Spring 4-8-2020

Qualitative Investigation of Gaseous Hydrodynamic Mixing Model Efficacy and Associated Sensitivity

Caleb White
University of New Mexico

Caleb White
Candidate

Mechanical Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Peter Vorobieff, Chair

Dr. Humberto Silva III, Co-Chair

Dr. Daniel Banuti, Member

Qualitative Investigation of Gaseous Hydrodynamic Mixing Model Efficacy and Associated Sensitivity

by

Caleb White

B.S., Mechanical Engineering, The University of New Mexico, 2019

THESIS

Submitted in the Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Mechanical Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2020

Dedication

To my Lord and Savior, Jesus Christ, for His eternal love and mercy.

"And whatsoever ye do in word or deed, do all in the name of the Lord Jesus, giving thanks to God and the Father by him." — Colossians 3:17

Acknowledgments

No one who achieves success does so without acknowledging the help of others. The wise and confident acknowledge this help with gratitude. — Alfred Whitehead

First and foremost, I would like to extend my gratitude to my co-advisor, Dr. Silva, for his patience and guidance throughout this research. His insight into numerical analyses has proven invaluable and the time he has invested in me is very much appreciated, thank you sir. I would like to thank Dr. Vorobieff for his oversight of this research, and Dr. Banuti for serving on my committee. I would like to thank Dr. Kyran "Kim" Mish, the CTH manager, for his support and funding of this research, without which this work would not exist. I am grateful for Dr. Mike Hobbs' tutelage on the use of TIGER and his input and advice on the thermodynamic equations of states. I would like to thank Leah Tuttle and David Peterson for their guidance on the intricacies of CTH and their willingness to thoroughly explain the underlying theory. I am grateful for Adam Stephens' assistance with DAKOTA. I would like to thank Dr. Wayne for his experimental support and Josiah Bigelow for providing his previous numerical work as a starting ground, both of whom answered countless questions during the early stages of my research. I would like to thank my manager and esteemed colleagues Adam Church, Shane Curtis, Josh Dye, Carla Montoya, Roberto Jimenez, Raymundo Gonzalez, and Daniel Gutierrez for their support, encouragement, and friendship throughout my internship at Sandia. Lastly, a special thanks to SIAM for granting me the use of their LaTeX template which I then modified to conform to UNM's Thesis Formatting Guidelines and personal preference.

Qualitative Investigation of Gaseous Hydrodynamic Mixing Model Efficacy and Associated Sensitivity

by

Caleb White

B.S., Mechanical Engineering, The University of New Mexico, 2019
M.S., Mechanical Engineering, The University of New Mexico, 2020

Abstract

A mixing model analyzes the mixing of helium (He) and sulfur hexafluoride (SF_6) according to two classical gaseous equations of state (EOS), namely, Amagat's Law and Dalton's Law, undergoing planar traveling shocks in three dimensions (3D). Numerical simulations utilize the Sandia National Laboratories (SNL) shock hydrodynamic code CTH and other codes including the SNL thermochemical equilibrium code TIGER and the uncertainty qualification (UQ) and sensitivity analysis code DAKOTA. Comparison with experimental results show that none of the equations of state are able to accurately predict the properties of the shocked mixture; similar discrepancies have been observed in previous works. A sensitivity study using incremental Latin Hy-

percube Sampling (iLHS) was performed upon the model and various metrics were used to establish convergence of model behaviour. Sensitivity results indicate that the mixing model is most sensitive to the initial temperature of the He/SF_6 mixture.

Contents

List of Figures

List of Tables

Nomenclature

Latin Letters

A	Coefficient matrix
A^+	Moore-Penrose pseudo-inverse of A
e	Specific energy [cal/g]
E_o	Volume potential function
f	Helmholtz free energy factor
G	Grüneisen function
k	BKW covolume factor
k_B	Boltzmann's constant [erg/K]
L_∞	Infinity Norm
M	Molecular mass [g/mol]
m	Mass [g]
n	Number of moles
P	Pressure [dyne/cm^2]
P_0	Internal pressure function

R	Universal gas constant [ergs/(mol·K)]
r	Radius
r^*	JCZ3 parameter
r_{xy}	Pearson correction coefficient
S	Entropy [erg/K]
s	Specific entropy [cal/(K·g)]
T	Absolute temperature [K]
V	Volume [cm^3]
v	Velocity [cm/s]
w	Mass fraction
x	Mole fraction
$\overline{x}$	Sample mean
z	Compressibility factor

Greek Letters

α	BKW parameter
β	BKW parameter
χ	Uncertainty variable
η	JCZ3 parameter
γ	Specific heat ratio
κ	BKW parameter

μ	Population mean
ν	Specific volume [cm^3/g]
Ω	Arbitrary variable
ω	Arbitrary specific variable
ρ	Density [g/cm^3]
σ	Population standard deviation
σ^2	Population variance
θ	BKW parameter
ε	JCZ3 parameter
φ	EXP6 potential function

Subscripts

i	ith component
m	Mixture
r	Ratio

Chapter 1

Introduction & Theory

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop." — Lewis Carroll, *Alice in Wonderland*

Shock tubes have a long history in the study of Equations of State (EOS). Shock tube equation of state studies have been conducted for many substances, including liquid nitrogen [1, 2], solid carbon dioxide [2], and gaseous argon [3, 4]. The necessity to account for real-gas properties in shocked gases has been noted at least as early as 1996 [4], based on discrepancies between predictions on ideal gas theory and experiments with pure argon. Until recently, however, there were few, if any, well-quantified shock tube experiments considering the equation of state for gas mixtures. A 2019 study [5] considered a binary mixture of two gases, helium and sulfur hexafluoride, both of which possess vastly different properties. The post-shock properties, such as pressure, temperature, and velocity measurements are taken from the experiments, and attempts are made to simulate the experimental conditions — using the Sandia National Laboratories hydrodynamic code CTH and the thermochemical equilibrium code TIGER — with a variety of equations of states in an attempt to determine if any of the EOS under consideration is more accurate at predicting the post-shock properties.

1.1 Equations of State

A gas is most often described by three physical properties, namely pressure (P), temperature (T), and specific volume (ν). An equation of state then defines the relationship between the properties, such that knowledge of any two properties fully defines the remaining state variables.

1.1.1 Ideal Gas

The ideal gas equation of state

$$P\nu = RT \tag{1.1}$$

is a simple and well known EOS which is a combination of Boyle's Law and Gay-Lussac's Law among others [6].

The ideal gas relationship is only applicable at low pressures and high temperatures (and thus a low density) [6]. A real gas departs from the ideal-gas behavior at higher pressures or when close to the saturation region or critical point [6]. To account for this deviation from the ideal-gas behavior, a correction parameter known as the compressibility factor (z) can be introduced to the ideal gas equation of state to result in the following [6]

$$P\nu = zRT \tag{1.2}$$

where z is unity for an ideal gas.

1.1.2 Amagat & Dalton

Equations of state for gas mixtures generally make assumptions about which of the properties can apply to the entire mixture (and thus are held constant) while the other properties are summed over the partial property components [6, 7]. Non-reacting gas mixtures behave as an ideal gas when under similar constraints as a pure gas. There are two classical laws used to predict the behavior of gas mixtures: Dalton's law of

additive pressures and Amagat's law of additive volumes [6, 7]. Thus, for Dalton

$$P_m = \sum_i^k P_i(T_m, V_m) \tag{1.3}$$

the pressure of the mixture is equal to the partial pressures the components of each gas would exert if they existed at the mixture temperature and volume [6, 7]. Likewise, for Amagat

$$V_m = \sum_i^k V_i(T_m, P_m) \tag{1.4}$$

the volume of the mixture is equal to the partial volumes the components of each gas would exert if they existed at the mixture temperature and pressure [6, 7].

For real gas mixtures, Equation (1.2) still applies to account for the deviation from ideal-gas behavior. Amagat and Dalton Law's are exact for ideal gas mixtures and yield identical results when the compressibility factor is unity [6, 7], however, they only serve as approximates for real gases. Amagat's Law implicitly accounts for the intermolecular forces between the molecules of the various gases in the mixture, while Dalton's Law disregards the influence of disparate molecules in the gas mixture [6]. Thus Amagat's Law is reported to be more suitable for higher pressures and Dalton for lower pressures [6, 7].

1.1.3 BKW

The Becker-Kistiakowsky-Wilson (BKW) EOS [8, 9] has been extensively used to calculate detonation properties and is defined by the equations

$$\frac{PV}{RT} = 1 + Xe^{\beta X}, \qquad X = \frac{\kappa \sum_i n_i k_i}{V(T+\theta)^\alpha} \tag{1.5}$$

where the values α, β, κ, and θ are empirically determined, constant fit parameters. The parameters $\alpha, \beta, \kappa, \theta$ and k_i may be adjusted to fit measured detonation proper-

ties, however, this unveils a weakness of the EOS, namely that *a priori* estimation of the parameters for any new chemical components introduced could prove unsatisfactory if dissimilar from components used in a parameter determination study [9]. The recommended values for the constants α, β, κ, and θ are $0.5, 0.298, 10.5$, and 6620, respectively [8], which were obtained from calibration of over sixty explosives across a wide range of densities and constitutes the BKWS EOS *.

1.1.4 JCZ3

The Jacobs-Cowperthwaite-Zwisler (JCZ) EOS [10] is of the form

$$P = \frac{G(V,T)nRT}{V} + P_0(V) \tag{1.6}$$

where the Grüneisen function G, and the internal pressure function P_0, are given by

$$G = 1 - \frac{V}{f}\left(\frac{\partial f}{\partial V}\right)_T, \qquad P_0 = -\frac{\mathrm{d}E_o}{\mathrm{d}V} \tag{1.7}$$

where E_o denotes the volume potential of a face-centered cube lattice and f is a factor based on the Helmholtz free energy of an ideal gas which guarantees that G maintains correct behavior across a wide range of densities [10].

The JCZ3 EOS was created (JCZ1 and JCZ2 are based on less accurate potentials and are no longer in use [11]) by modifying the Grüneisen and internal pressure function to be composed of the exponential 6 (EXP6) intermolecular potential function

$$P = \frac{G(V,T,\varphi)nRT}{V} + P_0(V,\varphi) \tag{1.8}$$

$$\varphi(r) = \varepsilon\left[\left(\frac{6}{\eta-6}\right)\exp\left[\eta(1-\frac{r}{r^*})\right] - \left(\frac{\eta}{\eta-6}\right)\left(\frac{r^*}{r}\right)^6\right] \tag{1.9}$$

*BKWS is SNL's optimized, re-parameterized BKW EOS.

which describes the P-V-T relationship of the gaseous product species that result from the detonation of energetic materials [10, 12, 13]. The molecular force parameters ε (often given as ε/k_B, where k_B is the Boltzmann constant) and r^* are the well depth for the pair potential and the radius of the minimum pair potential energy, respectively. It has been shown that the EXP6 potential function is relatively insensitive to perturbations in force constant η and yields the best agreement between measured and predicted pure liquid shock Hugoniots when using $\eta = 13$ [12, 13].

1.1.5 EXP6

The EXP6 EOS [14] has a similar formulation as the JCZ3 EOS — with slight variations in the definition of parameter mixture rules [11] — the main difference, however, is that the molecular force parameter η is no longer assumed to be a constant $\eta = 13$ and is instead calculated as the following mixture rule

$$\eta = \frac{\sum_{i,j} x_i x_j \eta_{ij} \varepsilon_{ij} r_{ij}^3}{\varepsilon_m r_m^3} \tag{1.10}$$

where i, j subscripts denotes parameters between chemical species i and j [11].

1.2 Shock Waves

A shock wave is defined as a traveling disturbance which is characterized by a sharp discontinuity in the field variables of a fluid [15]. A shock wave is formed from a finite-amplitude compression wave, as the velocity gradient steepens until asymptotic behaviour is reached. After the shock wave is formed, it continues to travel at an equilibrium speed [15].

The Rankine-Hugoniot Equations [15] relate the density ratio to the fluid velocity and pressure ratios across a shock wave; the equation for a normal shock (shock wave

is oriented normal to the velocity vector) may be given by:

$$\frac{\rho_2}{\rho_1} = \frac{1 + \frac{p_2}{p_1}\frac{(\gamma+1)}{(\gamma-1)}}{\frac{(\gamma+1)}{(\gamma-1)} + \frac{p_2}{p_1}} = \frac{v_1}{v_2} \tag{1.11}$$

where the subscripts 1 and 2 refer to the pre- and post- conditions of the shock, respectively.

Numerically modeling a shock wave is by no means trivial. Care must be taken to avoid numerical instabilities caused by cusps in the results due to discontinuities in the field variables [16]. However, only the post conditions of the shock are of interest — specifically speed, pressure, temperature — and should be accurately modeled, the characteristics of the shock wave — such as the width or structure — need not be accurately captured [17].

1.3 Adiabatic Flame Temperature

The adiabatic flame temperature is the maximum temperature that combustion products reach during a combustion process when there is no heat loss to the surroundings [6]. The adiabatic flame temperature depends on the following [6]:

i. reactant state

ii. the completeness of the reaction

iii. amount of air used

Incomplete combustion, heat transfer, and dissociation of the products all result in a lower temperature.

1.4 Sensitivity Analysis

Sensitivity analyses are vital in identifying which inputs to a complex system have the greatest influence on the outputs [18, 19, 20]. There exists a plethora of sensitivity techniques, one of which involves generating a distribution on each of the inputs and propagating them through the system which results in a distribution on the outputs [20]. The inputs which yield the greatest distribution on the outputs is considered to be the most sensitive, the identification of which allows the focus to be redirected to the critical inputs.

1.4.1 Incremental Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is a pseudo-random, stratified sampling technique where the cumulative distribution for each variable is divided into n non-overlapping intervals of equal probability [19, 20, 21, 22]. The size of the interval is determined by the specified probability distribution, for instance a normal distribution possesses smaller segments near the mean compared to the tails. A sample from each interval is then selected at random, such that the resulting sample set contains only one sample in every row and column of the hypercube [20]. A Latin Hypercube is the generalization of a Latin Square into an arbitrary number of dimensions, similarly, LHS is a k-dimensional extension of Latin Square sampling [19, 23]. A square grid is a Latin Square if and only if there is only one sample in each row and column, an example of which is depicted in Figure 1.1. The non-overlapping nature of the intervals along with the definition of a Latin Square restricts a sample from being used more than once.

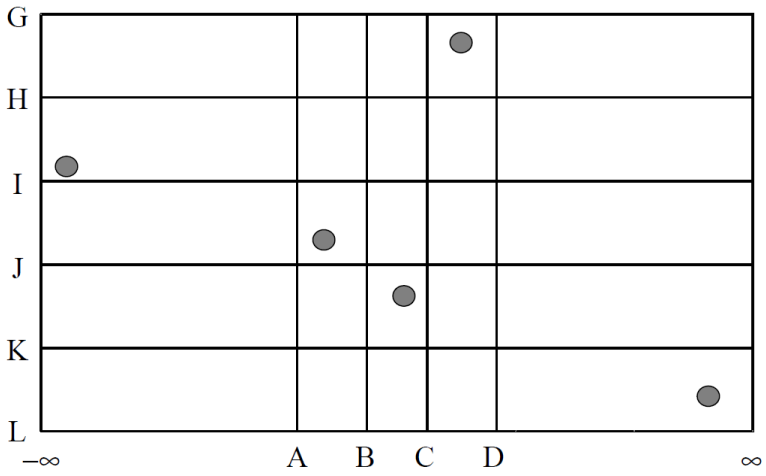


Figure 1.1: Latin Square example [21].

Incremental Latin Hypercube Sampling (iLHS) enables further sampling to be performed until convergence is reached. Each incremental sample doubles the total number of samples and contains the results of the previous hypercubes. The full sample is itself a Latin Hypercube — the stratification and correlation structure are maintained [20].

It is difficult to make a rigorous convergence assertion using random Monte Carlo sampling techniques. Because of the random sampling, previous samples are not considered when incrementing the random samples. The assumption cannot be made that the sample set has filled the probability space, nor that the entire space has been sampled with sufficient density. It typically takes orders of magnitude more samples to achieve convergence with random Monte Carlo than with iLHS [19].

