solid and the gas phase regions is shown in Figure 2.1

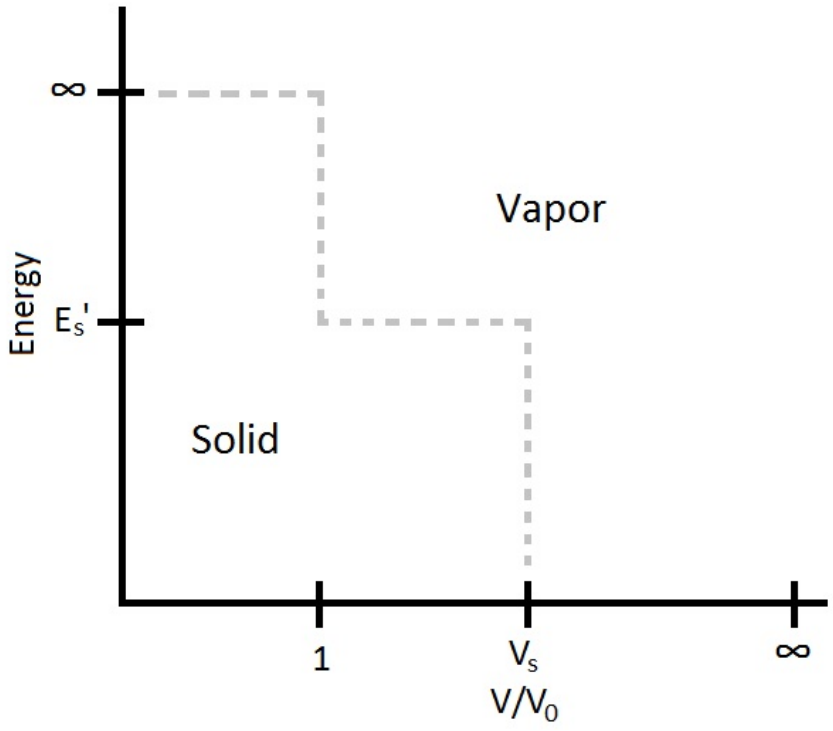


Figure 2.1 Tillotson EOS phase regions

For regions II ($V/V_0 < 1$ for $E > 0$) and III ($V/V_0 < V_s$ for $E < E'_s$)

$$P = \left(a + \frac{b}{\frac{E}{E_0\eta^2} + 1}\right)\frac{E}{V} + A\mu + B\mu^2 \tag{2.6}$$

where:

$\eta = \rho/\rho_0 = V_0/V$

$\mu = \eta - 1$

a, b, A, B are constants

The constants a, b, and A are fitted to a general EOS taking the form $P = \gamma(E, E) + E/V + f(V)$. The constant A is defined as $A = C^2/V_0$ and the constants a and b follow the conditions $a + b = \gamma_0$. The remaining constant, B, is reserved to

adjust the EOS to best fit the thermodynamic $P - V - E$ surface.

For region IV ($1 < V/V_0 < V_s$ for $E > E'_s$ and $V/V_0 > V_s$ for $E > 0$)

$$P = aE\rho + \left\{ \frac{bE\rho}{\frac{E}{E_0\eta^2} + 1} + A\mu e^{-\beta[(V/V_0)-1]} \right\} e^{-\alpha[(V/V_0)-1]^2} \tag{2.7}$$

where:

α and β are constants for ideal gas convergence

The Tillotson EOS formulation allows for the treatment of vaporization and other physics-driven changes in the material response, without a large increase in mathematical complexities, but no considerations are made about the melting process or the liquid phase. The material is either all solid or all vapor in this approach.

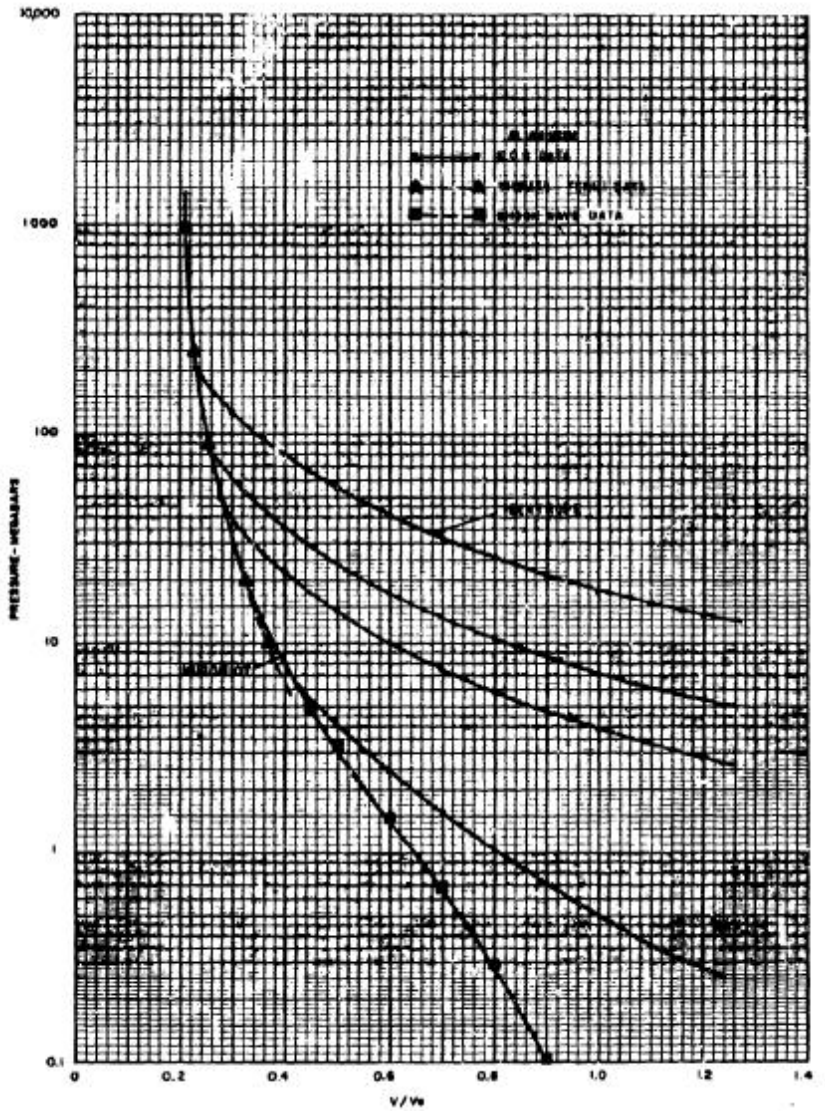


Figure 2.2 Hugoniot and Release Isentropes Results from Tillotson [33]

2.3 Multi-Branch Analytical EOS

The Multi-branch Analytic EOS (MBEOS) [3, 34] is a multi-phase EOS originally developed for Lithium. The EOS is broken into three regions, $\rho \geq \rho_0$, $\rho \leq \rho_0, E > E_c$, and $\rho \leq \rho_o, E < E_c$. Where E_c is the critical point energy on the saturation dome.

Figure 2.3 shows how the regions are divided up over ρ-E space.

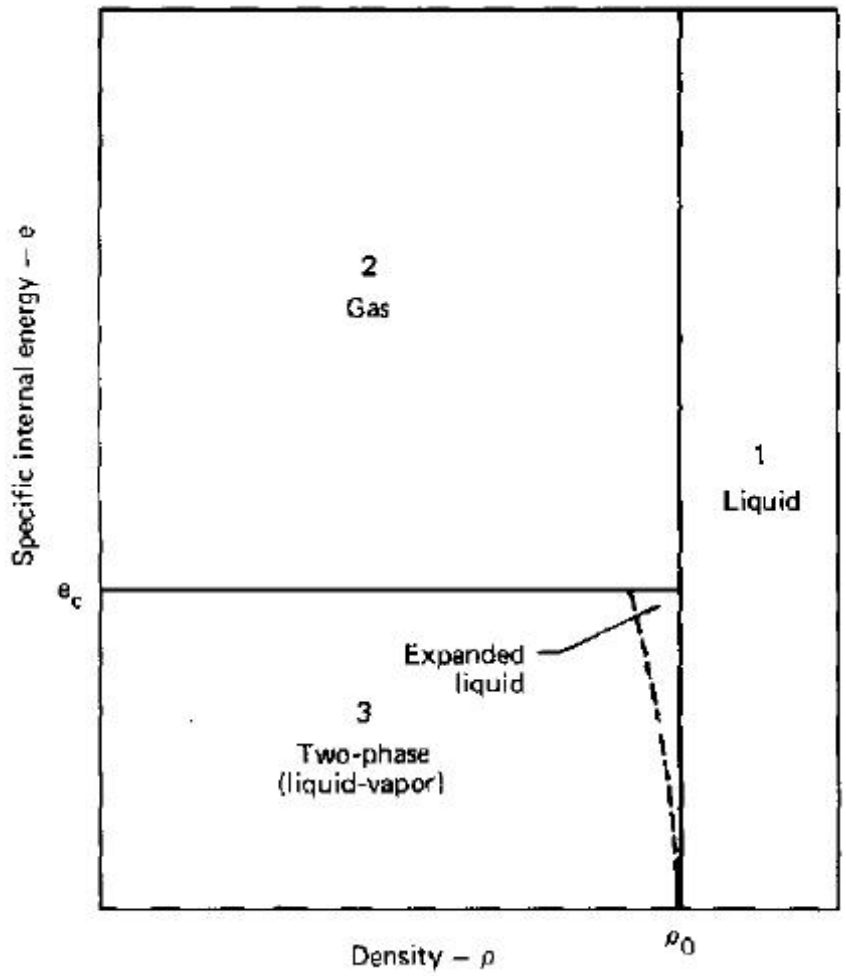


Figure 2.3 MBEOS Phase Space from Reference [3]

Region I: $\rho \geq \rho_0$

This region is the solid phase response, it is represented by the Mie-Grüneisen EOS arranged in a slightly different form than previously shown.

$$P = P_H \left(1 - \frac{\gamma_0}{2}\eta\right) + \gamma_0 \rho_0 E \tag{2.8}$$

where:

$$P_H = \frac{\rho_0 c_0^2 \eta}{(1 - s\eta)^2} \qquad \eta = 1 - \frac{\rho_0}{\rho} \tag{2.9}$$

Region II: $\rho \leq \rho_0, E > E_c$

This region lies well above the critical point, such that an ideal gas assumption and ionization is considered. The ionization term, $g(E)$, models the dissociation of the electron shell surrounding the Lithium nucleus.

$$P(\rho, E) = \frac{2}{3}(E - E_c)\rho g(E) + \left[\gamma_0 \rho_0 E - \frac{2}{3}(E - E_c)\rho_0 g(E)\right](\rho/\rho_0)^k \tag{2.10}$$

$$g(E) = 1 - 0.7e^{\left[-2(y-1.233)^2\right]} - 0.7e^{\left[-0.5(y-4.468)^2\right]} \tag{2.11}$$

where :

$$y = ln(E/E_c) \tag{2.12}$$

The constant k is fitted from tabular values in the expansion region. Region III: $\rho \leq \rho_0, E < Ec$

Region III is the region that contains the saturation dome and expanded liquid state. In the expanded liquid region, the region between ρ_0 and the saturation dome, is modeled using Equation 2.13. This allows for a smooth transition between the saturation dome and Region I. The saturation dome is determined through a Clausius-Clapeyron type fit, shown in Equation 2.13. P_c and E_c are the critical saturation pressure and energy. Constants m and n describe the saturation dome. For Lithium, m and n were fitted from tabulated data. On the vapor side of the saturation dome Region II and the saturation dome are joined using Equation 2.15.

$$P = \rho_0 C_0^2 \left(\rho/\rho_0 - 1\right) + \gamma_0 \rho_0 E \tag{2.13}$$

$$P = P_c exp\left(m - nE_c/E\right) \tag{2.14}$$

$$P = \gamma_0 \rho_0 E \left(\rho/\rho_0\right)^k \tag{2.15}$$

2.4 Bushman-Lomonosov EOS

The Bushman-Lomonosov EOS originates from *Intense Dynamic Loading of Condensed Matter* by A.V. Bushman et. al. [1]. This model was later modified by Lomonosov [35]. The EOS is a complete EOS taking a free energy, F, approach. The EOS further separates free energy into three components; the cold compression curve (F_c), atomic contributions (F_a), and electronic contributions (F_e). The cold and atomic contributions are further broken down into liquid and solid relationships.

$$F\left(V,T\right) = F_c\left(V\right) + F_a\left(V,T\right) + F_e\left(V,T\right) \tag{2.16}$$

The cold compression curve corresponds to the material behavior at the theoretical 0 K temperature state. Equation 2.17 behaves such that at $E_c\left(V_{0c}\right) = 0$ and at $E_c\left(V \to \infty\right) = E_s$, E_s is the sublimation energy. This assures that Equations 2.17 and 2.18 join smoothly. The value V_{0c} is the specific volume at zero pressure. The variable σ_c is the ratio of the specific volume at zero pressure over the current specific volume. The a_i variables are fitting constants for the solid phase cold curve.

$$F_c\left(V\right) = 3V_{0c} \sum_{i=1}^{5} \frac{a_i}{i} \left(\sigma_c^{i/3} - 1\right) \tag{2.17}$$

The liquid region cold curve differs between Bushman and Lomonosov, shown in Equations 2.18 and 2.19. Both curves join with the solid curve at $\sigma_c \geq 1$. The parameters A, B, m, n, l in Equation 2.18 are fitting parameters. The parameters used

in Lomonosov's equation, A_c, B_c, C_c, m, n, l, are also fitting constants.

$$F_c(V) = V_{0c}\left[A\left(\frac{\sigma_c^m}{m} - \frac{\sigma_c^l}{l}\right) + B\left(\frac{\sigma_c^n}{n} - \frac{\sigma_c^l}{l}\right)\right] + E_{sub} \tag{2.18}$$

$$F_c(V) = V_{0c}\left[A_c\frac{\sigma_c^m}{m} + B_c\frac{\sigma_c^n}{n} + C_c\frac{\sigma_c^l}{l}\right] + E_{sub} \tag{2.19}$$

Atomic or lattice contributions are divided into two phases, solid and liquid. The solid phase represents the vibration of atoms in a crystal lattice. Both Bushman and Lomonosov use the same relationships to describe solid atomic contributions, shown in Equations 2.20 and 2.21. These equations are high temperature approximations of the Debye theory. In Equation 2.21, Θ is an empirical relationship representing the characteristic temperature. The characteristic temperature represents vibrational frequency of the atoms in the lattice structure, for details see chapter 13 of Reference [36]. The variable B_s and D_s are constants fit to match the Grüneisen coefficient dependent on specific volume. Here ξ is defined by $\chi = ln\sigma$.

$$F_a(V,T) = 3RTln\left\{\Theta(\sigma)/T\right\} \tag{2.20}$$

$$\Theta(\sigma) = \Theta_{0s}\sigma^{2/3}exp\left\{\frac{\left(\gamma_{0s} - \frac{2}{3}\right)\left(B_s^2 + D_s^2\right)}{B_s}tan^{-1}\left[\frac{\chi B_s}{B_s^2 + D_s(\chi + D_s)}\right]\right\} \tag{2.21}$$

The atomic contributions in the liquid phases are shown in the below equations. Bushman and Lomonosov use the same relationship with the exception of the calculation of the characteristic temperature, Θ. The liquid atomic contributions are broken into two parts, F_t and F_m. The anharmonicity of the atoms at high temperatures is represented in the F_t term. F_m models the melting curve and the liquid-state near the the melting curve. F_m also models the change in density that occurs during melting.

$$F_a(V,T) = F_t(V,T) + F_m(V,T) \tag{2.22}$$

The terms that describe the anharmonicity at high temperature are shown in the Equation 2.23. The leading terms, $3R/2$, ensure the liquid model asymptotes to the ideal gas assumptions. The variables T_{sa}, T_{ca}, B_l, and D_l are parameters determined from shock compression experiments. The variable Θ_{0l} is found from $\Theta_a(0) = T_{ca}$. The difference between the Bushman, Equation 2.24, and Lomonosov, Equation 2.25, methods are the addition of the $T_{ca}\Theta$ to Θ in the Lomonosov method.

$$F_t(V,T) = \frac{3RT}{2}\left[1 + \frac{\sigma T_T}{(\sigma+\sigma_T)(T+T_T)}\right] ln\left\{\frac{\Theta(\sigma,T)}{T}\right\} \tag{2.23}$$

$$\Theta(\sigma,T) = T_{sa}\sigma^{2/3}\frac{\Theta_a(\sigma)+T}{T_{ca}+T} \tag{2.24}$$

$$\Theta(\sigma,T) = T_{sa}\sigma^{2/3}\frac{T_{ca}\Theta_a(\sigma)+T}{T_{ca}+T} \tag{2.25}$$

$$\Theta_a(\sigma) = \Theta_{0l} exp\left\{\frac{\left(\gamma_{0l}-\frac{2}{3}\right)(B_l^2+D_l^2)}{B_l} arctan\left[\frac{\chi B_l}{B_l^2+D_l(\chi+D_l)}\right]\right\} \tag{2.26}$$

The remaining liquid atomic contributions are the melting effect. Both Bushman and Lomonosov use the same form, shown in Equation 2.27. The relative density of the liquid phase is defined as $\sigma_l = \sigma/\sigma_m$. The constants σ_m and T_m are the specific volume ratio and temperature of melting at standard pressure.

$$F_m(V,T) = 3R\left\{\frac{2\sigma_l^2 T_m}{1+\sigma_l^3}\left[C_m + \frac{3A_m}{5}\left(\sigma_l^{5/3}-1\right)\right] + (B_m - C_m)T\right\} \tag{2.27}$$

The last component to the Bushman and Lomonosov models is the electronic contribution. Bushman and Lomonosov use the same relationships, shown in Equations 2.28-2.31. The electronic contributions are the same for the liquid and solid phases. At high temperatures, the electronic heat capacity, β, asymptotes to $3RZ/2$ which corresponds to an ideal electron gas with complete ionization.

$$F_e(V,T) = -c_e(\sigma,T)Tln\left\{1 + \frac{2\sigma^{-\gamma_e(\sigma,T)}}{3RTZ}\int^T\int^\tau \beta(\tau)\,d\tau dT\right\} \tag{2.28}$$

$$c_e\left(V,T\right)=\frac{3R}{2}\left[Z+\frac{\sigma_Z T_Z^2\left(1-Z\right)}{\left(\sigma+\sigma_Z\right)\left(T^2+T_Z^2\right)}exp\left\{-\frac{\tau_i\left(\sigma\right)}{T}\right\}\right] \tag{2.29}$$

$$\beta\left(T\right)=\beta_i+\left(\beta_0-\beta_i+\beta_m\frac{T}{T_b}\right)exp\left\{-\frac{T}{T_b}\right\} \tag{2.30}$$

$$\gamma_e\left(\sigma,T\right)=\frac{2}{3}+\left(\gamma_0-\frac{2}{3}+\gamma_m\frac{T}{T_g}\right)exp\left\{-\frac{T}{T_g}-\frac{\left(\sigma-\sigma_e\right)^2}{\sigma\sigma_d}\right\} \tag{2.31}$$

Both Bushman and Lomonosov generated a wide array of results. Figure 2.4 are U_s-u_p Hugoniot results. Both models show good agreement with the presented experimental data. This includes the high pressure regions where melting may occur.

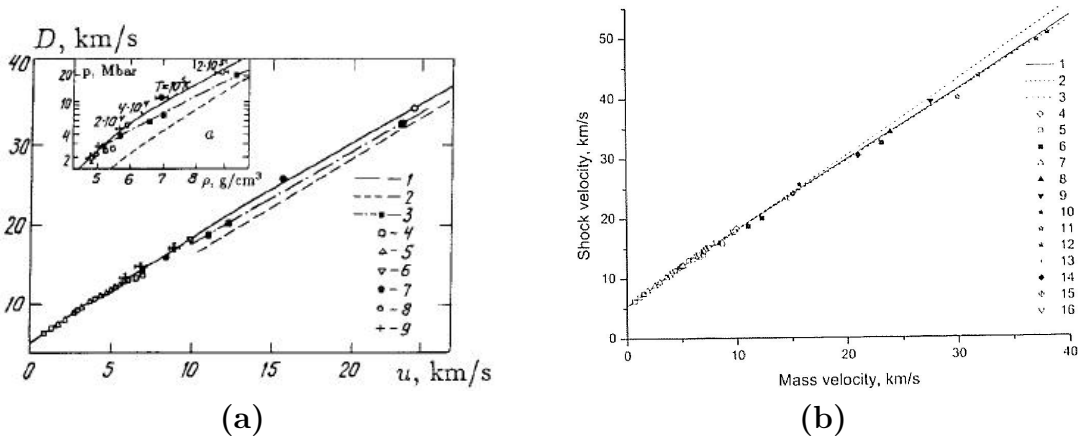


Figure 2.4 U_s-u_p Hugoniots from (a) Bushman [1] and (b) Lomonosov [35]

Lomonosov compared free surface velocity measurements in Figure 2.5. The model does a good job at lower free surface velocity but diverges significantly at higher values. The end result differs by approximately 10%.

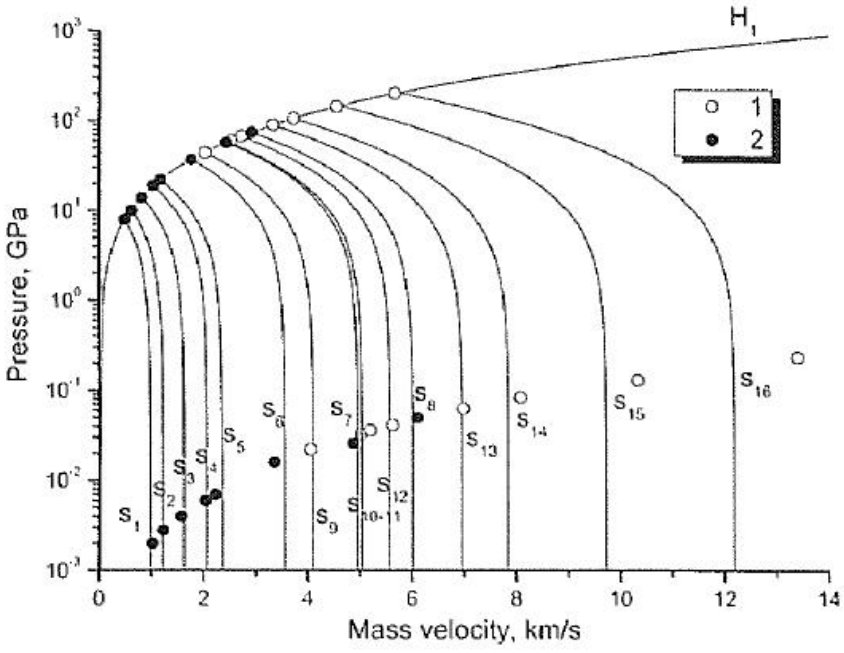


Figure 2.5 Release Isentropes Results from Lomonosov [35]

Both Bushman and Lomonosov made estimations of the critical properties. Bushman reported a critical pressure of 0.571 GPa, critical temperature 7222 K, and a critical specific volume of 1.24 cm^3/g. Lomonosov predicted a critical pressure of 0.197 GPa, critical temperature 5520 K, and a critical specific volume of 1.423 cm^3/g. The accompanying saturation domes are shown in Figure 2.6.

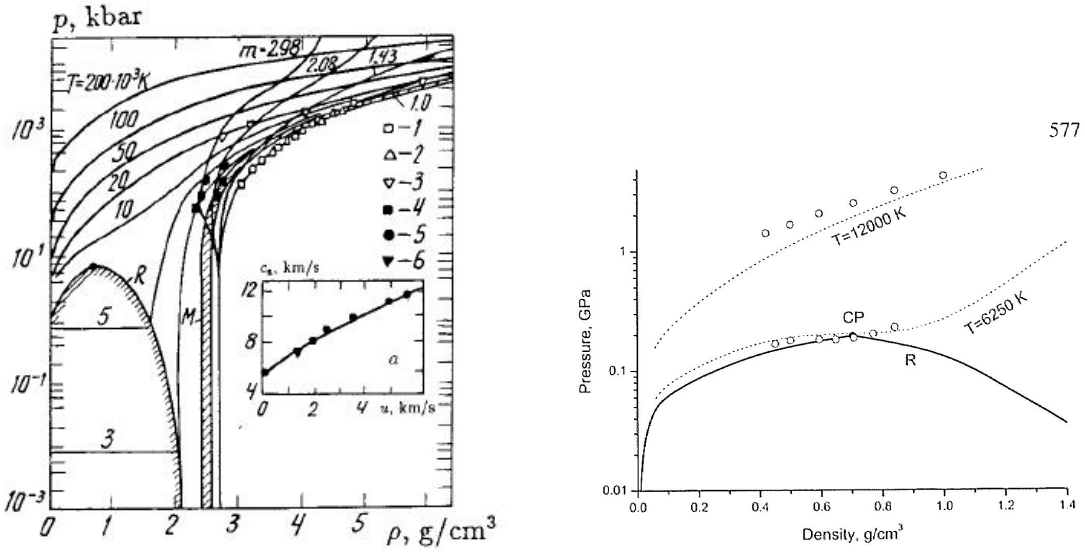


Figure 2.6 Saturation Domes from (a) Bushman [1] and (b) Lomonosov [35]

Chapter 3

Results

Several EOSs were simulated in this project, starting with the Mie-Grüneisen and Tillotson EOSs and moving to the more complex Multi-Branch and Bushman-Lomonosov EOSs. The Mie-Grüneisen, Tillotson, and Multi-Branch EOSs are all incomplete EOS while Bushman-Lomonosov EOS is a complete EOS.

The Mie-Grüneisen, Tillotson EOS, and the Multi-Branch EOS also used a shared program using the same routines to calculate Hugoniots and isentropes. Each EOS had its own module that included routines for initializing and calculating the thermodynamic state using the given inputs. The Bushman-Lomonosov EOS was considerably more complex. It was more straightforward to create an independent program to model the Bushman-Lomonosov EOS. The source code for both programs are given in the accompanying appendixes.

3.1 Mie-Grüneisen EOS

The first EOS modeled is the Mie-Grüneisen (MGR) equation of state. The Mie-Grüneisen was chosen as the starting point because it is a commonly used EOS in shock physics. As mentioned earlier the Mie-Grüneisen, given in Equation 3.1, uses a reference curve to model the pressure-volume space. This reference curve can take many forms but this work will look at a zero-Kelvin fit, room temperature fit, and the shock Hugoniot. One of the drawbacks to using the MGR EOS is that phase change is not modeled. Only the cold and vibrational contributions are included in the model.

$$P - P_R = \frac{\gamma_G}{V}\left(E - E_R\right) \tag{3.1}$$

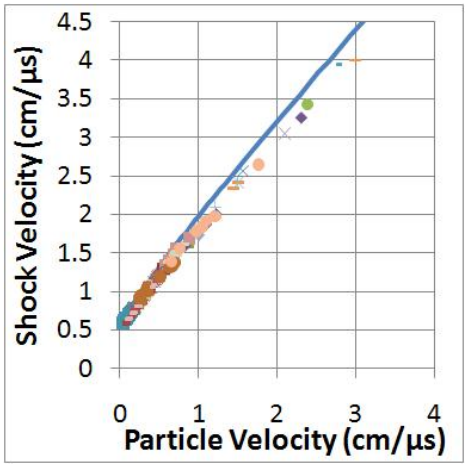
The zero-Kelvin fit used in this work is taken from the Bushman-Lomonosov EOS. The Mie-Grüneisen EOS was generated using the cold curve shown in Equation 3.2 and using the parameters listed in Table 3.1 taken from Reference [35]. The curve is constrained so that the energy and pressure are zero at V_{0c}. The pressure term is calculated by the $P_c\left(V\right) = -\frac{dE_c(V)}{dV}$ relationship. The variable σ_c is the ratio of the specific volume at zero pressure over the current specific volume. This relationship is used as the reference curve in the Mie-Grüneisen EOS

$$E_c^s\left(V\right) = 3V_{0c}\sum_{i=1,5}\frac{a_i}{i}\left(\sigma_c^{i/3} - 1\right) \tag{3.2}$$

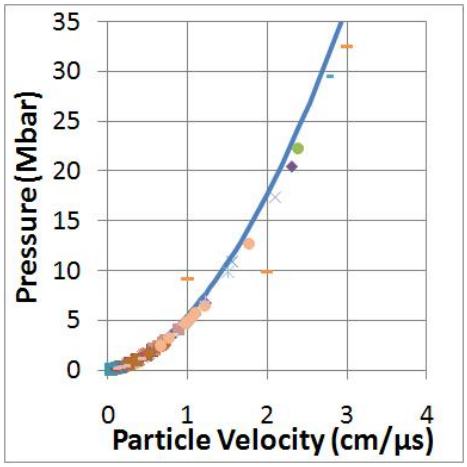
Table 3.1 Aluminum Zero-Kelvin Mie-Grüneisen EOS parameters

V_0 cm^3/g	γ_0	a_1	a_2	a_3	a_4	a_5
0.3614	2.19	326.35	-1035.44	858.51	-160.59	11.17

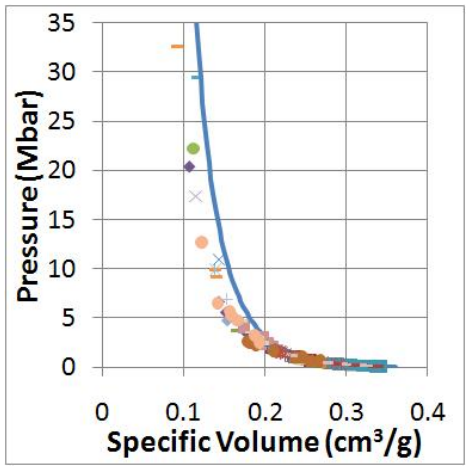
Results of zero-Kelvin Mie-Grüneisen EOS Hugoniot calculations are shown in Figure 3.1. The EOS does a reasonable job predicting the Hugoniot behavior.