1.4.2 Gaussian Distribution

The Gaussian (or normal) distribution function is widely used to model random continuous variables [24]. When the variation of measured data is due totally to random factors, and negative and positive deviation occurrences are equally probable, then the Gaussian distribution has been shown to describe the dispersion of the data [24].

The normal distribution may be given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{1.12}$$

where the two free parameters, μ and σ, are the population mean and standard deviation respectively. The probability of a value x existing between a lower limit x_1 and an upper limit x_2 may be given by

$$P(x_1 \leq x \leq x_2) = \int_{x_1}^{x_2} f(x)dx = \int_{x_1}^{x_2} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} dx \tag{1.13}$$

the integration of which must be evaluated numerically as $f(x)$ is in the form of an error function [24].

Figure 1.2 depicts a normal distribution with probability as a function of standard deviation†. The distribution curve is centered on the population mean μ.

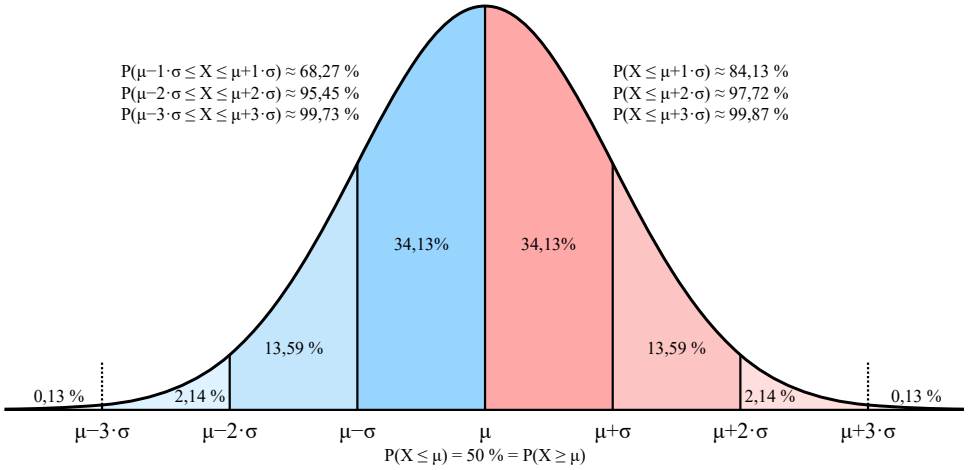


Figure 1.2: A normal distribution graph visualizing the values 1σ, 2σ, and 3σ.

1.4.3 Convergence Metrics

The incremental Latin Hypercube Sampling is considered to have sufficiently converged when the prescribed metric exhibits asymptotic behaviour for the latter few

†Wolfgang Kowarschick, Oct. 2012, Obtained from Wikimedia Commons.

incremental sample sets.

Mean, Variance, & Standard Deviation

A population contains the entire collection of measurements, observations, etc. about which some generalizations will be made. A sample is a subset of the population for which numerical data is obtained [24].

The mean is a common parameter used to describe the central tendency of a dataset [24]. The population mean may be given by

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1.14}$$

where N is the finite number of observations in the population. The sample mean, $\overline{x}$, possess a similar formulation as the population mean, where the sample observations is used in place of N. Given a sufficiently large sample size, the sample mean will be representative of the population mean [25].

Variance is a measure of dispersion or variability of a dataset [25]. The population variance may be given by

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} \tag{1.15}$$

and the sample variance may be given by

$$s^2 = \frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1} \tag{1.16}$$

where $N-1$ is the *degrees of freedom*. Since the population mean is rarely known, the sample mean $\overline{x}$ must be used instead, however this reduces the degrees of freedom from N to $N-1$ as only $N-1$ of the N deviations $(x_i - \overline{x})$ are freely determined [25].

The population and sample standard deviation are then the positive square roots of the respective variance [24].

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}}, \qquad s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}} \tag{1.17}$$

Infinity Norm

The infinity norm [26] (or uniform norm) is given by

$$L_\infty = ||x||_\infty = \max|x| \tag{1.18}$$

and simply just returns the value in a vector with the largest magnitude.

Mean Absolute Error

The Mean Absolute Error (MAE) has been showed to be the most natural measure of average error magnitude and is recommended over the typical Root Mean Square Error (RMSE) as there is no clear interpretation of the RMSE [27]. The MAE is given by

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i| \tag{1.19}$$

where x_i is the predicted value and y_i is the observed value [27].

Pearson Correlation Coefficient

The Pearson correlation coefficient (PCC, also known as the sample correction coefficient) is a parameter which can be used to determine if there exists a functional relationship between two variables [24, 25]. The magnitude of r_{xy} — given by Equation (1.20) — lies between -1 and 1 where a value of 1 indicates there is a positive

linear relationship, a value of -1 indicates a negative linear relationship, and a value of 0 indicates there is no linear correction between the two variables. Simply stated, the PCC measures the strength of the linear relationship between two variables [25].

$$r_{xy} = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}}, \qquad -1 \leq r_{xy} \leq 1 \tag{1.20}$$

Chapter 2

Methodology

In science one tries to tell people, in such a way as to be understood by everyone, something that no one ever knew before. But in poetry, it's the exact opposite.

— Paul Dirac

2.1 Software

2.1.1 DAKOTA

DAKOTA is a design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis framework developed at SNL [20]. DAKOTA possesses the ability to wrap around a simulation code supplied by the user and parameterize the code's input file as shown in Figure 2.1.

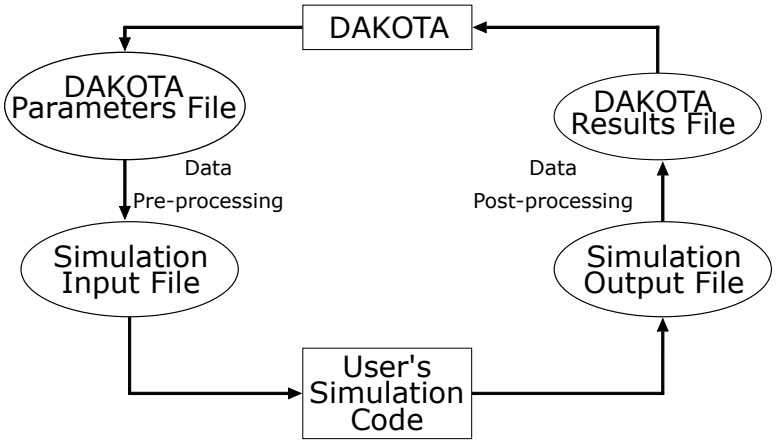


Figure 2.1: "Black-box" interface between DAKOTA and user's simulation code [20].

DAKOTA's "black-box" allows a generic simulation to be parameterized with a

unique set of variables as many times as desired in parallel. A multi-dimensional parameter study allowed for the creation of the nominal simulations. The uncertainty quantification study was then performed with DAKOTA's incremental Latin Hypercube sampling capability.

2.1.2 Tiger

The generation of custom, tabular EOS may be achieved through use of Tiger, which is a thermochemical equilibrium code that was originally developed by Stanford Research Institute and has since been updated and maintained by SNL [28]. For every EOS with the exception of Amagat and Dalton, the mixing of gases is handled by Tiger, which computes equilibrium thermodynamic states over a prescribed range of temperatures and volumes for the desired mixture composition. Tiger contains a species library of over 750 gases and elements which have been verified and validated by Hobbs, et. al [11].

2.1.3 CTH & BCAT

CTH is a multidimensional, multi-material, large deformation, strong shock, solid mechanics code (CTH is also known as a hydrocode) developed at SNL [16, 29]. CTH was the chosen analysis software primarily for its tabular equation of state input capability, which allows for usage of mixed material models that are created according to custom EOS formulations. There are six possible domain options available: one-dimensional linear, cylindrical, and spherical; two-dimensional cylindrical and rectangular; and three-dimensional rectangular. The numerical solver in CTH consists of a two-part solution scheme — a Lagrangian distortion step during which the mesh deforms to follow the material motion and a Eulerian remap step during which the distorted mesh is mapped back onto the original mesh [16, 29]. It should

be noted that CTH uses the cgs-eV[‡] unit system [29].

BCAT is a code in the CTH distribution package which is used to develop and test EOS models for CTH [30]. Before Tiger's tabular EOS can be utilized by CTH, the tables must be converted to the SESAME format developed by Los Alamos National Laboratory (LANL) [31]. The SESAME format consists of both plain text and binary tabular file formats, however, only the binary format will be used. The conversion is handled by BCAT.

2.2 Experimental Setup

The University of New Mexico (UNM) shock tube — depicted in Figure 2.2 — is constructed with a 1.97 m long driver section and a 3.2 m long driven section [5]. The driver section consists of a 1.22 m long cylindrical tube with a 7.62 cm inner diameter coupled to a 0.75 m long tube with a 7.62 cm inside square cross-section. The driven section contains a 7.62 cm inside square cross-section. The driver section was lengthened from its original length of 1.22 m [32] as the rarefaction shock wave can overtake and accelerate the incident shock wave in a driver section of insufficient length [33]. The driver and driven sections of the shock tube were separated by a thin-film polyester diaphragm which was then punctured by a pneumatically driven steel rod with a broad arrowhead at the initiation of the experiment [5].

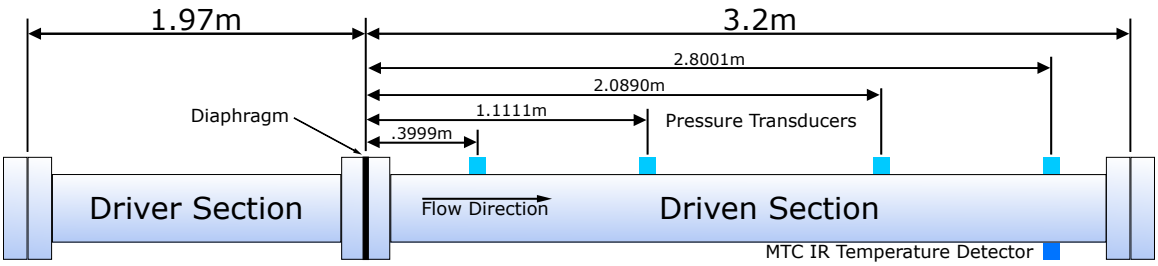


Figure 2.2: Modified notional depiction of UNM shock tube [5, 32].

The driver section was filled with Nitrogen (N_2) and pressurized to one of three

[‡] 1 eV = 11604.5 K

initial pressures, the driven section was filled with a mixture (one of two different molar concentrations) of He and SF_6 and also pressurized to one of three initial pressures[§]. Table 2.1 lists the varied experimental parameters which were multiplied through each columnar entry to create 18 unique experiments [5].

Table 2.1: Variable Experimental Parameters

P_{N_2} [kPa]	P_{He/SF_6} [kPa]	x_{He}
1006	39.3	50%
1145	78.6	75%
1282	118	

2.3 Mesh and Boundary Conditions

The UNM shock tube was modeled in CTH using a three-dimensional rectangular domain. Only the inside of the shock tube was modeled, therefore, the cylindrical portion of driver section was modeled as a rectangular tube with a 7.62 cm square cross-section; the length of the tube was shortened to preserve the volume of the driver section. The length ratio between a cylinder of radius r and a rectangular parallelepiped of square cross-sectional length $2r$, with both of length l, can be derived as

$$V_{rect} = 4r^2 l_{rect}, \qquad V_{cyl} = \pi r^2 l_{cyl}, \qquad \Rightarrow \qquad l_{rect} = \frac{\pi}{4} l_{cyl} \tag{2.1}$$

The polyester diaphragm was not modeled, at time $t = 0$ the assumption was made that the diaphragm has just been punctured and did not affect the subsequent flow development [17]. Lastly, reflective (symmetry) boundary conditions were applied on all boundaries of the domain [29].

[§]Due to elevation, 78.6 kPa is the average atmospheric pressure in Albuquerque, NM.

2.4 Amagat and Dalton Mixing

For the generation of tabular Amagat or Dalton EOS, it is required that Tiger computes equilibrium thermodynamic states over a prescribed range of either temperatures and pressures or temperatures and volumes for Amagat or Dalton respectively, and creates a pure gas tabular EOS for each gas in the mixture composition. The mixing of the pure gas tables to create a mixed tabular EOS is then performed manually through the use of Python scripts according to the desired EOS formulation. The tabular EOS from Tiger consists of five state variables: temperature T, specific volume ν, pressure P, specific energy e, and specific entropy s [28].

For Dalton, temperature and volume are assumed to be constant throughout the mixture, while for Amagat the temperature and pressure are assumed to be constant throughout the mixture; the values for the remaining mixture variables are obtained by summation across the pure gas EOS tables. However, Tiger outputs several specific variables

$$\omega = \frac{\Omega}{m}, \qquad m = Mn \tag{2.2}$$

(where ω is an arbitrary specific variable, Ω is the arbitrary variable, and m is the mass) instead of the variables themselves, which prevents direct summation of the variables. Note that the number of moles in the tabular EOS is normalized to one; as such, the mass may also be given as

$$m_i = M_i x_i, \qquad x_i = \frac{n_i}{n_m} \tag{2.3}$$

where x is the mole fraction of the species and the subscripts i and m denote the ith mass and total mixture mass respectively. The specific variables must first be

weighted by mass fraction w as follows

$$w_i = \frac{m_i}{m_m} \tag{2.4}$$

for summation to occur.

2.4.1 Dalton

For Dalton, the formulation of the variables may be given as

$$T_m = T_i, \qquad \nu_m = w_i \nu_i, \qquad P_m = \sum_i P_i(T, V) \tag{2.5a}$$

$$e_m = \sum_i w_i e_i(T, V), \qquad s_m = \sum_i w_i s_i(T, V) \tag{2.5b}$$

where the specific volume is still required to be weighted by mass fraction in order to obtain the specific volume of the mixture despite not being summed.

Dalton Mixing Nuance

An important nuance to note is that during generation of the pure gas Dalton EOS tables with Tiger, the prescribed volume range is given as a prescribed specific volume range and thus the range needs to be scaled between the pure gas tables in order to create tables of equivalent thermodynamic states. The scaling may be given as follows

$$\nu_j = \nu_i \frac{m_i}{m_j} \tag{2.6}$$

where the subscripts i and j denote the chemical species i and j respectively.

2.4.2 Amagat

For Amagat, the formulation of the variables may be given as

$$T_m = T_i, \qquad \nu_m = \sum_i w_i \nu_i(T, P), \qquad P_m = P_i \tag{2.7a}$$

$$e_m = \sum_i w_i e_i(T, P), \qquad s_m = \sum_i w_i s_i(T, P) \tag{2.7b}$$

Amagat Mixing Nuance

Another important nuance to note is that BCAT demands a constant specific volume range across all the isotherms during the conversion to the SESAME EOS format. Due to the specific volume range being calculated instead of prescribed for Amagat, the specific volume ranges vary between the different isotherms. To remedy this quandary, the global maximum and minimum specific volume values are calculated and a new specific volume range — which is evenly spaced on a logarithmic scale (a geometric progression) — is generated using:

$$\nu_{\text{new}} = \text{geomspace}\left(\min(\nu_{\text{old}}), \max(\nu_{\text{old}}), n_P\right) \tag{2.8}$$

where `geomspace` is the Python function (in the Numpy module) which produces the geometric progression, and n is the number of pressure discretizations. A new pressure range must then be calculated for every isotherm, however, as pressure is the independent variable and specific volume is the dependent variable — that is, specific volume is a function of pressure — a power law curve fit of the form:

$$f(x) = c_0 x^{c_1} \tag{2.9}$$

must obtain pressure as a function of the specific volume and then substitute the new specific volume range to acquire the final pressure.

For every isotherm, the following system of equations must be solved:

$$\begin{bmatrix} 1 & \ln(\nu_1) \\ \vdots & \vdots \\ 1 & \ln(\nu_n) \end{bmatrix} \begin{bmatrix} \ln(c_0) \\ c_1 \end{bmatrix} = \begin{bmatrix} \ln(P_1) \\ \vdots \\ \ln(P_n) \end{bmatrix} \tag{2.10a}$$

$$\begin{bmatrix} \ln(c_0) \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 & \ln(\nu_1) \\ \vdots & \vdots \\ 1 & \ln(\nu_n) \end{bmatrix}^{-1} \begin{bmatrix} \ln(P_1) \\ \vdots \\ \ln(P_n) \end{bmatrix} \tag{2.10b}$$

The system is overdetermined, that is, there is no unique solution as the number of equations is greater than the number of unknowns. Thus, the matrix inverse must be computed using a Moore-Penrose pseudo-inverse [34] using singular value decomposition (SVD). The pseudo-inverse of A (denoted A^+) is unique and contains the least-squares solution. It can be shown [26] that if

$$Q_1 \Sigma Q_2^T = A \tag{2.11}$$

is the singular value decomposition of A, then

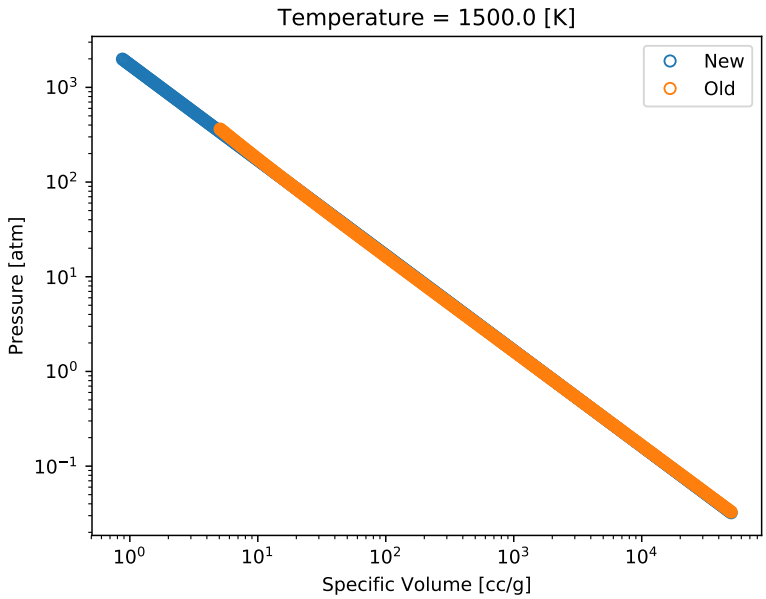
$$A^+ = Q_2 \Sigma^+ Q_1^T \tag{2.12}$$

where Σ is a diagonal matrix containing the singular values of A, Σ^+ is the diagonal matrix containing the reciprocal singular values of A, and Q_1, Q_2 are orthogonal matrices. Finally, the new pressure range may be given by

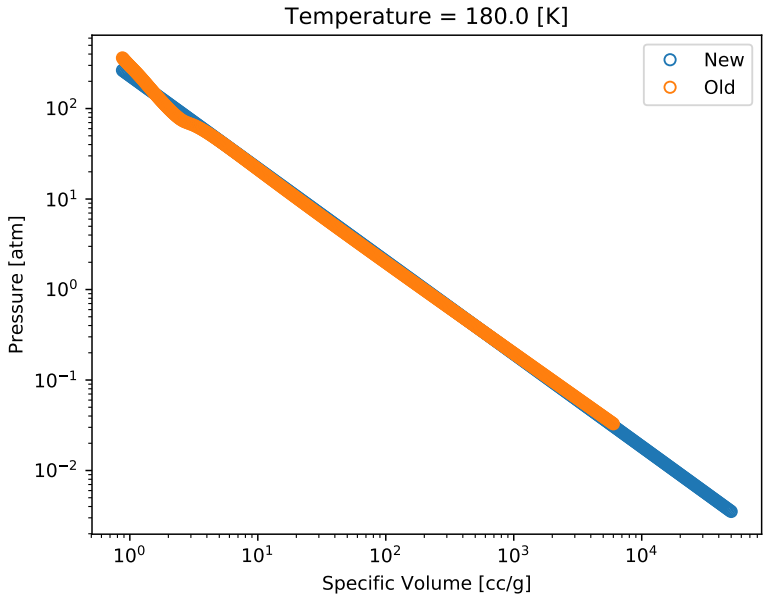
$$P_{\text{new}} = \exp(\ln(c_0)) v_{\text{new}}^{c_1} \tag{2.13a}$$

$$P_{\text{new}} = c_0 v_{\text{new}}^{c_1} \tag{2.13b}$$

Figure 2.3 depicts the original dataset superimposed upon the interpolated dataset for the bounding isotherms of the EOS.



(a) Highest temperature isotherm.



(b) Lowest temperature isotherm.

Figure 2.3: Power law interpolation of prescribed pressure.

To quantify the error introduced by the interpolation, both temperature and pres-

sure, and temperature and specific volume were prescribed during the creation of the JCZ3 EOS in Tiger. The resultant post-shock variables were visually indistinguishable.

2.5 Adiabatic Flame Calculations

Examination of the He/SF_6 mixture at the initial experimental conditions with Tiger's adiabatic flame temperature calculations revealed that:

i. there was no dissociation of the SF_6 into its constituents.

ii. the compressibility factor z of the mixture was essentially unity ($z \approx 1.02$), indicating a weak shock.

During the generation of the mixed EOS tables, Tiger failed to converge over the thermodynamic state prescribed. The output from Tiger revealed that that the EOS table contained several species other than SF_6. To resolve the convergence issue, Tiger was restricted to only contain pure SF_6 as the adiabatic temperature calculations showed there was no dissociation of the SF_6.

2.6 Simulation Procedure

For this research, two different studies were performed. The first was a nominal parameter study in which simulations were performed for the unique experimental conditions. The second was a sensitivity study in which the initial conditions of the simulation were independently varied via iLHS to assess the correlation between the input and output variables.

For each study, DAKOTA generates the unique parameters for each simulation in parallel. The variables are then passed to a collection of Python scripts which perform the following steps for each simulation:

1. Generate the Tiger input deck and execute Tiger to create the tabular EOS(s).
2. IF Amagat or Dalton: Manually create mixed EOS.
3. Generate BCAT input deck and execute BCAT to convert EOS to SESAME format.
4. Generate CTH input deck and execute CTH to initialize the shock tube simulation.

where an "input deck" is a plain text file containing the instruction set for the software.

After the completion of all simulations, DAKOTA post-processes the output data from CTH for each simulation. The post-processing utilizes another Python script which plots desired variable time-histories but more importantly calculates a single scalar value from each of the time-histories to represent each of the post-shock variables of interest for that particular simulation. DAKOTA then collects all the post-shock quantities and conveniently tabulates them in a text file against the initial input parameters which allows for further post-processing outside of DAKOTA to visualize the global results.

2.6.1 Nominal Parameter Study

The experimental parameters in Table 2.1 were combined with the six EOS to create Table 2.2. DAKOTA then performed a multidimensional parameter study by multiplying through the columnar entries to create 108 unique simulations.

Table 2.2: Nominal Parameters

EOS	P_{N_2} [kPa]	P_{He/SF_6} [kPa]	x_{He}
Ideal	1006	39.3	50%
Amagat	1145	78.6	75%
Dalton	1282	118	
BKW			
JCZ3			
EXP6			

2.6.2 Sensitivity Study

For the sensitivity study, only the Amagat and Dalton EOS were of interest. DAKOTA does not contain the ability to perform iLHS sampling on top of a multidimensional parameter study. Therefore, two individual studies were performed — one for Amagat and one for Dalton — as part of the overall sensitivity study. The five parameters to be perturbed are the driver initial pressure and temperature, the driven initial pressure and temperature, and the molar fraction of the Helium in the mixture. Table 2.3 contains the mean values of the parameters for each study. It was unrealistic due to the computational cost to perform a sensitivity study for every unique experiment — 18 in total — therefore, to reduce the number of studies necessary, the mean values were chosen from Table 2.1 in such a way to mathematically bound the problem. That is, the Amagat EOS was pared with the maximum driver pressure, the minimum driven pressure, and the lowest Helium molar fraction — paring the strongest resultant shock against the heaviest mixture. The Dalton EOS was pared with the minimum driver pressure, the maximum driven pressure, and the highest Helium molar fraction — paring the weakest resultant shock with the lightest mixture.

Table 2.3: iLHS Perturbed Variables

Variable	Symbol	Amagat	Dalton
N_2 Initial Pressure	χ_1	1282 [kPa]	1006 [kPa]
He/SF_6 Initial Pressure	χ_2	39.3 [kPa]	118 [kPa]
Helium Molar Fraction	χ_3	50%	75%
N_2 Initial Temperature	χ_4	295 [K]	295 [K]
He/SF_6 Initial Temperature	χ_5	295 [K]	295 [K]

Each variable was prescribed a Gaussian distribution; for simplicity, a normalized Gaussian distribution — depicted in Figure 2.4 — was generated in DAKOTA with $\mu = 1,\ \sigma = .03$ yielding an uncertainty range of approximately $\pm 10\%$. The resulting samples were then multiplied by the variable's respective mean before being passed to the Python scripts.

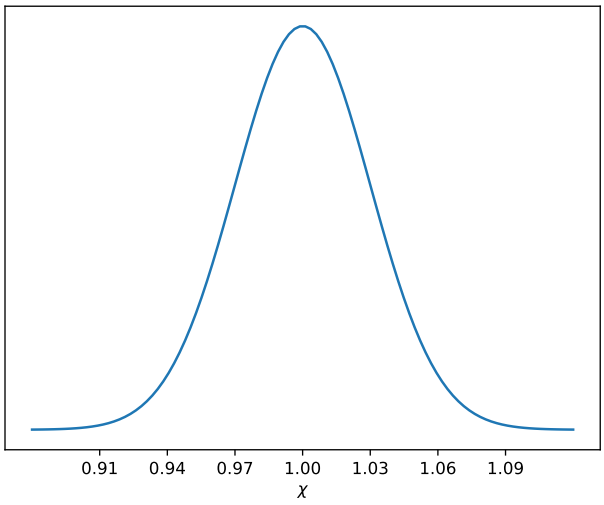


Figure 2.4: Normalized Gaussian distribution, $\approx \pm 10\%$ uncertainty range.

Gaussian distributions arise in many areas of physical phenomena, such as the height and the Intelligent Quotient (IQ) of the population, and viral infection rates. Due to this, it was expected that the uncertainty variables would follow suit and

possess a Gaussian distribution.

2.7 Post Processing

Figures 2.5 to 2.7 depict the typical time histories of the post shock variables of interest for each simulation. The time histories are collected from tracers in the same locations as the experimental pressure transducers. A scalar value is calculated for each of the variables of interest by taking the average of the maximum values from all of the transducers.

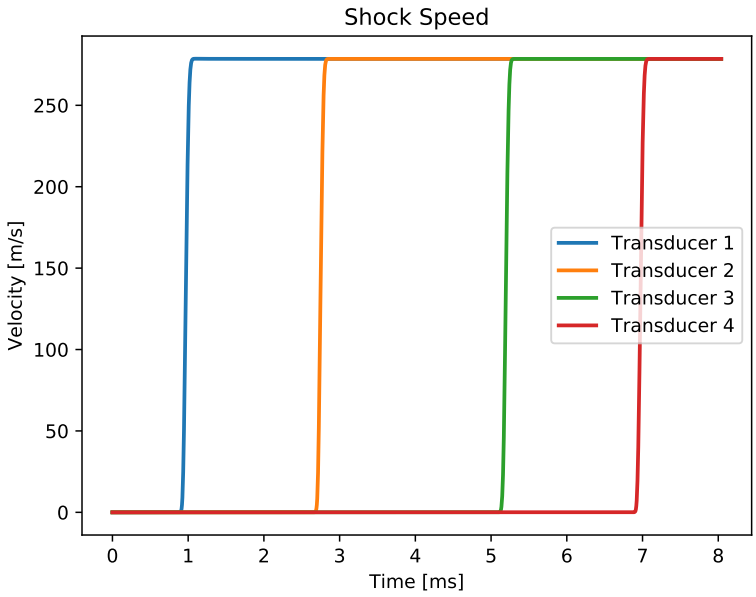


Figure 2.5: Typical velocity time history.

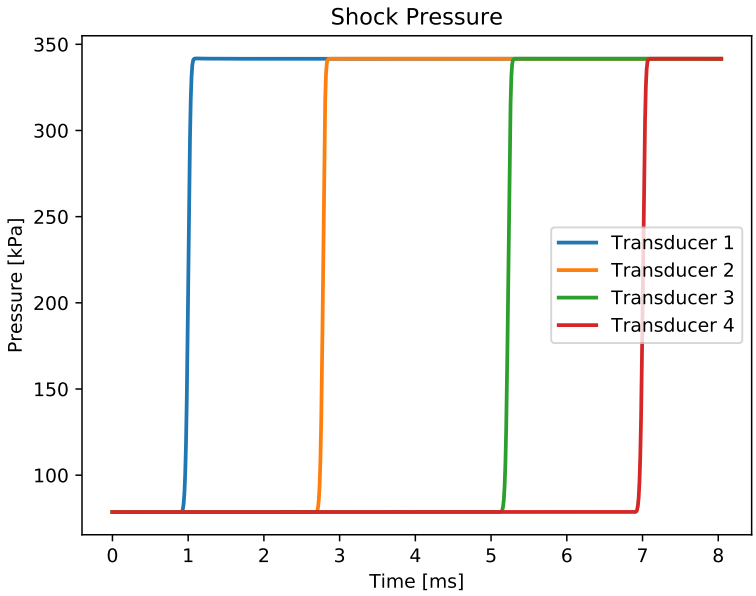


Figure 2.6: Typical pressure time history.

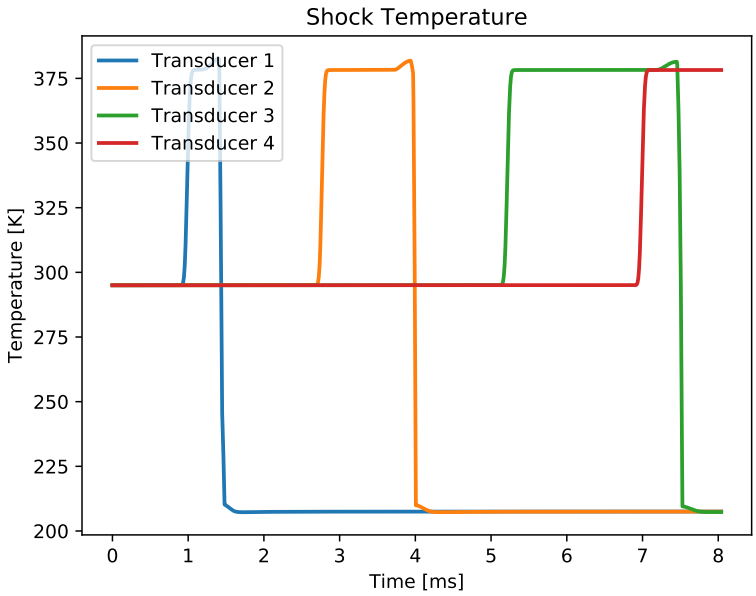


Figure 2.7: Typical temperature time history.

2.8 Mesh/Grid Convergence

Mesh convergence studies were not performed for the spatial mesh in CTH nor the discretized EOS grid in the SESAME tables. The results from Bigelow's convergence studies were utilized instead [17]. Table 2.4 lists the T, P, and ν ranges and the number of discretizations which yielded a relative error of less than 10^{-3} for the tabular EOS.

Table 2.4: Tabular EOS Variable Ranges [17]

Variable	Min	Max	N
T [K]	180	1500	224
P [atm]	.0328	363.1	484
ν [cc/g]	5	6005	484

For the spatial mesh, Bigelow showed that for 3D the values listed in Table 2.5 yielded an error of less than 5% [17]. Due to the shock tube growing in length since Bigelow's research, the number of discretizations also grew and were adjusted slightly to obtain as close to a 1:1:1 length ratio for the computational cells as possible.

Table 2.5: Spatial Mesh Discretization [17]

Variable	Bigelow	White
N_x	406	516
N_y	8	8
N_z	8	8
Δ_x [cm]	1.088	.9516
Δ_y [cm]	.9525	.9525
Δ_z [cm]	.9525	.9525

Chapter 3

Results & Discussion

The beginning of knowledge is the discovery of something we do not understand.