(a) U_s - u_p



(b) P - u_p



(c) P - V

Figure 3.1 Zero Kelvin Mie-Grüneisen Hugoniot Plots. Experimental data from References [5-28]

The next method was a room temperature fit. This fit was described by Wilkin in reference [37]. This method defines energy and pressure as zero at room temperature (300 K) and $V/V_0 = 1$. The reference curve for energy and pressure are shown in Equations 3.3 and 3.4. Parameters used in the room temperature fit are shown in Table 3.2. The parameters were calculated using Hugoniot fit and the equations given in Equation 3.5. Here $x = 1 - V/V_0$.

$$\epsilon_0 = \epsilon_{00} + \epsilon_{01}x + \epsilon_{02}x^2 + \epsilon_{03}x^3 + \epsilon_{04}x^4 \tag{3.3}$$

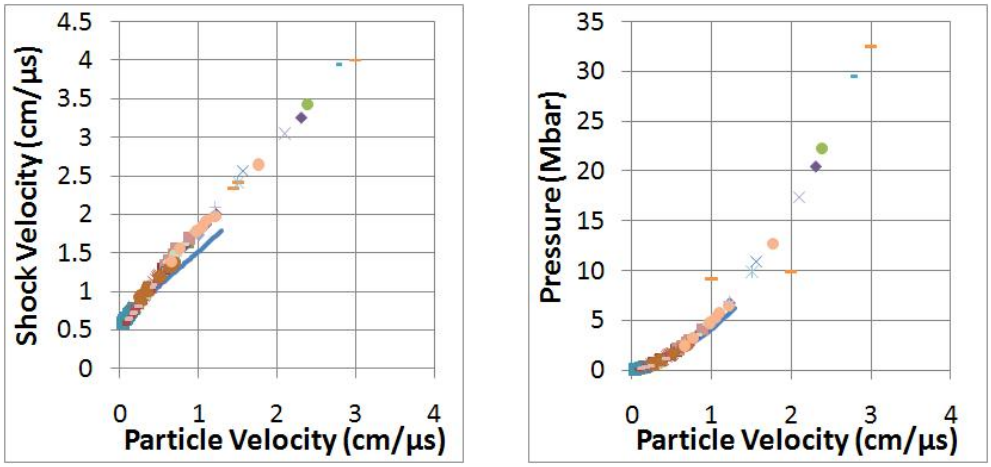
$$P_0 = -\frac{d\epsilon_0}{dV} = \frac{1}{V_0}\left(\epsilon_{01} + 2\epsilon_{02}x + 3\epsilon_{03}x^2 + 4\epsilon_{04}x^3\right) \tag{3.4}$$

$$\begin{aligned} e_0 &= e_{00} + e_{01}x + e_{02}x^2 + e_{03}x^3 + e_{04}x^4 \\ e_{00} &= -900R \\ e_{01} &= \gamma_0 e_{00} \\ e_{02} &= \frac{1}{2}\left(C + \gamma_0^2 e_{00}\right) \\ e_{03} &= \frac{1}{6}\left(4SC_0 + \gamma_0^3 e_{00}\right) \\ e_{04} &= \frac{1}{24}\left(-2\gamma_0 SC_0^2 + 18S^2C_0^2 + \gamma_0^4 e_{00}\right) \end{aligned} \tag{3.5}$$

Table 3.2 Aluminum Room Temperature Mie-Grüneisen EOS parameters

ρ_0 cm^3/g	γ_0	ϵ_{00}	ϵ_{01}	ϵ_{02}	ϵ_{03}	ϵ_{04}
2.712	2.14	-2.773e-3	-5.9342e-3	1.4226e-1	2.4498e-1	2.8429e-1

Results of room temperature Mie-Grüneisen EOS Hugoniot calculations are shown in Figure 3.2. The room temperature fit doesn't match the experimental data very well. It does give a reasonable match to low pressure region $P < 1Mbar$, but diverges from the experimental data at higher pressure.



(a) U_s - u_p **(b)** P - u_p

(c) P - V

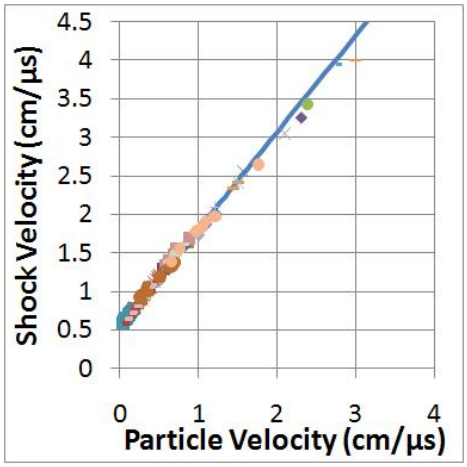
Figure 3.2 Room Temperature Mie-Grüneisen Hugoniot Plots. Experimental data from References [5-28]

The final method is the Mie-Grüneisen EOS using the Hugoniot as a reference curve. This is a common method for calculating the Mie-Grüneisen EOS. A linear $U_s - u_p$ was fitted to the experimental data and was used as the reference curve in the Mie-Grüneisen EOS. The parameters used in Mie-Grüneisen EOS are shown in Table 3.3.

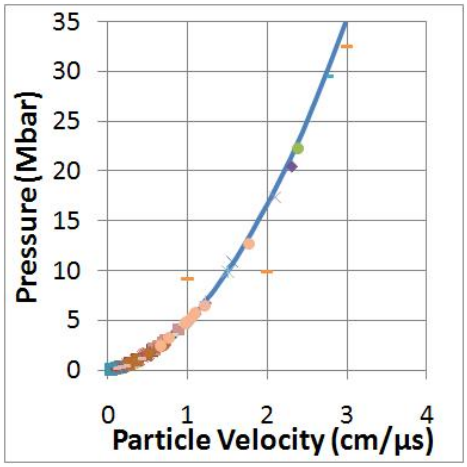
Table 3.3 Aluminum Mie-Grüneisen EOS parameters

ρ_0 g/cm^3	C_0 $cm/\mu s$	S	γ_0
2.712	0.54518	1.2592	2.14

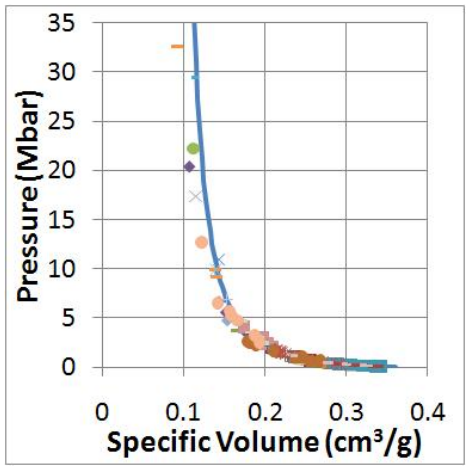
Results from the Hugoniot curve Mie-Grüneisen is shown in Figure 3.3. It gives good results matching the experimental principle Hugoniot. This is expected since the reference curve is the Hugoniot derived from the experimental data. The release isentropes, shown in Figure 3.4, match the low pressure states but don't do as well of a job at the higher states where phase change exists.



(a) U_s - u_p



(b) P - u_p



(c) P - V

Figure 3.3 Mie-Grüneisen Hugoniot Plots. Experimental data from References [5-28]

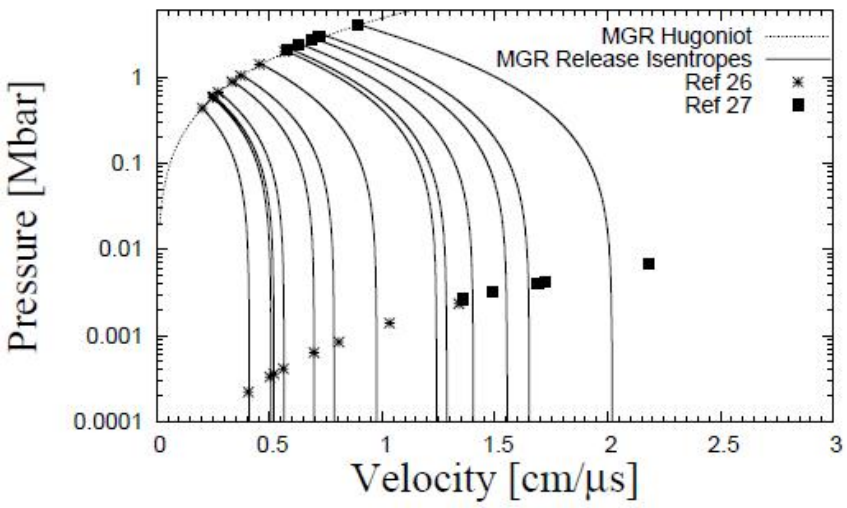
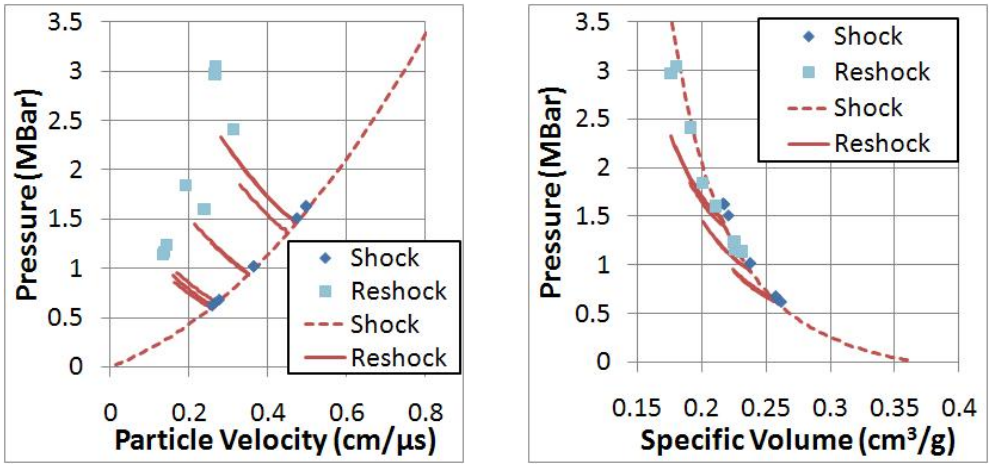


Figure 3.4 Mie-Grüneisen Release Isentrope Results. Experimental data from References [21-29]

Reshock results were calculated using the method described in the reshock section, Section 1.1.3. In the Mie-Grüneisen type EOSs the Grüneisen, γ, strongly influences the off-Hugoniot behavior. Using the Hugoniot, parameters listed in Table 3.3 Hugoniot reshocks were calculated, shown in Figure 3.5.



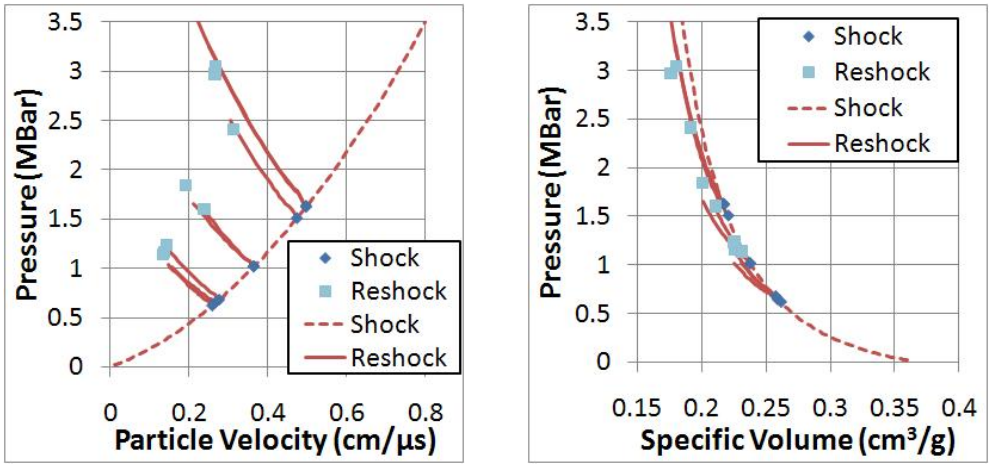
(a) P - u_p **(b)** P - V

Figure 3.5 Mie-Grüneisen Reshock Results. Experimental data from Reference [30].

In Reference [30], Nellis et. al., provided a U_s-u_p Hugoniot fit given in Table 3.4. Using these parameters the reshock experiments were recalcuated. The resulting P-u_p and P-V curves are shown in Figure 3.6. The Nellis fit more accurately matches the experimental data. This is largely due to the difference in the Grüneisen parameter.

Table 3.4 Aluminum Mie-Grüneisen EOS parameters from Reference [30]

ρ_0 g/cm^3	C_0 $cm/\mu s$	S	γ_0
2.712	0.5386	1.339	1.35



(a) P - u_p **(b)** P - V

Figure 3.6 Nellis Mie-Grüneisen Reshock Results. Experimental data from Nellis [30].

3.2 Tillotson EOS

The Tillotson EOS was modeled using the parameters found in Table 3.5. These parameters were taken from Ref. [33]. Along the the EOS parameters, Tillotson also provided some additional numerical results for several metals. The numerical results consist of the Hugoniot and several release isentropes. First the results from Tillotson's [33] report were compared to the data calculated using the EOS code. This was done to verify that the Tillotson EOS is being modeled correctly.

Table 3.5 Aluminum Tillotson EOS parameters

a	b	A Mbar	B Mbar	E_0 cm^3/g	α	β	E'_s $Mbarcm^3/g$	V_s^a cm^3/g
0.5	1.63	0.752	0.65	0.05	5.0	5.0	0.03	1.1

In Tillotson's report [33] a method for calculating the compression of materials and their release from high pressures was addressed. Tillotson reported on the principle Hugoniot and the release isentrope of several initial pressure/volume states. Calculation of the Hugoniot is straightforward and follows the approach demonstrated in Section 1.1.1.

To calculate the principle Hugoniot a Newton's solver was needed because particle velocity cannot be directly solved for in the compression portion of the Tillotson EOS shown in Equation 2.6. The Newton's solver, Equation 3.6, uses an initial guess, x_0, and calculates the desired function f and it's derivative f'. A better guess of the correct answer is returned as x_1. This process is repeated until the function is close to the desired solution. In this particular problem the function we are solving for is $f = P_{EOS} - P_{hugo}$, where P_{EOS} is the pressure calculated by EOS using $E = u_p^2/2$. P_{hugo} is the pressure calculated using the Hugoniot relationships, i.e. $P_{hugo} = u_p^2/(V_0 - V_1)$.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3.6}$$

Figure 3.7 shows the comparison between the data calculated by Tillotson and what was generated during this work. The calculated values are in good agreement

with Tillotson's data, assuring that the model is being properly implemented.

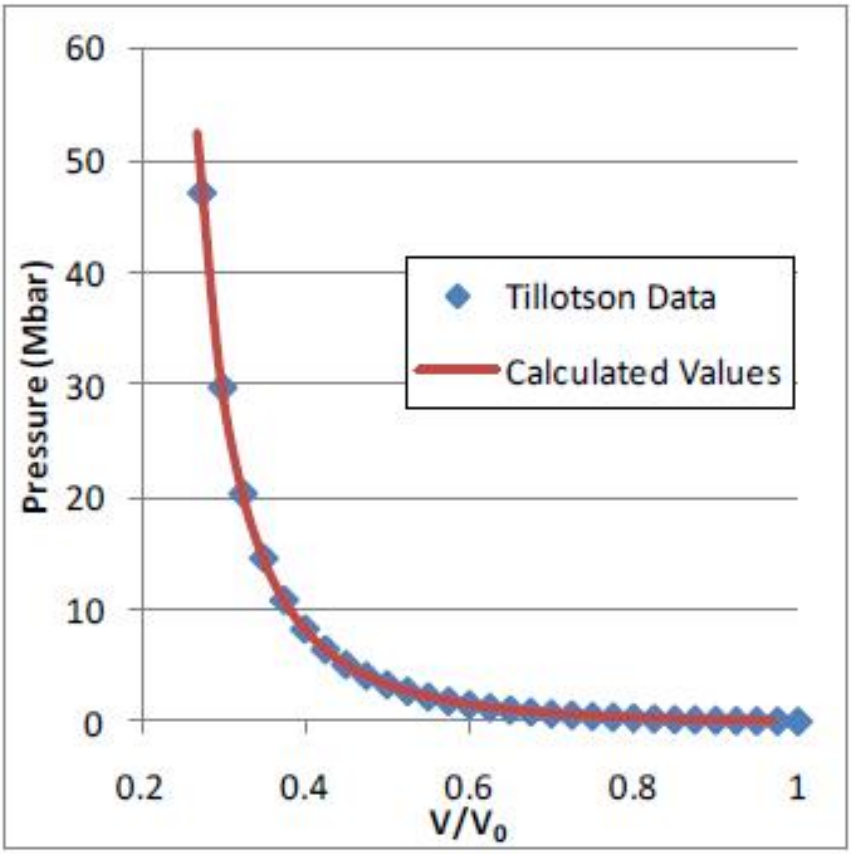


Figure 3.7 Tillotson Hugoniot comparision with numerical data

In addition to the principle Hugoniot data, Tillotson provides seven release isentropes. The isentropes were found using the method described in Section 1.1.2. Two of the seven isentropes are shown in Figure 3.8. These two isentropes were chosen because one release exhibits no phase change, Figure 3.8a, while the other releases to the vapor phase, Figure 3.8b. The dramatic change in slope seen in Figure 3.8b is a result of the material transition from solid to vapor states.

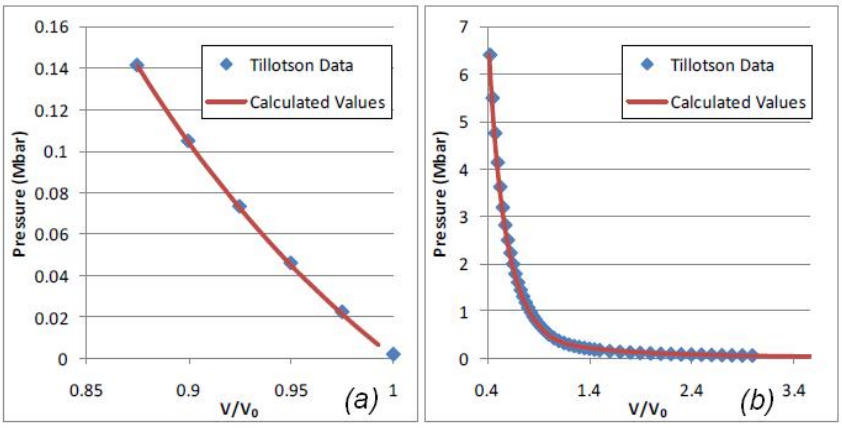
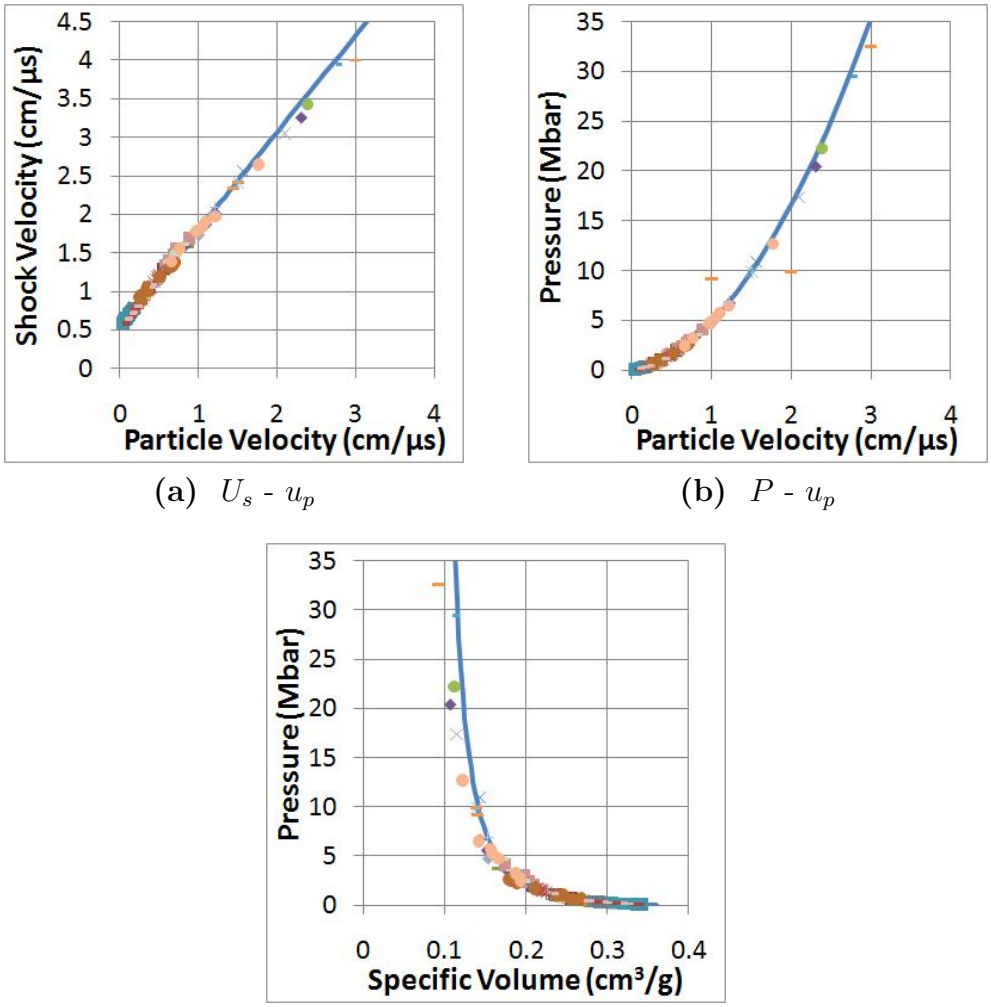


Figure 3.8 Tillotson release isentrope comparision with numerical data

In addition to the numerical data provided by Tillotson, the EOS was compared to experimental Hugoniot and release isentrope data shown in Figure 3.9. In the Tillotson EOS the Grüneisen parameter isn't limited to the fixed ratio, $\gamma/V = \gamma_0/V_0$. The Grüneisen parameter in the Tillotson EOS is a function of energy and specific volume. The change of the Grüneisen parameter along the hugoniot is shown in Figure 3.10



(a) U_s - u_p **(b)** P - u_p

(c) P - V

Figure 3.9 Tillotson Hugoniot Plots. Experimental data from References [5-28]

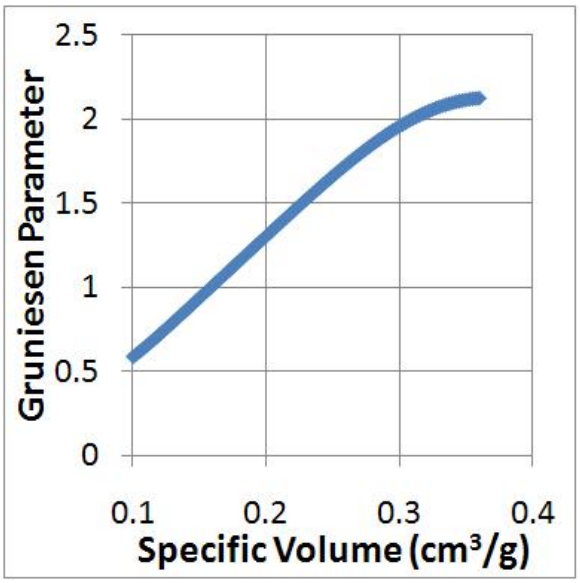


Figure 3.10 Tillotson EOS Grüneisen parameter

The Tillotson EOS does a good job of matching the release isentrope free-surface velocity shown in Figure 3.11. In general Tillotson matches the release isentropes better than most EOS because it is a two-phase model. The material transitions from all solid to all vapor on release, which is sufficient for high-pressure release cases where melting and the liquid state aren't thought to play a large role in the release isentrope.

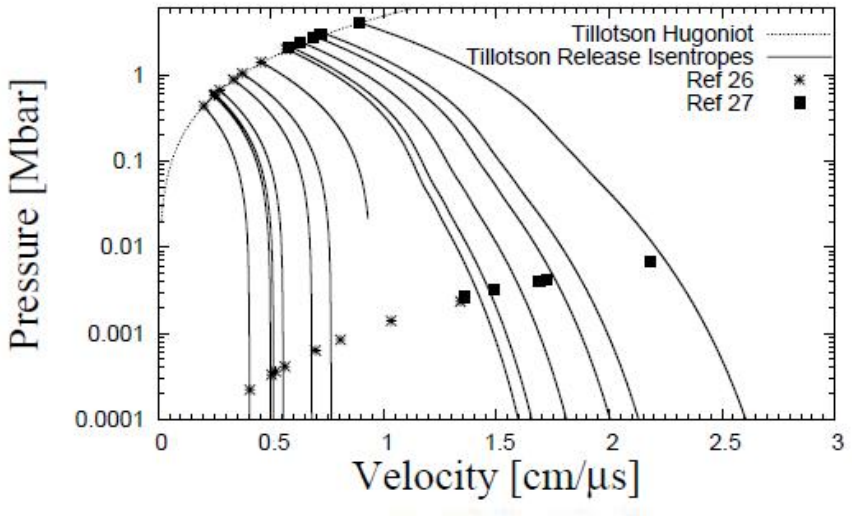
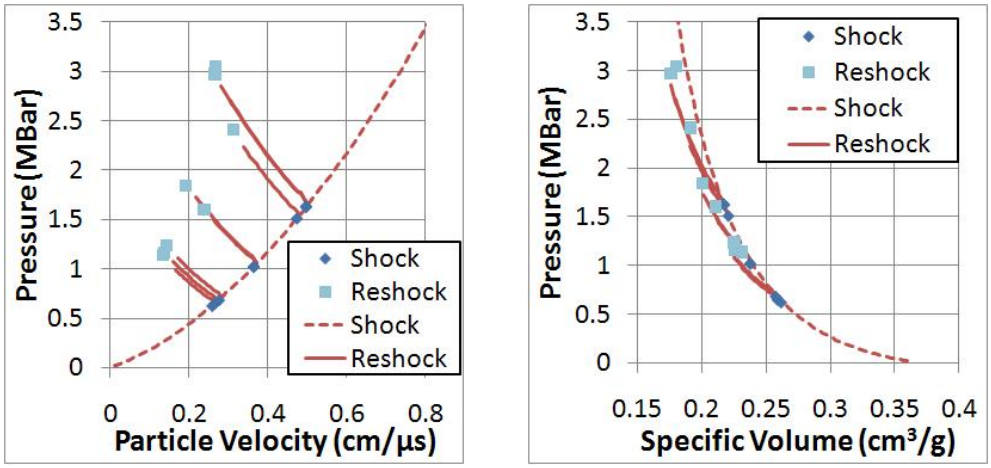


Figure 3.11 Tillotson Release Isentrope Results. Experimental data from References [21-29]

Reshock states for the Tillotson EOS are shown in Figure 3.12. The Tillotson EOS does a better job predicting the reshock state because the Grüneisen parameter varies with specific volume and energy.



(a) P - u_p **(b)** P - V

Figure 3.12 Tillotson Reshock Results. Experimental data from Reference [30].

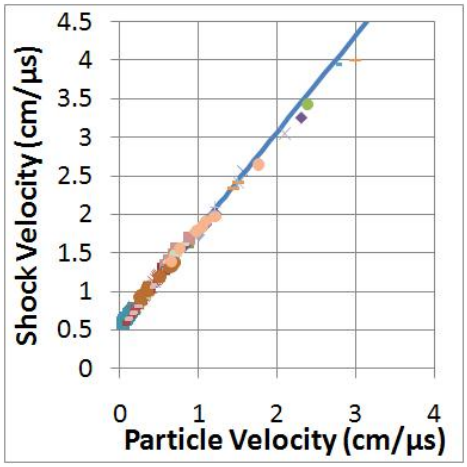
3.3 Multi-Branch Analytical EOS

The next EOS is the Multi-Branch Analytical EOS (MBEOS). The parameters used in the EOS are shown in Table 3.6. The effects of ionization shown in Equation 2.11 are neglected since it is particular to Lithium.

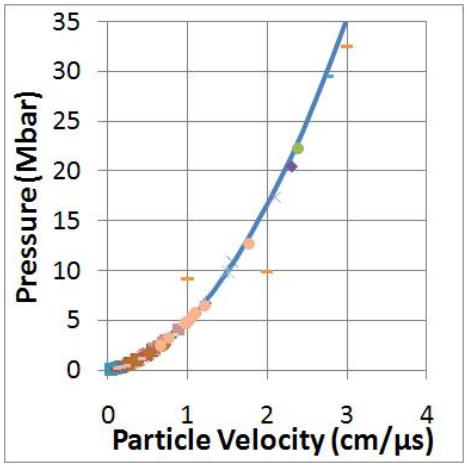
Table 3.6 Aluminum MBEOS parameters

ρ_0 g/cm^3	γ_0	T_0 K	P_c $Mbar$	E_c $Mbar * cm^3/g$	E_1 $Mbar * cm^3/g$	C_0 $cm/\mu s$	S
2.712	2.14	298	0.0018202	0.122	0.01	0.54518	1.2592
m	n	k	ξ	R $Mbar * cm^3/(gK)$			
1.0	1.1	1.1	2/3	3.08173e-6			

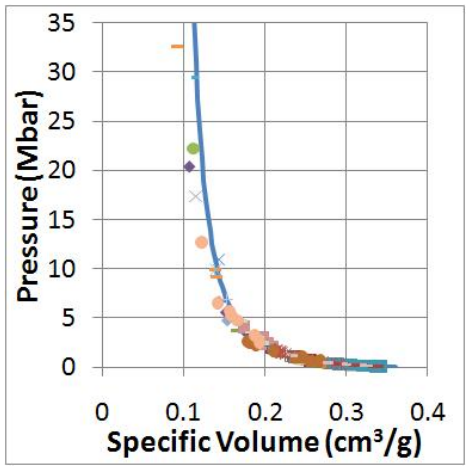
The Hugoniot, Figure 3.13, was compared to the experimental data. The EOS shows reasonable agreement with experimental data.This is expected since the MBEOS uses the Mie-Grüneisen model in the condensed phase region.