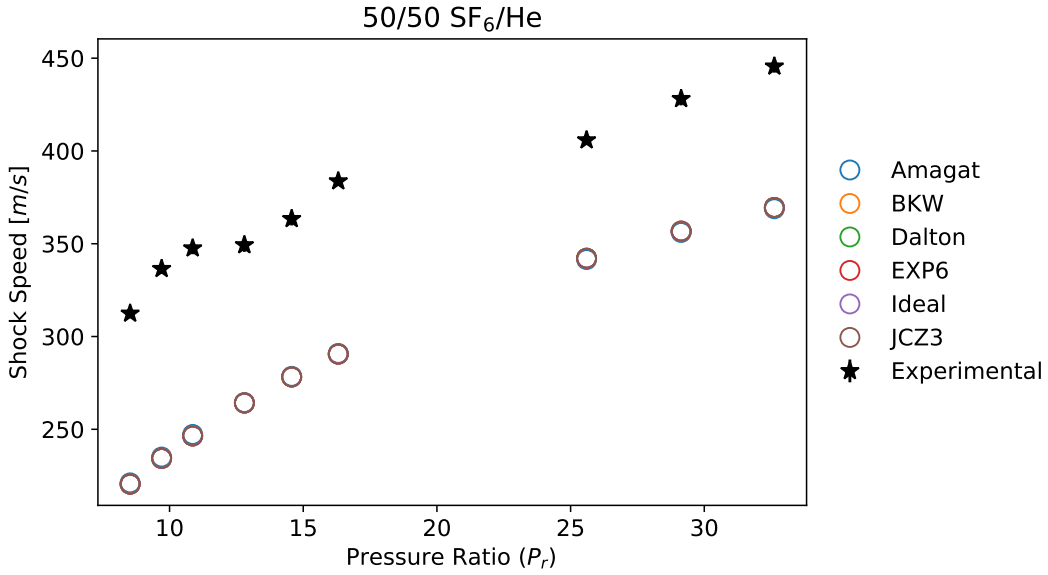
— Frank Herbert

3.1 Nominal Parameter Study

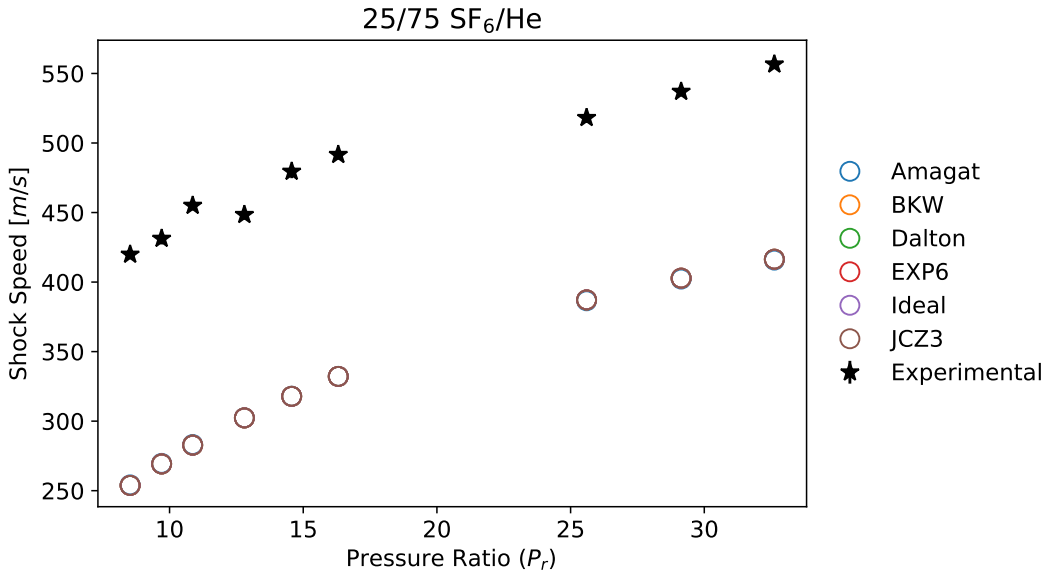
The results from the DAKOTA multi-dimensional parameter study — three post-shock quantities of interest: speed, temperature, and pressure — are compared against experimental data with uncertainty bars (in most cases, the uncertainty bars do not extend past the size of the marker) [5] and depicted in Figures 3.1 to 3.3. A dimensionless pressure ratio defined as

$$P_r = \frac{P_{\mathrm{N_2}}}{P_{\mathrm{He/SF_6}}} \tag{3.1}$$

allows for a more concise comparison between the post-shock properties and initial conditions.

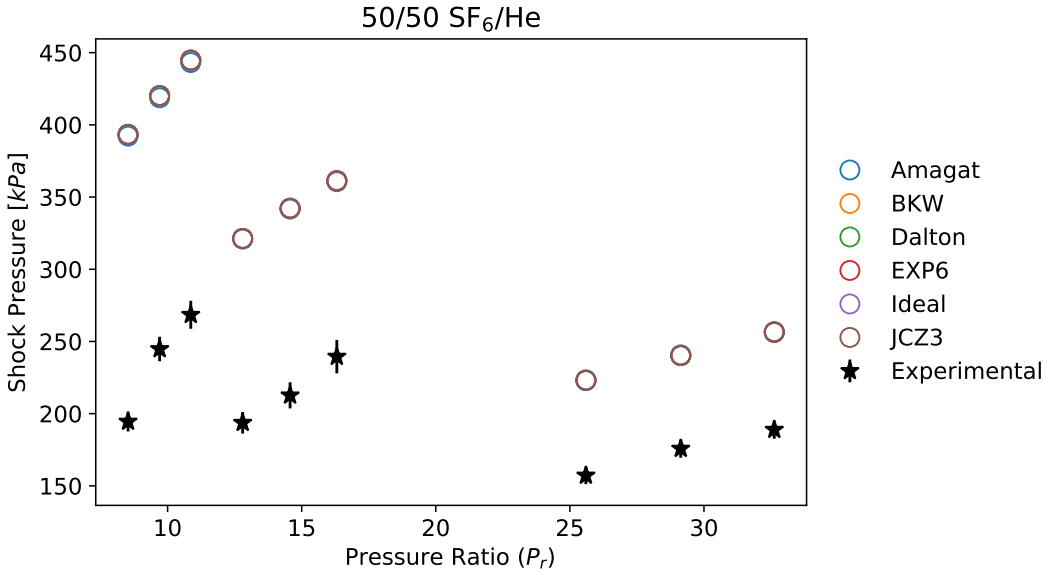


(a) 50% Helium molar fraction.

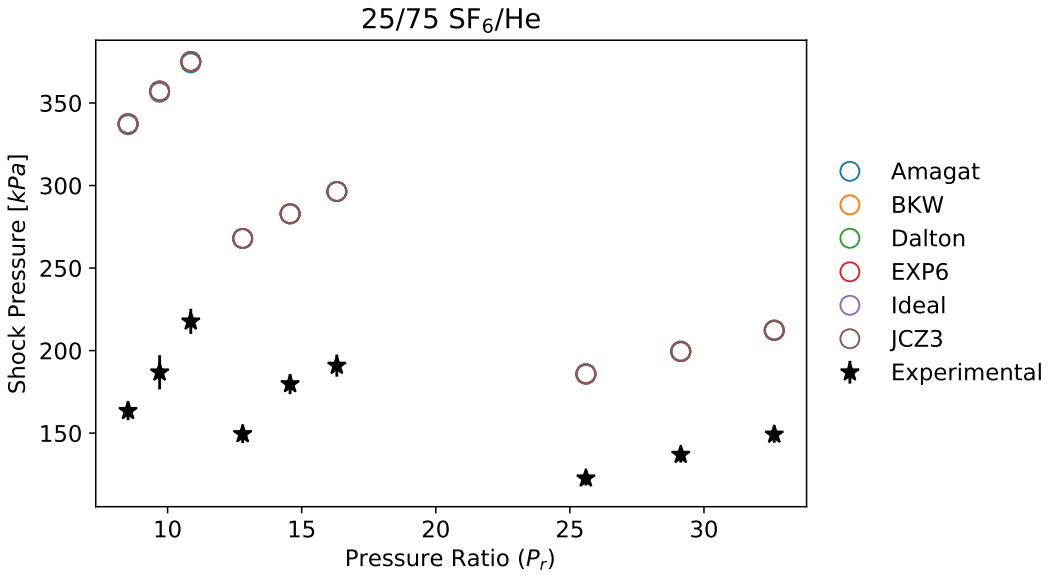


(b) 75% Helium molar fraction.

Figure 3.1: Comparison of simulational post-shock velocity against experimental data for the various mixing EOS and mixture ratios.

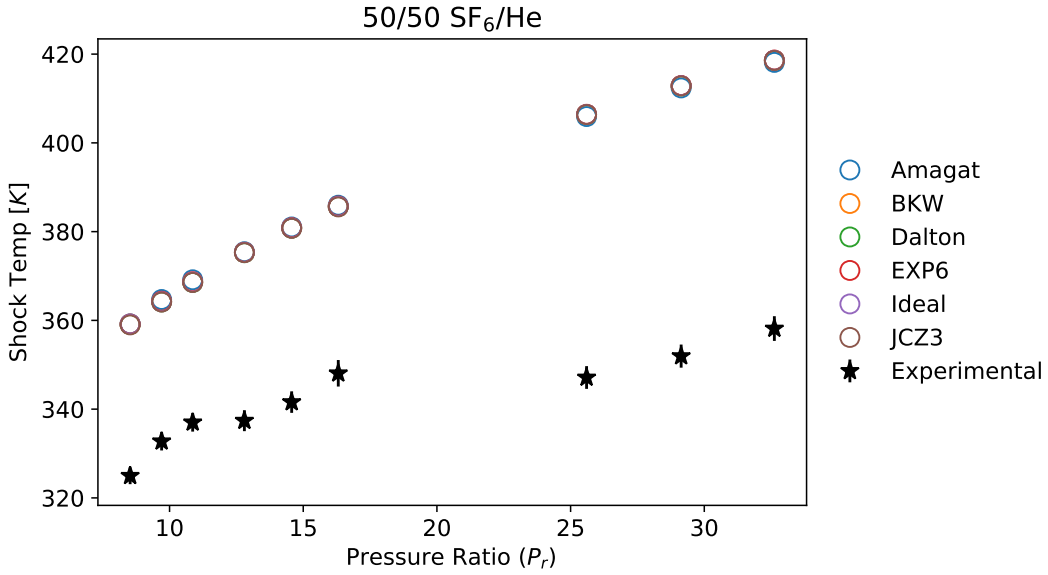


(a) 50% Helium molar fraction.

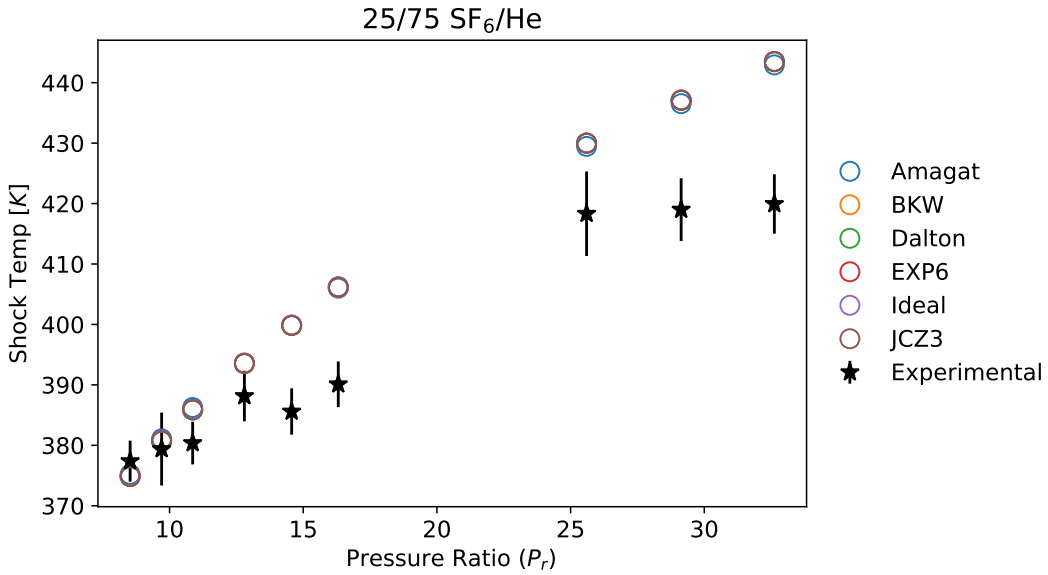


(b) 75% Helium molar fraction.

Figure 3.2: Comparison of simulational post-shock pressure against experimental data for the various mixing EOS and mixture ratios.



(a) 50% Helium molar fraction.



(b) 75% Helium molar fraction.

Figure 3.3: Comparison of simulational post-shock temperature against experimental data for the various mixing EOS and mixture ratios.

The results from all the EOS are visually indistinguishable from one another, which is to be expected as z is essentially unity — signifying a weak shock. The overall data trends are matched nicely. However, in spite of the weak shock there are large discrepancies between experiments and simulations. Similar results were

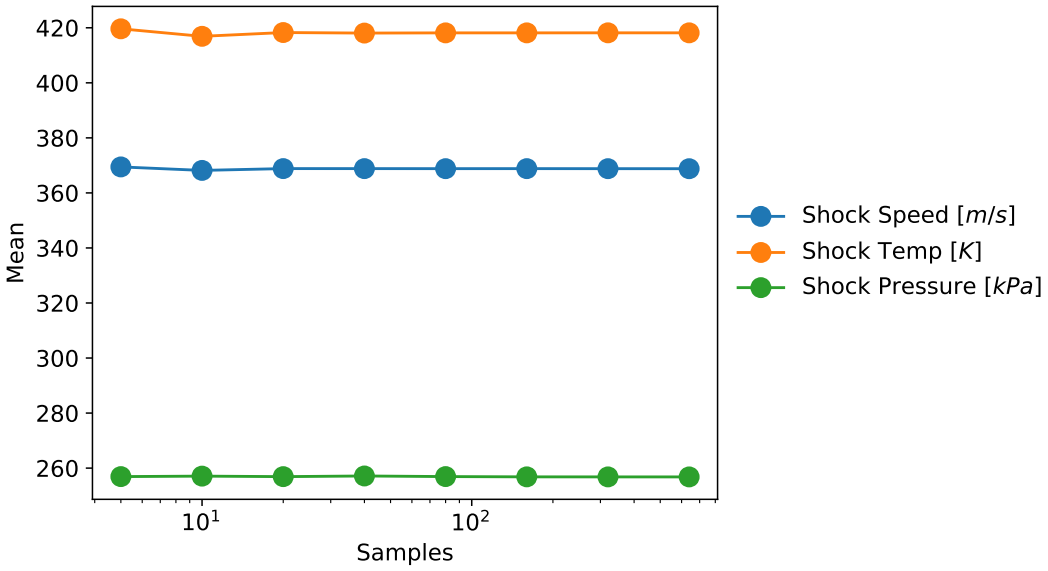
obtained from previous works [5, 17] with comparisons of experimental data against both analytical and simulation results. The corroboration of the results across the various works presents a strong assertion that the EOS fail to take into account the "time scale ... of the experiment associated with the shock passage" [5]. Kinetic Molecular Theory (KMT) appears to provide an explanation as to the cause of the discrepancies. An attempt to provide at least a qualitative explanation of the disagreement resulted in the introduction of a parameter, which is based on the difference in mean free path collision times of the species, describing the disparity in behavior between the species at the microscopic scale [5].

3.2 Sensitivity Study

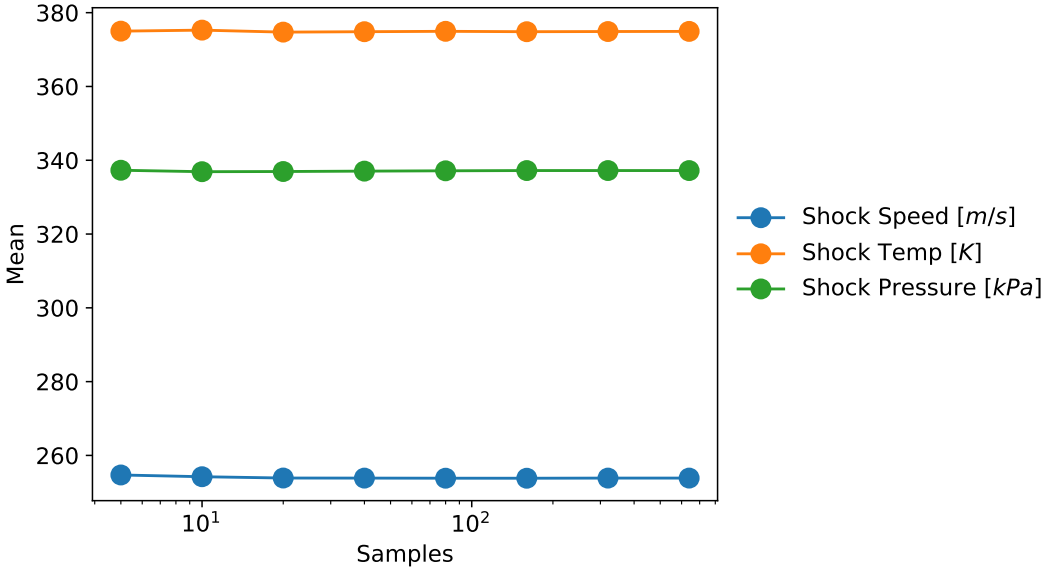
Both studies started with five samples, and subsequently doubled the number of samples until ending with 640 samples for each study — resulting in 1280 total samples for the entire sensitivity study. The distributions of the outputs were examined with respect to the aforementioned metrics and are depicted in Figures 3.4 to 3.9. From visual inspection it is evident that sufficient samples were considered to permit a convergence assertion for all metrics.

The majority of the linear correlations presented in Figures 3.7 to 3.9 are intuitive: shock speed is directly proportional to all inputs with the exception of the driven pressure, similarly the shock pressure is directly proportional to all inputs with the exception of molar fraction and the mixture temperature (an increase of the temperature yields a density increase).

Of great interest, however, is that the correction between the initial He/SF_6 temperature and the post-shock temperature is almost identically 1 for all sample sets.

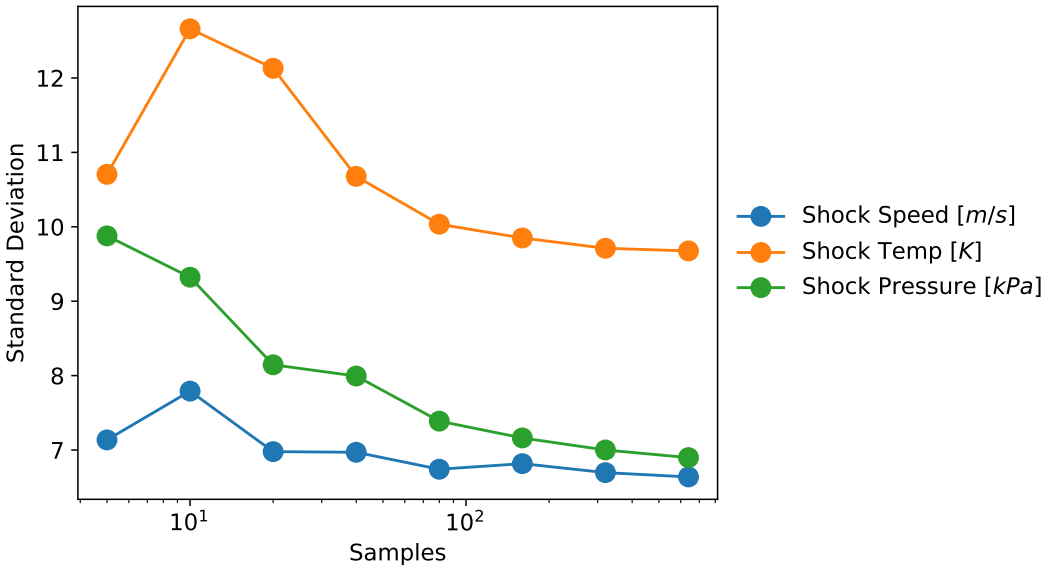


(a) Amagat EOS

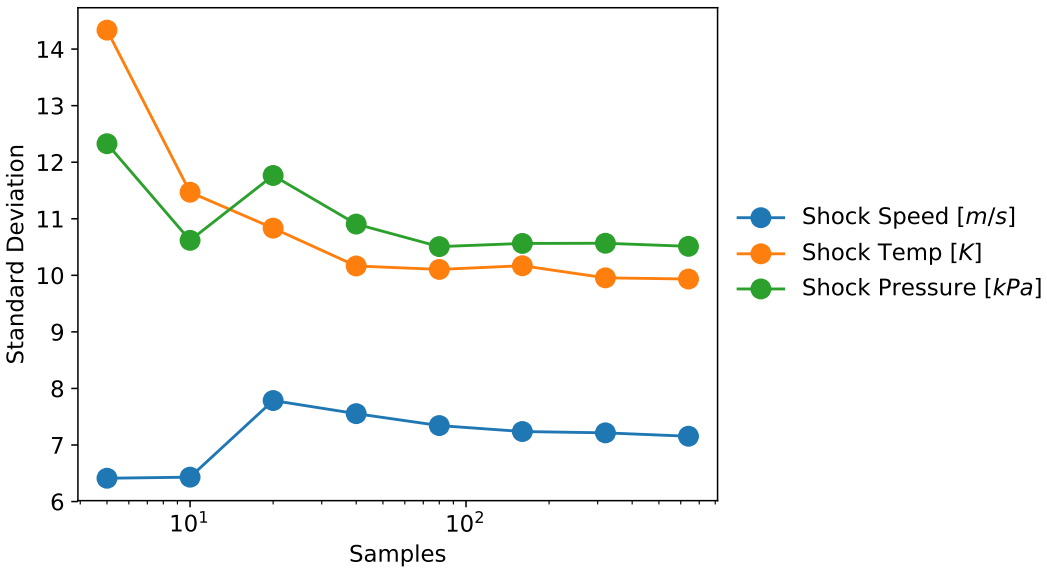


(b) Dalton EOS

Figure 3.4: Mean QOI for all incremental sample sets.

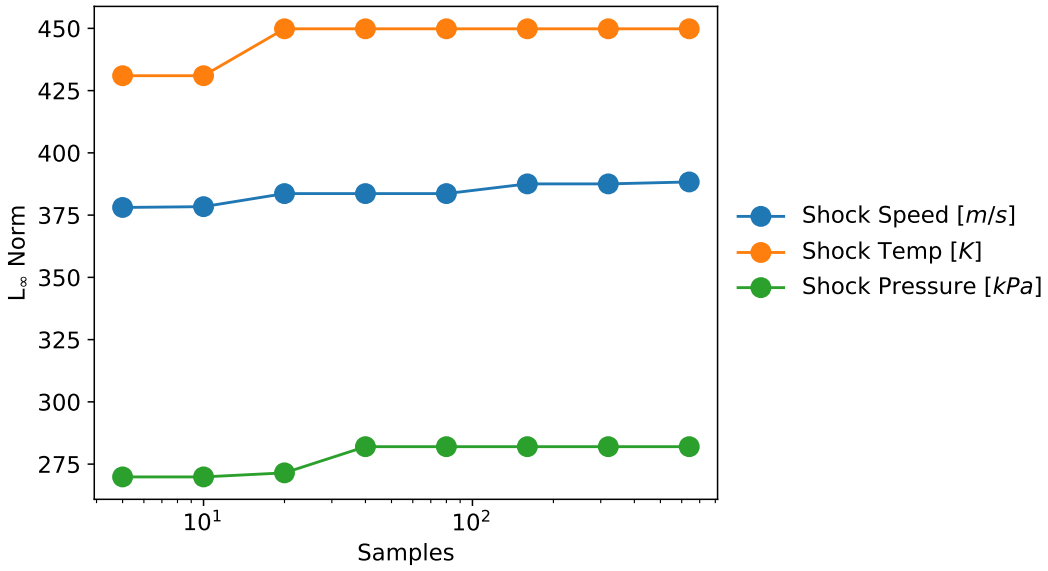


(a) Amagat EOS

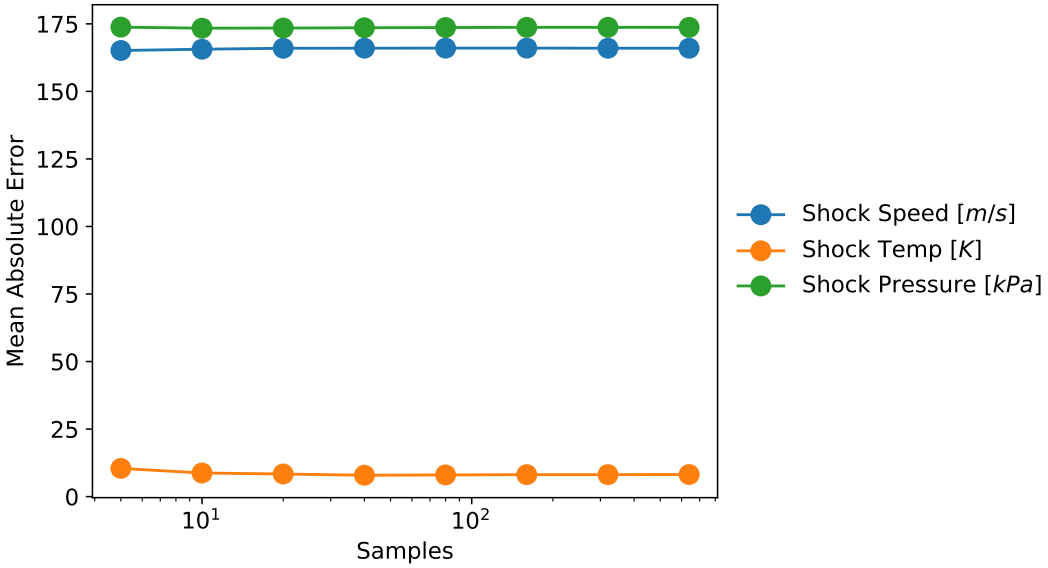


(b) Dalton EOS

Figure 3.5: QOI standard deviation for all incremental sample sets.

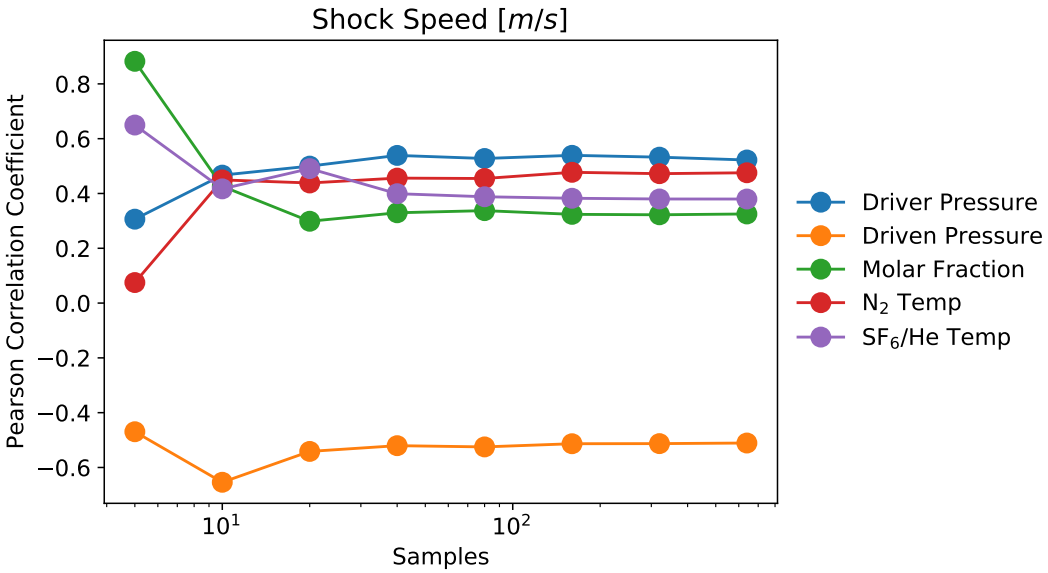


(a) Amagat EOS

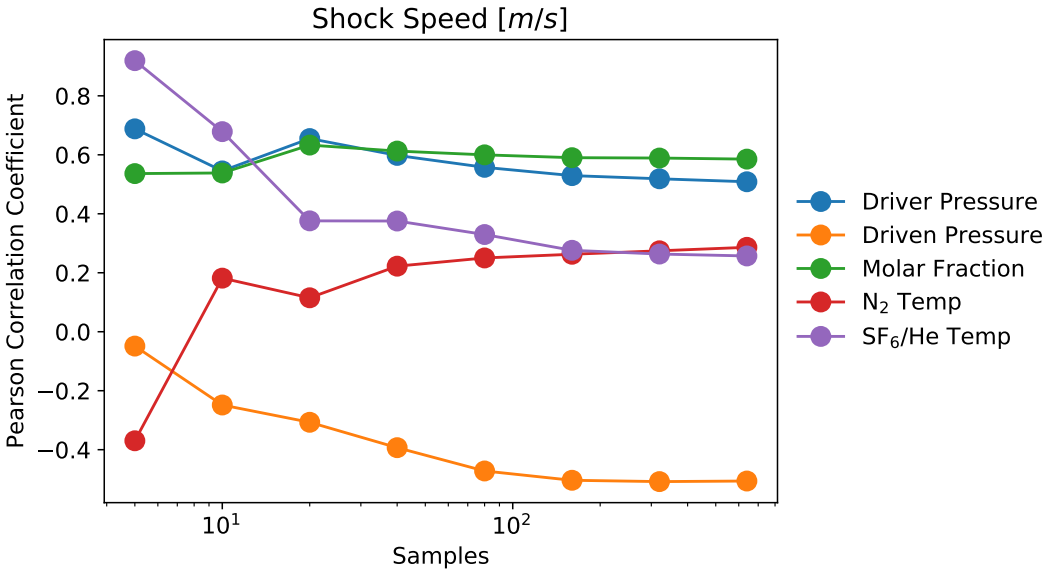


(b) Dalton EOS

Figure 3.6: Infinity Norm & Mean Absolute Error for all incremental sample sets.