(a) U_s - u_p



(b) P - u_p



(c) P - V

Figure 3.13 MBEOS Hugoniot Plots. Experimental data from References [5-28]

The release isentropes were also generated for this EOS, Figure 3.14. The release isentropes show very little improvement from the Mie-Grüneisen results. Figure 3.15 is a series of constant energy lines to show the change in phase state. The dashed lines in the figure are a selection of high-pressure release isentopes. This shows that the material passes through the mixed phase region under a strong enough initial shock state. Recall that the phases are separated by the Clausius-Clapeyron given in

Equation 2.13.

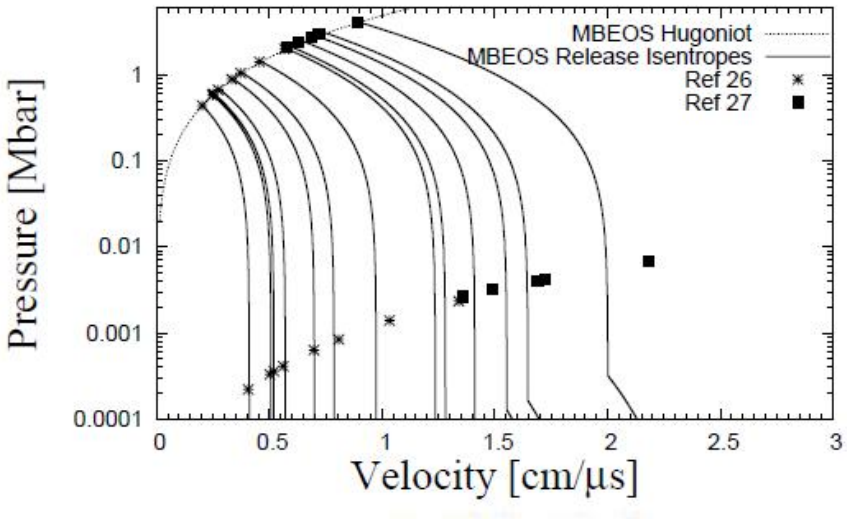


Figure 3.14 MBEOS Release Isentropes Results. Experimental data from References [21-29]

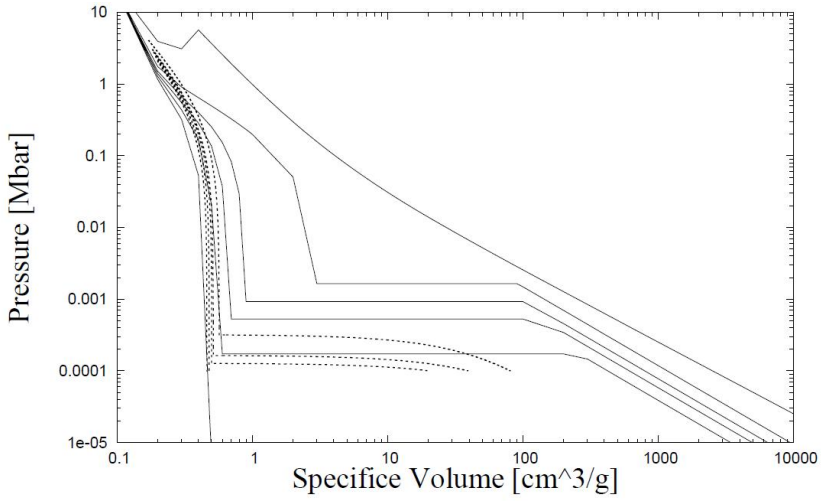
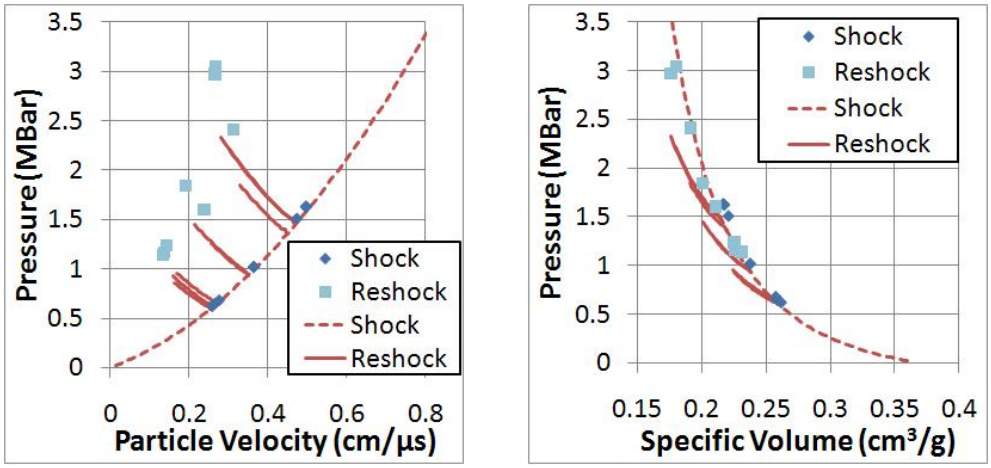


Figure 3.15 MBEOS Phase Diagram Results.

The results for the MBEOS reshock data are shown in Figure 3.16. The results are the same as the Mie-Grüneisen EOS. This is expected since the MBEOS uses the

Mie-Grüneisen EOS in the compression region.



(a) P - u_p **(b)** P - V

Figure 3.16 MBEOS Reshock Results. Experimental data from Reference [30].

3.4 Bushman-Lomonosov EOS

While reviewing Lomonosov [35] EOS approach, some mathematical discrepancies were found that didn't agree with the Bushman EOS method. This was particularly evident in the liquid atomic terms. So a hybrid approach was used. The cold contribution and the electronic terms in Table 3.7 are taken from the Lomonosov work while the atomic terms in Table 3.7 are taken from the Bushman model. A summary of the EOS parameters used are shown in Table 3.7.

This EOS is a multiphase EOS with three distinct phases; solid, liquid, and vapor. The liquid and vapor phases are modeling the liquid relationship described earlier. Instead of calculating the saturation points independently and interpolating between

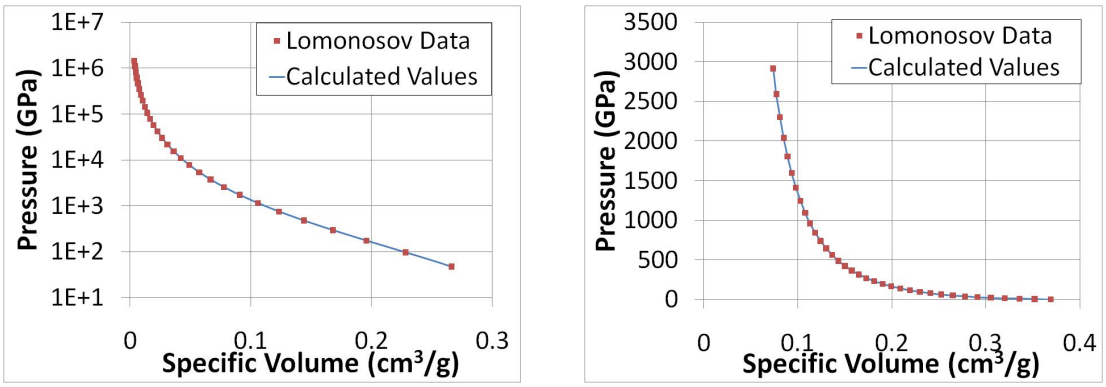
them, the saturation points are calculated during each set of the calculation. This is a computationally expensive method but removes errors associated with interpolation.

The saturation points were calculated in a multi-step process. First it is determined if the the thermodynamic state lies in the mixed-phase region. This is accomplished by tracking along a given isotherm for multiple specific volume with the same pressure. The next step is to determine saturation points for the given isotherm. The saturation points are calculated using Gibb's phase equilibrium using the method described in Section 1.1.4. The final step is to determine if the temperature-volume state lies in the mix-phase region. The transition between the solid and liquid region is also calculated using Gibb's phase equilibrium. The state with the lower Gibb's free energy is the stable state. No consideration of a mixed-phase region between the solid and liquid states were considered. It is assumed that the material is either all solid or all liquid.

Table 3.7 Aluminum Bushman-Lomonosov EOS parameters

V_0 g/cm^3	V_{0c} g/cm^3	Z	R	a_1	a_2	a_3	a_4	a_5
0.370	0.361	13	0.31	326.35	-1035.44	858.51	-160.59	11.17
E_{sub} g/cm^3	A_c g/cm^3	B_c	C_c	m	n	l	T_m kK	σ_m
-12.1	-12.91	40.96	-28.05	8.0	4.99	0.70	0.933	0.923
Θ_{0s} kK	γ_{0s}	B_s	D_s	T_T kK	σ_T	T_{sa} kK	T_{ca} kK	Θ_{0l} kK
0.1	2.19	0.7	0.7	30	0.14	6	25	157
γ_{0l} kK	B_l	D_l	A_m	B_m	C_m	T_Z kK	σ_Z	T_i kK
1.78	1.05	0.0	2.24	-5.64	0.21	200	0.8	50
σ_i	γ_0	γ_m	T_g kK	σ_e	σ_d	T_b	β_i	β_0
0.3	0.7	-0.5	300	1.0	9.99e9	8	0.0242	0.050
β_m								
0.0								

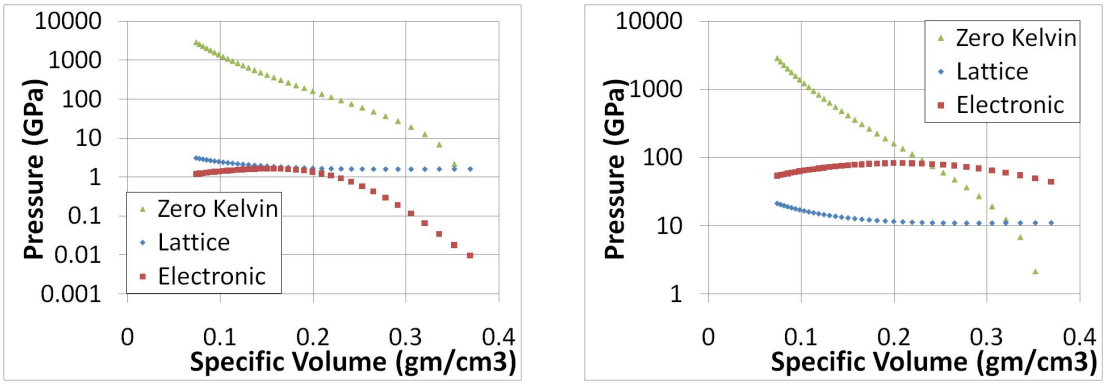
Lomonosov [35] included some numerical data including 0 K isotherm, 293 K isotherm, and the principal Hugoniot. Comparisons between the numerical data and the EOS are shown in Figure 3.17. The EOS is in good agreement with the numerical data from Lomonosov.



(a) 0 K Isotherm **(b)** 293 K Isotherm

Figure 3.17 Isotherms compared to Lomonosov Data

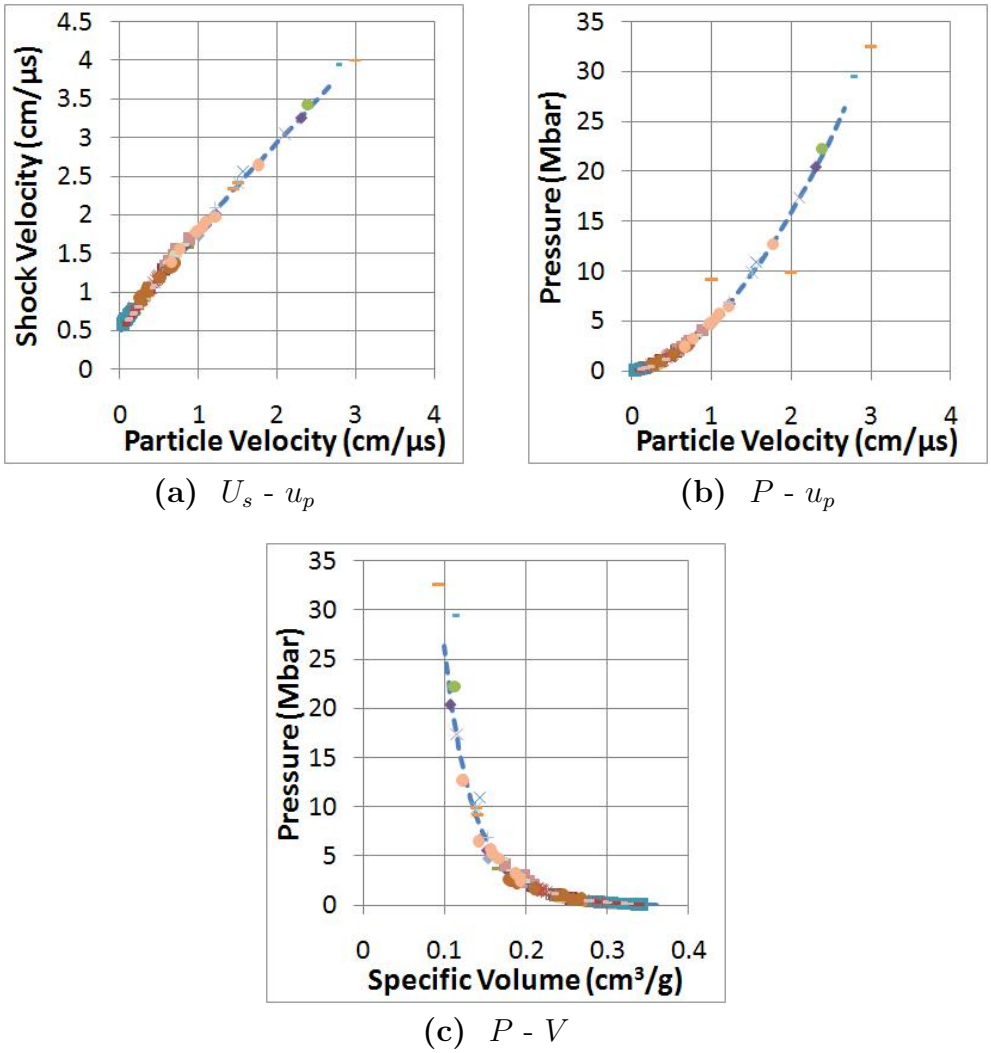
The three components that make up the Bushman-Lomonosov EOS; cold curve, lattice, and electronic contributions, can be evaluated independently as was done in Figure 3.18. The figure shows a low temperature and a higher temperature isotherm. In low temperature isotherms, the main contributor is the cold curve terms. As temperature increases, the lattice and the electronic terms play a more important role.



(a) 293 K Isotherm **(b)** 2000 K Isotherm

Figure 3.18 Bushman-Lomonosov EOS Contributions

Figure 3.19 shows the Hugoniot plots using the Bushman-Lomonosov EOS. The EOS transitions from a solid to a liquid during compression, something not captured in other EOSs. This is shown by the transition from the solid line to the dashed line.



(a) U_s - u_p **(b)** P - u_p

(c) P - V

Figure 3.19 Bushman-Lomonosov Hugoniot Plots. Experimental data from References [5-28]

The release isentropes, Figure 3.20, show good agreement at low initial pressure. As the pressure increases the EOS under-predicts the free surface velocity. A sharp

discontinuity seen in Figure 3.20 is a result of the solver not iterating to the correct solution. The Newton method was unable to converge on a solution because it was iterating between the solid and liquid phases.

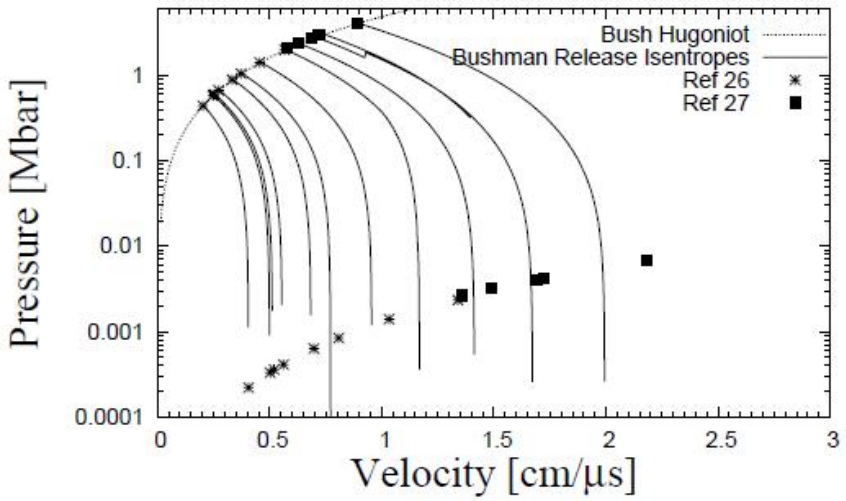
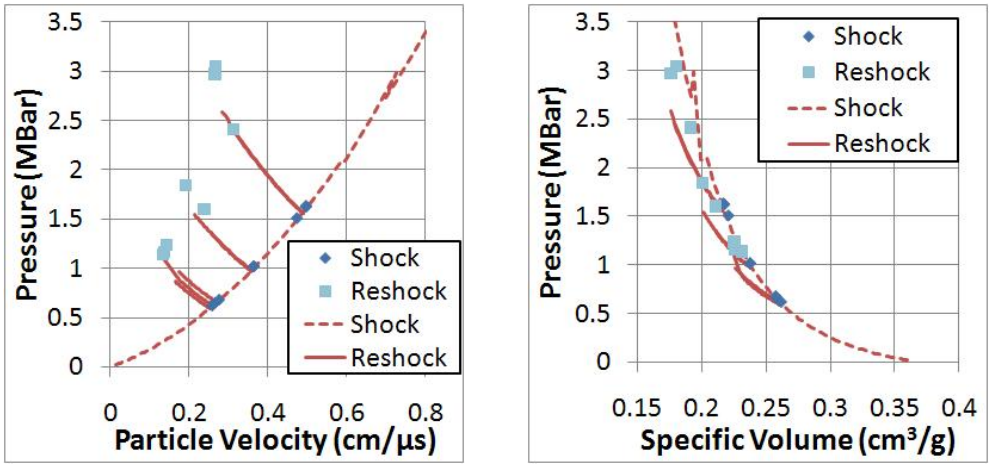


Figure 3.20 Bushman - Lomonosov Release Isentrope Results. Experimental data from References [21-29]

The calculated reshock states for the Bushman-Lomonosov EOS are shown in Figure 3.21. The Bushman-Lomonosv EOS does a reasonable job predicting the reshock state, though it under-predicts pressure.



(a) P - u_p **(b)** P - V

Figure 3.21 Bushman-Lomonosov Reshock Results. Experimental data from Reference [30].

The EOS predicts a critical temperature around 5500 K and a critical pressure of 1 GPa. These values differ from Bushman and Lomonosov results. Lomonosov predicted a critical temperature of 6250 K and a critical pressure of 0.197 GPa, while Bushman predicts a critical temperature of 7222 K and a critical pressure of 0.571 GPa.

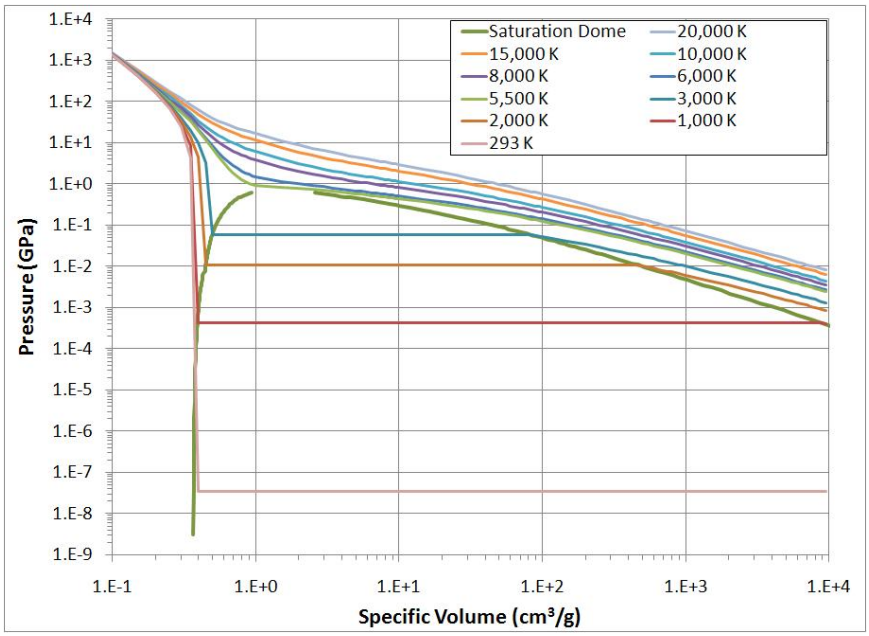


Figure 3.22 Bushman-Lomonosov Phase Diagram Results

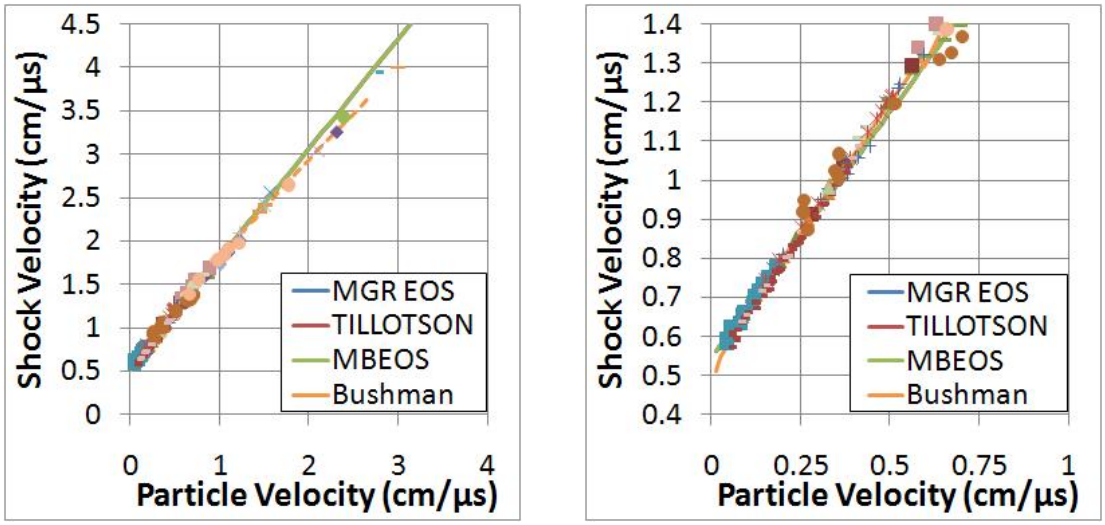
Chapter 4

Conclusions

Several equations of state (EOS) for Aluminum were investigated in this thesis. Different types of EOS were also considered. An incomplete EOS formulation requires an additional relationship to fully define the thermodynamic state. This contradicts a complete EOS formulation which fully defines the thermodynamic state. Examples of incomplete EOSs modeled in this work are Mie-Grüneisen, Tillotson, and Multi-branch analytical EOS (MBEOS). The Bushman-Lomonosov EOS is the only complete EOS modeled here.

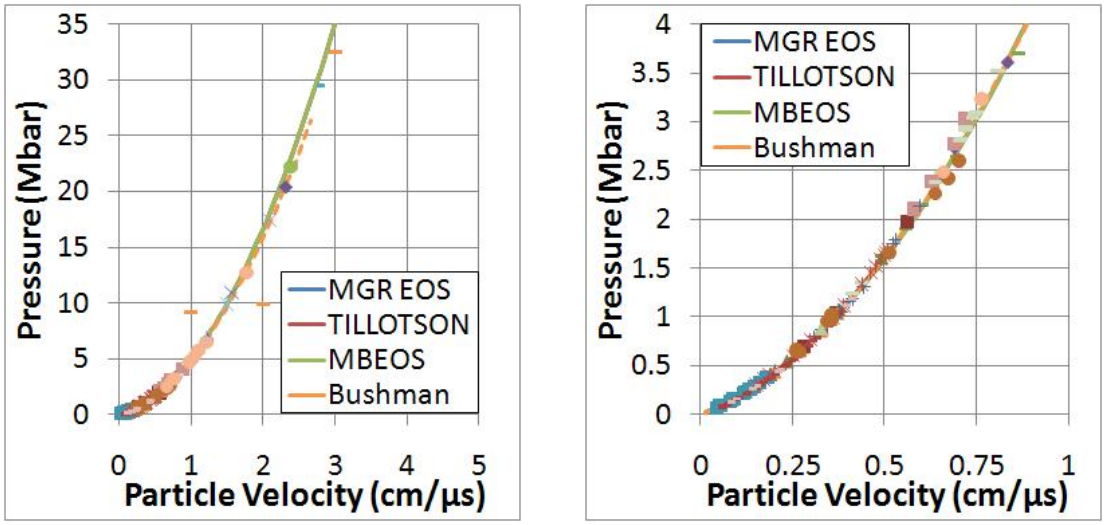
All EOS examined here do a good job of modeling the the Hugoniot at lower pressures shown in Figures 4.1 - 4.3. Mie-Grüneisen, Tillotson, and MBEOS have matching Hugoniot responses. This is expected since all three EOS use a Mie-Grüneisen form to represent the condensed phase. Bushman-Lomonosov EOS shows a different response. This is particularly evident at stronger shocks. Under stronger shocks, a solid-liquid phase change is predicted in the Bushman-Lomonosov EOS that is not captured in the other EOS. This is shown in dashed lines in Figures 4.1-4.3. The Bushman-Lomonosov best represents the shock Hugoniots over the range of states

examined here. Under lower pressure, all EOS give good results.



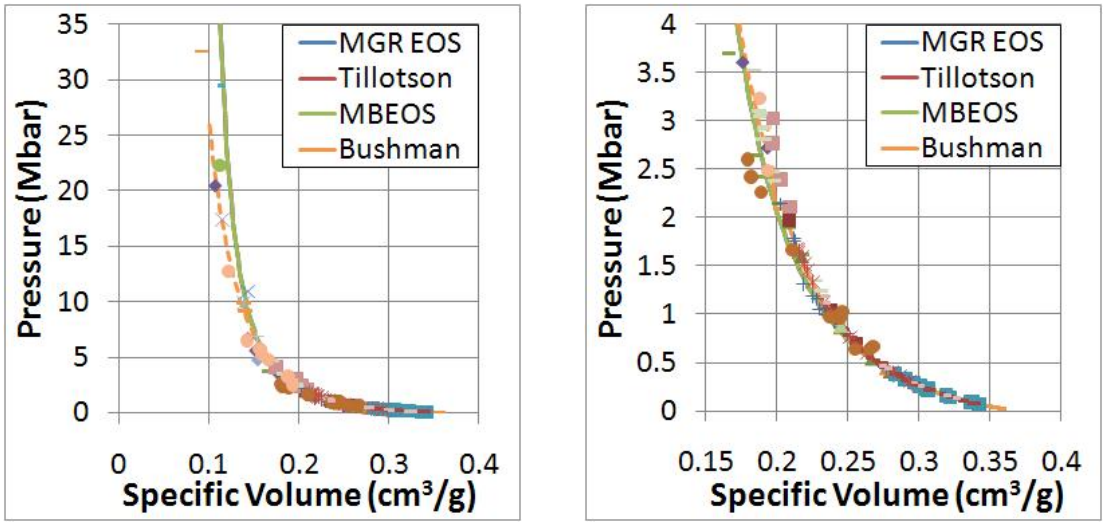
(a) (b)

Figure 4.1 U_s-u_p Results



(a) (b)

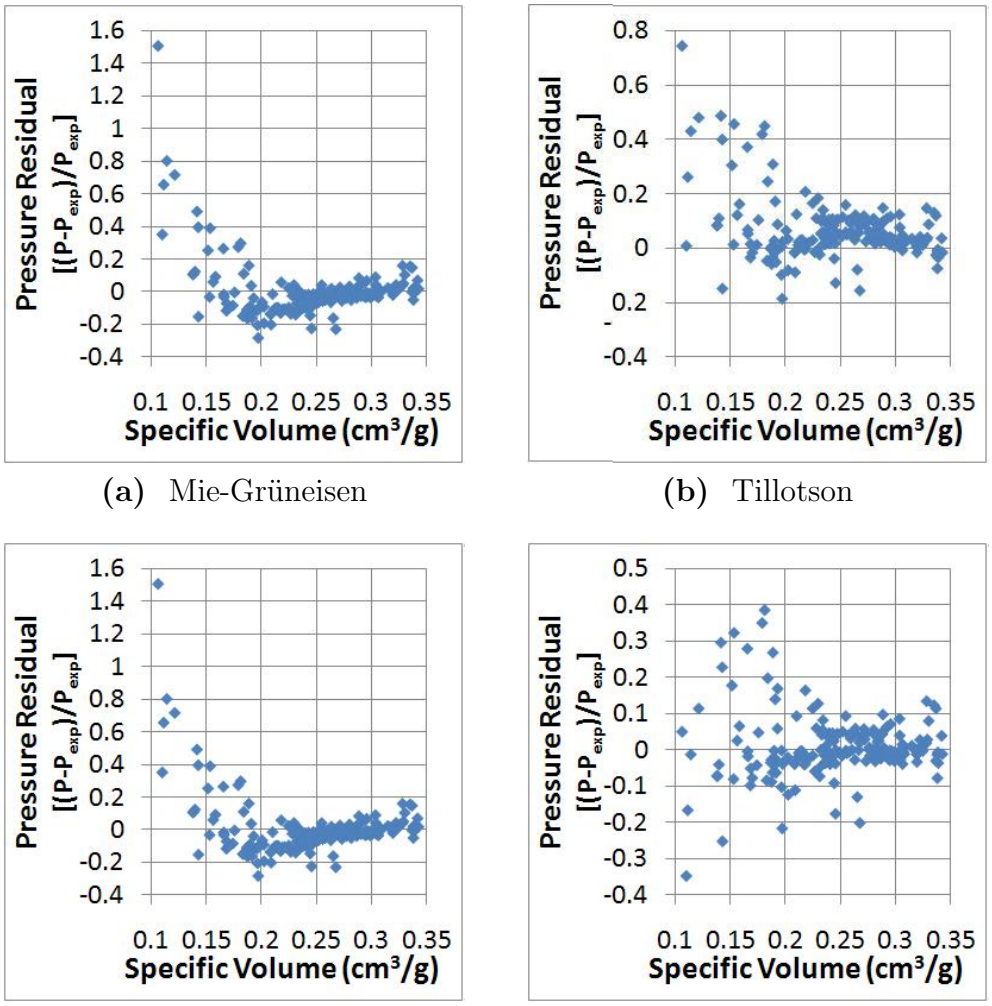
Figure 4.2 P-u_p Results



(a) (b)

Figure 4.3 *P-V* Results

Pressure residuals plotted in Figure 4.4 show how the compuation results deviate from the experimental results. All the models reasonably match the low compression region within 20 %. Under large compressions the Mie-Grüneisen and MBEOS EOS vary by 160 % while the Tillotson EOS only varies by a maximum of 80 %. The Bushman EOS peforms the best with the largest deviation of 40 % over the entire experimental range.