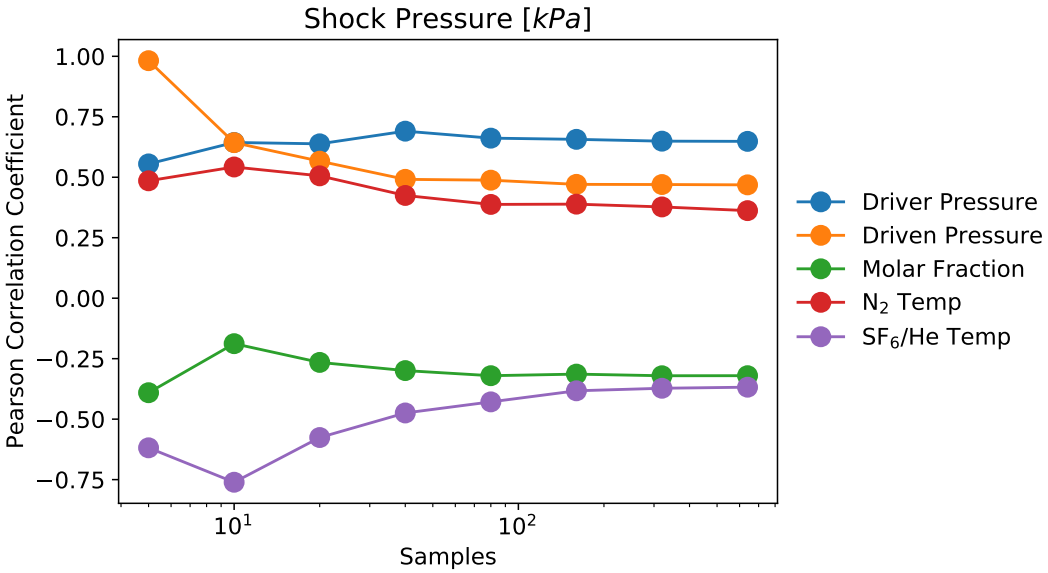


(a) Amagat EOS

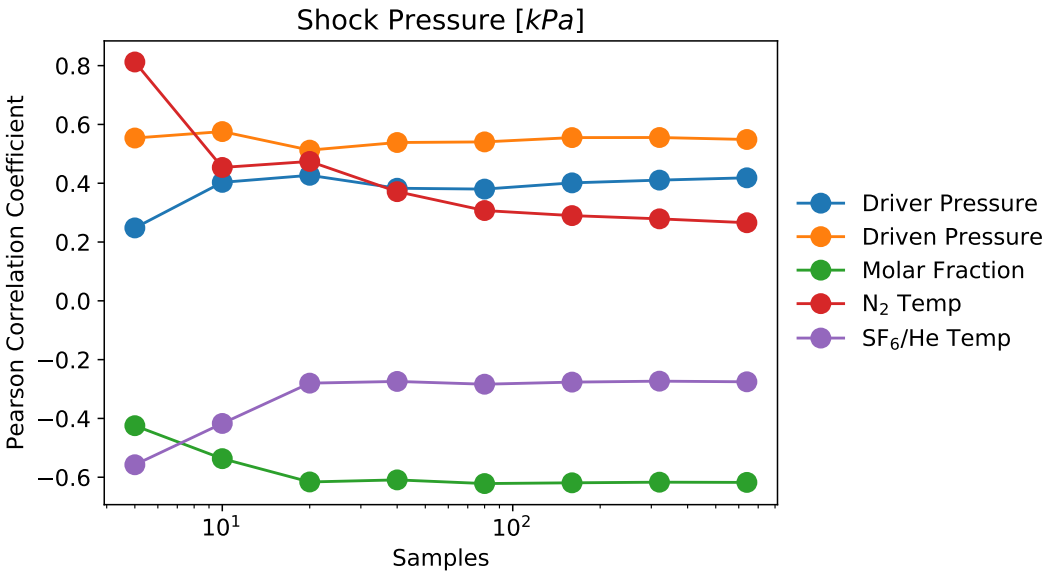


(b) Dalton EOS

Figure 3.7: PCC correlation between inputs and post-shock speed.

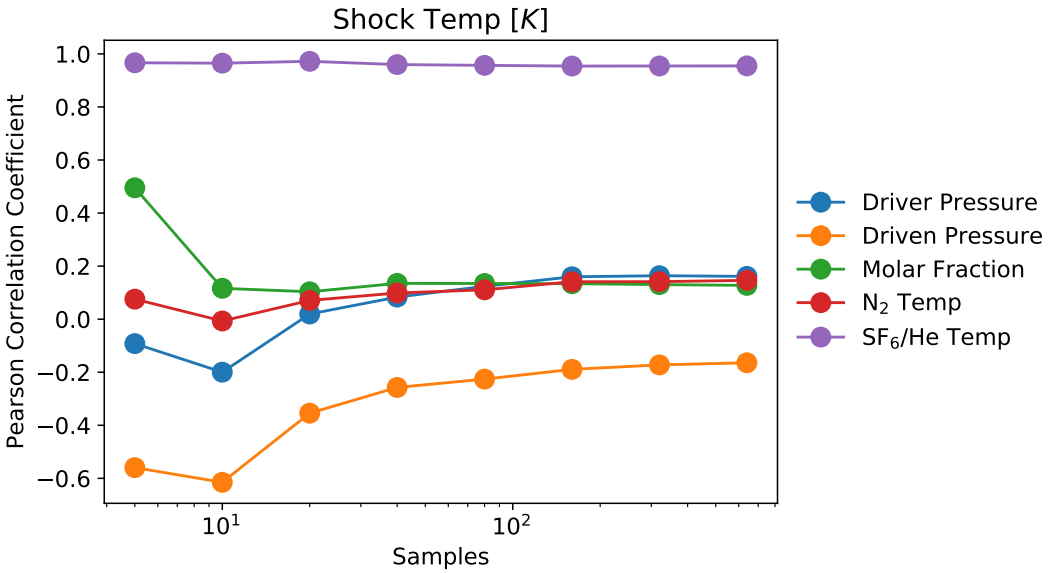


(a) Amagat EOS

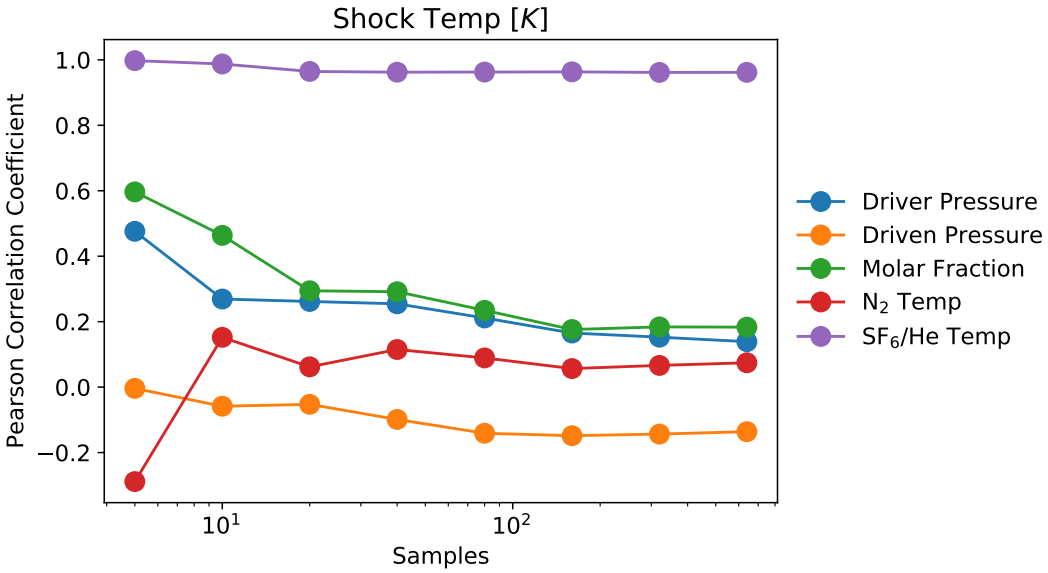


(b) Dalton EOS

Figure 3.8: PCC correlation between inputs and post-shock pressure.



(a) Amagat EOS



(b) Dalton EOS

Figure 3.9: PCC correlation between inputs and post-shock temperature.

Chapter 4

Conclusions & Future Work

It is finished. — Jesus Christ

4.1 Conclusions

The main conclusion from comparison between experimental and analytical data is that the ideal gas laws used to predict the post-shock properties of the gas mixtures do not contain the necessary physics to account for the non-equilibrium effects of the shock. The results in Figures 3.1 to 3.3 from the numerical simulations confirm that there indeed is model discrepancy. An explanation using KMT has been proposed in literature which correlates the disparity of the gases to the discrepancy in the post-shock properties. Moreover, this discrepancy is too prominent to be accounted for solely by using, for example, van der Waals equations for component gases instead of ideal gas equations [5].

Incremental Latin Hypercube Sampling revealed that the model and equations of state are most sensitive to the initial temperature of the mixture.

4.2 Future Work

As with any research project of respectable size, there is no end to the advancements which could be made. This thesis could be expanded upon in a number of ways:

- Consider other mixture ratios.

- Study other highly disparate gaseous mixtures.
- Develop a correction parameter based on KMT and include it in the Amagat and Dalton EOS.
- Additional experiments will be necessary to drive EOS development.

The list above is not all-inclusive, but merely serves to provide a future researcher a few possible directions to pursue.

Appendices

Appendix A

Python Scripts

A.1 Driver

```
#!/usr/bin/env python3

import subprocess as sp
import os

import tiger
import bcat
import cth

{%
setfmt('%0.4e')
R = 8.314e7
mol_wgt_he_sf6 = (mol_frac_he*4.002602
                  + (1-mol_frac_he)*146.055)
density_he_sf6 = driven_pressure*mol_wgt_he_sf6/(R*295)
density_n2 = driver_pressure*28.0134/(R*295)
int_ratio = list(mol_frac_he.as_integer_ratio())
int_ratio[1] -= int_ratio[0]
end
%}


def run(pv_max, pv_min, pv_type, mol_frac_he, bcat_mixed, bcat_input, sesame,
        driver_density, driver_pressure, driven_density, driven_pressure, title,
        eosnum, t_max=1500, t_min=180, t_points=224, pv_points=484,
        tiger_he=None, tiger_sf6=None, tiger_mixed=None, lib_he='lib, jczs2',
        lib_sf6='lib, jczs2', lib_mixed='lib, jczs2', cho_he=None, cho_sf6=None,
        cho_mixed=None, com_he=None, com_sf6=None, com_mixed=None, geos_he=None,
        geos_sf6=None, geos_mixed=None, scale=False, bcat_he=None,
        bcat_sf6=None, amagat=False, dalton=False, power_law=False, eshift=0,
        cth_input='./cth.in', _3d=False, n2_temp=295, mix_temp=295):
    """This puts all of the functions in tiger.py, bcat.py and cth.py together.
       First, tiger is used to generate tables of thermodynamic states, either
       two pure tables of Helium and SF6 which are then mixed according to
```

```
35          Amagat or Dalton's law, or a single table which tiger mixed. Next the
36          mixed tables are fed into BCAT to create a Sesame table for CTH. Next a
37          CTH input deck is written.
38
39      Parameters
40      ----------
41      pv_max : float
42          The max pressure/volume of the isolines generated.
43      pv_min : float
44          The min pressure/volume of the isolines generated.
45      pv_type : str
46          String to denote what variable the pv values are.
47          Either 'v' for volume or 'P' for pressure.
48      mol_frac_he : float
49          The percentage of helium in the mixture.
50      bcat_mixed : str
51          File name of mixed table.
52      bcat_input : str
53          The file name of the bcat input deck.
54      sesame : str
55          The file name of the sesame table.
56      driver_density : float
57          The density of the driver gas.
58      driver_pressure : float
59          The pressure of the driver gas.
60      driven_density : float
61          The density of the driven mixture.
62      driven_pressure : float
63          The pressure of the driven mixture.
64      title : str
65          The title of the CTH run.
66      eosnum : int
67          The sesame eos number.
68      t_max : float
69          The max temperature of the isolines generated.
70      t_min : float
71          The min temperature of the isolines generated.
72      t_points : int
73          The number of isolines to generate.
74      pv_points : int
75          The number of pressure/volume points to generate.
76      tiger_he : str
77          The name of the helium tiger input deck to be written.
78      tiger_sf6 : str
79          The name of the sf6 tiger input deck to be written.
80      tiger_mixed : str
81          The name of the mixed tiger input deck to be written.
82      lib : str
83          The desired 'lib' command in tiger.
84      cho : str
85          The desired 'cho' command in tiger.
86      com : str
87          The desired 'com' command in tiger.
88      geos : str
```

```
89          The desired 'geos' command in tiger.
90      scale : bool, optional
91          Flag to scale helium specific volumes. This is needed
92          for the Dalton law as Dalton must sum over volumes and
93          not specific volumes. Thus the helium volumes is scaled
94          by mass fraction from the sf6 specific volumes.
95      bcat_he : str
96          File name of pure helium table.
97      bcat_sf6 : str
98          File name of pure sf6 table.
99      dalton : bool
100         Flag to use dalton mixing laws.
101     amagat : bool
102         Flag to use amagat mixing laws.
103     power_law : bool
104         Flag to iterpolate genereate consistent specific
105         volume and interpolate new pressures. Only needed
106         when (T,P) points are specified in Tiger rather
107         than (v,T) points.
108     eshift : float
109         The density at the reference state (ambient temp).
110     cth_input : str
111         The cth input deck file name.
112     _3d : bool
113         Flag to specify 3D simulation.
114     n2_temp : real
115         Intitial temperature [K] of N2 in CTH.
116     mix_temp : real
117         Intitial temperature [K] of He_SF6 mixture in CTH.
118     """
119     def run_tiger(tiger_input, t_max, t_min, t_points, pv_max, pv_min,
120                   pv_points, pv_type, mol_frac_he, com, cho, geos, lib,
121                   bcat_input, scale):
122         tiger.write_tiger(tiger_input, t_max, t_min, t_points, pv_max,
123                           pv_min, pv_points, pv_type, mol_frac_he, com,
124                           cho, geos, lib, scale)
125         shcmd = sp.Popen('/home/cwhite3/bin/tiger',
126                          stdin=open(tiger_input),
127                          stdout=open('/dev/null', 'w'),
128                          stderr=open(tiger_input + '.log', 'w'))
129         shcmd.wait()
130         os.rename('./tiger.plt', bcat_input)
131     if tiger_he is not None:
132         run_tiger(tiger_he, t_max, t_min, t_points, pv_max, pv_min, pv_points,
133                   pv_type, mol_frac_he, com_he, cho_he, geos_he, lib_he,
134                   bcat_he, scale)
135         run_tiger(tiger_sf6, t_max, t_min, t_points, pv_max, pv_min, pv_points,
136                   pv_type, mol_frac_he, com_sf6, cho_sf6, geos_sf6, lib_sf6,
137                   bcat_sf6, scale=False)
138     if tiger_he is None:
139         run_tiger(tiger_mixed, t_max, t_min, t_points, pv_max, pv_min,
140                   pv_points, pv_type, mol_frac_he,
141                   com_mixed, cho_mixed, geos_mixed, lib_mixed, bcat_mixed,
142                   scale)
```

```
143
144     dens = bcat.write_cheetah(bcat_mixed, mol_frac_he, pv_points, t_points,
145                               bcat_he, bcat_sf6, dalton, amagat, power_law)
146     bcat.write_bcat(bcat_input, bcat_mixed, pv_points, t_points, eshift,
147                     {{mol_wgt_he_sf6}}, 1.01325e-4, dens, eosnum, sesame)
148     shcmd = sp.Popen('bcat',
149                      stdin=open(bcat_input),
150                      stdout=open(bcat_input + '.log', 'w'))
151     shcmd.wait()
152     cth.write_cth(cth_input, sesame + '{}'.format(eosnum), title,
153                   eosnum, driver_density, driver_pressure,
154                   driven_density, driven_pressure, _3d, n2_temp, mix_temp)
155
156
157 if __name__ == '__main__':
158     if '{{gas_law}}' == 'Dalton':
159         run(6005, 5, 'v', {{mol_frac_he}}, './he_sf6_dalton.plt',
160             './bcat_dalton.i', 'dalton', {{density_n2}},
161             {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
162             'Dalton EOS', 9000, tiger_he='./tiger_dalton_he.i',
163             tiger_sf6='./tiger_dalton_sf6.i', cho_sf6='cho, sf6, f',
164             com_he='com, helium, 1, mole', com_sf6='com, sf6, 1, mole',
165             scale=True, bcat_he='./he_dalton.plt', bcat_sf6='./sf6_dalton.plt',
166             dalton=True, _3d={{_3d}})
167     elif '{{gas_law}}' == 'Amagat':
168         run(363.1, .0328, 'P', {{mol_frac_he}}, './he_sf6_amagat.plt',
169             './bcat_amagat.i', 'amagat', {{density_n2}},
170             {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
171             'Amagat EOS', 9001, tiger_he='./tiger_amagat_he.i',
172             tiger_sf6='./tiger_amagat_sf6.i', cho_sf6='cho, sf6, f',
173             com_he='com, helium, 1, mole', com_sf6='com, sf6, 1, mole',
174             bcat_he='./he_amagat.plt', bcat_sf6='./sf6_amagat.plt',
175             amagat=True, power_law=True, _3d={{_3d}})
176     elif '{{gas_law}}' == 'Ideal':
177         run(6005, 5, 'v', {{mol_frac_he}}, './ideal.plt',
178             './bcat_ideal.i', 'ideal', {{density_n2}},
179             {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
180             'Ideal EOS', 9002, tiger_mixed='./tiger_ideal.i',
181             cho_mixed='cho, sf6, he, f', geos_mixed='geos, ideal',
182             com_mixed='com, helium, {:.0f}, sf6, {:.0f}, mole'
183                       .format(*{{int_ratio}}), _3d={{_3d}})
184     elif '{{gas_law}}' == 'BKW':
185         run(6005, 5, 'v', {{mol_frac_he}}, './bkw.plt',
186             './bcat_bkw.i', 'bkw', {{density_n2}},
187             {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
188             'BKW EOS', 9003, tiger_mixed='./tiger_bkw.i',
189             cho_mixed='cho, sf6, he, f', geos_mixed='geos, bkw',
190             com_mixed='com, helium, {:.0f}, sf6, {:.0f}, mole'
191                       .format(*{{int_ratio}}), _3d={{_3d}})
192     elif '{{gas_law}}' == 'JCZ3':
193         run(6005, 5, 'v', {{mol_frac_he}}, './jcz3.plt',
194             './bcat_jcz3.i', 'jcz3', {{density_n2}},
195             {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
196             'JCZ3 EOS', 9004, tiger_mixed='./tiger_jcz3.i',
```

```
            cho_mixed='cho, sf6, he, f', geos_mixed='geos, jcz3',
            com_mixed='com, helium, {:.0f}, sf6, {:.0f}, mole'
                      .format(*{{int_ratio}}), _3d={{_3d}})
    elif '{{gas_law}}' == 'EXP6':
        run(6005, 5, 'v', {{mol_frac_he}}, './exp6.plt',
            './bcat_exp6.i', 'exp6', {{density_n2}},
            {{driver_pressure}}, {{density_he_sf6}}, {{driven_pressure}},
            'EXP6 EOS', 9005, tiger_mixed='./tiger_exp6.i',
            cho_mixed='cho, sf6, he, f', geos_mixed='geos, exp6',
            com_mixed='com, helium, {:.0f}, sf6, {:.0f}, mole'
                      .format(*{{int_ratio}}), _3d={{_3d}})