(a) Mie-Grüneisen **(b)** Tillotson

(c) MBEOS **(d)** Bushman

Figure 4.4 Residual pressure along the principle Hugoniot

The free surface velocity calculation shows that the Tillotson EOS is the most accurate for the experimental data. The remaining EOSs do an acceptable job under lower initial states. Once the shock pressure is greater than 1.5 MBar the free surface results diverge considerably.

The reshock calculations show how the EOSs perform off the principle Hugoniot. In the Mie-Grüneisen, Tillotson, and MBEOS EOSs the Grüneisen parameter has a

major effect on the calculated reshock state. The Tillotson does not use a constant Grüneisen ratio but calculated the Grüneisen parameter as a function of energy and specific volume. The Tillotson EOS performs the best of the three incomplete EOSs. The Bushman-Lomonosov EOS does a good job predicting the reshock state but underestimates pressure.

Chapter 5

Future Work

The work presented here could be extended in several different directions. First, the method created here could be extended to include additional EOS models and theory. The code could be improved to handle phase transitions more accurately and with less numerical noise. This would be accomplished by improving the numerical methods for determining the stable solution maybe via a mixing region between the solid-liquid transition. The code could be changed to include SESAME tabular output which would allow for new EOS table creation. The code is currently limited on which thermodynamic variables are needed to define a thermodynamic state. The code could be expanded so that for any two thermodynamic variables the remaining ones are found.

Another possible direction for future work is to incorporate models presented here into existing shock physics codes. This could be accomplished by using the individual modules that define each EOS. The shock physics code would need to provide thermodynamic variables and the remaining ones would be calculated and returned.

Finally, this work could be extended to develop new EOS models. The model theory could be created and tested using the framework developed here. A composite approach could be utilized to use a wide array of models. The composite model would require smoothing between each individual theory but may lead to more accurate behavior at extreme states.

Bibliography

[1] A. V. Bushman, G. I. Kanel', A. L. Ni, and V. E. Fortov, *Intense Dynamic Loading of Condensed Matter*. Washington, DC: Taylor & Francis, 1993.

[2] M. A. Meyers, *Dynamic Behavior of Materials*. Wiley-Interscience, 1 ed., Sept. 1994.

[3] G. R. Gathers, *Selected Topics in Shock Wave Physics and Equation of State Modeling*. River Edge, NJ: World Scientific, 1994.

[4] Y. B. Zel'dovich and Y. P. Raizer, *Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena*. Dover Publications, Mar. 2002.

[5] L. V. Al'tshuler, S. B. Kormer, A. A. Bakanova, and R. F. Trunin, "Equations of state for aluminum, copper and lead in the high pressure region," *Journal of Experimental and Theoretical Physics*, vol. 11, pp. 573–579, 1960.

[6] L. V. Al'tshuler, S. B. Kormer, M. I. Brazhnik, L. A. Vladimirov, M. P. Speranskaya, and A. I. Funtikov, "The isentropic compressibility of aluminum, copper, lead at high pressures," *Journal of Experimental and Theoretical Physics*, vol. 11, no. 4, pp. 766–775, 1960.

[7] L. V. Al'tshuler and A. P. Petrunin, "Rentgenographic investigation of compressibility of light substances under obstacle impact of shock waves," *Zhurnal Tekhnicheskoi Fiziki*, vol. 36, no. 6, pp. 717–725, 1961.

[8] I. C. Skidmore and E. Morris, "Experimental equation-of-state data for uranium and its interpretation in the critical region," IAEA 173-216, Vienna, 1962.

[9] W. H. Isbell, F. H. Shipman, and A. H. Jones, "Hugoniot equation of state measurements for eleven materials to five megabars," Materials Science Laboratory MSL-68-13, General Motors Corp., 1968.

[10] R. G. McQueen, S. P. Marsh, J. W. Taylor, J. N. Fritz, and W. J. Carter, "The equation of state of solids from shock wave studies," in *High Velocity Impact Phenomena*, pp. 293–417, New York: Academic Press, 1970.

[11] L. V. Al'tshuler and B. S. Chekin, "Metrology of high pulsed pressures," in *All-Union Pulsed Pressures Simposium*, vol. 1, (Moscow), pp. 5–22, 1974.

[12] L. V. Al'tshuler, N. N. Kalitkin, L. V. Kuz'mina, and B. S. Chekin, "Shock adiabats for ultrahigh pressures," *Journal of Experimental and Theoretical Physics*, vol. 45, no. 1, pp. 167–171, 1977.

[13] S. P. Marsh, *LASL Shock Hugoniot Data*. Berkeley: Univ. California Press, 1980.

[14] L. V. Al'tshuler, A. A. Bakanova, I. P. Dudoladov, E. A. Dynin, R. F. Trunin, and B. S. Chekin, "Shock adiabats for metals. new data, statistical analysis and general regularities," *Journal of Applied Mechanics and Technical Physics*, vol. 22, p. 145, 1981.

[15] L. P. Volkov, N. P. Voloshin, A. S. Vladimirov, V. N. Nogin, and V. A. Simonenko, "Shock compressibility of aluminum at pressure 10 mbar," *Journal of Experimental and Theoretical Physics Letters*, vol. 31, pp. 588–591, 1981.

[16] A. C. Mitchell and W. J. Nellis, "Shock compression of aluminum, copper, and tantalum," *Journal of Applied Physics*, vol. 52, no. 5, pp. 3363–3374, 1981.

[17] C. E. Ragan, "Shock compression measurements at 1 to 7 TPa," *Physical Review A*, vol. 25, p. 3360, June 1982. Copyright (C) 2009 The American Physical Society; Please report any problems to prola@aps.org.

[18] C. E. Ragan, "Shock-wave experiments at threefold compression," *Physical Review A*, vol. 29, p. 1391, Mar. 1984. Copyright (C) 2009 The American Physical Society; Please report any problems to prola@aps.org.

[19] V. A. Simonenko, N. P. Voloshin, A. S. Vladimirov, A. P. Nagibin, V. Nogin, V. A. Popov, V. A. Sal'nikov, and Y. A. Shoidin, "Absolute measurements of shock compressibility of aluminum at pressures p>= 1 TPa," *Journal of Experimental and Theoretical Physics*, vol. 61, no. 4, pp. 869–873, 1985.

[20] R. F. Trunin, "Compressibility of various substances at high shock pressures," *Physics of the Solid Earth*, vol. 22, p. 103, 1986.

[21] B. L. Glushak, A. P. Zharkov, M. V. Zhernokletov, V. Ternovoi, A. S. Filimonov, and V. E. Fortov, "Experimental investigation of the thermodynamics of dense plasmas formed from metals at high energy concentrations," *Journal of Experimental and Theoretical Physics*, vol. 69, no. 4, pp. 739–749, 1989.

[22] M. A. Podurets, G. V. Simakov, and R. F. Trunin, "Transition of stishovite to a denser phase," *Fizika Zemli*, vol. 4, pp. 30–37, 1990.

[23] M. A. Podurets, V. M. Ktitorov, R. F. Trunin, V. A. Popov, A. Y. Matveev, B. V. Pechenkin, and A. G. Sevast'yanov, "Shock wave compression of aluminum at pressures of 1,7 TPa," *Teplofizika Vysokikh Temperatur*, vol. 32, no. 6, pp. 952–955, 1994.

[24] R. Trunin, M. Panov, and A. Medvedev, "Compressibility of iron, aluminum, molybdenum, titanium, and tantalum at shock-wave pressures of 1-2.5 TPa," *Journal of Experimental and Theoretical Physics Letter*, vol. 62, no. 7, pp. 591–594, 1995.

[25] R. F. Trunin, M. A. Podurets, G. V. Simakov, V. A. Popov, and A. G. Sevast'yanov, "New data on aluminum, plexiglas and quartz compression obtained in strong shock wave of underground nuclear explosion," *Journal of Experimental and Theoretical Physics*, vol. 81, no. 3, p. 464, 1995.

[26] R. F. Trunin, M. Panov, and A. Medvedev, "Shock compressibilities of iron, aluminum, and tantalum at terapascal pressures," *Khim. Fiz.*, vol. 14, no. 2-3, pp. 97–99, 1995.

[27] R. F. Trunin, L. F. Gudarenko, M. V. Zhernokletov, and G. V. Simakov, "Experimental data on shock compressibility and adiabatic expansion of condensed substances," in *RFNC*, (Sarov), 2001.

[28] M. D. Knudson, R. W. Lemke, D. B. Hayes, C. A. Hall, C. Deeney, and J. R. Asay, "Near-absolute hugoniot measurements in aluminum to 500 GPa using a magnetically accelerated flyer plate technique," *Journal of Applied Physics*, vol. 94, pp. 4420–4431, Oct. 2003.

[29] A. A. Bakanova, I. P. Dudoladov, M. V. Zhernokletov, V. N. Zubarev, and G. V. Simakov, "Vaporization of shock-compressed metals on expansion," *Journal of Applied Mechanics and Technical Physics*, vol. 24, pp. 204–209, Mar. 1983.

[30] W. J. Nellis, A. C. Mitchell, and D. A. Young, "Equation-of-state measurements for aluminum, copper, and tantalum in the pressure range 80–440 GPa (0.8–4.4 mbar)," *Journal of Applied Physics*, vol. 93, no. 1, pp. 304–310, 2003.

[31] J. C. Maxwell, *The scientific paper of James Clerk Maxwell*. Mineola, NY: Dover Publications, 2003.

[32] K. Wark, *Advanced thermodynamics for engineers*. McGraw-Hill, 1995.

[33] J. H. Tilltson, "Metallic equations of state for hypervelocity impact," Technical Report GA-3216, General Atomic, San Diego, CA, July 1963.

[34] L. Glenn and D. Young, "Dynamic loading of the structural wall in a lithium-fall fusion reactor," *Nuclear Engineering and Design*, vol. 54, pp. 1–16, Sept. 1979.

[35] I. Lomonosov, "Multi-Phase equation of state for aluminum," *Laser and Particle Beams*, vol. 25, no. 04, pp. 567–584, 2007.

[36] N. M. Laurendeau, *Statistical Thermodynamics*. New York, NY: Cambridge University Press, 2005.

[37] M. L. Wilkins, *Computer Simulation of Dynamic Phenomena*. Scientific Computation, New York: Springer-Verlag, 1999.

Appendix A

Equation of State Source Code

The main EOS routine is two parts. A shared module where common variables are stored and the main driver routine. Below is the shared module.

```
MODULE EOS
  IMPLICIT NONE
    DOUBLE PRECISION rho0, C0, s1, v0,gamma0
    DOUBLE PRECISION Pr, Er
    DOUBLE PRECISION P,E,Up, Us, T, S,H,G
    DOUBLE PRECISION, dimension(10) :: a, ti, mb
    DOUBLE PRECISION, dimension(50) :: be
    INTEGER, dimension(5) :: nEOS
END MODULE EOS
```

This is the main driver routine for the EOS code.

```
PROGRAM eos_1_0
USE EOS
!-------------------------------------------------------------------------------
!
!   CODE NAME:          eos_1_0
!
!   VERSION NUMBER:     1.0
!
!   DATE CREATED:       28 JANUARY 201‘
!
!   LANGUAGE:           FORTRAN 90
!
!   AUTHOR:             Aaron Ward
!                       email: aaron.ward@corvidtec.com
!
!   DESCRIPTION:
```

```
!
!     Calulate shock properties and relationships via different equation
!     of state (EOS).
!
!     The Mie-Gruneisen EOS states that P(V,E) = Pr + gamma/V*(E-Er)
!     here the 'r' represents the references state. The simplist case is
!     where the reference state is the known principle hugoniot.
!
!  UPDATES:
!        DATE              WHO       Ver.     CHANGES
!        07/27/09          AJW       0.1      ADDED Mie-Gruneisen EOS to program
!        08/05/09          AJW       0.2      ADDED Tillotson EOS inputs
!        09/07/09          AJW       0.3      ADDED Release isentrope with
!                                               calculating the free surface velocity
!        09/14/09          AJW       0.4      ADDED improved method for calculating
!                                               the principle hugoniot
!        09/15/09          AJW       0.4      ADDED Tillotson principle hugoniot
!                                               and single phase isentropic release
!        09/16/09          AJW       0.4      ADDED Tillotson 2 phase isentropic
!                                               release
!        09/22/09          AJW       0.5      The EOSs have been removed form the
!                                               main program and instead have been
!                                               writen as Modules
!        09/22/09          AJW       0.5      The findV, isentrope sections have
!                                               been rewriten with improvements.
!                                               the initial guess of v in findV is
!                                               still a consern an new method of
!                                               determining should be found.
!        10/05/09          AJW       0.6      Tillotson EOS validated
!        10/18/09          AJW       0.7      EOS Modues rewriten, new aritecture
!                                               setup for finding EOS variables.
!                                               Given any two the remaining
!                                               variables are found. This allows
!                                               hugoniont, isentropic, etc logic
!                                               to be moved back into the main
!                                               program. Requiring a significate
!                                               rewrite which concluded version 0.7
!        10/21/09          AJW       0.8      Multi-branch EOS modulus added
!        05/02/11          AJW       1.0      Added Compression Isentropes
!
!-----------------------------------------------------------------------------
      IMPLICIT NONE
      INTEGER sel
!      DOUBLE PRECISION P1,v1
! Start of the Program
```

```
!    - Menu
!    ( I would like something like panda

      WRITE(*,100) 'EOS CODE', 'VERSION 0.9'


 1000 CALL MENU(sel)

 9000 WRITE(*,*) 'BYE'
!
  100 format(25x,A/,23x,A/)
      END PROGRAM eos_1_0
!-----------------------------------------------------------------------
!  Subroutine: MENU
!
!  DESCRIPTION:
!    Main Menu Subroutine tells options
!-----------------------------------------------------------------------
      SUBROUTINE MENU(sel)
      IMPLICIT NONE
      integer sel
!       sel =0
 1000 WRITE(*,101) 'Select EOS to Model'
      WRITE(*,100) '----------------------MENU----------------------'
      WRITE(*,101) '(1) Mie-Gruneisen EOS'
      WRITE(*,101) '(2) Tillotson EOS'
      WRITE(*,101) '(3) Multi-branch EOS'
      WRITE(*,101) '(4) '
      WRITE(*,101) '(5) USER'
      WRITE(*,101) '(6) Exit System'
      read(*,*) sel
      if(sel.eq.1) then
         CALL mgrinput(sel)
      elseif(sel.eq.2) then
         CALL tilinput(sel)
      elseif(sel.eq.3) then
         CALL mbinput(sel)
!       elseif(sel.eq.4) then
!          CALL beinput(sel)
      elseif(sel.eq.5) then
         write(*,100) 'This feature will be added later to allow'
         write(*,100) 'user EOS to be added for testing pruposes'
         write(*,*) ''
         goto 1000
      elseif(sel.eq.6) then
```

```
        return
      else
        write(*,*) 'Not a valid option'
        goto 1000
      endif
      CALL OMENU()
      if (sel.ne.6) then
        goto 1000
      endif
      return
  100 format(15x,A)
  101 format(18x,A)

      END SUBROUTINE MENU
!-----------------------------------------------------------------------
!  Subroutine: OMENU
!
!  DESCRIPTION:
!    The OMENU or hugoniot menu allow you to select what is to be done next
!    plotting, calculating based on the Mie-Gruneisen Hugoniot form.
!-----------------------------------------------------------------------
      SUBROUTINE OMENU()
      IMPLICIT NONE
      integer sel
 1000 WRITE(*,100) '----------Options MENU----------'
      WRITE(*,101) '(1) Hugoniot'
      WRITE(*,101) '(2) Isotherms'
      WRITE(*,101) '(3) Reshock'
      WRITE(*,101) '(4) Isoenergy'
      WRITE(*,101) '(5) Release Isentrope'
      WRITE(*,101) '(6) Start Over'
      read(*,*) sel
      if(sel.eq.1) then
        CALL inpHugo()
      elseif(sel.eq.2) then
        CALL ISOTHERM()
      elseif(sel.eq.3) then
!       CALL inpRels()
        CALL reshock()
      elseif(sel.eq.4) then
        CALL ISOENERGY()
      elseif(sel.eq.5) then
        CALL HRISENTROPE()
      elseif(sel.eq.6) then
        RETURN
```

```
      else
         RETURN
      endif
      goto 1000
      return
  100 format(15x,A)
  101 format(18x,A)
      END SUBROUTINE OMENU

!-----------------------------------------------------------------------
!  Subroutine: inpHugo()
!
!  DESCRIPTION:
!    Values to be using in calculating princilpe hugo
!-----------------------------------------------------------------------
      SUBROUTINE inpHugo()
      USE EOS
      integer sel,step
      DOUBLE PRECISION L,U

 1000 WRITE(*,101) 'Select Independent Variable'
      WRITE(*,101) '(1) v'
      read(*,*) sel
      if(sel.eq.1) then
         goto 2000
      elseif(sel.eq.2) then
         goto 3000
      elseif(sel.eq.2) then
         goto 4000
      else
         write(*,101)'Not a valid input'
         goto 1000
      endif
 2000 WRITE(*,101) 'Enter Low, Upper V and increments'
      READ(*,*) L,U,step
      CALL Hugoniot(L,U,step)
      GOTO 5000
 3000 WRITE(*,*) 'Feature not added yet'
      goto 2000
 4000 WRITE(*,*) 'Feature not added yet'
      goto 2000
 5000 WRITE(*,*) 'Plot again yes (1)/No (2)'
      read(*,*)  sel
      if(sel.eq.1) then
         goto 1000
```

```
      endif
      RETURN
  100 format(10x,'-----',A)
  101 format(18x,A)
      END SUBROUTINE inpHugo

!-----------------------------------------------------------------------
!  Subroutine: inpRels()
!
!  DESCRIPTION:
!    Values to be using in calculating princilpe hugo
!-----------------------------------------------------------------------
      SUBROUTINE inpRels()
      USE EOS
      integer sel, step
      DOUBLE PRECISION L,U
 1000 WRITE(*,101) 'Select Independent Variable'
      WRITE(*,101) '(1) V'
!       WRITE(*,101) '(2) u'
!       WRITE(*,101) '(3) P'
      read(*,*) sel
      if(sel.eq.1) then
         goto 2000
      elseif(sel.eq.2) then
         goto 3000
      elseif(sel.eq.2) then
         goto 4000
      else
         write(*,101)'Not a valid input'
         goto 1000
      endif
 2000 WRITE(*,101) 'V and increments'
      READ(*,*) L,step
      U = v0
      write(*,*) L,U,step
!       CALL ISENTROPE(U,L,step)
      call Comp_Isen(L,step)
      GOTO 5000
 3000 WRITE(*,*) 'Not added yet'
      goto 2000
 4000 WRITE(*,*) 'Not added yet'
      goto 2000
 5000 WRITE(*,*) 'Plot again yes (1)/No (2)'
      read(*,*)  sel
      if(sel.eq.1) then
```

```
        goto 1000
      endif

      RETURN
100   format(10x,'-----',A)
101   format(18x,A)
      END SUBROUTINE inpRels


      SUBROUTINE mgrinput(sel)
      USE MGR
      INTEGER sel
      call eosINP(sel)
      RETURN
      END SUBROUTINE

      SUBROUTINE tilinput(sel)
      USE TILLOTSON
      INTEGER sel
      call eosINP(sel)
      RETURN
      END SUBROUTINE

      SUBROUTINE mbinput(sel)
      USE MBEOS
      INTEGER sel
      call eosINP(sel)
      RETURN
      END SUBROUTINE

      SUBROUTINE mgrFindV(P1,vo)
      USE MGR
      DOUBLE PRECISION P1,vo
      call findV(P1,vo)
      RETURN
      END SUBROUTINE

      SUBROUTINE tilFindV(P1,vo)
      USE TILLOTSON
      DOUBLE PRECISION P1,vo
      call findV(P1,vo)
      RETURN
      END SUBROUTINE

      SUBROUTINE mbFindV(P1,vo)
```

```
      USE MBEOS
      DOUBLE PRECISION P1,vo
       call findV(P1,vo)
      RETURN
      END SUBROUTINE

      SUBROUTINE mgrprop(P1,E1,V1,T1,Ss,temp)
      USE MGR
      DOUBLE PRECISION P1,E1,V1,T1,Ss
      DOUBLE PRECISION,DIMENSION(3)   ::   temp
       call mgreos(P1,E1,V1,T1,Ss,temp)
      RETURN
      END SUBROUTINE

      SUBROUTINE tillprop(P1,E1,V1,T1,Ss,temp)
      USE TILLOTSON
      DOUBLE PRECISION P1,E1,V1,T1,Ss
      DOUBLE PRECISION,DIMENSION(3)   ::   temp
       call tilleos(P1,E1,V1,T1,Ss,temp)
      RETURN
      END SUBROUTINE

      SUBROUTINE mbprop(P1,E1,V1,T1,Ss,temp)
      USE MBEOS
      DOUBLE PRECISION P1,E1,V1,T1,Ss
      DOUBLE PRECISION,DIMENSION(3)   ::   temp
       call multieos(P1,E1,V1,T1,Ss,temp)
      RETURN
      END SUBROUTINE


!-----------------------------------------------------------------------
!  Subroutine: Hugoniot
!
!  DESCRIPTION:
!    Plots/exports isotherms for over a   given temperature range and number
!    of steps
!-----------------------------------------------------------------------
      SUBROUTINE Hugoniot(hig,low,incr)
       use EOS
       IMPLICIT NONE
      DOUBLE PRECISION L, U, delta, Cu, low, hig,dump,E2,Up2,f2
      DOUBLE PRECISION f,df,ierror,error,Peos,Ut,V2,P2,dsmall,E1
      DOUBLE PRECISION,DIMENSION(3)   :: temp
      DOUBLE PRECISION,DIMENSION(10)  :: junk
```

```
      integer sel, step, i, incr

!       write(*,*) 'v0: ',v0
      open(11,file='hugo.dat',access='append')
      open(9)
      L = low
      U = hig
      step = incr
      delta = (U-L)/(step*1.)
      WRITE(11,100) 'Principle Hugoniot Calculation'
      if(nEOS(2).eq.1.or.nEOS(2).eq.2) then
         WRITE(11,104) rho0, C0, s1, gamma0
         if(nEOS(2).eq.2) then
            WRITE(11,105) a(1), a(2), a(3), a(4), a(5)
         endif
      endif
      WRITE(11,102) 'v[cm3/g]','P[Mbar]','Up[cm/micro-s]',
     &              'Us[cm/micro-s]'
      dsmall=0.99999
      if(Up.eq.0.) then
         Up=0.1
      endif
      temp(1:3)=0.0
 2100 do i=0,step
         Cu=L+delta*i
         dump=0
         error=0.01
         ierror=1.0
         do while (ierror.ge.error)
            Peos=0.0
            junk(1:10)=0.0
            E1=(Up**2./2.)
            write(9,*) (Up**2)/(v0-cu), E1
            if(nEOS(1).eq.1) then
               call mgrprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(Peos,E1,cu,junk(2),junk(3),temp)
            endif
            f=Peos-(Up**2)/(v0-cu)
            write(9,*) cu, Peos, E1, temp(2)
            Up2=dsmall*Up
            E2=(Up2**2./2.)
            junk(1:10)=0.0
```

```
            write(9,*) (Up2**2)/(v0-cu), E1
            if(nEOS(1).eq.1) then
               call mgrprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(junk(1),E2,cu,junk(2),junk(3),temp)
            endif
            f2 = junk(1)-(Up2**2)/(v0-cu)
            df = (f2-f)/(Up2-Up)
            Ut=Up-f/df
            write(9,*) cu, Peos, E1, f2, df, Ut
            ierror = abs((Ut-Up)/Up)
            Up=abs(Ut)
         end do
         Up=Ut
         Us=(Up*v0) / (v0-cu)
         P = Up*Us/v0
         E = Up**2/2
         write(11,103) Cu,P, Up, Us
         write(*,103)  Cu,P, Up, Us
      enddo
      write(11,*) ''
      write(11,*) ''
      close(11)
      close(9)
      return
  100 format(10x,A)
  101 format(18x,A)
  102 format(16x,A,7x,A,3x,A,3x,A)
  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x)
  104 format(4x,'rho0: ',e10.4,4x,'c0: ',e10.4,4x,'s1: ',
     &     e10.4,4x,'g0: ',e10.4)
  105 format(4x,'Wilkins Constants: ',e10.4,' : ',e10.4,
     &     ' : ', e10.4,' : ',e10.4)
      END SUBROUTINE Hugoniot
!-----------------------------------------------------------------------
! Subroutine: Reshock
!
! DESCRIPTION:
!    Calulated reshock state given principle shock strength, and reshock
!    strength
!
!-----------------------------------------------------------------------
      SUBROUTINE reshock()
```

```
      use EOS
      IMPLICIT NONE
      DOUBLE PRECISION up_1, P_1, E_1, V_1
      DOUBLE PRECISION up_2, P_2, E_2, V_2
      DOUBLE PRECISION L, U, delta, Ut,Cu,E1,E2,Up2
      DOUBLE PRECISION f, f2,df, ierror, error, Peos, V2, dsmall
      DOUBLE PRECISION,DIMENSION(3) :: temp
      DOUBLE PRECISION,DIMENSION(10) :: junk
      INTEGER step1, step2,i

      open(11,file='reshock.data',access='append')
      dsmall = 0.99999
      WRITE(*,101) 'Enter Hugonoit shock specific volume and increments'
      READ(*,*) V_1, step1
      WRITE(*,101) 'Enter reshock specific volume and increments'
      READ(*,*) V_2, step2
      write(11,102) 'v[cm3/g]','P[Mbar]','Up[cm/micro-s',
     &              'Us[cm/micro-s]'
      WRITE(*,*) "Calculating principle hugoniot"
      Up = 0.1
      temp(1:3) = 0.0
      L = V0
      U = V_1
      delta = (U-L)/(step1*1.0)
      do i=1, step1
         Cu = L+delta*i
!         dump(1:10) = 0.0
         error = 0.01
         ierror = 1.0
         do while (ierror.ge.error)
            Peos = 0.0
            E1 = (Up**2./2.)
            if (nEOS(1).eq.1) then
               call mgrprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(Peos,E1,cu,junk(2),junk(3),temp)
            endif
            f=Peos-(Up**2)/(v0-cu)
!            write(9,*) cu, Peos, E1, temp(2)
            Up2=dsmall*Up
            E2=(Up2**2./2.)
            junk(1:10)=0.0
!            write(9,*) (Up2**2)/(v0-cu), E1
```

```
            if(nEOS(1).eq.1) then
               call mgrprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(junk(1),E2,cu,junk(2),junk(3),temp)
            endif
            f2 = junk(1)-(Up2**2)/(v0-cu)
            df = (f2-f)/(Up2-Up)
            Ut=Up-f/df
!            write(9,*) cu, Peos, E1, f2, df, Ut
            ierror = abs((Ut-Up)/Up)
            Up=abs(Ut)
         end do
         Up=Ut
         Us=(Up*v0) / (v0-cu)
         P = Up*Us/v0
         E = Up**2/2
         write(11,103) Cu,P, Up, Us
         write(*,103)  Cu,P, Up, Us
      enddo
      write(*,*) "Principle Hugoniot calculation complete"
      write(*,103)  Cu,P, Up, Us
      write(*,*) "Reshock calculation"
      up_1 = Up
      P_1 = P
      E_1 = E
      write(*,*) "Up_0    ,    P_0     ,   E_0"
      write(*,*) up_1, P_1, E_1

      temp(1:3) = 0.0
      L = Cu
      U = V_2
      delta = (U-L)/(step2*1.0)
      Up_2 = 0.01
      do i=1, step2
         Cu = L+delta*i
!         dump(1:10) = 0.0
         error = 0.01
         ierror = 1.0
         do while (ierror.ge.error)
            Peos = 0.0
            P_2 = P_1+((Up_1-Up_2)**2.)/(V_1-cu)
!            E1 = (Up**2./2.)
            E1 = E_1 + 0.5*(P_2-P_1)*(V_1-cu)
```

```
            if (nEOS(1).eq.1) then
               call mgrprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(Peos,E1,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(Peos,E1,cu,junk(2),junk(3),temp)
            endif
            f=Peos-P_2
!           write(9,*) cu, Peos,P_2, E1
            Up2=dsmall*Up_2
!           E2=(Up2**2./2.)
            P_2 = P_1+((Up_1-Up2)**2.)/(V_1-cu)
            E2 = E_1 + 0.5*(P_2-P_1)*(V_1-cu)
            junk(1:10)=0.0
            write(9,*) P_2, E1
            if(nEOS(1).eq.1) then
               call mgrprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.2) then
               call tillprop(junk(1),E2,cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(junk(1),E2,cu,junk(2),junk(3),temp)
            endif
            f2 = junk(1)-P_2
            df = (f2-f)/(Up2-Up_2)
            Ut=Up_2-f/df
            write(9,*) cu, Peos, E1, f2, df, Ut
            ierror = abs((Ut-Up_2)/Up_2)
            write(9,*) ierror, error
            Up_2=abs(Ut)
         end do
         Up=Ut
         Us=(Up-Up_1)*v0 / (v0-cu)
         P = P_1+((Up_1-Up2)**2.)/(V_1-cu)
         E = E_1 + 0.5*(P-P_1)*(V_1-cu)
         write(11,103) Cu,P, Up, Us
         write(*,103)  Cu,P, Up, Us
      enddo


      return
  100 format(10x,A)
  101 format(18x,A)
  102 format(16x,A,7x,A,3x,A,3x,A)
  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x)
      END SUBROUTINE Reshock
```

```
!-----------------------------------------------------------------------
!  Subroutine: ISOTHERMS
!
!  DESCRIPTION:
!    Plots/exports isotherms for over a  given temperature range and number
!    of steps
!-----------------------------------------------------------------------
      SUBROUTINE ISOTHERM()
      IMPLICIT NONE
      DOUBLE PRECISION Ts    ! Start temperature
      DOUBLE PRECISION Te    ! End temperature
      integer num            ! number or increments

      WRITE(*,*) 'This Feature has not been added yet'
      return

      END SUBROUTINE ISOTHERM

!-----------------------------------------------------------------------
!  Subroutine: ISOENERGY
!
!  DESCRIPTION:
!    Plots/exports Lines of constant energy
!-----------------------------------------------------------------------
      SUBROUTINE ISOENERGY()
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION E1        ! ENERGY
      DOUBLE PRECISION vl,vu     ! start and engine specific energy
      DOUBLE PRECISION delta,cu
      DOUBLE PRECISION,DIMENSION(3) :: temp
      DOUBLE PRECISION,DIMENSION(10):: junk
      integer num,i,m              ! number or increments
      DOUBLE PRECISION level
!       open(12)
      open(11,file='isoenergy.dat',access='append')
      write(11,103) 'v[cm3/g]','P[Mbar]','E[Mbar-cm3/g]',
     &              'T[K]','S[Mbar-cm3/g-K]'
      temp(1:3) = 0.
      write(*,101) 'Enter energy state'
      read(*,*)  E1
      write(*,101) 'Enter lower and upper specific volume'
      read(*,*) vl,vu
      write(*,101) 'Enter number of increments to be calculated'
```

```
      read(*,*) num
!       delta = (vu-vl)/(num*1.)
      delta = vl
      cu=vl
      do i=0,num
         junk(1:10) = 0.
         if(nEOS(1).eq.1) then
            call mgrprop(junk(1),E1,cu,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.2) then
            call tillprop(junk(1),E1,cu,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.3) then
            call mbprop(junk(1),E1,cu,junk(2),junk(3),temp)
         endif
         write(11,102) cu,junk(1),E1,junk(2),junk(3),nEOS(2)
         m=mod(i,9)
!          write(*,*) 'mode(i,10)',m
!          pause
!          write(12,*) i,m, cu, delta
         if(m.eq.0) then
            delta = cu
         endif
         cu=cu+delta
         if(cu.gt.vu) goto 1000
      enddo
 1000 write(11,*) ''
      write(11,*) ''
      close(11)
      return
  101 format(18x,A)
  102 format(15x,e10.4,4x,e10.4,5x,e10.4,6x,e10.4,6x,e10.4,4x,i)
  103 format(16x,A,7x,A,3x,A,3x,A,3x,A)
      END SUBROUTINE ISOENERGY
!-----------------------------------------------------------------------
! Subroutine: Compression Isentrope
!
!   Plots/exports compression isentrope starting at V0 to pressure P1
!-----------------------------------------------------------------------
      SUBROUTINE Comp_Isen(Upper,incr)
      use EOS
      IMPLICIT NONE
      DOUBLE PRECISION Pi, Ei, Po,E2,Eo
      DOUBLE PRECISION cu,l,Upper, V2
      DOUBLE PRECISION f, f2, df, delta ,dsmall
      DOUBLE PRECISION dP2,dP1
```

```
      DOUBLE PRECISION, dimension(3) :: temp
      DOUBLE PRECISION, dimension(10) :: junk

      INTEGER          incr,step,i
!
      open(11,file='isen_compress.dat',access='append')
      l = V0
      step=incr
      delta = ((v0*1.1)-l)/step*1.
      dsmall=0.99999
      Ei = 0.00000001
      Pi = 0.0
      Up = 0.0

      WRITE(11,102) 'v[cm3/g]','P[Mbar]','E[Mbar-cm3/g]',
     &              'Ur[cm/micro-s]','Region'
      write(11,103) l,Pi,Ei,Up,nEOS(2)
      write(*,103) l,Pi,Ei,Up
      junk(1:10) = 0.0
      V2 = l*dsmall
      E2 = Ei-(V2-L)*Pi
      if(nEOS(1).eq.1) then
         call mgrprop(junk(1),E2,V2,junk(2),junk(3),temp)
      elseif(nEOS(1).eq.2) then
         call tillprop(junk(1),E2,V2,junk(2),junk(3),temp)
      elseif(nEOS(1).eq.3) then
         call mbprop(junk(1),E2,V2,junk(2),junk(3),temp)
      endif
!       write(*,*) Pr, Er, E2,V2,v0,junk(1)
      dP1 = (sqrt(-(junk(1)-Pi)/(V2-L)))
!       write(*,*) Pi, junk(1), V2,L, dP1
      delta = ((v0*1.1)-l)/step*1.
      i=0
      Po = 1

      cu = l
      do while(cu.gt.Upper)
         i=i+1
         Cu=l-delta*i ! Current location (v)
!!         Calculate Eo using foward difference method
!!         Shouldn't Pr+gamma0/v0(E-Er) be replaced with P ??
         Eo = Ei-delta*(Pi)
         junk(1:10)=0.0
         if(nEOS(1).eq.1) then
            call mgrprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
```

```
            elseif(nEOS(1).eq.2) then
               call tillprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
            endif
            Po = junk(1)
            V2=Cu*dsmall
            E2 = Eo-(V2-Cu)*Po
            junk(1:10)=0.0
!                 Find P2
            if(nEOS(1).eq.1) then
               call mgrprop(junk(1),E2,V2,junk(2),junk(3),temp)
            elseif (nEOS(1).eq.2) then
               call tillprop(junk(1),E2,V2,junk(2),junk(3),temp)
            elseif(nEOS(1).eq.3) then
               call mbprop(junk(1),E2,V2,junk(2),junk(3),temp)
            endif
            dP2 =(sqrt(-(junk(1)-Po)/(V2-Cu)))
!           write(*,*) Po, junk(1), V2,Cu, dP2
!           pause
            Up=Up+0.5*(dP2+dP1)*(delta)
            write(11,103) Cu, Po, Eo,Up,nEOS(2)
            write(*,103) Cu, Po, Eo, Up
            Ei=Eo
            Pi=Po
            dP1=dP2
         enddo
!
         close(11)
  102  format(16x,A,6x,A,4x,A,3x,A,3x,A)
  103  format(15x,e10.4,4x,e10.4,4x,e10.4,6x,e10.4,7x,i1)
       END SUBROUTINE Comp_Isen

!-----------------------------------------------------------------------
!  Subroutine: ISENTROPE
!
!  DESCRIPTION:
!    Plots/exports ISENTROP
!-----------------------------------------------------------------------
       SUBROUTINE ISENTROPE(low,hig,incr)
       use EOS
       IMPLICIT NONE
       DOUBLE PRECISION low, hig
       DOUBLE PRECISION L, U, Cu, delta
       DOUBLE PRECISION Pi, Po, dump
```

```
      DOUBLE PRECISION Ei, Eo
      DOUBLE PRECISION Ur, dP1,dP2, E2,E1
      DOUBLE PRECISION error, ierror, f,df,Up2,Peos,v2,dsmall,Ut,f2
      DOUBLE PRECISION,dimension(3)  :: temp
      DOUBLE PRECISION,dimension(10) :: junk

      INTEGER          step, i, sel, incr

      open(11,file='isen.dat',access='append')
      L=low
      U=hig
      step=incr
      dsmall=0.99999
!       dsmall=1.00001
!       L = v0 ! Lower bound
!       U = 0.203478 ! Upper Bound
!       step = 100
      WRITE(11,100) 'Isentrope Calculation'
      WRITE(11,*) 'Ploting from ',L,' to ',U
!       Up=0
!      Find initial Hugoniot Pressure given specific volume
      error=0.0001
      ierror=1.0
      Up=0.1
      do while (ierror.ge.error)
         Peos=0.0
         junk(1:10)=0.0
         E1=(Up**2./2.)
         if(nEOS(1).eq.1) then
            call mgrprop(Peos,E1,L,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.2) then
            call tillprop(Peos,E1,L,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.3) then
            call mbprop(Peos,E1,L,junk(2),junk(3),temp)
         endif
         f=Peos-(Up**2)/(v0-L)
         Up2=dsmall*Up
         E2=(Up2**2./2.)
         junk(1:10)=0.0
         if(nEOS(1).eq.1) then
            call mgrprop(junk(1),E2,L,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.2) then
            call tillprop(junk(1),E2,L,junk(2),junk(3),temp)
         elseif(nEOS(1).eq.3) then
            call mbprop(junk(1),E2,L,junk(2),junk(3),temp)
```

```
            endif
            f2 = junk(1)-(Up2**2)/(v0-L)
            df = (f2-f)/(Up2-Up)
            Ut=Up-f/df
            ierror = abs((Ut-Up)/Up)
            Up=abs(Ut)
            write(*,*) 'Up:',Up
          enddo
          Ur=Up                    ! units cm/micro-s
          Ei= Ur**2./2.
          if(nEOS(1).eq.1) then
            call mgrprop(junk(1),Ei,L,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.2) then
            call tillprop(junk(1),Ei,L,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.3) then
            call mbprop(junk(1),Ei,L,junk(2),junk(3),temp)
          endif
!         Us=(Up*v0)/(v0-L)
!         Pi = Ur*Us/v0            ! units MBar
          Pi=junk(1)
          write(*,*) L, Pi, low
!         pause
!         Ei = Ur**2./2.           ! units MBar-cc/gm
          WRITE(11,102) 'v[cm3/g]','P[Mbar]','E[Mbar-cm3/g]',
     &                  'Ur[cm/micro-s]','Region'
          write(11,103)  L,Pi,Ei,Ur,nEOS(2)
          write(*,103)  L,Pi,Ei,Ur
          junk(1:10) = 0.0
          V2 = L*dsmall
          E2 = Ei-(V2-L)*Pi
          if(nEOS(1).eq.1) then
            call mgrprop(junk(1),E2,V2,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.2) then
            call tillprop(junk(1),E2,V2,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.3) then
            call mbprop(junk(1),E2,V2,junk(2),junk(3),temp)
          endif
          write(*,*) Pr, Er, E2,V2,v0,junk(1)
          dP1 = (sqrt(-(junk(1)-Pi)/(V2-L)))
          write(*,*) Pi, junk(1), V2,L, dP1
          delta = ((v0*1.1)-l)/step*1.
          i=0
          Po = 1
          do while(Po.gt.0.0001)
            i=i+1
```

```
          Cu=L+delta*i ! Current location (v)
!!          Calculate Eo using foward difference method
!!          Shouldn't Pr+gamma0/v0(E-Er) be replaced with P ??
          Eo = Ei-delta*(Pi)
          junk(1:10)=0.0
          if(nEOS(1).eq.1) then
             call mgrprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.2) then
             call tillprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.3) then
             call mbprop(junk(1),Eo,Cu,junk(2),junk(3),temp)
          endif
          if(junk(1).lt.0) then
             goto 1000
          endif
          Po = junk(1)
          V2=Cu*dsmall
          E2 = Eo-(V2-Cu)*Po
          junk(1:10)=0.0
!               Find P2
          if(nEOS(1).eq.1) then
             call mgrprop(junk(1),E2,V2,junk(2),junk(3),temp)
          elseif (nEOS(1).eq.2) then
             call tillprop(junk(1),E2,V2,junk(2),junk(3),temp)
          elseif(nEOS(1).eq.3) then
             call mbprop(junk(1),E2,V2,junk(2),junk(3),temp)
          endif
          dP2 =(sqrt(-(junk(1)-Po)/(V2-Cu)))
!           write(*,*) Po, junk(1), V2,Cu, dP2
!           pause
          Ur=Ur+0.5*(dP2+dP1)*(delta)
          write(11,103) Cu, Po, Eo,Ur,nEOS(2)
          write(*,103) Cu, Po, Eo, Ur
          Ei=Eo
          Pi=Po
          dP1=dP2
        enddo
 1000   write(11,*) ''
        write(11,*) ''
        close(11)
        return
  100   format(10x,A)
  101   format(18x,A)
  102   format(16x,A,6x,A,4x,A,3x,A,3x,A)
  103   format(15x,e10.4,4x,e10.4,4x,e10.4,6x,e10.4,7x,i1)
```

```
  104 format(4x,'rho0: ',e10.4,4x,'c0: ',e10.4,4x,'s1: ',
     &      e10.4,4x,'g0: ',e10.4)
  105 format(4x,'Wilkins Constants: ',e10.4,' : ',e10.4,
     &      ' : ', e10.4,' : ',e10.4)

      END SUBROUTINE ISENTROPE

!-----------------------------------------------------------------------
!  Subroutine: HRISENTROPE
!
!  DESCRIPTION:
!    Plots/exports The release isentrope from a given pressure
!-----------------------------------------------------------------------
      SUBROUTINE HRISENTROPE()
      use EOS
      IMPLICIT NONE
      DOUBLE PRECISION L, U, Cu, delta
      DOUBLE PRECISION Pg
      DOUBLE PRECISION v, P1, P2, E1, E2
      DOUBLE PRECISION df, Pi, Po, Ur,dP1,dP2, dump
      DOUBLE PRECISION Ei, Eo, ierror, error , P0
      DOUBLE PRECISION,DIMENSION(10) :: junk
      DOUBLE PRECISION,DIMENSION(3)  :: temp
      INTEGER          step , i, sel

 1000 write(*,101) 'Pressure which released from'
      read(*,*) Pg
      if(nEOS(1).eq.1) then
         call mgrFindV(Pg,L)
      elseif(nEOS(1).eq.2) then
         call tilFindV(Pg,L)
      elseif(nEOS(1).eq.3) then
         call mbFindV(Pg,L)
      endif
      write(*,*) 'P:',Pg,'v:',L
      step = 10000
      U=L*10
      CALL ISENTROPE(L,U,step)
      goto 5000
 5000 WRITE(*,*) 'Plot again yes (1)/No (2)'
      read(*,*)  sel
      if(sel.eq.1) then
         goto 1000
      endif
```

```
      RETURN
100   format(10x,A)
101   format(18x,A)

      END SUBROUTINE HRISENTROPE
```

Appendix B

Mie-Grüneisen EOS Source Code

The Mie-Grüneisen Module used in the EOS program

```
      MODULE MGR
      IMPLICIT NONE

      CONTAINS

!-----------------------------------------------------------------------
!    SUBROUTINE: mgreos
!
!    DESCRIPTION:
!          Caculaties the thermodynamic variables using the
!          Mie-Gruniesen EOS give two thermodynamic variables
!
!-----------------------------------------------------------------------
      SUBROUTINE mgreos(P1,E1,V1,T1,Ss,temp)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION P1, E1, V1, T1, Ss
      DOUBLE PRECISION,DIMENSION(3)    :: temp
      DOUBLE PRECISION f,df,cu, error,ierror
      DOUBLE PRECISION P2,E2,V2,T2,S2
!       temp has additional arguments that may not be necessary
      DOUBLE PRECISION,DIMENSION(5)    :: junk
      INTEGER,DIMENSION(5)   :: num
!       DOUBLE PRECISION eta,mu

!
!     Checking for non zero variables
!
       num(1:5) = 0
       if(P1.ne.0) then
          num(1)=1
```

```
      endif
      if(E1.ne.0) then
         num(2)=1
      endif
      if(V1.ne.0) then
         num(3)=1
      endif
      if(T1.ne.0) then
         num(4)=1
      endif
      if(Ss.ne.0) then
         num(5)=1
      endif
!
!     Finding the two give thermodynamic variables
!
      if(num(1).eq.1) then
         if(num(2).eq.1) then
!           Pressure and Energy
            goto 1000
         elseif(num(3).eq.1) then
!           Pressure and Volume
            goto 1001
         elseif(num(4).eq.1) then
!           Pressure and Temperature
            goto 1002
         elseif(num(5).eq.1) then
!           Pressure and Entropy
            goto 1003
         else
            write(*,*) 'Two thermodynamic variables must be specified'
            return
         endif
      elseif(num(2).eq.1) then
         if(num(3).eq.1) then
!           Energy and Volume
            goto 1004
         elseif(num(4).eq.1) then
!           Energy and Temperature
            goto 1005
         elseif(num(5).eq.1) then
!           Energy and Entropy
            goto 1006
         else
            write(*,*) 'Two thermodynamic variables must be specified'
```

```
            return
         endif
      elseif(num(3).eq.1) then
         if(num(4).eq.1) then
!        Volume and Temperature
            goto 1007
         elseif(num(5).eq.1) then
!        Volume and Entropy
            goto 1008
         else
            write(*,*) 'Two thermodynamic variables must be specified'
            return
         endif
      elseif(num(4).eq.1) then
         if(num(5).eq.1) then
!        Temperature and Entropy
            goto 1009
         else
            write(*,*) 'Two thermodynamic variables must be specified'
            return
         endif
      else
         write(*,*) 'Two thermodynamic variables must be specified'
         return
      endif
!
!     Solving the MGR EOS using given Pressure and Energy
!     to find specific volume, temperature and entropy
 1000 if(temp(1).ne.0) then
         cu=temp(1)    ! Initial guess
      else
         cu=v0*0.25
      endif

      error=0.01
      ierror=1.0

      do while (ierror.ge.error)
        call hugoRef(cu)
        P2=Pr+gamma0/v0*(E1-Er)
        f=P1-P2
        V2=cu*0.9999
        call hugoRef(V2)
        P2=Pr+gamma0/v0*(E1-Er)
        junk(1)=P1-P2
```

```
        df = (junk(1)-f)/(V2-cu)
        cu=cu-f/df
        call hugoRef(cu)
        P2 =Pr+gamma0/v0*(E1-Er)
        ierror=abs(P1-P2)/P1
      enddo
!     Found the V that satisfies the given P and E
      V1=cu
!
!     Need to figure out how to find entropy and Temperature
!
      return
!
!     Solving the MGR EOS using given Pressure and Volume
!     to find entergy, temperature and entropy
!
 1001 call hugoRef(V1)
      if(temp(1).ne.0) then
         cu=temp(1)    ! Initial guess
      else
         cu=Er  ! Reference Energy
      endif

      error=0.01
      ierror=1.0

      do while (ierror.ge.error)
!        hugoRef(V1)
        P2=Pr+gamma0/v0*(cu-Er)
        f=P1-P2
!     Need to replace with analytical expression
        E2=cu*0.9999
        P2 = Pr+gamma0/v0*(E2-Er)
        junk(1)=P1-P2
        df = (junk(1)-f)/(E2-cu)
        cu=cu-f/df
        P2 = Pr+gamma0/v0*(cu-Er)
        ierror=abs(P1-P2)/P1
      enddo
!     Found the E that satisfies the given P and v
      E1=cu
!
!     Need to figure out how to find entropy and Temperature
!
      return
```

```
!
!     Solving the MGR EOS using given Pressure and Temperature
!     to find energy, specific volume, and entropy
!
 1002 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Pressure and Entropy
!     to find energy, specific volume, and temperature
!
 1003 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Energy and Volume
!     to find pressure, temperature and entropy
!
 1004 call hugoRef(V1)
      P1=Pr+gamma0/v0*(E1-Er)
      return
!
!     Solving the MGR EOS using given Energy and Temperature
!     to find pressure, specific volume, and entropy
!
 1005 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Energy and Entropy
!     to find pressure, specific volume, and temperature
!
 1006 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Volume and Temperature
!     to find pressure, energy, and entropy
!
 1007 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Volume and Entropy
!     to find pressure, energy, and temperature
!
 1008 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the MGR EOS using given Temperature and Entropy
```

```
!      to find pressure, energy, and specific volume
!
 1009 write(*,*) 'This Feature needs to be added'
      return

      END SUBROUTINE


      SUBROUTINE hugoRef(vi)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION vi, x,x2,E2

      if(nEOS(2).eq.1) then
         goto 1000
      elseif(nEOS(2).eq.2) then
         goto 2000
      elseif(nEOS(2).eq.3) then
         goto 3000
      endif
 1000 Pr = (C0**2*(v0-vi))/((v0-s1*(v0-vi))**2)
      Er = 0.5*Pr*(v0-vi)
      goto 5000
 2000 x = 1.-(vi/v0)
      Er =(a(1)+a(2)*x+a(3)*x**2.+a(4)*x**3.+a(5)*x**4.)
      Pr =(a(2)+2*a(3)*x+3*a(4)*x**2.+4*a(5)*x**3.)*rho0
      goto 5000
 3000 x = v0/vi
      Er = 3*v0*(a(1)*(x**(1./3.)-1)+a(2)/2.*(x**(2./3.)-1)
     &  + a(3)/3.*(x-1)+a(4)/4.*(x**(4./3.)-1)+a(5)/5.*(x**(5./3.)-1))
      Er = Er/100000
      Pr=(v0**(4./3.)*(a(1)*vi**(4./3.)+a(2)*vi*v0**(1./3.)+
     &    a(3)*(vi*v0)**(2./3)+a(4)*vi**(1./3.)*v0+
     &    a(5)*v0**(4./3.)))/vi**(8./3.)
      Pr = Pr/100
      goto 5000
 5000 return
      END SUBROUTINE


      SUBROUTINE eosINP(sel)
      USE EOS
      IMPLICIT NONE
      integer sel
      nEOS(1) = sel  ! MGR EOS Parameters
```

```
      WRITE(*,100) '-------------------MENU-------------------'
      WRITE(*,101) '(1) Enter Hugoniot Paramters'
      WRITE(*,101) '(2) Enter Wilkin Energy Parameters'
      WRITE(*,101) '(3) Elastic Cold Curve'
      WRITE(*,101) '(5) Exit System'
      read(*,*) sel
      nEOS(2) = sel ! Reference Curve Definition
      if(sel.eq.1.or.sel.eq.2) then
         write(*,100) 'Enter rho0,C0, s1, and gamma0'
         read(*,*) rho0, C0, s1, gamma0
         v0   = 1./rho0
         if(sel.eq.2) then
            write(*,100) 'Enter e00, e01, e02, e03, e04'
            write(*,100) 'see Wilkins pgs 60-63'
            read(*,*)  a(1), a(2), a(3), a(4), a(5)
         endif
      elseif(sel.eq.3) then
         write(*,100) 'Cold curve approximation'
         write(*,100) 'Enter v0, gamma0, a1,a2,a3,a4,a5'
         write(*,100) 'see Lomonosov'
         read(*,*) v0, gamma0,a(1),a(2),a(3),a(4),a(5)
         rho0 = 1./v0
      endif
      RETURN
  100 format(15x,'-----',A)
  101 format(18x,A)
      END SUBROUTINE



!       SUBROUTINE hugoProp(vi)
!       USE EOS
!       IMPLICIT NONE
!       DOUBLE PRECISION vi
!       CALL hugoRef(vi)
!       Up = sqrt(((Pr-(gamma0/v0)*Er)/(1/(v0-vi)-gamma0/(2*v0))))
!       Up = Up*10
!       Us = (Up*v0) / (v0-vi)
!       P = Up*Us/v0*0.01
!       E = 0.5*P*(v0-vi)
!       return
!       END SUBROUTINE

      SUBROUTINE findV(Pg,vout)
      USE EOS
```

```
      IMPLICIT NONE
      DOUBLE PRECISION Pg,f,df,Peos,Peos2,f2,vout
      DOUBLE PRECISION ierror, error,cu,P1,E1

      error= 0.001
      ierror = 1.0

      cu=v0*0.5
      do while(ierror.ge.error)
!       Up=sqrt(Pg*(v0-cu))
!       E1=(Up**2)/2.
        CALL hugoRef(cu)
        Up=sqrt(Pg*(v0-cu))
        E1=(Up**2)/2.
        Peos = Pr+(gamma0/v0)*(E1-Er)
        f=Pg-Peos
        Up=sqrt(Pg*(v0-cu*0.9999))
        E1=(Up**2)/2.
        CALL hugoRef(cu*0.9999)
        Peos2 = Pr+(gamma0/v0)*(E1-Er)
        f2 = Pg-Peos2
        df= (f2-f)/(cu*0.9999-cu)
        cu = cu -f/df
        CALL hugoRef(cu)
        Up=sqrt(Pg*(v0-cu))
        E1=(Up**2)/2.
        P1=Pr+(gamma0/v0)*(E1-Er)
        ierror = abs((P1-Pg)/Pg)
      end do
      vout = cu
      write(*,*) vout,Pg,P1
      pause
      return
      END SUBROUTINE

      END MODULE
```