```

A.2 Tiger

```
#!/usr/bin/env python3

import numpy as np


def write_tiger(file_name, t_max, t_min, t_points, pv_max, pv_min, pv_points,
                pv_type, mol_frac_he, com, cho=None, geos=None, lib='lib jczs',
                scale=False):
    """Generates thermodynamic states for either a pure gas or a mixture using
       (T,P) or (v, T) states. Writes the tiger input deck.

    Parameters
    ----------
    file_name : str
        The name of the input deck to be written.
    t_max : float
        The max temperature of the isolines generated.
    t_min : float
        The min temperature of the isolines generated.
    t_points : int
        The number of isolines to generate.
    pv_max : float
        The max pressure/volume of the isolines generated.
    pv_min : float
        The min pressure/volume of the isolines generated.
    pv_points : int
        The number of pressure/volume points to generate.
    pv_type : str
        String to denote what variable the pv values are.
        Either 'v' for volume or 'P' for pressure.
    mol_frac_he : float
        The percentage of helium in the mixture.
    com : str
        The desired 'com' command in tiger.
    cho : str
```

```
        The desired 'cho' command in tiger.
    geos : str
        The desired 'geos' command in tiger.
    lib : str
        The desired 'lib' command in tiger.
    scale : bool, optional
        Flag to scale helium specific volumes. This is needed
        for the Dalton law as Dalton must sum over volumes and
        not specific volumes. Thus the helium volumes is scaled
        by mass from the sf6 specific volumes.
    """
    mol_mass_he = 4.002602
    mol_mass_sf6 = 146.055
    mass_he = mol_mass_he*mol_frac_he
    mass_sf6 = mol_mass_sf6*(1 - mol_frac_he)
    t = np.linspace(t_min, t_max, num=t_points, endpoint=True)[::-1]

    if pv_type == 'v':
        pv = np.asarray([pv_min, pv_max])
    else:
        pv = np.asarray([pv_max, pv_min])

    if scale:
        pv = pv*mass_sf6/mass_he
    lines = []
    for temp1, temp2 in zip(t[:-1:2], t[1::2]):
        lines.append('iso, T, {:.4f}, {}, {}, {}, {}, log\n'
                     .format(temp1, pv_type, pv[0], pv_points-1, pv[1]))
        lines.append('iso, T, {:.4f}, {}, {}, {}, {}, log\n'
                     .format(temp2, pv_type, pv[1], pv_points-1, pv[0]))
    with open(file_name, 'w') as f:
        f.write('{}\n'.format(lib))
        if cho is not None:
            f.write('{}\n'.format(cho))
        f.write('{}\n'.format(com))
        if geos is not None:
            f.write('{}\n'.format(geos))
        f.write('poi, T, 1000, P, 3000\n')
        f.write('plt, T, V, P, E, S\n')
        f.writelines(lines)
        f.write('stop')
```

A.3 BCAT

```
#!/usr/bin/env python3

import numpy as np
import pandas as pd
from scipy.interpolate import griddata
import scipy as scp
```

```
7  import datetime
8
9  import matplotlib.pyplot as plt
10 from matplotlib import cm
11 import matplotlib.ticker as mticker
12 from mpl_toolkits.mplot3d import Axes3D
13 plt.switch_backend('agg')
14
15
16 def write_cheetah(file_name_mixed, mol_frac_he, pv_points, t_points,
17                   file_name_he=None, file_name_sf6=None, dalton=False,
18                   amagat=False, power_law=False):
19     """Mixes TIGER tables according to different laws and writes the new
20        mixture to a table for BCAT. Also queries the material surface for
21        density at the reference state
22
23     Parameters
24     ----------
25     file_name_mixed : str
26         File name of mixed table.
27     mol_frac_he : float
28         The percentage of helium in the mixture.
29     pv_points : int
30         The number of pressure/volume points.
31     t_points : int
32         The number of temperature points.
33     file_name_he : str
34         File name of pure helium table.
35     file_name_sf6 : str
36         File name of pure sf6 table.
37     dalton : bool
38         Flag to use dalton mixing laws.
39     amagat : bool
40         Flag to use amagat mixing laws.
41     power_law : bool
42         Flag to iterpolate genereate consistent specific
43         volume and interpolate new pressures.
44
45     Returns
46     -------
47     dens : float
48         The density at reference state (ambient temp).
49     """
50     mol_mass_he = 4.002602
51     mol_mass_sf6 = 146.055
52     mass_he = mol_mass_he*mol_frac_he
53     mass_sf6 = mol_mass_sf6*(1 - mol_frac_he)
54     mass_frac_sf6 = mass_sf6/(mass_he + mass_sf6)
55     mass_frac_he = mass_he/(mass_he + mass_sf6)
56     if file_name_he is not None:
57         he = pd.read_table(file_name_he, delim_whitespace=True,
58                            skiprows=2, names=['T', 'v', 'p', 'e', 's'])
59     if file_name_sf6 is not None:
60         sf6 = pd.read_table(file_name_sf6, delim_whitespace=True,
```

```
                            skiprows=2, names=['T', 'v', 'p', 'e', 's'])
    if file_name_he is None and file_name_sf6 is None:
        data = pd.read_table(file_name_mixed, delim_whitespace=True,
                             skiprows=2, names=['T', 'v', 'p', 'e', 's'])
    if dalton:
        data = pd.DataFrame()
        data['T'] = he['T']
        data['v'] = he['v']*mass_frac_he
        data['p'] = he['p'] + sf6['p']
        data['e'] = he['e']*mass_frac_he + sf6['e']*mass_frac_sf6
        data['s'] = he['s']*mass_frac_he + sf6['s']*mass_frac_sf6

    if amagat:
        data = pd.DataFrame()
        data['T'] = he['T']
        data['v'] = he['v']*mass_frac_he + sf6['v']*mass_frac_sf6
        data['p'] = he['p']
        data['e'] = he['e']*mass_frac_he + sf6['e']*mass_frac_sf6
        data['s'] = he['s']*mass_frac_he + sf6['s']*mass_frac_sf6

    create_plots('./surface.pdf', pv_points, t_points, data)

    if power_law:
        data['v'], data['p'] = power_law_interp(data['v'], data['p'], data['T'],
                                                pv_points, t_points,
                                                xlabel='Specific Volume',
                                                ylabel='Pressure',
                                                )
    with open(file_name_mixed, 'w') as f:
        f.write('variables = T_K v_cc/g p_atm e_cal/g s_cal/(g-k)\n')
        f.writelines(['   {:.3e}    {:.3e}    {:.3e}    {:.3e}    {:.3e}\n'
                      .format(*data.values[i, :])
                     for i in range(len(data.values[:,0]))])
    v = griddata((data['T'], data['p']), data['v'], (298, 1), method='linear')
    return 1/v


def power_law_interp(x_old, y_old, t, x_points, t_points, linear=False,
                     plot=False, xlabel='x', ylabel='y'):
    """Power law interpolation function. BCAT expects the volumes to be
       constant but in the case of Amagat the volumes are calculated and
       not specified. Thus the volumes values are not equal across all the
       isotherms. This is used to create a constant discretization of the
       x array and then generates the respective y array via interpolation.

    Parameters
    ----------
    x_old : 1D Pandas DataFrame
        Values to be discretized consistently.
    y_old : 1D Pandas DataFrame
        Y values matching the x_old input.
    t : 1D Pandas DataFrame
        Temperatures array.
    x_points : int
```

```
115         Number of x points.
116     t_points : int
117         Number of isotherms.
118     linear : bool, optional
119         Flag to discretize x values in a linear or logrithmic fashion.
120     plot : bool, optional
121         Flag to generate plots of the old vs. new values for the
122         isotherms.
123     xlabel : str, optional
124         Label of the x-axis for the plot command.
125     ylabel : str, optional
126         Label of the y-axis for the plot command.
127
128     Returns
129     -------
130     x : 1D Pandas DataFrame
131         Newly discretized x values.
132     y : 1D Pandas DataFrame
133         Interpolated y values.
134     """
135     x_max = np.max(x_old)
136     x_min = np.min(x_old)
137     x = np.zeros((x_points, t_points))
138     if linear:
139         x[:, 0] = np.linspace(x_min, x_max, x_points, endpoint=True)
140     else:
141         x[:, 0] = np.geomspace(x_min, x_max, x_points, endpoint=True)
142     for i in range(t_points)[::2]:
143         x[:, i] = x[:, 0].copy()
144         x[:, i+1] = x[::-1, i].copy()
145     x_old = np.asarray(x_old).reshape((x_points, t_points), order='F')
146     y = np.asarray(y_old).reshape((x_points, t_points), order='F')
147     y_old = y.copy()
148     b = np.ones((x_points, 2))
149     for i in range(t_points):
150         b[:, -1] = np.log(x_old[:, i])
151         coefficients = np.linalg.pinv(b)@np.log(y[:, i])
152         y[:, i] = np.exp(coefficients[0])*x[:, i]**coefficients[-1]
153         if plot:
154             plt.figure()
155             plt.plot(x[:, i], y[:, i], label="New", marker='o', linestyle='None',
156                      markerfacecolor='none')
157             plt.plot(x_old[:, i], y_old[:, i], label="Old", marker='o',
158                      linestyle='None', markerfacecolor='none')
159             plt.xscale('log')
160             plt.yscale('log')
161             plt.xlabel(xlabel)
162             plt.ylabel(ylabel)
163             plt.legend()
164             plt.title('T = {}'.format(t[i*x_points+1]))
165             plt.savefig('figs/interp{}.pdf'.format(i))
166             plt.close('all')
167     x = x.reshape(x_points*t_points, order='F')
168     y = y.reshape(x_points*t_points, order='F')
```

```
169     return x, y
170
171
172 def create_plots(surface_plt, pv_points, t_points, df):
173     """Creates various plots of material tables.
174
175     Parameters
176     ----------
177     surface_plt : str
178         Name of the surface plot.
179     pv_points : int
180         The number of pressure/volume points.
181     t_points : int
182         The number of isolines.
183     df : Pandas DataFrame
184         Dataframe of a mixed he_sf6 table.
185     """
186     def log_tick(val, pos=None):
187         """Custom function for FuncFormatter which creates log tick labels
188            on a 3D plot. Log scale on 3D plots in python have been broken
189            since 2011. This is a simple work around.
190         """
191         return r'$10^{' + '{:.0f}'.format(val) + '}$'
192
193     T = np.asarray(df['T']).reshape((pv_points, t_points), order='F')
194     p = np.asarray(df['p']).reshape((pv_points, t_points), order='F')
195     v = np.asarray(df['v']).reshape((pv_points, t_points), order='F')
196
197     fig = plt.figure()
198     ax = fig.gca(projection='3d')
199     ax.plot_surface(np.log10(v), p, T)
200     ax.xaxis.set_major_formatter(mticker.FuncFormatter(log_tick))
201     ax.set_xlabel('Density (g/cc)')
202     ax.set_ylabel('Pressure (Pa)')
203     ax.set_zlabel('Temperature (K)')
204     plt.savefig(surface_plt)
205     plt.close()
206
207
208
209 def write_bcat(file_name, tiger_table, pv_points, t_points, eshift, fw, pres,
210                dens, eosnum, file_name_sesame):
211     """Write the bcat input deck.
212
213     Parameters
214     ----------
215     file_name : str
216         The file name of the bcat input deck.
217     tiger_table : str
218         The file name of the tiger plt table.
219     pv_points : int
220         The number of pressure/volume points.
221     t_points : int
222         The number of isolines.
```

```
    eshift : float
        The energy shift.
    fw : int
        The formula weight.
    dens : float
        The density at the reference state (ambient temp).
    eosnum : int
        The sesame material number.
    file_name_sesame : str
        The file name of the sesame table.
    """
    with open(file_name, 'w') as f:
        f.writelines(['\n',
                      'mod use\n',
                      '{}\n'.format(tiger_table),
                      '{}, {}, {}\n'.format(pv_points, t_points, eshift),
                      'slib use\n',
                      '201\n',
                      '36, {:.3f}, {:.4e}, {:d}, {:.4e}\n'
                      .format(fw, pres, 298, dens),
                      '301\n',
                      'no\n',
                      '\n',
                      '{}, {}, {}, {}\n'
                      .format(eosnum, datetime.date.today().strftime("%m%d%y"),
                              file_name_sesame + str(eosnum), 'trash'),
                      'quit'])
```