Appendix C

Tillotson EOS Source Code

The Tillotson Module used in the EOS program.

```
      MODULE TILLOTSON
      IMPLICIT NONE

      CONTAINS

!-----------------------------------------------------------------------
!   SUBROUTINE: tilleos
!
!   DESCRIPTION:
!         Caculaties the thermodynamic variables using the
!         Tillotson EOS give two thermodynamic variables
!
!-----------------------------------------------------------------------

      SUBROUTINE tilleos(P1,E1,V1,T1,Ss,temp)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION P1, E1, V1, T1, Ss
      DOUBLE PRECISION,DIMENSION(3)    :: temp
      DOUBLE PRECISION f,df,cu, error,ierror
      DOUBLE PRECISION P2,E2,V2,T2,S2
!       temp has additional arguments that may not be necessary
      DOUBLE PRECISION,DIMENSION(5)    :: junk
      INTEGER,DIMENSION(5)   :: num
      DOUBLE PRECISION eta,mu
!
!     Checking for non zero variables
!
       num(1:5) = 0
       if(P1.ne.0) then
          num(1)=1
```

```
      endif
      if(E1.ne.0) then
         num(2)=1
      endif
      if(V1.ne.0) then
         num(3)=1
      endif
      if(T1.ne.0) then
         num(4)=1
      endif
      if(Ss.ne.0) then
         num(5)=1
      endif
!
!     Finding the two give thermodynamic variables
!
      if(num(1).eq.1) then
         if(num(2).eq.1) then
!           Pressure and Energy
            goto 1000
         elseif(num(3).eq.1) then
!           Pressure and Volume
            goto 1001
         elseif(num(4).eq.1) then
!           Pressure and Temperature
            goto 1002
         elseif(num(5).eq.1) then
!           Pressure and Entropy
            goto 1003
         else
            write(*,*) 'Two thermodynamic variabls must be specified'
            return
         endif
      elseif(num(2).eq.1) then
         if(num(3).eq.1) then
!           Energy and Volume
            goto 1004
         elseif(num(4).eq.1) then
!           Energy and Temperature
            goto 1005
         elseif(num(5).eq.1) then
!           Energy and Entropy
            goto 1006
         else
            write(*,*) 'Two thermodynamic variabls must be specified'
```

```
            return
          endif
       elseif(num(3).eq.1) then
          if(num(4).eq.1) then
!         Volume and Temperature
            goto 1007
          elseif(num(5).eq.1) then
!         Volume and Entropy
            goto 1008
          else
            write(*,*) 'Two thermodynamic variabls must be specified'
            return
          endif
       elseif(num(4).eq.1) then
          if(num(5).eq.1) then
!         Temperature and Entropy
            goto 1009
          else
            write(*,*) 'Two thermodynamic variabls must be specified'
            return
          endif
       else
          write(*,*) 'Two thermodynamic variabls must be specified'
          return
       endif
 1000  if(temp(1).ne.0) then
          cu=temp(1)    ! Initial guess
       else
          cu=v0*0.25
       endif

       error=0.01
       ierror=1.0
!
!     Need to add logic for adding multi-phase check
!
       eta = v0/cu
       mu = eta-1
       do while (ierror.ge.error)
         CALL findP(P2,E1,cu)
         f=P1-P2
         V2=cu*0.9999
         eta =v0/V2
         mu=eta-1
         CALL findP(P2,E1,V2)
```

```
        junk(1)=P1-P2
        df = (junk(1)-f)/(V2-cu)
        cu=cu-f/df
        eta = v0/cu
        mu = eta-1
        CALL findP(P2,E1,cu)
        ierror=abs(P1-P2)/P1
      enddo
!     Found the V that satisfies the given P and E
      V1=cu
!
!     Need to figure out how to find entropy and Temperature
!
      return
!
!     Solving the Tillotson EOS using given Pressure and Volume
!     to find entergy, temperature and entropy
!
 1001 if(temp(1).ne.0) then
        cu=temp(1)    ! Initial guess
      else
        cu=ti(1)*0.25   ! Eo
      endif

      error=0.01
      ierror=1.0

      eta = v0/V1
      mu = eta-1
      do while (ierror.ge.error)
        CALL findP(P2,cu,V1)
        f=P1-P2
        E2=cu*0.9999
        CALL findP(P2,E2,V1)
        junk(1)=P1-P2
        df = (junk(1)-f)/(E2-cu)
        cu=cu-f/df
        CALL findP(P2,cu,V1)
        ierror=abs(P1-P2)/P1
      enddo
!     Found the E that satisfies the given P and v
      E1=cu
!
!     Need to figure out how to find entropy and Temperature
!
```

```
      return
!     Solving the Tillotson EOS using given Pressure and Temperature
!     to find energy, specific volume, and entropy
!
!
 1002 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Pressure and Entropy
!     to find energy, specific volume, and temperature
!
 1003 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Energy and Volume
!     to find pressure, temperature and entropy
!
 1004 CALL findP(P1,E1,V1)

      return
!
!     Solving the Tillotson EOS using given Energy and Temperature
!     to find pressure, specific volume, and entropy
!
 1005 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Energy and Entropy
!     to find pressure, specific volume, and temperature
!
 1006 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Volume and Temperature
!     to find pressure, energy, and entropy
!
 1007 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Volume and Entropy
!     to find pressure, energy, and temperature
!
 1008 write(*,*) 'This Feature needs to be added'
      return
!
```

```
!     Solving the Tillotson EOS using given Temperature and Entropy
!     to find pressure, energy, and specific volume
!
 1009 write(*,*) 'This Feature needs to be added'
      return
      END SUBROUTINE

      SUBROUTINE findP(P1,E1,V1)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION P1,E1,V1
      DOUBLE PRECISION check, mu,eta

      check = V1/v0
      eta = v0/V1
      mu = eta-1.

      if(check.ge.ti(3)) then
         nEOS(2)=4
      elseif((check.gt.1).and.(E1.gt.ti(2))) then
         nEOS(2)=4
      else
         nEOS(2)=3
      endif

      if(nEOS(2).eq.3) then
         P1 = (ti(4)+ti(5)/(E1/(ti(1)*eta**2.)
     &          +1.))*E1/V1+ti(6)*mu+ti(7)*mu**2.
      elseif(nEOS(2).eq.4) then
         P1 =ti(4)*E1/V1+(ti(5)*E1/(V1*(E1/(ti(1)*(eta**2.))+1.))+
     &      ti(6)*mu*exp(-ti(9)*(V1/v0-1.)))*exp(-ti(8)*((V1/v0)-1.)**2.)
      endif
      RETURN
      END SUBROUTINE

      SUBROUTINE eosINP(sel)
      USE EOS
      IMPLICIT NONE
      integer sel
      nEOS(1) = sel ! Tillotson EOS Parameters
      write(*,100) 'Tillotson EOS Parameters'
      write(*,100) 'Enter rho0,E0,Es,Vs'
      read(*,*) rho0, ti(1), ti(2), ti(3)

      write(*,100) 'Enter a,b,A,B'
```

```
      read(*,*) ti(4),ti(5),ti(6),ti(7)
      write(*,100) 'Enter Alpha and Beta'
      read(*,*) ti(8),ti(9)
      v0 = 1./rho0
      RETURN
  100 format(15x,'-----',A)
      END SUBROUTINE

      SUBROUTINE findV(Pg,vout)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION Pg,f,df,Peos,Peos2,f2,vout
      DOUBLE PRECISION ierror, error, cu
      DOUBLE PRECISION eta,mu,eta2,mu2,vi2

      error=0.001
      ierror=1.0

      cu=v0*0.25

      do while(ierror.ge.error)
         eta=v0/cu
         mu=eta-1
         Up=sqrt(Pg*(v0-cu))
         E=(Up**2)/2.
         Peos = (ti(4)+ti(5)/(E/(ti(1)*eta**2)
     &          +1))*(E/cu)+ti(6)*mu+ti(7)*mu**2
         f=Pg-Peos
         vi2=cu*0.9999
         eta2=v0/vi2
         mu2=eta2-1
         Up=sqrt(Pg*(v0-vi2))
         E=(Up**2)/2.
         Peos2 = (ti(4)+ti(5)/(E/(ti(1)*eta2**2)
     &          +1))*(E/vi2)+ti(6)*mu2+ti(7)*mu2**2
         f2=Pg-Peos2
         df = (f2-f)/(vi2-cu)
         cu = cu -f/df
         P = (ti(4)+ti(5)/(E/(ti(1)*eta**2)
     &          +1))*(E/cu)+ti(6)*mu+ti(7)*mu**2
!          write(*,*) cu,P,Pg
         ierror = abs((P-Pg)/Pg)
      enddo
      vout = cu
```

```
    RETURN
    END SUBROUTINE

    END MODULE
```

Appendix D

Multi-Branch Analytical EOS Source Code

The Multi-Branch Analytical Module used in the EOS program.

```
MODULE MBEOS
IMPLICIT NONE

CONTAINS

!-----------------------------------------------------------------------
! CURRENT VALUES
!   rho0    = 2.712 g/cm^3
!   T0      = 298  K
!   Pc      = 0.0018202 Mbar
!   Ec      = 0.122 Mbar-cm^3/g
!   E1      = 0.01 Mbar-cm^3/g! ~1 kJ/g This is esencietly the bottom of
!                                       the sat dome
!   gamma0 = 2.14
!   c0      = 0.54518
!   s       = 1.2592
!  m        = 8      ! not correct
!   n       = 0.7    ! not correct
!   k       = 1  !assumed since i don't have gas data to compare
!                !k is a fitting paramter and is only used in
!                !the gas region
!   xhi     = 2/3    !No internal degree of freedome
!   R       = 3.08173e-6 (Mbar-cm^3)/(g-K)
!
!-----------------------------------------------------------------------


!-----------------------------------------------------------------------
```

```
!    SUBROUTINE: multieos
!
!    DESCRIPTION:
!          Caculaties the thermodynamic variables using the
!          Multi-branch analytical EOS give two thermodynamic
!          variables
!
!          The first attempt at this is using the models and
!          presentation shown in Gathers. This will probable
!          need to be tuned for Aluminum (the original work
!          was developed for Lithium)
!
!-----------------------------------------------------------------------

      SUBROUTINE multieos(P1,E1,V1,T1,Ss,temp)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION P1, E1, V1, T1, Ss
      DOUBLE PRECISION,DIMENSION(3)    :: temp
      DOUBLE PRECISION f,df,cu, error,ierror
      DOUBLE PRECISION P2,E2,V2,T2,S2
!       temp has additional arguments that may not be necessary
      DOUBLE PRECISION,DIMENSION(5)    :: junk
      INTEGER,DIMENSION(5)   :: num
      DOUBLE PRECISION eta,mu
!
!     Checking for non zero variables
!
      num(1:5) = 0
      if(P1.ne.0) then
         num(1)=1
      endif
      if(E1.ne.0) then
         num(2)=1
      endif
      if(V1.ne.0) then
         num(3)=1
      endif
      if(T1.ne.0) then
         num(4)=1
      endif
      if(Ss.ne.0) then
         num(5)=1
      endif
!
```

```
!       Finding the two give thermodynamic variables
!
        if(num(1).eq.1) then
           if(num(2).eq.1) then
!          Pressure and Energy
              goto 1000
           elseif(num(3).eq.1) then
!          Pressure and Volume
              goto 1001
           elseif(num(4).eq.1) then
!          Pressure and Temperature
              goto 1002
           elseif(num(5).eq.1) then
!          Pressure and Entropy
              goto 1003
           else
              write(*,*) 'Two thermodynamic variabls must be specified'
              return
           endif
        elseif(num(2).eq.1) then
           if(num(3).eq.1) then
!          Energy and Volume
              goto 1004
           elseif(num(4).eq.1) then
!          Energy and Temperature
              goto 1005
           elseif(num(5).eq.1) then
!          Energy and Entropy
              goto 1006
           else
              write(*,*) 'Two thermodynamic variabls must be specified'
              return
           endif
        elseif(num(3).eq.1) then
           if(num(4).eq.1) then
!          Volume and Temperature
              goto 1007
           elseif(num(5).eq.1) then
!          Volume and Entropy
              goto 1008
           else
              write(*,*) 'Two thermodynamic variabls must be specified'
              return
           endif
        elseif(num(4).eq.1) then
```

```
          if(num(5).eq.1) then
!         Temperature and Entropy
             goto 1009
          else
             write(*,*) 'Two thermodynamic variabls must be specified'
             return
          endif
       else
          write(*,*) 'Two thermodynamic variabls must be specified'
          return
       endif
!
!      Solving the Tillotson EOS using given Pressure and Energy
!      to find specific volume, temperature and entropy
 1000  if(temp(1).ne.0) then
          cu=temp(1)    ! Initial guess
       else
          cu=v0*0.25
       endif

       error=0.01
       ierror=1.0
!
!     Need to add logic for adding multi-phase check
!
       do while (ierror.ge.error)
         CALL findP(P2,E1,cu)
          f=P1-P2
          V2=cu*0.9999
         CALL findP(P2,E1,V2)
          junk(1)=P1-P2
          df = (junk(1)-f)/(V2-cu)
          cu=cu-f/df
         CALL findP(P2,E1,cu)
          ierror=abs(P1-P2)/P1
       enddo
!      Found the V that satisfies the given P and E
       V1=cu
       if(nEOS(2)==1) then   ! Solid Phase
          junk(1) = -3*mb(9)*mb(1)
          junk(2) = 1-V1/v0
          junk(3) = junk(1)+gamma0*junk(1)*junk(2)+
     &         0.5*(c0**2.+gamma0**2.*junk(1))*junk(2)**2.+
     &         1./6.*(4*s1*C0**2.+gamma0**3.*junk(1))*junk(2)**3.
          T1=(E1-junk(3))/(3*mb(9))
```

```
      elseif(nEOS(2)==2) then ! Expanded phase, E greater than Ec
!          T1 = Tg+(v0/V1)**0.2*(Ts-Tg)
      elseif(nEOS(2)==3) then ! Interpolation region
                              ! E greater than E1 (what is E1???)

      else  ! nEOS(2) =4 ! below dome

      endif

!
!     Need to figure out how to find entropy and Temperature
!
      junk(1:10) = 0.
      return
!
!     Solving the Tillotson EOS using given Pressure and Volume
!     to find entergy, temperature and entropy
!
 1001 if(temp(1).ne.0) then
         cu=temp(1)    ! Initial guess
      else
         cu=ti(1)*0.25   ! Eo
      endif

      error=0.01
      ierror=1.0

      do while (ierror.ge.error)
        CALL findP(P2,cu,V1)
        f=P1-P2
!     Need to replace with analytical expression
        E2=cu*0.9999
        CALL findP(P2,E2,V1)
        junk(1)=P1-P2
        df = (junk(1)-f)/(E2-cu)
        cu=cu-f/df
        CALL findP(P2,cu,V1)
        ierror=abs(P1-P2)/P1
      enddo
!     Found the E that satisfies the given P and v
      E1=cu
!
!     Need to figure out how to find entropy and Temperature
!
      return
```

```
!
!     Solving the Tillotson EOS using given Pressure and Temperature
!     to find energy, specific volume, and entropy
!
 1002 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Pressure and Entropy
!     to find energy, specific volume, and temperature
!
 1003  write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Energy and Volume
!     to find pressure, temperature and entropy
!
 1004 CALL findP(P1,E1,V1)

      return
!
!     Solving the Tillotson EOS using given Energy and Temperature
!     to find pressure, specific volume, and entropy
!
 1005 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Energy and Entropy
!     to find pressure, specific volume, and temperature
!
 1006 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Volume and Temperature
!     to find pressure, energy, and entropy
!
 1007 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Volume and Entropy
!     to find pressure, energy, and temperature
!
 1008 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Tillotson EOS using given Temperature and Entropy
```

```
!      to find pressure, energy, and specific volume
!
 1009 write(*,*) 'This Feature needs to be added'
      return
      END SUBROUTINE

      SUBROUTINE findP(P1,E1,V1)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION P1,E1,V1
      DOUBLE PRECISION check, mu,eta
      DOUBLE PRECISION ph, x
      DOUBLE PRECISION,DIMENSION(10)    :: junk
!       open(12,access='append')
      if(V1.lt.v0) then
         nEOS(2) = 1    ! Solid Phase ~ using MGR
      elseif(E1.gt.mb(3)) then
         nEOS(2) = 2    ! Expanded phase, E greater than Ec
      elseif(E1.gt.mb(4)) then
         nEOS(2) = 3    ! Interpolation region
                        ! E greater than E1 (what is E1???)
      else
         nEOS(2) = 4    ! below dome
      endif

      if(nEOS(2).eq.1) then
!  There's probable better ways to do this but this is how it's presented
!  in gathers, optimization comes second
!  MGR eos for compression (solid) region (region 1)
         x = 1-V1/v0
         ph= rho0*c0**2.*x/(1.-S1*x)**2.
         P1=ph*(1-gamma0*x/2.)+gamma0*rho0*E1
      elseif(nEOS(2).eq.2) then
!  Gas Model (region 2)
!         write(12,*) V1, E1, mb(3), mb(8), mb(7),v0
         P1=mb(8)*(E1-mb(3))/V1+(gamma0/v0*E1-mb(8)*(E1-mb(3))/v0)
     &         *(v0/V1)**mb(7)/V1

      elseif(nEOS(2).eq.3) then
!  Interpolation Region 3
         junk(1) = rho0*c0*(v0/V1-1.)+gamma0*rho0*mb(4)   ! p41
         junk(2) = c0**2.                                 ! dpr41
         junk(3) = 0.0                                    ! dpe41
         junk(4) = mb(2)*exp(mb(5))*exp(-mb(6)*mb(3)/E1)  ! p511
         junk(5) = 0.0                                    ! dpr511
```

```
      junk(6) = junk(4)*mb(6)*mb(3)/E1**2.                    ! dpe511
      junk(7) = gamma0*rho0*E1*(v0/V1)**mb(7)                 ! p512
      junk(8) = junk(7)*mb(7)*V1                                ! dpr512
      junk(9) = junk(7)/E1                                      ! dpe512
      if (junk(4).gt.junk(7)) then                            ! p511>p512
         junk(4)=junk(7)                                      ! now p51
         junk(5)=junk(8)                                      ! dpr51
         junk(6)=junk(9)                                      ! dpe51
      endif
      junk(7)=gamma0*rho0*mb(3)*(v0/V1)**mb(7)                !p2
      junk(8)=junk(7)*mb(7)*V1                                !pdr2
      junk(9)=0.0                                             !pde2
      if(junk(1).gt.junk(4)) then                             !p41>p51
         P1=junk(1)+(E1-mb(4))*(junk(7)-junk(1))/(mb(3)-mb(4))
      else
         P1=junk(4)+(E1-mb(4))*(junk(7)-junk(4))/(mb(3)-mb(4))
      endif
   elseif(nEOS(2).eq.4) then
      junk(1)=mb(2)*exp(mb(5))*exp(-mb(6)*mb(3)/E1)           !p51
      junk(2)=gamma0*rho0*E1*(v0/v1)**mb(7)                   !p52
      if (junk(1).gt.junk(2)) then                            !p51>p52
         junk(1) = junk(2)                                    !p52=p5
      endif
!                                                          junk(1) is now p51
      junk(2)=rho0*c0**2*(v0/V1-1)+gamma0*rho0*E1             !p4
      if (junk(2).gt.junk(1)) then
         P1=junk(2)
      else
         P1=junk(1)
      endif
   endif
   junk(1:10)=0
   RETURN
   END SUBROUTINE
!
   SUBROUTINE eosINP(sel)
   USE EOS
   IMPLICIT NONE
   integer sel
   nEOS(1) = sel ! Tillotson EOS Parameters
   write(*,100) 'Multi=Branch EOS Parameters'
   write(*,100) 'Enter rho0, gamma0, T0'
!    read(*,*) rho0,gamma0, mb(1)
   rho0 = 2.712
   gamma0=2.14
```

```
      mb(1) = 298
      write(*,*) rho0,gamma0,mb(1)
      write(*,100) 'Enter Pc,Ec,E1,C0,S'
!       read(*,*) mb(2),mb(3),mb(4),C0,S1
      mb(2) = 0.0018202
      mb(3) = 0.122
!       mb(4) = 0.01
      mb(4) = mb(3) ! Still not sure what E1 does
      C0 = 0.54518
      s1 = 1.2592
      write(*,*) mb(2),mb(3),mb(4),C0,S1
      write(*,100) 'Enter m, n, k, xi, R'
!       read(*,*) mb(5),mb(6),mb(7),mb(8),mb(9)
      mb(5) = 1.0     ! wrong
      mb(6) = 1.1   ! wrong
      mb(7) = 1.1
      mb(8) = 2./3.
      mb(9) = 3.08173e-6
      write(*,*) mb(5),mb(6),mb(7),mb(8),mb(9)
      v0 = 1./rho0
      write(*,*) v0
      RETURN
  100 format(15x,'-----',A)
      END SUBROUTINE

      SUBROUTINE findV(Pg,vout)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION Pg,f,df,Peos,Peos2,f2,vout
      DOUBLE PRECISION ierror, error, cu,E1
      DOUBLE PRECISION eta,mu,eta2,mu2,vi2

      error=0.001
      ierror=1.0

      cu=v0*0.25

!       Up=sqrt(Pg*(v0-cu))
!       E1=(Up**2)/2.
!      Peos = (ti(4)+ti(5)/(E/(ti(1)*eta**2)
!     &           +1))*(E/cu)+ti(6)*mu+ti(7)*mu**2
!       call findP(Peos,E1,cu)

      do while(ierror.ge.error)
         Up=sqrt(Pg*(v0-cu))
```

```
            E1=(Up**2)/2.
            call findP(Peos,E1,cu)
            f=Pg-Peos
            vi2=cu*0.9999
            Up=sqrt(Pg*(v0-vi2))
            E1=(Up**2)/2.
!             Peos2 = (ti(4)+ti(5)/(E/(ti(1)*eta2**2)
!     &             +1))*(E/vi2)+ti(6)*mu2+ti(7)*mu2**2
            call findP(Peos2,E1,vi2)
            f2=Pg-Peos2
            df = (f2-f)/(vi2-cu)
            cu = cu -f/df
            Up=sqrt(Pg*(v0-cu))
            E1=(Up**2)/2.
!             write(*,*) 'Before hugoProp call'
!             P = (ti(4)+ti(5)/(E/(ti(1)*eta**2)
!     &             +1))*(E/cu)+ti(6)*mu+ti(7)*mu**2
            call findP(Peos,E1,cu)
            write(*,*) cu,Peos,Pg
!             pause
            ierror = abs((Peos-Pg)/Pg)
          enddo
          vout = cu

          RETURN
          END SUBROUTINE

          END MODULE
```