A.4 CTH

```
import numpy as np

def write_cth(file_name, file_name_sesame, title, eosnum, driver_density,
              driver_pressure, driven_density, driven_pressure, _3d,
              n2_temp, mix_temp):
    """Writes the CTH input deck.

    Parameters
    ----------
    file_name : str
        The cth input deck file name.
    file_name_sesame : str
        The file name of the sesame table.
    title : str
        The title of the CTH run.
    eosnum : int
        The sesame eos number.
    driver_density : float
        The density of the driver gas.
    driver_pressure : float
```

```
        The pressure of the driver gas.
    driven_density : float
        The density of the driven mixture.
    driven_pressure : float
        The pressure of the driven mixture.
    _3d : bool
        Flag to specify 3D simulation.
    n2_temp : real
        Intitial temperature [K] of N2 in CTH.
    mix_temp : real
        Intitial temperature [K] of He_SF6 mixture in CTH.
    """
    with open(file_name, 'w') as f:
        f.writelines(['*eor* cthin\n',
                      '{}\n\n'.format(title)])
        f.writelines(['control\n',
                      'mmp3\n',
                      'matcs = 1\n',
                      '\tprint\n',
                      'endcontrol\n\n'])
        f.writelines(['mesh\n'])
        if _3d:
            f.writelines(['\tgeometry = 3dr\n'])
        else:
            f.writelines(['\tgeometry = 2dr\n'])
        f.writelines(['\t\tx0 = -171.0\n',
                      '\t\t\tx1 n=516 w=491.0 r=1.0\n',
                      '\t\tendx\n',
                      '\t\ty0 = 0.0\n',
                      '\t\t\ty1 n=8 w=7.62 r=1.0\n',
                      '\t\tendy\n'])
        if _3d:
            f.writelines(['\t\tz0 = 0.0\n',
                          '\t\t\tz1 n=8 w=7.62 r=1.0\n',
                          '\t\tendz\n'])
        f.writelines(['endm\n\n'])
        f.writelines(['eos ses=500000\n',
                      '\tmat1 ses user eos={} feos=\'{}\'\n'
                      .format(eosnum, file_name_sesame),
                      '\tmat2 idg n2\n',
                      'endeos\n\n'])
        f.writelines(['diatom\n',
                      '\tpackage \'driver\'\n',
                      '\t\tmaterial 2\n',
                      '\t\tpressure {:.5e}\n'.format(driver_pressure),
                      '\t\ttemperature {:.5e}\n'.format(n2_temp/11604.5),
                      '\t\tinsert box\n'])
        if _3d:
            f.writelines(['\t\t\tp1 = -171, 0, 0\n',
                          '\t\t\tp2 = 0, 7.62, 7.62\n'])
        else:
            f.writelines(['\t\t\tp1 = -171, 0\n',
                          '\t\t\tp2 = 0, 7.62\n'])
        f.writelines(['\t\tendi\n',
```

```
                    '\tendp\n',
                    '\tpackage \'driven\'\n',
                    '\t\tmaterial 1\n',
                    '\t\tpressure {:.5e}\n'.format(driven_pressure),
                    '\t\ttemperature {:.5e}\n'.format(mix_temp/11604.5),
                    '\t\tinsert box\n'])
        if _3d:
            f.writelines(['\t\t\tp1 = 0, 0, 0\n',
                          '\t\t\tp2 = 320, 7.62, 7.62\n'])
        else:
            f.writelines(['\t\t\tp1 = 0, 0\n',
                          '\t\t\tp2 = 320, 7.62\n'])
        f.writelines(['\t\tendi\n',
                      '\tendp\n',
                      'enddiatom\n\n'])
        f.writelines(['epdata\n',
                      'ende\n\n'])
        if _3d:
            f.writelines(['tracer\n',
                          '\tadd -022.00, 3.81, 3.81 fixed=xyz\n',
                          '\tadd 039.985, 7.62, 3.81 fixed=xyz\n',
                          '\tadd 111.105, 7.62, 3.81 fixed=xyz\n',
                          '\tadd 208.895, 7.62, 3.81 fixed=xyz\n',
                          '\tadd 280.015, 7.62, 3.81 fixed=xyz\n',
                          '\tadd 039.985, 3.81, 3.81 fixed=xyz\n',
                          '\tadd 111.105, 3.81, 3.81 fixed=xyz\n',
                          '\tadd 208.895, 3.81, 3.81 fixed=xyz\n',
                          '\tadd 280.015, 3.81, 3.81 fixed=xyz\n',
                          '\tadd 000.000, 7.62, 3.81\n',
                          '\tadd 000.000, 3.81, 3.81\n',
                          '\tadd 320.000, 3.81, 3.81\n',
                          'endt\n\n'])
        else:
            f.writelines(['tracer\n',
                          '\tadd -022.00, 3.81 fixed=xy\n',
                          '\tadd 039.985, 7.62 fixed=xy\n',
                          '\tadd 111.105, 7.62 fixed=xy\n',
                          '\tadd 208.895, 7.62 fixed=xy\n',
                          '\tadd 280.015, 7.62 fixed=xy\n',
                          '\tadd 039.985, 3.81 fixed=xy\n',
                          '\tadd 111.105, 3.81 fixed=xy\n',
                          '\tadd 208.895, 3.81 fixed=xy\n',
                          '\tadd 280.015, 3.81 fixed=xy\n',
                          '\tadd 000.000, 7.62\n',
                          '\tadd 000.000, 3.81\n',
                          '\tadd 320.000, 3.81\n',
                          'endt\n\n'])
        f.writelines(['convct\n',
                      '\tinterface=smyra\n',
                      'endc\n\n'])
        f.writelines(['edit\n',
                      '\tshortc\n',
                      '\t\tcy 0 dc 100\n',
                      '\tends\n',
```

```
                    'ende\n\n'])
        f.writelines(['boundary\n',
                    '\tbhydro\n',
                    '\t\tbxb=0 bxt=0\n',
                    '\t\tbyb=0 byt=0\n'])
        if _3d:
            f.writelines(['\t\tbzb=0 bzt=0\n'])
        f.writelines(['\tendh\n',
                    'endb\n\n'])
        f.writelines(['mindt\n',
                    '\ttime 0 dtmin 1e-9\n',
                    'endm\n\n'])
        f.writelines(['spy\n',
                    '\tSave("M,VOLM,VX,VY,VMAG,P,PM,T,TM,DENS,DENSM,CS,CSM");',
                    '\n',
                    '\tSaveTime(0.0,5e-5);\n',
                    '\tPlotTime(0.0,5e-5);\n',
                    '\n',
                    '\tImageFormatHD();\n',
                    '\n',
                    '\tdefine main(){\n',
                    '\t\tpprintf(" PLOT: Cycle=%d, Time=%e\\n",CYCLE,TIME);',
                    '\n',
                    '\t\tXLimits(-171,320);\n',
                    '\t\tYLimits(0,7.62);\n'])
        if _3d:
            f.writelines(['\t\tZLimits(0,7.62);\n'])
        if _3d:
            f.writelines(['\t\tHotMap1;\n',
                        '\n',
                        '\t\tImage("figs/Pressure");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(1e5,2e7,LOG_MAP);\n',
                        '\t\t\tLabel(sprintf("Pressure at %0.2e s.",TIME));',
                        '\n',
                        '\t\t\tPaint3DMats("P");\n',
                        '\t\t\tDrawColorMap("(dyn/cm^2^)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Velocity");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(1e0,1e6,LOG_MAP);\n',
                        '\t\t\tLabel(sprintf("Velocity at %0.2e s.",TIME));',
                        '\n',
                        '\t\t\tPaint3DMats("VMAG");\n',
                        '\t\t\tDrawColorMap("(cm/s)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Temperature");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(0,.05);\n',
                        '\t\t\tLabel(sprintf("Temperature at %0.2e s.",TIME))'
                              + ';\n',
                        '\t\t\tPaint3DMats("T");\n',
```

```
                        '\t\t\tDrawColorMap("(eV)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Density");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(0,1e-2);\n',
                        '\t\t\tLabel(sprintf("Density at %0.2e s.",TIME));\n',
                        '\t\t\tPaint3DMats("DENS");\n',
                        '\t\t\tDrawColorMap("(g/cc)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Mats");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tLabel(sprintf("Mats at %0.2e s.",TIME));\n',
                        '\t\t\tMatColors(GREEN,BLUE);\n',
                        '\t\t\tPlot3DMats;\n',
                        '\t\tEndImage;\n',
                        '\t}\n'])
    else:
        f.writelines(['\t\tHotMap1;\n',
                        '\n',
                        '\t\tImage("figs/Pressure");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(1e5,2e7,LOG_MAP);\n',
                        '\t\t\tLabel(sprintf("Pressure at %0.2e s.",TIME));',
                        '\n',
                        '\t\t\tPlot2D("P");\n',
                        '\t\t\tDraw2DMatContour;\n',
                        '\t\t\tDraw2DTracers;\n',
                        '\t\t\tDrawColorMap("(dyn/cm^2^)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Velocity");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(1e0,1e6,LOG_MAP);\n',
                        '\t\t\tLabel(sprintf("Velocity at %0.2e s.",TIME));',
                        '\n',
                        '\t\t\tPlot2D("VMAG");\n',
                        '\t\t\tDraw2DMatContour;\n',
                        '\t\t\tDraw2DTracers;\n',
                        '\t\t\tDrawColorMap("(cm/s)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Temperature");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(0,.05);\n',
                        '\t\t\tLabel(sprintf("Temperature at %0.2e s.",TIME))'
                              + ';\n',
                        '\t\t\tPlot2D("T");\n',
                        '\t\t\tDraw2DMatContour;\n',
                        '\t\t\tDraw2DTracers;\n',
                        '\t\t\tDrawColorMap("(eV)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
```

```
                        '\t\tImage("figs/Density");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tColorMapRange(0,1e-2);\n',
                        '\t\t\tLabel(sprintf("Density at %0.2e s.",TIME));\n',
                        '\t\t\tPlot2D("DENS");\n',
                        '\t\t\tDraw2DMatContour;\n',
                        '\t\t\tDraw2DTracers;\n',
                        '\t\t\tDrawColorMap("(g/cc)",.15,.3,.25,.8);\n',
                        '\t\tEndImage;\n',
                        '\n',
                        '\t\tImage("figs/Mats");\n',
                        '\t\t\tWindow(0,0,1,1);\n',
                        '\t\t\tLabel(sprintf("Mats at %0.2e s.",TIME));\n',
                        '\t\t\tMatColors(GREEN,BLUE);\n',
                        '\t\t\tPlot2DMats;\n',
                        '\t\t\tDraw2DTracers;\n',
                        '\t\tEndImage;\n',
                        '\t}\n'])
    f.writelines(['\n',
                  '\tSaveHis("GLOBAL,VOLM,M,P,PM,T,TM,DENS,DENSM,VX,VY,'
                              + 'VMAG,CVMAG,MAT_GLOBAL,CS,CSM");\n',
                  '\tSaveTracer(ALL);\n',
                  '\tStopTracer("P",12,{:.5e}'.format(driver_pressure*.5),
                  ',"CEIL",1);\n'
                  '\tHisTime(0,1e-6);\n',
                  'endspy\n\n'])
```

References

[1] V. N. Zubarev and G. S. Telegin. "Shock Compressibility of Liquid Nitrogen and Solid Carbon Dioxide". In: *Doklady Akademii Nauk SSSR* 142.2 (1962), pp. 309–312.

[2] M. Ross and F. H. Ree. "Repulsive forces of simple molecules and mixtures at high density and temperature". In: *The Journal of Chemical Physics* 73.12 (1980), pp. 6146–6152. DOI: 10.1063/1.440106.

[3] D. F. Davidson et al. "Shock Tube Measurements of the Equation of State of Argon". In: *International Journal of Thermophysics* 19.6 (1998), pp. 1585–1594. DOI: 10.1007/BF03344910.

[4] D. F. Davidson and R. K. Henson. "Real Gas Corrections in Shock Tube Studies at High Pressures". In: *Israel Journal of Chemistry* 36.3 (1996), pp. 321–326. DOI: 10.1002/ijch.199600044.

[5] P. Wayne et al. "Dalton's and Amagat's laws fail in gas mixtures with shock propagation". In: *Science Advances* 5.12 (2019). DOI: 10.1126/sciadv.aax4749.

[6] Y. A. Çengel and M. A. Boles. *Thermodynamics: An Engineering Approach*. 6th ed. McGraw-Hill, 2008, pp. 137–144, 701–708. ISBN: 978-0071257718.

[7] K. W. Woo and S. I. Yeo. "Dalton's Law vs. Amagat's Law for the Mixture of Real Gases". In: *SNU Journal of Education Research* 5 (1995), pp. 127–134. ISSN: 1225-5335. URL: http://hdl.handle.net/10371/72458.

[8] M. L. Hobbs and M. R. Baer. "Calibrating the BKW-EOS with a large product species data base and measured C-J properties". In: Proceedings - Tenth International Detonation Symposium. Boston, MA, July 1992. SAND92-1931C.

[9] R. D. Cowan and W. Fickett. "Calculation of the Detonation Properties of Solid Explosives with the Kistiakowsky-Wilson Equation of State". In: *The Journal of Chemical Physics* 24.5 (1956), pp. 932–939. DOI: 10.1063/1.1742718.

[10] M. Cowperthwaite and W. H. Zwisler. "The JCZ Equations of State for Detonation Products and Their Incorporation into the TIGER Code". In: Proceedings - Sixth Symposium (International) on Detonation. Vol. 221. ACR. Coronado, CA: Office of Naval Research, June 1976, pp. 162–172.

[11] M. L. Hobbs, R. G. Schmitt, and H. K. Moffat. "JCZS3 — An Improved Database for EOS Calculations". In: Proposed for Proceedings - 16th International Detonation Symposium. Cambridge, MD, July 2018. SAND2018-6389C.

[12] B. C. McGee, M. L. Hobbs, and M. R. Baer. *Exponential 6 parameterization for the JCZ3-EOS*. Sandia Technical Report SAND98-1191. Sandia National Laboratories, July 1998. DOI: 10.2172/639774.

[13] M. R. Baer, M. L. Hobbs, and B. C. McGee. "JCZS: An Intermolecular Potential Database for Performing Accurate Detonation and Expansion Calculations". In: *Propellants, Explosives, Pyrotechnics* 24 (1999), pp. 269–279. SAND98-2452J.

[14] L. E. Fried, W. M. Howard, and P. C. Souers. "EXP6: A New Equation of State Library for High Pressure Thermochemistry". In: Proceedings - 12th International Detonation Symposium. San Diego, CA, Aug. 2002, pp. 567–575.

[15] I. G. Currie. *Fundamental Mechanics of Fluids*. 4th ed. CRC Press, 2013. ISBN: 978-1439874608.

[16] J. M. McGlaun, S. L. Thompson, and M. G. Elrick. "CTH: A Three-Dimensional Shock Wave Physics Code". In: *International Journal of Impact Engineering* 10 (1990), pp. 351–360. DOI: 10.1016/0734-743X(90)90071-3.

[17] J. M. Bigelow. "A Thermodynamic Study of Binary Real Gas Mixtures Undergoing Normal Shocks". Master's Thesis. The University of New Mexico, 2017. URL: https://digitalrepository.unm.edu/me_etds/168.

[18] A. Saltelli et al. *Global Sensitivity Analysis*. Wiley, 2008. ISBN: 978-0470059975.

[19] J. Helton and F. Davis. "Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems". In: *Reliability Engineering & System Safety* 81.1 (2003), pp. 23–69. DOI: 10.1016/S0951-8320(03)00058-9.

[20] B. M. Adams et al. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual*. Sandia Technical Report SAND2014-4633. Sandia National Laboratories, July 2014. Updated May 2019 (Version 6.10).

[21] G. Wyss and K. Jorgensen. *A user's guide to LHS: Sandia's Latin Hypercube Sampling Software*. Sandia Technical Report SAND-98-0210. Sandia National Laboratories, Feb. 1998. DOI: 10.2172/573301.

[22] B. Tang. "Orthogonal Array-Based Latin Hypercubes". In: *Journal of the American Statistical Association* 88.424 (1993), pp. 1392–1397. DOI: 10.2307/2291282.

[23] M. D. McKay, R. J. Beckman, and W. J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from

a Computer Code". In: *Technometrics* 21.2 (1979), pp. 239–245. DOI: 10.2307/1268522.

[24] A. J. Wheeler and A. R. Ganji. *Introduction to Engineering Experimentation*. 3rd ed. Pearson, 2010. ISBN: 978-0131742765.

[25] D. C. Montgomery, G. C. Runger, and N. F. Hubele. *Engineering Statistics*. 3rd ed. Wiley, 2006. ISBN: 978-0471742227.

[26] S. J. Leon. *Linear Algebra with Applications*. 9th ed. Pearson, 2015. ISBN: 978-0321962218.

[27] C. J. Willmott and K. Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". In: *Climate Research* 30.1 (2005), pp. 79–82. DOI: 10.3354/cr030079.

[28] M. Cowperthwaite and W. H. Zwisler. *TIGER Computer Program Documentation*. Stanford Research Institute, Mar. 1974.

[29] M. K. Wong. *CTH User's Manual*. Version 12.1. Sandia National Laboratories, Nov. 2019.

[30] G. I. Kerley. *BCAT User's Manual and Input Instructions*. Version 1.30. Sandia National Laboratories, Mar. 2015.

[31] S. Lyon and J. Johnson. *SESAME: The Los Alamos National Laboratory Equation of State Database*. LANL Technical Report LA-UR-92-3407. Los Alamos National Laboratory, 1992.

[32] J. Bigelow et al. "Mixing-Model Sensitivity to Input Parameter Variation". In: vol. 115. WIT Transactions on Engineering Sciences. June 2017, pp. 85–96. DOI: 10.2495/MPF170101.

[33] P. J. Wayne. "Characterization of single- and multi-phase shock-accelerated flows". Ph. D. Dissertation. The University of New Mexico, 2019. URL: https://digitalrepository.unm.edu/me_etds/170.

[34] R. Penrose. "A generalized inverse for matrices". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3 (1955), pp. 406–413. DOI: 10.1017/S0305004100030401.