Appendix E

Bushman-Lomonosov EOS Source Code

The Bushman-Lomonosov EOS was calculated using a separate program that the other EOSs. The complete program is listed below.

```
      PROGRAM BushmanEOS
!-------------------------------------------------------
!
!  This program calulates shock hugoniot, release isentrop
!  end points using bushman EOS and
!  gibbs relationship
!
!-------------------------------------------------------
      IMPLICIT NONE
      DOUBLE PRECISION :: V1,vu,vl,vg,Ps
      DOUBLE PRECISION :: T1,P1,E1,Ss,G
      DOUBLE PRECISION :: vliq,vgas,Pg
      DOUBLE PRECISION :: P2,gl, gg, dg1, dg2, dp,dg
      DOUBLE PRECISION :: cu,Ptemp, f1,df,dv
      DOUBLE PRECISION :: Tl, Tu
      DOUBLE PRECISION :: delta
      DOUBLE PRECISION :: error
      DOUBLE PRECISION :: ierror,ierror2
      DOUBLE PRECISION :: Plow, Pup
      DOUBLE PRECISION :: Fl1, Fl2, Fs1, Fs2,V2, Pl
      DOUBLE PRECISION :: high, low
      DOUBLE PRECISION :: v_1, v_2

      DOUBLE PRECISION,DIMENSION(3)    :: temp
      DOUBLE PRECISION,DIMENSION(5)    :: dump
      DOUBLE PRECISION,DIMENSION(6)    :: guess
      DOUBLE PRECISION,DIMENSION(6,1000)::dome
```

```
      DOUBLE PRECISION :: Fl, Fs

      INTEGER          :: step,tstep
      INTEGER          :: step1, step2
      INTEGER          :: i
      INTEGER          ::counter, maxcount,check
      INTEGER          :: flag

      INTEGER          :: incr


      maxcount = 100
      delta = 1.001
      error = 0.01
      ierror = 1.0
      ierror2 = 1.0
      counter = 20
      check =0

!       E1 =


!       E1 = 27.48
      E1 = 0.
      P1 = 0.
      Ss = 0.
!       Ss = 5.195
!       V1 = 0.2083
!       T1 = 0
!       temp(1) = 11000
      V1= 0.37
!       T1 = 0.
      T1 = .293
!       temp(1) = 1121.14833403170
!       temp(1) = 300
!       call busheos(P1,E1,V1,T1,Ss,temp)
      call bush(P1,E1,V1,T1,Ss,temp)
      write(*,101) V1,T1,P1,E1,Ss,temp(3),temp(2)
! Isentrope calculation
!       call HRISENTROPE()

! Reshock calculation
!       v_1 = 0.25885
!       v_1 = 0.237611
```

```
!         v_1 = 0.216409
!         v_1 = 0.260656
!         v_1 = 0.256342
!         v_1 = 0.216409
!         v_1 = 0.219653
        v_1 = 0.237869
        step1= 40
!         v_2 = 0.225701
!         v_2 = 0.201069
!         v_2 = 0.176217
!         v_2 = 0.230568
!         v_2 = 0.224189
!         v_2 = 0.179572
!         v_2 = 0.190892
        v_2 = 0.210988
        step2 = 40
        call reshock(v_1,step1,v_2,step2)


! Hugoniot calculation
!         high = 0.36
!         low = 0.1
!         incr = 105
!         call Hugoniot(low,high,incr)

!         open(11,file='Isotherm.dat')

!         T1 = 20.0
!         vu = 10000
!         step = 1000
!         delta = 1./counter
!         V1 = 0.1
!         write(11,102) "v","T","P","E","S","G","State"
!         do while (check ==0)
!            do i=1,counter-2
!               P1 = 0.0
!               E1 = 0.0
!               Ss = 0.0
!               temp(1:3) = 0.0
!               call bush(P1,E1,V1,T1,Ss,temp)
!               G = E1+P1*V1-T1*Ss
!               write(11,101) V1,T1,P1,E1,Ss,G,temp(2)
!               V1 =V1+delta
!               if (V1.gt.vu) then
!                  check = 1
```

```
!               endif
!            enddo
!            delta = delta*10
!         enddo
!!
!         close(11)
!

  101 format(3x,e10.4,4x,e10.4,4x,e10.4,
     &          4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.3)
  102 format(3x,A,6x,A,6x,A,6x,A,6x,A,6x,A,6x,A)
  103 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
  104 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
      END PROGRAM



!-----------------------------------------------------------------------
!  Subroutine: Hugoniot
!
!  DESCRIPTION:
!    Plots/exports isotherms for over a   given temperature range and
!    number of steps
!-----------------------------------------------------------------------
      SUBROUTINE Hugoniot(hig,low,incr)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION L, U, delta, Cu, low, hig,dump,E2,Up2,f2
      DOUBLE PRECISION f,df,ierror,error,Peos,Ut,V2,P2,dsmall,E1
!       DOUBLE PRECISION E,Up,Us,P,v0
      DOUBLE PRECISION,DIMENSION(3)  :: temp
      DOUBLE PRECISION,DIMENSION(10) :: junk
      integer sel, step, i, incr
      integer counter, maxcount

!       write(*,*) 'v0: ',v0
      v0 =  0.370
      counter = 0
      maxcount = 100

      open(11,file='hugo.dat',access='append')
      open(9)
      L = low
      U = hig
      step = incr
```

```
      delta = (U-L)/(step*1.)
      WRITE(11,100) 'Principle Hugoniot Calculation'
      WRITE(11,102) 'v[cm3/g]','P[Mbar]','Up[cm/micro-s]',
     &              'Us[cm/micro-s]'
!       dsmall=0.99999
      dsmall = 1.01
      if(Up.eq.0.) then
         Up=0.1
      endif
      temp(1:3)=0.0
!       L = .300
 2100 do i=0,step
         Cu=L+delta*i
         write(9,*) "v:", cu, "Up:",Up
         dump=0
         error=0.001
         ierror=1.0
!          E1 = 5.0e-3
!          call busheos(Peos,E1,cu,junk(2),junk(3),temp)

!          return
         do while (ierror.ge.error)
            Peos=0.0
            junk(1:10)=0.0
            E1=((Up**2.)/2.)
            write(9,107) cu, Up, (Up**2)/(v0-cu), E1
            call busheos(Peos,E1,cu,junk(2),junk(3),temp)
            f=Peos-(Up**2)/(v0-cu)
            write(9,107) cu, Peos, E1, temp(2)
            Up2=dsmall*Up
            E2=((Up2**2./2.))
            temp(1) = junk(2)
            junk(1:10)=0.0
            write(9,106) (Up2**2)/(v0-cu), E1, temp(1)
            call busheos(junk(1),E2,cu,junk(2),junk(3),temp)
            f2 = junk(1)-(Up2**2)/(v0-cu)
            df = (f2-f)/(Up2-Up)
            Ut=Up-f/df
            write(9,108) cu, Peos, E1, f2, df, Ut
            ierror = abs((Ut-Up)/Up)
            Up=abs(Ut)
            write(9,*) "ierror=",ierror
            write(9,*)
            if (counter.ge.maxcount) then
               write(9,*) "Cant converge  moving to next point"
```

```
            ierror = 0.
          endif
          counter =counter +1
        end do
        write(9,*)
        Up=Ut
        Us=(Up*v0) / (v0-cu)
        P = Up*Us/v0
        E = Up**2/2
        write(11,103) Cu,P, Up, Us, temp(2)
        write(*,103)  Cu,P, Up, Us, temp(2)
      enddo
      write(11,*) ''
      write(11,*) ''
      close(11)
      close(9)
      return
  100 format(10x,A)
  101 format(18x,A)
  102 format(16x,A,7x,A,3x,A,3x,A)
  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
  104 format(4x,'rho0: ',e10.4,4x,'c0: ',e10.4,4x,'s1: ',
     &     e10.4,4x,'g0: ',e10.4)
  105 format(4x,'Wilkins Constants: ',e10.4,' : ',e10.4,
     &     ' : ', e10.4,' : ',e10.4)
  106 format(3x,e10.4,4x,e10.4,4x,e10.4)
  107 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
  108 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
      END SUBROUTINE Hugoniot

!-----------------------------------------------------------------------
! Subroutine: Reshock
!
! DESCRIPTION:
!     Calulated reshock state given principle shock strength, and reshock
!     strength
!
!-----------------------------------------------------------------------
      SUBROUTINE reshock(V_1,step1,V_2,step2)
      use EOS
      IMPLICIT NONE
      DOUBLE PRECISION up_1, P_1, E_1, V_1
      DOUBLE PRECISION up_2, P_2, E_2, V_2
      DOUBLE PRECISION L, U, delta, Ut,Cu,E1,E2,Up2
      DOUBLE PRECISION f, f2,df, ierror, error, Peos, V2, dsmall
```

```
      DOUBLE PRECISION,DIMENSION(3) :: temp
      DOUBLE PRECISION,DIMENSION(10) :: junk
      INTEGER step1, step2,i, counter, maxcount

      counter = 0
      maxcount = 20
      v0 = 0.37

      open(11,file='reshock.data',access='append')
      dsmall = 0.99999
!      WRITE(*,101) 'Enter Hugonoit shock specific volume and increments'
!      READ(*,*) V_1, step1
!      WRITE(*,101) 'Enter reshock specific volume and increments'
!      READ(*,*) V_2, step2
      write(*,*) " V1, step1, v2, step 2"
      write(*,*) V_1, step1, V_2, step2
      write(11,102) 'v[cm3/g]','P[Mbar]','Up[cm/micro-s',
     &              'Us[cm/micro-s]'
      WRITE(*,*) "Calculating principle hugoniot"
      Up = 0.1
      temp(1:3) = 0.0
      L = 0.37
      U = V_1
      delta = (U-L)/(step1*1.0)
!      junk(2) = 0.325
      do i=1, step1
         Cu = L+delta*i
!         dump(1:10) = 0.0
         error = 0.01
         ierror = 1.0
         counter = 0
         do while (ierror.ge.error)
            Peos = 0.0
!            temp(1) = junk(2)
            junk(1:10) = 0.0
            E1 = (Up**2./2.)
            call busheos(Peos,E1,cu,junk(2),junk(3),temp)
            f=Peos-(Up**2)/(v0-cu)
            Up2=dsmall*Up
            E2=(Up2**2./2.)
!            temp(1) = junk(2)
            junk(1:10)=0.0
            call busheos(junk(1),E2,cu,junk(2),junk(3),temp)
            f2 = junk(1)-(Up2**2)/(v0-cu)
            df = (f2-f)/(Up2-Up)
```

```
            Ut=Up-f/df
            ierror = abs((Ut-Up)/Up)
!             ierror = abs(f/((Up**2)/(v0-cu)))
            Up=abs(Ut)
            if (counter.ge.maxcount) then
               write(9,*) "Cant converge  moving to next point at"
               write(9,*) "V =", Cu
               ierror = 0.
            endif
            counter =counter +1
         end do
         if (counter.lt.maxcount) then
            Up=Ut
            Us=(Up*v0) / (v0-cu)
            P = Up*Us/v0
            E = Up**2/2
            write(11,103) Cu,P, Up, Us,temp(2)
            write(*,103)  Cu,P, Up, Us,temp(2)
         else
            write(*,*) "Solution not found at V =",Cu
         endif

      enddo
      write(*,*) "Principle Hugoniot calculation complete"
      write(*,103)  Cu,P, Up, Us
      write(*,*) "Reshock calculation"
      up_1 = Up
      P_1 = P
      E_1 = E
      write(*,*) "Up_0    ,    P_0     ,   E_0"
      write(*,*) up_1, P_1, E_1

      temp(1:3) = 0.0
      L = Cu
      U = V_2
      counter = 0
      delta = (U-L)/(step2*1.0)
      Up_2 = 0.01
      do i=1, step2
         Cu = L+delta*i
!          dump(1:10) = 0.0
         error = 0.01
         ierror = 1.0
         counter = 0
         do while (ierror.ge.error)
```

```
            Peos = 0.0
            P_2 = P_1+((Up_1-Up_2)**2.)/(V_1-cu)
!            E1 = (Up**2./2.)
            E1 = E_1 + 0.5*(P_2-P_1)*(V_1-cu)
            temp(1) = junk(2)
            junk(1:10)=0.0
            call busheos(Peos,E1,cu,junk(2),junk(3),temp)
            f=Peos-P_2
!            write(9,*) cu, Peos,P_2, E1
            Up2=dsmall*Up_2
!            E2=(Up2**2./2.)
            P_2 = P_1+((Up_1-Up2)**2.)/(V_1-cu)
            E2 = E_1 + 0.5*(P_2-P_1)*(V_1-cu)
            temp(1) = junk(2)
            junk(1:10)=0.0
!            write(9,*) P_2, E1
            call busheos(junk(1),E2,cu,junk(2),junk(3),temp)
            f2 = junk(1)-P_2
            df = (f2-f)/(Up2-Up_2)
            Ut=Up_2-f/df
!            write(9,*) cu, Peos, E1, f2, df, Ut
!            ierror = abs((Ut-Up_2)/Up_2)
            ierror = abs((P_2-Peos)/P_2)
!            write(9,*) ierror, error
            Up_2=abs(Ut)
            if (counter.ge.maxcount) then
               write(9,*) "Can't converge  moving to next point at"
               write(9,*) "V =", Cu
               ierror = 0.
            endif
            counter =counter +1
         end do
         if (counter.lt.maxcount) then
            Up=Ut
            Us=(Up-Up_1)*v0 / (v0-cu)
            P = P_1+((Up_1-Up2)**2.)/(V_1-cu)
            E = E_1 + 0.5*(P-P_1)*(V_1-cu)
            write(11,103) Cu,P, Up, Us, temp(2)
            write(*,103)  Cu,P, Up, Us, temp(2)
         else
            write(*,*) "Solution not found at V =",Cu
         endif
      enddo

      return
```

```
  100 format(10x,A)
  101 format(18x,A)
  102 format(16x,A,7x,A,3x,A,3x,A)
!  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x)
  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)

      ENDSUBROUTINE Reshock

!-----------------------------------------------------------------------
!  Subroutine: ISENTROPE
!
!  DESCRIPTION:
!    Plots/exports ISENTROP
!-----------------------------------------------------------------------
      SUBROUTINE ISENTROPE(low,hig,incr,temp)
       use EOS
       IMPLICIT NONE
       DOUBLE PRECISION low, hig
       DOUBLE PRECISION L, U, Cu, delta
       DOUBLE PRECISION Pi, Po, dump
       DOUBLE PRECISION Ei, Eo, T0
       DOUBLE PRECISION Ur, dP1,dP2, E2,E1
       DOUBLE PRECISION error, ierror, f,df,Up2,Peos,v2,dsmall,Ut,f2
       DOUBLE PRECISION S0,Ss
       DOUBLE PRECISION,dimension(3)  :: temp
       DOUBLE PRECISION,dimension(10) :: junk

       INTEGER          step, i, sel, incr,counter

       open(11,file='isen.dat',access='append')
       open(66)
       L=low
       U=hig
       step=incr
!       dsmall=0.99999
       dsmall = 0.999
       counter = 0
!       dsmall=1.00001
!       L = v0 ! Lower bound
!       U = 0.203478 ! Upper Bound
!       step = 100
       WRITE(11,100) 'Isentrope Calculation'
        WRITE(11,*) 'Ploting from ',L,' to ',U
!       Up=0
!      Find initial Hugoniot Pressure given specific volume
```

```
      error=0.0001
      ierror=1.0
!       Up=0.1
      Up = temp(3)
      junk(2) = temp(1)
      write(*,*) "temp 1:", temp(1)
      write(*,*) "Up:",Up
      do while (ierror.ge.error)
         Peos=0.0
         temp(1) = junk(2)
         junk(1:10)=0.0
         E1=(Up**2.)/2.
         call  busheos(Peos,E1,L,junk(2),junk(3),temp)
         f=Peos-(Up**2)/(v0-L)
         Up2=dsmall*Up
         E2=(Up2**2.)/2.
         temp(1) = junk(2)
         junk(1:10)=0.0
         call busheos(junk(1),E2,L,junk(2),junk(3),temp)
         f2 = junk(1)-(Up2**2)/(v0-L)
         df = (f2-f)/(Up2-Up)
         Ut=Up-f/df
         ierror = abs((Ut-Up)/Up)
         Up=abs(Ut)
         write(*,*) 'Up:',Up
      end do
      Ur=Ut                    ! units cm/micro-s
      Ei= Ur**2./2.
      temp(1) = junk(2)
      junk(1:10) = 0.0
      call busheos(junk(1),Ei,L,junk(2),junk(3),temp)
      Pi = Peos
      write(*,*) L, Pi, low
      pause
      WRITE(11,102) 'v[cm3/g]','P[Mbar]','E[Mbar-cm3/g]',
     &              'Ur[cm/micro-s]','Region'
      write(11,103)  L,Pi,Ei,junk(2),junk(3),Ur, temp(2)
      write(*,*)  L,Pi,Ei,junk(2),Ur, temp(2)
      temp(1) = junk(2)
      write(*,*) "Tguess:",temp(1)
      junk(1:10) = 0.0
      V2 = L*dsmall
      E2 = Ei-(V2-L)*Pi
      call busheos(junk(1),E2,V2,junk(2),junk(3),temp)
      S0 = junk(3)
```

```
      write(*,*) Pi, Ei, E2,V2,v0,junk(1)
      dP1 = (sqrt(-(junk(1)-Pi)/(V2-L)))
      write(*,*) Pi, junk(1), V2,L, dP1, temp(2)
!      delta = ((v0*1.1)-l)/step*1.
      delta = 0.001
      i=0
      Po = 1
      Ss = S0
!      return
      do while(Po.gt.0.0001)
!      do while(counter.lt.100000)
!         Ss = S0
         i=i+1
         Cu=L+delta*i ! Current location (v)
!!         Calculate Eo using foward difference method
!!         Shouldn't Pr+gamma0/v0(E-Er) be replaced with P ??
         temp(1) = junk(2)
         junk(1:10)= 0.0
         call busheos(junk(1),junk(3), Cu, junk(2), Ss,temp)
         write(66,*) Cu, Eo, junk(1)
         if(junk(1).lt.0) goto 1000
         Po = junk(1)
         Eo = junk(3)
         T0 = junk(2)
         V2=Cu*dsmall
         temp(1) = junk(2)
         junk(1:10)=0.0
!              Find P2
         call busheos(junk(1), junk(3), V2, junk(2), Ss, temp)
         write(66,*) V2, E2, junk(1)
         dP2 =(sqrt(-(junk(1)-Po)/(V2-Cu)))
!         write(*,*) Po, junk(1), V2,Cu, dP2
!         pause
         Ur=Ur+0.5*(dP2+dP1)*(delta)
         write(11,103) Cu, Po, Eo,T0,Ss,Ur, temp(2)
         write(*,103) Cu, Po, Eo,T0, Ss,Ur,temp(2)
         write(66,*) dP1, dp2, delta, Ur
         write(66,*) ""
         Ei=Eo
         Pi=Po
         dP1=dP2
!         counter = counter + 1
      enddo
 1000 write(11,*) ''
      write(11,*) ''
```

```
      close(11)
      write(*,*) "End", junk(1)
      write(*,103) Cu, Po, Eo, U
      return
  100 format(10x,A)
  101 format(18x,A)
  102 format(16x,A,6x,A,4x,A,3x,A,3x,A)
!  103 format(15x,e10.4,4x,e10.4,4x,e10.4,6x,e10.4,7x,i1)
  103 format(15x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,
     &      4x,e10.4,4x,e10.4)
  104 format(4x,'rho0: ',e10.4,4x,'c0: ',e10.4,4x,'s1: ',
     &      e10.4,4x,'g0: ',e10.4)
  105 format(4x,'Wilkins Constants: ',e10.4,' : ',e10.4,
     &      ' : ', e10.4,' : ',e10.4)

      END SUBROUTINE ISENTROPE


!-----------------------------------------------------------------------
!  Subroutine: HRISENTROPE
!
!  DESCRIPTION:
!    Plots/exports The release isentrope from a given pressure
!-----------------------------------------------------------------------
      SUBROUTINE HRISENTROPE()
      use EOS
      IMPLICIT NONE
      DOUBLE PRECISION L, U, Cu, delta
      DOUBLE PRECISION Pg
      DOUBLE PRECISION v, P1, P2, E1, E2
      DOUBLE PRECISION df, Pi, Po, Ur,dP1,dP2, dump
      DOUBLE PRECISION Ei, Eo, ierror, error , P0
      DOUBLE PRECISION,DIMENSION(10) :: junk
      DOUBLE PRECISION,DIMENSION(3)  :: temp
      INTEGER          step , i, sel

 1000 write(*,101) 'Pressure which released from'
      read(*,*) Pg
      call findV(Pg,L,temp)
      write(*,*) "after findV"
      write(*,*) 'P:',Pg,'v:',L,'T:',temp(1)
      step = 10000
      U=L*10
      CALL ISENTROPE(L,U,step,temp)
      goto 5000
```

```
 5000 WRITE(*,*) 'Plot again yes (1)/No (2)'
      read(*,*)  sel
      if(sel.eq.1) then
         goto 1000
      endif

      RETURN
  100 format(10x,A)
  101 format(18x,A)

      END SUBROUTINE HRISENTROPE



       SUBROUTINE domeCheck(Pg,T1,vl,vu,step,flag, guess)
!
!     This subroutine scans along the given isotherm checking the
!     pressure state is below the given pressure (Pg)
!     indicating the estimated location of specific volumn with
!     satisfies the P and T
!
!     vl and vu are bounds of the search
!     step is a legacy and can be removed
!     flag is an indicator of how many solutions where found
!     guess are the specific density which bound the given pressure
!
       IMPLICIT NONE
       DOUBLE PRECISION :: P2, P1,Pg,v2,v1, vl, vu, T1
       DOUBLE PRECISION :: check, delta,delta2, vt
       DOUBLE PRECISION :: fl1,fl2,fs1,fs2,Pl,Ps
       DOUBLE PRECISION :: Gl, Gs
       DOUBLE PRECISION,DIMENSION(6) :: guess
       DOUBLE PRECISION,DIMENSION(5) :: dump
       DOUBLE PRECISION,DIMENSION(3) :: temp

       INTEGER :: flag
       INTEGER :: i
       INTEGER :: step
       INTEGER :: crap
       INTEGER :: counter


       crap = 0
       open(14)
       flag = 1
```

```
      guess(1:6) = 0.0
!      delta = 0.05
      delta2 = 1.001
!      check = 1.0
!      v1 = vl
!      v1 = 0.1
      if (vu.le.0.0) then
         vu = 10000000.
      endif
      if (vl.le.0.0) then
         vl = 0.1
      endif
!      v1 = 10000000.
!      vl = 0.1
      v1 = vu
      counter = 20


      if (Pg.le.0.0) then
! In the event that the given pressure is negative,
!   then only two "real" solutions are possible
         flag = -1
         guess(1) = -9.99e9
         guess(5) = -9.99e9
         return
      endif

      delta =  v1/counter
!      check = vl/10

!      P1 = 0.0
!      dump(1:5) = 0.
! LEGACY          call bush(P1,dump(1),v1,T1,dump(2),temp)
      CALL getF(v1,T1,fl1,fs1)
      vt = v1*delta2
      CALL getF(vt,T1,fl2,fs2)
      Pl = -(fl2-fl1)/(vt-v1)
      Ps = -(fs2-fs1)/(vt-v1)
      Gl = Fl1+Pl*v1
      Gs = Fs1+Ps*v1
      if (v1.gt.0.5) then
         P1=Pl
      else if (Gl > Gs) then
         P1=Ps
      else
```

```
            P1=Pl
          endif

          write(14,103) P1,v1

          if (P1.gt.Pg) then
! If P at the maximum specific volume returns a pressure
!   greater the the given pressure it will never pick the
!   correct vgass since it is greater the the max
            flag = -2
            guess(1) = -9.99e9
            guess(5) = -9.99e9
            return
          endif

!         P1= -1000

!         do while (v1.lt.vu)
          do while (v1.gt.vl)
            do i=1,(counter-2)
!             write(*,*) "i=",i,delta
              P2 = P1
              v2 = v1
              P1 = 0.0
              dump(1:5) = 0.
              v1 = v1-delta
! LEGACY            call bush(P1,dump(1),v1,T1,dump(2),temp)
              CALL getF(v1,T1,fl1,fs1)
              vt = v1*delta2
              CALL getF(vt,T1,fl2,fs2)
              Pl = -(fl2-fl1)/(vt-v1)
              Ps = -(fs2-fs1)/(vt-v1)
              Gl = Fl1+Pl*v1
              Gs = Fs1+Ps*v1
              if (v1.gt.0.5) then
                P1=Pl
              else if (Gl > Gs) then
                P1=Ps
              else
                P1=Pl
              endif
              write(14,103) P1,v1,P2,v2
              if(((P1.gt.Pg).and.(P2.lt.Pg)).or.
     &                    ((P1.lt.Pg).and.(P2.gt.Pg))) then
!               pause
```

```
                write(14,*) '********'
                guess(flag) = v2
                guess(flag+1) = v1
                flag=flag+2
              endif
              if(flag.eq.7) then
                v1=0.1*vl
                exit
              endif
            enddo
            delta = delta/10
          enddo
          flag = (flag-1)/2
!         write(*,*) 'Number of possible solutions : ',flag
          write(14,*) 'Number of possible solutions : ',flag
          close(14)
          return
  103   format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
        END SUBROUTINE

        SUBROUTINE endPoints(P1,T1,guess,vl,vg)
!
!     This subroutine finds the actual endpoints of the saturation dome
!     P1 is
!
        IMPLICIT NONE
        DOUBLE PRECISION :: vl,vg,cu
        DOUBLE PRECISION :: T1, P1,P2,Ptemp,f1,f2,df
        DOUBLE PRECISION :: v1, v2,dv,delta,delta2
        DOUBLE PRECISION :: error, ierror
        DOUBLE PRECISION :: vt,fl1,fl2,fs1,fs2,Gl,Gs,Pl,Ps
        DOUBLE PRECISION,DIMENSION(5)    :: dump
        DOUBLE PRECISION,DIMENSION(3)    :: temp
        DOUBLE PRECISION,DIMENSION(6)    :: guess
!  guess is an initial guess for lower and upper
        INTEGER              :: counter, maxcount
!
!         write(*,*) 'P1 :', P1, ' T1: ',T1
        open(17)
        write(17,*) "EndPoints: ",P1, T1

        delta2 = 1.001
        delta = 1.001
        error = 0.01
        ierror= 1.0
```

```
      dump(1:5) = 0.
      temp(1:3) = 0.
      Ptemp=0.
      vl = .3   ! liquid state
      vg = 100  ! gas state
      counter = 0
      maxcount = 100 ! limits number of itermations
!
!       cu =guess(5)
!       cu = 0.5*(guess(5)+guess(6))
      cu = guess(5)
      do while (ierror >= error)
!!         cu = vl
         dump(1:5) = 0.
         Ptemp = 0.
!          call bush(Ptemp,dump(1),cu,T1,dump(2),temp)
         CALL getF(cu,T1,fl1,fs1)
         vt = cu*delta2
         CALL getF(vt,T1,fl2,fs2)
         Pl = -(fl2-fl1)/(vt-cu)
         Ps = -(fs2-fs1)/(vt-cu)
         Gl = Fl1+Pl*cu
         Gs = Fs1+Ps*cu
         if (cu.gt.0.5) then
            Ptemp = Pl
         else if (Gl > Gs) then
            Ptemp=Ps
         else
            Ptemp=Pl
         endif
         f1 = P1-Ptemp
         write(17,101) cu, T1, Ptemp, f1, -9.9e9, -9.9e9
         Ptemp = 0
         dump(1:5) = 0.
!          call bush(Ptemp,dump(1),cu*delta,T1,dump(2),temp)
         CALL getF(cu*delta,T1,fl1,fs1)
         vt = cu*delta*delta2
         CALL getF(vt,T1,fl2,fs2)
         Pl = -(fl2-fl1)/(vt-cu*delta)
         Ps = -(fs2-fs1)/(vt-cu*delta)
         Gl = Fl1+Pl*cu*delta
         Gs = Fs1+Ps*cu*delta
         if ((cu*delta).gt.0.5) then
            Ptemp = Pl
         else if (Gl > Gs) then
```

```
            Ptemp=Ps
         else
            Ptemp=Pl
         endif
         df=(P1-Ptemp)-f1
         dv = delta*cu-cu
         df = df/dv
         write(17,101) delta*cu, T1, Ptemp, (P1-Ptemp), df, dv
         cu = cu - f1/df

         if ((cu.lt.guess(6)).or.(cu.gt.guess(5))) then
            if(cu.gt.guess(5))then
               write(17,*) cu,">",guess(5)
               cu = cu+f1/df
               cu = 0.5*(cu+guess(5))
               write(17,*) "new cu =",cu
            else
               write(17,*) cu,">",guess(6)
               cu = cu+f1/df
               cu = 0.5*(cu+guess(6))
               write(17,*) "new cu =",cu
            endif
         endif
         Ptemp = 0
         dump(1:5) = 0.
!          call bush(Ptemp,dump(1),cu,T1,dump(2),temp)
         CALL getF(cu,T1,fl1,fs1)
         vt = cu*delta2
         CALL getF(vt,T1,fl2,fs2)
         Pl = -(fl2-fl1)/(vt-cu)
         Ps = -(fs2-fs1)/(vt-cu)
         Gl = Fl1+Pl*cu
         Gs = Fs1+Ps*cu
         if (cu.gt.0.5) then
            Ptemp = Pl
         else if (Gl > Gs) then
            Ptemp=Ps
         else
            Ptemp=Pl
         endif
         ierror = ABS((P1-Ptemp)/P1)
!          ierror = ABS(P1-Ptemp)
         write(17,101) cu, T1, Ptemp,ierror, -9.9, -9.9
         counter= counter+1
         if(counter.ge.maxcount) then
```

```
                write(17,*) "Maximum iteration reached taking last value"
                ierror = 0.0
!               cu = vl + 1.0
!               counter = 0
            endif
        enddo
        vl = cu
        write(17,*) "End of vl loop"
        write(17,*) vl

!
        ierror= 1.0

        counter = 0
!        cu = guess(1)
        cu = 0.5*(guess(1)+guess(2))
        do while (ierror >= error)
!            cu = vg
            dump(1:5) = 0.
            Ptemp = 0.
!            call bush(Ptemp,dump(1),cu,T1,dump(2),temp)
            CALL getF(cu,T1,fl1,fs1)
            vt = cu*delta2
            CALL getF(vt,T1,fl2,fs2)
            Pl = -(fl2-fl1)/(vt-cu)
            Gl = Fl1+Pl*cu
!            Gs = Fs1+Ps*cu
            if (cu.gt.0.5) then
               Ptemp = Pl
!            else if (Gl > Gs) then
!               Ptemp=Ps
            else
               Ptemp=Pl
            endif
            f1 = P1-Ptemp
            Ptemp = 0
            dump(1:5) = 0.
!            call bush(Ptemp,dump(1),cu*delta,T1,dump(2),temp)
            CALL getF(cu*delta,T1,fl1,fs1)
            vt = cu*delta*delta2
            CALL getF(vt,T1,fl2,fs2)
            Pl = -(fl2-fl1)/(vt-cu*delta)
            Gl = Fl1+Pl*cu*delta
!            Gs = Fs1+Ps*cu*delta
            if ((cu*delta).gt.0.5) then
```

```
                Ptemp = Pl
!             else if (Gl > Gs) then
!                Ptemp=Ps
              else
                Ptemp=Pl
              endif
              df=(P1-Ptemp)-f1
              dv = delta*cu-cu
              df = df/dv
              cu = cu - f1/df
              Ptemp = 0
              dump(1:5) = 0.
!             call bush(Ptemp,dump(1),cu,T1,dump(2),temp)
              write(17,101) cu,T1,Ptemp, f1,df,dv
              CALL getF(cu,T1,fl1,fs1)
              vt = cu*delta2
              CALL getF(vt,T1,fl2,fs2)
              Pl = -(fl2-fl1)/(vt-cu)
              Gl = Fl1+Pl*cu
!             Gs = Fs1+Ps*cu
              if (cu.gt.0.5) then
                Ptemp = Pl
!             elseif (Gl > Gs) then
!                Ptemp=Ps
              else
                Ptemp=Pl
              endif
              ierror = ABS((P1-Ptemp)/P1)
              counter= counter+1
              if(counter.ge.maxcount) then
                 write(17,*) "Maximum iteration reached taking last value"
                 ierror = 0.0
!                cu = vg - 1.0
!                vg = cu
!                counter = 0
!                write(*,*) "cu reset to : ", cu
!                pause
              endif
            enddo
            vg = cu
            write(17,*) vg

            close(17)
!           write(*,*) 'Vl : ',vl, ' Vg : ', vf
```

```
      return
  101 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
      END SUBROUTINE

      SUBROUTINE getBounds(T1,Plow,Pup)
!       This subroutine returns the lower and upper pressure in the cubic
!       region of the isotherm
      IMPLICIT NONE
      DOUBLE PRECISION :: T1, Plow, Pup
      DOUBLE PRECISION :: P2, P1, V1,V2,vt
      DOUBLE PRECISION :: Fl1, Fs1,Fl2,Fs2,Pl,Ps,Gl,Gs
      DOUBLE PRECISION :: delta,delta2
      DOUBLE PRECISION :: ierror
      DOUBLE PRECISION :: error
      DOUBLE PRECISION :: Vtop

      INTEGER :: check, counter, i, inflec, state, crap

      error = 1.0
      ierror = 0.001
      V1 = 0.1
      Vtop = 100000.
      check = 0
      delta2 = 1.001

      inflec = 1
      state = 0

      open(77)

!     Sweep through the isotherm starting at v=0.1 to 1e5 on a log scale
!     check fro inflection points between possitive and negative slopes
        counter = 10
        delta = 1./counter
        crap = 0

        call getF(V1,T1,Fl1,Fs1)
        vt=V1*delta2
        call getF(vt,T1,Fl2,Fs2)
        Pl = -(Fl2-Fl1)/(vt-V1)
        Ps = -(Fs2-Fs1)/(vt-V1)
        Gl = Fl1+Pl*V1
        Gs = Fs1+Ps*V1
        if (V1.gt.0.5) then
           P1 = Pl
```

```
            state = 2
         else if (Gl>Gs) then
            P1 = Ps
            state = 1
         else
            P1 = Pl
            state = 2
         endif
         if (P1.lt.0.0) then
            crap = 1
         endif

         write(77,103) T1, V1, P1, Gl,Gs,state, inflec

         do while (check == 0)
           do i=1,counter-1
              V2=V1+delta
              call getF(V2,T1,Fl1,Fs1)
              vt=V2*delta2
              call getF(vt,T1,Fl2,Fs2)
              Pl = -(Fl2-Fl1)/(vt-V2)
              Ps = -(Fs2-Fs1)/(vt-V2)
              Gl = Fl1+Pl*V2
              Gs = Fs1+Ps*V2
              if (V2.gt.0.5) then
                 P2 = Pl
                 state = 2
              else if (Gl.gt.Gs) then
                 P2 = Ps
                 state = 1
              else
                 P2 = Pl
                 state = 2
              endif
              write(77,103) T1, V2, P2,Gl,Gs,state, inflec
              if (crap.eq.1) then
                 if(P2.lt.0.0) then
                    crap = 1
                 else
                    crap = 0
                 endif
              else if ((P2.gt.P1).and.(inflec.eq.1)) then
!     Found first inflection point
                 Plow = P1
                 inflec = 2
```

```
            elseif ((P2.lt.P1).and.(inflec.eq.2)) then
!     Found second inflection point
               Pup = P2
               inflec = 3
               check =1
            else if (V2.gt. Vtop) then
!               write(*,*) "V2 greater the Vtop"
               check = 1
               exit
            endif
            V1 = V2
            P1 = P2
!
         enddo
         delta = delta*10
       enddo

       close(77)
       return
  103 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,i1,4x,i1)
       END SUBROUTINE

       SUBROUTINE satPoints(T1,Ps,vl,vg)
!      This subroutine returns the saturation endpoints of the isotherm T
!      using only the getF routine
!      Outputs incluse the saturation pressure and the specific volume,
!      energy, and entropy at the liquid and gas endpoints
       IMPLICIT NONE
       DOUBLE PRECISION T1, Ps, vl, vg, el,eg,sl,sg
       DOUBLE PRECISION P1,P2, dg1, dg2, dp, dg
       DOUBLE PRECISION f1, df,dv, delta, ierror
       DOUBLE PRECISION Pl, Pu, Pg
       DOUBLE PRECISION vliq, vgas, Fl, Fs
       DOUBLE PRECISION vbottom, vtop
       DOUBLE PRECISION Gl, Gs,Glower,Ggas
       DOUBLE PRECISION error

       DOUBLE PRECISION,DIMENSION(3)     :: temp
       DOUBLE PRECISION,DIMENSION(5)     :: dump
       DOUBLE PRECISION,DIMENSION(6)     :: guess

       INTEGER :: flag, step, i, check, counter, pcount, maxcount,count2
       INTEGER :: check2

       delta = 1.001
```

```
      ierror = 1
      error = 0.001
      step = 1000 ! This may not be needed
      vbottom = 0.1
      vtop = 10000
      maxcount = 100
      counter = 0
      pcount = 1
      count2 = 0
      check2 =1



      open(15)
      write(15,*) "Temperature=",T1
      !
      CALL getBounds(T1,Pl,Pu)
      write(15,*) "getBound returns"
      write(15,101) T1, Pl, Pu
!     Initial pressure guess is the average of the Pl and Pu
!     representing the max and min pressure seen in the cubic section
!      P1 = 0.5*(Pl+Pu)
      P1 = 0.5*(Pu+Pl)
      if (P1.lt.0.0) then
         P1=0.5*Pu
      endif
!      write(*,*) P1

!      P1=0.5*Pu
!     domeCheck find rough location of specific volum
!     endPoints routine finds the specific volume which corespond to the
!     given Temperautre and guessed pressure
      guess(1:6) = 0.0
      call domeCheck(P1,T1,vbottom,vtop,step,flag,guess)
      write(15,*) "domeCheck returns"
      write(15,102) T1, P1, guess(1), guess(5)
      if (flag.lt.3) then
         write(15,*) "Flag =",flag
         if(flag.eq.-2) then
            count2 = 1
            do while (check2==1)
               write(15,*) "Increasing vtop x10"
               vtop = vtop*10
               write(15,*) "vtop=",vtop
               call  domeCheck(P1,T1,vbottom,vtop,step,flag,guess)
```

```
                if (flag.eq.3) then
                   write(15,*) "Success"
                   check2 = 0
                endif
                if (count2.eq.6) then
                   write(15,*) "Failure"
                   write(15,*) "Gas volume greater than max volume"
                   vl = -9.99e-9
                   vg = -9.99e-9
                   Ps = -9.99e-9
                return
                endif
                count2=count2+1
             enddo
          else
             write(15,*) "Outside saturation region"
             vl = -9.99e-9
             vg = -9.99e-9
             Ps = -9.99e-9
             return
          endif
       endif
       check2 = 1

!       if (flag.lt.3) then
!          write(*,*) "less than three solutions found new guess"
!          check = 0
!          do while (check ==0)
!             counter = counter + 1
!             P1 = 0.5*(Pu+P1)
!             guess(1:6) = 0.0
!             flag = 0
!             CALL domeCheck(P1,T,vbottom,vtop,step,flag,guess)
!             if (flag.eq.3) then
!                check=1
!             endif
!             if (counter==maxcount) then
!                return
!             endif
!          enddo
!       endif
       CALL endPoints(P1,T1,guess,vliq,vgas)

       write(15,*) "endPoints returns"
       write(15,102) T1, P1, vliq, vgas
```

```
CALL getF(vgas,T1,Fl,Fs)
Gl = Fl + P1*Vgas
Gs = Fs + P1*Vgas
Ggas = Gl

CALL getF(vliq,T1,Fl,Fs)
Gl = Fl + P1*vliq
Gs = Fs + P1*vliq
if(vliq.gt.0.5) then
   Glower = Gl
else if(Gl>Gs) then
   Glower = Gs
else
   Glower = Gl
endif

dg1  = Glower- Ggas
write(15,*) "First gibbs"
write(15,103) T1, P1, vliq,vgas,dg1


P2=P1*delta
CALL domeCheck(P2,T1,vbottom,vtop,step,flag,guess)
write(15,*) "domeCheck returns"
write(15,102) T1, P2, guess(1), guess(5)
CALL endPoints(P2,T1,guess,vliq,vgas)
write(15,*) "endPoints returns"
write(15,102) T1, P2, vliq, vgas

CALL getF(vgas,T1,Fl,Fs)
Gl = Fl + P2*vgas
Gs = Fs + P2*vgas
Ggas = Gl

CALL getF(vliq,T1,Fl,Fs)
Gl = Fl + P2*vliq
Gs = Fs + P2*vliq

if(vliq.gt.0.5) then
   Glower = Gl
else if(Gl>Gs) then
   Glower = Gs
else
   Glower = Gl
```

```
        endif
        dg2  = Glower - Ggas

        write(15,*) "second gibbs"
        write(15,103) T1, P2, vliq,vgas,dg2

        ierror = abs(dg2/Glower)

!        write(15,*) Glower
!
!      P2 must never be negative because as v--> inf P--> 0
!         P will never be less than zero, hence the endpoints
!         rountine finds the wrong endpoints
!
        dP = P2-P1
        dg = dg2-dg1
        dg = dg/dp
        P1 = P2
        dg1 = dg2
        P2 = P2-dg2/dg
        if (P2.lt.0.0) then
           P2=P1*0.5
        endif
!
        do while (ierror >= error)
           pcount = pcount + 1
           CALL domeCheck(P2,T1,vbottom,vtop,step,flag,guess)
           write(15,*) "domeCheck returns"
           write(15,102) T1, P2, guess(1), guess(5)
!           write(15,*) "Flag=", flag
!           if (flag.eq.9) then
!              write(15,*) "Gas volume greater than max volume"
!              vl = -9.99e-9
!              vg = -9.99e-9
!              Ps = -9.99e-9
!              return
!           endif

           if (flag.lt.3) then
              write(15,*) "Flag=", flag
              if (flag.eq.-2) then
                 count2 = 1
                 do while (check2==1)
                    write(15,*) "Increasing vtop x10"
                    vtop = vtop*10
```

```
            write(15,*) "vtop=",vtop
            call domeCheck(P1,T1,vbottom,vtop,step,flag,guess)
            if (flag.eq.3) then
               write(15,*) "Success"
               check2 = 0
            endif
            if (count2.eq.6) then
               write(15,*) "Failure"
               write(15,*) "Gas volume greater than max volume"
               vl = -9.99e-9
               vg = -9.99e-9
               Ps = -9.99e-9
               check = 0
               return
            endif
            count2=count2+1
         enddo
      else
         write(15,*) "Outside saturation region"
         return
         vl = -9.99e-9
         vg = -9.99e-9
         Ps = -9.99e-9
      endif
   endif
   check2=1


   CALL endPoints(P2,T1,guess,vliq,vgas)
   write(15,*) "endPoints returns"
   write(15,102) T1, P2, vliq, vgas

   CALL getF(vgas,T1,Fl,Fs)
   Gl = Fl + P2*vgas
   Gs = Fs + P2*vgas
   Ggas = Gl

   CALL getF(vliq,T1,Fl,Fs)
   Gl = Fl + P2*vliq
   Gs = Fs + P2*vliq

   if (vliq.gt.0.5) then
      Glower = Gl
   else if(Gl>Gs) then
      Glower = Gs
```

```
            else
               Glower = Gl
            endif
            dg2  = Glower-Ggas

            write(15,*) "loop gibbs"
            write(15,103) T1, P2, vliq,vgas,dg2

            ierror = abs(dg2/Glower)

            dP = P2-P1
            dg = dg2-dg1
            dg = dg/dp
            P1 = P2
            dg1 = dg2
            P2 = P2-dg2/dg
            if (P2.lt.0.0) then
               P2 = 0.5*P1
            endif
            if (P2>Pu) then
               P2 = Pu
            else if (P2<Pl) then
               P2 = Pl
            endif
            if (pcount.eq.maxcount) then
               write(15,*) "less than three solutions found"
               vl = -9.99e-9
               vg = -9.99e-9
               Ps = -9.99e-9
               return
            endif
         enddo
         Ps=P1
         vl = vliq
         vg = vgas
         write(15,*) "vl=",vl,"vg=",vg
         close(15)
!         vg,el,eg,sl,sg
         return !
   101 format(3x,e10.4,4x,e10.4,4x,e10.4)
   102 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
   103 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
   104 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
       END SUBROUTINE
```

```
      SUBROUTINE bush(P1,E1,V1,T1,Ss,temp)
      IMPLICIT NONE
      DOUBLE PRECISION P1, E1, V1, T1, Ss,F1
      DOUBLE PRECISION P2, E2, V2, T2, F2
      DOUBLE PRECISION Fl1,Fl2, Fs1,Fs2
      DOUBLE PRECISION Pl1,Ps1, Sl1, Ss1
      DOUBLE PRECISION Pl2, Ps2, Sl2,Ss2
      DOUBLE PRECISION El2, Es2
      DOUBLE PRECISION El1, Es1, Gl,Gs
      DOUBLE PRECISION n1, n2,dn
      DOUBLE PRECISION deltaV, deltaT
      DOUBLE PRECISION cu
      DOUBLE PRECISION error, ierror
      DOUBLE PRECISION v0, ti
      DOUBLE PRECISION Ps, vl, vg,el,eg,sl,sg
      DOUBLE PRECISION Fs
      DOUBLE PRECISION x
      DOUBLE PRECISION,DIMENSION(3)    :: temp

      INTEGER,DIMENSION(5)   :: num
      v0 = 0.3690
      ti = .293
      deltaV = 1.0001
      deltaT = 1.001
!
      open(18)
!
!     Checking for non zero variables
!
      num(1:5) = 0
      if(P1.ne.0) then
         num(1)=1
      endif
      if(E1.ne.0) then
         num(2)=1
      endif
      if(V1.ne.0) then
         num(3)=1
      endif
      if(T1.ne.0) then
         num(4)=1
      endif
      if(Ss.ne.0) then
         num(5)=1
      endif
```

```
!
!      Finding the two give thermodynamic variables
!
       if(num(3).eq.1) then
          if(num(4).eq.1) then
!         Volume and Temperature
             goto 1007
          endif
       endif
!      Should this call be it's own subroutine
!
 1007 CALL getF(V1,T1,Fl1,Fs1)
       V2 = deltaV*V1
       CALL getF(V2,T1,Fl2,Fs2)
       Pl1 = -(Fl2-Fl1)/(V2-V1)
       Ps1 = -(Fs2-Fs1)/(V2-V1)
!      Caluclate S
       T2 = deltaT*T1
       CALL getF(V1,T2,Fl2,Fs2)
       Sl1 = -(Fl2-Fl1)/(T2-T1)
       Ss1 = -(Fs2-Fs1)/(T2-T1)
       El1 = Fl1 + T1*Sl1
       Es1 = Fs1 + T1*Ss1
!       Gl = Fl1 + Pl1*V1
       Gl = El1 + Pl1*V1 - T1*Sl1
!       Gs = Fs1 + Ps1*V1
       Gs = Es1 + Ps1*V1 - T1*Ss1
       write(18,*) T1,V1,Pl1,Ps1,El1,Es1,Gl,Gs
!
!      temp(2) = 1 Solid
!      temp(2) = 2 liquid
!      temp(2) = 3 liquid-vapor
!      temp(2) = 4 solid-vapor
!      temp(2) = 5 vapor
!
!       if (V1 > 0.5) then
! Assume Vapor
!          P1 = Pl1
!          E1 = El1
!          Ss = Sl1
!          temp(2) = 2
!       else if(Gl>Gs) then

       CALL satPoints(T1,Ps,vl,vg)
       write(18,103) T1, Ps, vl, vg, V1
```

```
!       write(*,*) "Finding Sat Points"
        if ((V1.gt.vg).or.(V1.lt.vl).or.(Ps < 0.0)) then
!          write(*,*) "Not in a mixed region"
           write(18,*) "Not in a mixed region"
           if( V1 > 0.5) then
              P1 = Pl1
              E1 = El1
              Ss = Sl1
              temp(2) = 2
           else if(T1.ge.10.) then
              write(18,*) "Liquid"
              P1 = Pl1
              E1 = El1
              Ss = Sl1
! temp(2) = 2 --- liquid is equilibrium state
              temp(2) = 2

           else if(Gl>Gs) then
              write(18,*) "Solid"
              P1 = Ps1
              E1 = Es1
              Ss = Ss1
! temp(2) = 1 --- Solid is equilibrium state
              temp(2) = 1
           else
              write(18,*) "Liquid"
              P1 = Pl1
              E1 = El1
              Ss = Sl1
! temp(2) = 2 --- liquid is equilibrium state
              temp(2) = 2
           endif
!       else if ((V1<vg).and.(V1>vl)) then
        else
           write(18,*) "In mix phase region"
!       V1 is between Vgas and Vliq is a mixed phase
!            temp(2) = 3
!       Find pressure, energy, entropy at gas endpoint
           CALL getF(vg,T1,Fl1,Fs)
           V2 = deltaV*vg
           CALL getF(V2,T1,Fl2,Fs2)
           Pl1 = -(Fl2-Fl1)/(V2-Vg)
           T2 = deltaT*T1
           CALL getF(Vg,T2,Fl2,Fs2)
           Sl1 = -(Fl2-Fl1)/(T2-T1)
```

```
          El1 = Fl1+T1*Sl1
!       Find pressure, energy, entropy at liquid/solid endpoint
          CALL getF(Vl,T1,Fl1,Fs1)
          V2 = deltaV*vl
          CALL getF(V2,T1,Fl2,Fs2)
          Pl2 = -(Fl2-Fl1)/(V2-Vl)
          Ps2 = -(Fs2-Fs1)/(V2-Vl)
          CALL getF(Vl,T2,Fl2,Fs2)
          Sl2 = -(Fl2-Fl1)/(T2-T1)
          Ss2 = -(Fs2-Fs1)/(T2-T1)
          El2 = Fl1+T1*Sl2
          Es2 = Fs1+T1*Ss2
          Gl = El2 + Pl2*Vl - T1*Sl2
          Gs = Es2 + Ps2*Vl - T1*Ss2
          x = (vl-V1)/(vl-vg)
          if(Gl>Gs) then
             write(18,*) "Solid -Vapor"
             temp(2) = 4 ! Solid - Vapor
             temp(3) = x
!             P1 = 0.5*(Ps1+Ps2)
             P1 = Ps
             E1 = El1-x*(El1-Es2)
             Ss = Sl1-x*(Sl1-Ss2)
          else
             write(18,*) "Liquid - Vapor"
             temp(2) = 3 ! Liquid - Vapor
             temp(3) = x
!             P1 = 0.5*(Pl1+Pl2)
             P1 = Ps
             E1 = El1-x*(El1-El2)
             Ss = Sl1-x*(Sl1-Sl2)
          endif
       endif
!
       return
  104 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
  103 format(3x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4,4x,e10.4)
      END SUBROUTINE

      SUBROUTINE getF(v,T,Fl,Fs)
!-----------------------------------------------------------
!
! The liquid part of the Bushman EOS
! returns the value of Free Energy
!
```

```
!-----------------------------------------------------------------
      IMPLICIT NONE
      DOUBLE PRECISION :: R
      DOUBLE PRECISION :: Z
      DOUBLE PRECISION :: V0c
      DOUBLE PRECISION :: V0
      DOUBLE PRECISION :: a1, a2, a3, a4, a5
      DOUBLE PRECISION :: Ac, Bc, Cc
      DOUBLE PRECISION :: m, n, l
      DOUBLE PRECISION :: Esub
      DOUBLE PRECISION :: Ta
      DOUBLE PRECISION :: sigmaa
      DOUBLE PRECISION :: Tca
      DOUBLE PRECISION :: Tsa
      DOUBLE PRECISION :: theta0l
      DOUBLE PRECISION :: theta0s
      DOUBLE PRECISION :: gamma0l
      DOUBLE PRECISION :: gamma0s
      DOUBLE PRECISION :: Bl, Dl
      DOUBLE PRECISION :: Bs, Ds
      DOUBLE PRECISION :: sigmam0
      DOUBLE PRECISION :: Tm0
      DOUBLE PRECISION :: Am, Bm, Cm
      DOUBLE PRECISION :: betai,betao,betam
      DOUBLE PRECISION :: Tb
      DOUBLE PRECISION :: sigmaz
      DOUBLE PRECISION :: Tz
      DOUBLE PRECISION :: sigmai
      DOUBLE PRECISION :: Ti
      DOUBLE PRECISION :: gammaei,gammaeo,gammaem
      DOUBLE PRECISION :: Tg
      DOUBLE PRECISION :: sigmae, sigmad
      DOUBLE PRECISION :: tth
      DOUBLE PRECISION :: v,T,P,s,F
      DOUBLE PRECISION :: sigmaC, sigma, sigmam, x
      DOUBLE PRECISION :: taui, Cei, Ce, gammaE, Be
      DOUBLE PRECISION :: Fcl, Fal, Fe, Fll, Fm, Ca, theta, thetacl
      DOUBLE PRECISION :: Fcs, Fas, Fl, Fs, thetacs

      R      =  0.31
      Z      =  13.
      V0c    =  0.361
      V0     =  0.370
      a1     =  326.35
      a2     = -1035.44
```

```
a3      =  858.51
a4      = -160.59
a5      =  11.17
Ac      = -12.91
Bc      =  40 .96
Cc      = -28.05
 m      =  8.0
 n      =  4.99
 l      =  0.7
Esub    =  12.1
Ta      =  30.0
sigmaa  =  0.14
Tca     =  25.0
Tsa     =   6.0
theta0s=    0.1
gamma0s=    2.19
Bs      =   0.6
Ds      =   0.36
theta0l=  157.0
gamma0l=  1.78
Bl      =  1.05
Dl      =  0.0
sigmam0=  0.923
Tm0     =  0.933
Am      =  2.24
Bm      = -5.64
Cm      =  0.21
betai   =  0.0242
betao   =  0.05
betam   =  0.0
Tb      =  8.
sigmaz  =  0.8
Tz      =  200.
sigmai  =  0.3
Ti      =  50.
gammaei=  0.4
gammaeo=  0.7
gammaem= -0.5
Tg      =  300.
sigmae  =  1.0
sigmad  =  9.99e9
tth     =  2./3.

sigmaC  = V0c/v
sigma   = V0/v
```

```
      sigmam = sigma/sigmam0
      x      = log(sigma)
      taui     = Ti*exp(-sigma/sigmai)
!       write(*,*) "Tau : ", taui

! Cold Contribution (Lomonosov)
! Solid
      Fcs = 3.*V0c*(a1/1*(sigmaC**(1./3.)-1)+a2/2*(sigmaC**(2./3.)-1)+
     &              a3/3*(sigmaC**(3./3.)-1)+a4/4*(sigmaC**(4./3.)-1)+
     &              a5/5*(sigmaC**(5./3.)-1))
! Liquid
      if (sigmaC.ge.1.) then
         Fcl = Fcs
      else
         Fcl =V0c*(Ac*(sigmaC**m)/m+Bc*(sigmaC**n)/n+Cc*(sigmaC**l)/l)
     &        +Esub
      endif
! Atomic Contribution (Bushman)
! Solid
      thetacs=theta0s*sigma**(tth)*
     &        exp((((gamma0s-(tth))*(Bs**2+Ds**2))/Bs)*
     &        atan((x*Bs)/(Bs**2+Ds*(x+Ds))))
      Fas=3*R*T*log(thetacs/T)

! Liquid
      Ca = 3.*R/2.*(1+((sigma*Ta)/((sigma+sigmaa)*(T+Ta))))
      thetacl = theta0l*exp(((gamma0l-tth)*(Bl**2.+Dl**2.))/Bl*
     &          atan((x*Bl)/(Bl**2.+Dl*(x+Dl))))
      theta = (sigma**tth)*Tsa*((thetacl+T)/(Tca+T))
      Fll = Ca*T*log(theta/T)
      Fm = 3.*R*(((2.*(sigmam**2.)*Tm0)/(1.+(sigmam**3.)))*
     &     (Cm+3*Am/5*((sigma**(5./3.))-1.))+(Bm-Cm)*T)
      Fal = Fll + Fm
! Electronic Contribution (Lomonosov)
!       Ce = 3.*R/2.*(Z+((sigmaz*sigma*(Tz**2.)*(1.-Z))/((sigma+sigmaz)*
!     &      (T**2.+Tz**2.)))*exp(-taui/T))
! Electronic Contribution (Bushman)
! Liquid and Solid
      Ce = 3.*R/2.*(Z+((sigmaz*(Tz**2.)*(1.-Z))/((sigma+sigmaz)*
     &     (T**2.+Tz**2.)))*exp(-taui/T))
      Cei = 3.*R*Z/2.
      gammaE = gammaei+(gammaeo-gammaei+gammaem*T/Tg)*exp(-T/Tg-
     &         ((sigma-sigmae)**2./(sigma*sigmad)))
      Be = (2./T**2) *(Tb*(betam*T-(betai-betao-2*betam)*Tb)*exp(-T/Tb)
     &     + betai*(T**2.)/2 - (betai-betao-betam)*Tb*T +
```

```
     &          (betai-betao-2*betam)*Tb**2)
      Fe = -Ce*T*log(1+((Be*T)/(2*Cei))*sigma**(-1.*gammaE))
!       write(*,*) 'Ce :',Ce," Cei :",Cei," gammaE:",gammaE," Be :",Be,""

!       write(*,*) 'Fc :',Fcs," Fa :",Fas," Fe:",Fe,""

      Fl = Fcl + Fal + Fe
      Fs = Fcs + Fas + Fe
!      F = Fl
!      F=Fs
      END SUBROUTINE

      SUBROUTINE busheos(P1,E1,V1,T1,Ss,temp)
      USE EOS
!-----------------------------------------------------------------------
!   SUBROUTINE: busheos
!
!   DESCRIPTION:
!         Caculaties the thermodynamic variables using the
!         Bushman EOS given two thermodynamic variables
!
!-----------------------------------------------------------------------
      IMPLICIT NONE
      DOUBLE PRECISION P1, E1, V1, T1, Ss
      DOUBLE PRECISION,DIMENSION(3)    :: temp

      DOUBLE PRECISION  f, df, cu, error, ierror
      DOUBLE PRECISION  P2, E2, V2, T2, S2,E0,P0,S0
      DOUBLE PRECISION,DIMENSION(5)    :: junk

      INTEGER, DIMENSION(5) :: num
      INTEGER  counter, maxcount, count2
      v0 = 0.37


      open(88)
      write(88,*) "Input parameters"
      write(88,101)V1 , T1 , P1 , E1 , Ss , temp(1)
!
!     Checking for non zero variables
!
      num(1:5) = 0
      if(P1.ne.0) then
         num(1)=1
      endif
```

```
      if(E1.ne.0) then
         num(2)=1
      endif
      if(V1.ne.0) then
         num(3)=1
      endif
      if(T1.ne.0) then
         num(4)=1
      endif
      if(Ss.ne.0) then
         num(5)=1
      endif
!
!     Finding the two give thermodynamic variables
!
      if(num(1).eq.1) then
         if(num(2).eq.1) then
!           Pressure and Energy
            goto 1000
         elseif(num(3).eq.1) then
!           Pressure and Volume
            goto 1001
         elseif(num(4).eq.1) then
!           Pressure and Temperature
            goto 1002
         elseif(num(5).eq.1) then
!           Pressure and Entropy
            goto 1003
         else
!             write(*,*) 'Two thermodynamic variabls must be specified'
            return
         endif
      elseif(num(2).eq.1) then
         if(num(3).eq.1) then
!           Energy and Volume
            goto 1004
         elseif(num(4).eq.1) then
!           Energy and Temperature
            goto 1005
         elseif(num(5).eq.1) then
!           Energy and Entropy
            goto 1006
         else
!             write(*,*) 'Two thermodynamic variabls must be specified'
            return
```

```
         endif
      elseif(num(3).eq.1) then
         if(num(4).eq.1) then
!        Volume and Temperature
            goto 1007
         elseif(num(5).eq.1) then
!        Volume and Entropy
            goto 1008
         else
!             write(*,*) 'Two thermodynamic variabls must be specified'
            return
         endif
      elseif(num(4).eq.1) then
         if(num(5).eq.1) then
!        Temperature and Entropy
            goto 1009
         else
!             write(*,*) 'Two thermodynamic variabls must be specified'
            return
         endif
      else
!         write(*,*) 'Two thermodynamic variabls must be specified'
         return
      endif
!
!     Solving the Bushman EOS using given Pressure and Energy
!     to find specific volume, temperature and entropy
 1000 if(temp(1).ne.0) then
         cu=temp(1)    ! Initial guess
      else
         cu=v0*0.25
      endif

      error = 0.001
      ierror = 1.0

      E1 = E1*100
      P1 = P1/100


      return
!
!     Solving the Tillotson EOS using given Pressure and Volume
!     to find entergy, temperature and entropy
 1001 if(temp(1).ne.0) then
```

```
         cu=temp(1)    ! Initial guess
      else
         cu=v0*0.25    ! Initial temperature guess
      endif


      return
!
!     Solving the Bushman EOS using given Pressure and Temperature
!     to find energy, specific volume, and entropy
!
 1002 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Bushman EOS using given Pressure and Entropy
!     to find energy, specific volume, and temperature
!
 1003  write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Bushman EOS using given Energy and Volume
!     to find pressure, temperature and entropy
!
 1004 error = 0.005
      ierror = 1.0
!      open(88)
      maxcount = 20
      counter = 0
      count2 = 0

      if (temp(1).ne.0) then
         cu=temp(1)/1000
      else
         cu=0.3   ! Initial guess 293 K
!         cu = 7
      endif
      E0 = 0.2748
      P0 = 1.839
!      S0 = 1.889
!      E0 = 0.1397e2
!      P0 = 0.3446e-7
      S0 = 0.5283e+2
!      S0 = 0.0
      E1 = E1*100+E0
      write(88,*) "E1 =",E1, "T1=",cu
```

```
      E2 = 0.0
      do while (ierror.ge.error)
         CALL bush(junk(2),E2,V1,cu,junk(3),temp)
         write(88,*) cu, V1, E2
         f=E1-E2
         T2 = cu*1.001
         junk(1:10) = 0.
         E2 = 0.0
         CALL bush(junk(2),E2,V1,T2,junk(3),temp)
         write(88,*) T2, V1, E2
         junk(1) = E1-E2
         df = (junk(1)-f)/(T2-cu)
         cu = cu - f/df
         if (cu.lt.0) then
            cu = cu+f/df + 1.
            write(88,*) "New T:",cu
         endif
         junk(1:10) = 0.
         E2 = 0.0
         write(88,*) cu, V1,E2
         CALL bush(junk(2),E2,V1,cu,junk(3),temp)
         ierror = abs(E1-E2)/E1
         write(88,*) cu, V1,E2,ierror
         if (counter.ge.maxcount) then
            write(88,*) "Maximum number of iteration reached"
            cu = 0.5*(cu+cu+f/df)
            write(88,*) "Reseting Cu to:" , cu
            count2 = count2+1
            counter = 0
         endif
         if (count2.eq.5) then
            write(88,*) "Could not converge on a answer"
            ierror =0.0
         endif
         counter = counter + 1
      enddo
      T1=cu*1000
      P1=(junk(2)-P0)/100
      E1= (E1-E0)/100
      Ss=junk(3)-S0
!       close(88)

      return
!
!     Solving the Bushman EOS using given Energy and Temperature
```

```
!     to find pressure, specific volume, and entropy
!
 1005 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Bushman EOS using given Energy and Entropy
!     to find pressure, specific volume, and temperature
!
 1006 write(*,*) 'This Feature needs to be added'
      return
!
!     Solving the Bushman EOS using given Volume and Temperature
!     to find pressure, energy, and entropy
!
 1007 CALL bush(P1,E1,V1,T1,Ss,temp)
      write(*,*) "Volume and Temperature"
      return

!
!     Solving the Bushman EOS using given Volume and Entropy
!     to find pressure, energy, and temperature
!
 1008 error = 0.005
      ierror = 1.0
!      open(88)
      maxcount = 40
      counter = 0
      count2 = 0
      E0 = 0.2748
      P0 = 1.839
!      S0 = 1.889
!      E0 = 0.1397e2
!      P0 = 0.3446e-7
      S0 = 0.5283e2
!      S0 = 0.0
      Ss = Ss - S0

      if (temp(1).ne.0) then
         cu=temp(1)/1000
      else
         cu=0.3   ! Initial guess 293 K
!         cu = 7
      endif
      S2 = 0.0
      do while (ierror.ge.error)
```

```
          CALL bush(junk(2),junk(3),V1,cu,S2,temp)
          write(88,*) cu, V1, S2
          f=Ss-S2
          T2 = cu*1.001
          junk(1:10) = 0.
          E2 = 0.0
          CALL bush(junk(2),junk(3),V1,T2,S2,temp)
          write(88,*) T2, V1, S2
          junk(1) = Ss-S2
          df = (junk(1)-f)/(T2-cu)
          cu = cu - f/df
          if (cu.lt.0) then
             cu = cu+f/df + 1.
             write(88,*) "New T:",cu
          endif
          junk(1:10) = 0.
          E2 = 0.0
          write(88,*) cu, V1,E2
          CALL bush(junk(2),junk(3),V1,cu,S2,temp)
          ierror = abs(Ss-S2)/Ss
          write(88,*) cu, V1,S2,ierror
          if (counter.ge.maxcount) then
             write(88,*) "Maximum number of iteration reached"
             cu = 0.5*(cu+cu+f/df)
             write(88,*) "Reseting Cu to:" , cu
             count2 = count2+1
             counter = 0
          endif
          if (count2.eq.5) then
             write(88,*) "Could not converge on a answer"
             ierror =0.0
          endif
          counter = counter + 1
       enddo
       T1=cu*1000
       P1=(junk(2)-P0)/100
       E1=(junk(3)-E0)/100
       Ss=S2+S0
!       close(88)
       return
!
!      Solving the Bushman EOS using given Temperature and Entropy
!      to find pressure, energy, and specific volume
!
 1009  write(*,*) 'This Feature needs to be added'
```

```
      return

  101 format(3x,e10.4,4x,e10.4,4x,e10.4,
     &          4x,e10.4,4x,e10.4,4x,e10.4)


      END SUBROUTINE

!-------------------------------------------------------
!
!  Find the specific volume at the given pressure
!    assume Pg was achieved via a shockwave
!
!-------------------------------------------------------
      SUBROUTINE FindV(Pg,vout,temp)
      USE EOS
      IMPLICIT NONE
      DOUBLE PRECISION Pg, vout
      DOUBLE PRECISION f, df, Peos, Peos2, f2
      DOUBLE PRECISION ierror, error, cu, E1,v2


      DOUBLE PRECISION,DIMENSION(3)  :: temp
      DOUBLE PRECISION,DIMENSION(10) :: junk

      v0 = 0.37

      error = 0.001
      ierror = 1.0
      cu = v0*0.7
!       cu = 0.276
      temp(1:3) = 0.
      junk(1:10) = 0.0
      open(99)
      write(*,*) "Start findV"

      do while (ierror.ge.error)
         Up=sqrt(Pg*(v0-cu))
         E1=(Up**2)/2.
!          E1 = 0.5*Pg*(v0-cu)
         temp(1) = junk(2)
         write(99,*) Up, E1, temp(1)
         Peos = 0.0
         junk(1:10) = 0.0
         call busheos(Peos,E1,cu,junk(2),junk(3),temp)
```

```
        f = Pg - Peos
        write(99,*) Peos, Pg, cu

        v2 = cu*1.01
        Up=sqrt(Pg*(v0-v2))
        E1=(Up**2)/2.
!         E1 = 0.5*Pg*(v0-v2)
        temp(1) = junk(2)
        write(99,*) Up, E1, temp(1)
        Peos = 0.0
        Peos2 = 0.0
        junk(1:10) = 0.0
        call busheos(Peos2,E1,v2,junk(2),junk(3),temp)
        f2 = Pg- Peos2
        write(99,*) Peos2, Pg, v2

        df = (f2-f)/(v2-cu)
        cu = cu - f/df
        write(99,*) f, f2, df, cu

        Up=sqrt(Pg*(v0-cu))
        E1=(Up**2)/2
        temp(1) = junk(2)
        Peos2 = 0.0
        junk(1:10) = 0.0

        call busheos(Peos,E1,cu,junk(2),junk(3),temp)
        ierror = abs((Peos-Pg)/Pg)
        write(99,*) Peos, v2, ierror

      enddo
      temp(3) = Up
      vout = cu
      close(99)
      write(*,*) "Found V"
      return

      END SUBROUTINE
```