MULTIVARIABLE OPTIMIZATION OF LIQUID ROCKET ENGINES
USING PARTICLE SWARM ALGORITHMS

by
DANIEL RAY JONES

DR. JOHN BAKER, COMMITTEE CHAIR
DR. SEMIH M. OLCMEN
DR. PAUL S. RAY

A THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Aerospace Engineering and Mechanics in the Graduate School of The University of Alabama

TUSCALOOSA, ALABAMA
2013
ABSTRACT

Liquid rocket engines are highly reliable, controllable, and efficient compared to other conventional forms of rocket propulsion. As such, they have seen wide use in the space industry and have become the standard propulsion system for launch vehicles, orbit insertion, and orbital maneuvering. Though these systems are well understood, historical optimization techniques are often inadequate due to the highly non-linear nature of the engine performance problem. In this thesis, a Particle Swarm Optimization (PSO) variant was applied to maximize the specific impulse of a finite-area combustion chamber (FAC) equilibrium flow rocket performance model by controlling the engine’s oxidizer-to-fuel ratio and de Laval nozzle expansion and contraction ratios. In addition to the PSO-controlled parameters, engine performance was calculated based on propellant chemistry, combustion chamber pressure, and ambient pressure, which are provided as inputs to the program. The performance code was validated by comparison with NASA’s Chemical Equilibrium with Applications (CEA) and the commercially available Rocket Propulsion Analysis (RPA) tool. Similarly, the PSO algorithm was validated by comparison with brute-force optimization, which calculates all possible solutions and subsequently determines which is the optimum. Particle Swarm Optimization was shown to be an effective optimizer capable of quick and reliable convergence for complex functions of multiple non-linear variables.
DEDICATION

This thesis is dedicated to my dearest friends and family who have supported me throughout life and have enabled me to complete this thesis. In particular, I would like to acknowledge my mother for encouraging my curiosity as a child and making many sacrifices to ensure that I had the opportunity to follow my passions through education.
LIST OF ABBREVIATIONS AND SYMBOLS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>area, $m^2$</td>
</tr>
<tr>
<td>$a$</td>
<td>speed of sound, $m/s$</td>
</tr>
<tr>
<td>$a_i$</td>
<td>thermodynamic curve fit coefficient</td>
</tr>
<tr>
<td>$a_{ij}$</td>
<td>matrix of stoichiometric coefficients</td>
</tr>
<tr>
<td>$b$</td>
<td>matrix of element mole numbers</td>
</tr>
<tr>
<td>$C.F.$</td>
<td>correction control factor, <em>unitless</em></td>
</tr>
<tr>
<td>$C_F$</td>
<td>coefficient of thrust, <em>unitless</em></td>
</tr>
<tr>
<td>$C_P$</td>
<td>molar specific heat at $P = c, J/kmol-K$</td>
</tr>
<tr>
<td>$C_v$</td>
<td>molar specific heat at $V = c, J/kmol-K$</td>
</tr>
<tr>
<td>$c^*$</td>
<td>characteristic velocity, $s$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>particle self-trust parameter, <em>unitless</em></td>
</tr>
<tr>
<td>$c_2$</td>
<td>particle collective trust parameter, <em>unitless</em></td>
</tr>
<tr>
<td>$F$</td>
<td>thrust, $N$</td>
</tr>
<tr>
<td>$G$</td>
<td>Gibbs free energy, $J/kg$</td>
</tr>
<tr>
<td>$g_0$</td>
<td>standard gravitational acceleration, $m/s^2$</td>
</tr>
<tr>
<td>$H$</td>
<td>molar specific enthalpy, $J/kmol$</td>
</tr>
<tr>
<td>$h$</td>
<td>specific enthalpy, $J/kg$</td>
</tr>
<tr>
<td>$I$</td>
<td>total impulse, $N\cdot s$</td>
</tr>
<tr>
<td>$I_{SP}$</td>
<td>specific impulse, $s$</td>
</tr>
<tr>
<td>$n$</td>
<td>total number of moles</td>
</tr>
</tbody>
</table>
\( nEL \)  number of elements
\( nSP \)  number of species
\( M \)  molecular weight, \( \text{kg/kmol} \)
\( Mach \)  Mach number, \textit{unitless}
\( m \)  mass, \( \text{kg} \)
\( \dot{m} \)  mass flow rate, \( \text{kg/s} \)
\( o/f \)  oxidizer-to-fuel mass ratio, \textit{unitless}
\( P \)  static pressure, \( \text{Pa} \)
\( R \)  universal gas constant, \( \text{J/kmol-K} \)
\( r \)  freshly generated uniformly-distributed random number, \textit{unitless}
\( s \)  specific entropy, \( \text{J/kg-K} \)
\( T \)  temperature, \( \text{K} \)
\( t \)  current particle swarm iteration, \textit{unitless}
\( u \)  velocity, \( \text{m/s} \)
\( V \)  particle swarm velocity vector
\( X \)  particle swarm position vector
\( x \)  mole fraction, \textit{unitless}
\( \gamma \)  ratio of specific heats, \textit{unitless}
\( \gamma_s \)  isentropic exponent, \textit{unitless}
\( \eta \)  particle swarm global/local balance parameter, \textit{unitless}
\( \theta \)  nozzle angle, \textit{degrees}
\( \lambda \)  nozzle thrust efficiency, \textit{unitless}
\( \mu \) chemical potential, \( J/kg \)

\( \pi \) Lagrangian multiplier divided by RT

\( \chi \) velocity constriction factor, \( \text{unitless} \)

Subscripts

\( a \) ambient condition

\( c \) condition at the initial nozzle contraction point

\( e \) exit condition

\( i \) \( i^{\text{th}} \) iteration, species, or element

\( inf \) infinite area combustor condition

\( inj \) condition at the propellant injector

\( j \) \( j^{th} \) iteration, species, or element

\( t \) throat condition

\( 0 \) reactant mixture condition

\( ref \) reference state

Superscripts

\( o \) standard-state condition

\( CB \) best solution of the collective

\( NB \) best solution within a particle’s neighborhood

\( PB \) best solution of a specific particle
ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Dr. John Baker, for his continued support through my work on this project. Our many discussions of the scope and technical direction of this research saved me countless hours of confusion and indecision. Gratitude is also due to Dr. Semih Olcmen and Dr. Paul Ray for serving on my thesis committee. I would also like to acknowledge the financial support of The University of Alabama Graduate Council, without which I would not have accomplished this work.
**CONTENTS**

ABSTRACT................................................................................................................... ii

DEDICATION .................................................................................................................... iii

LIST OF ABBREVIATIONS AND SYMBOLS ............................................................. iv

ACKNOWLEDGMENTS ................................................................................................... vii

LIST OF TABLES ........................................................................................................... x

LIST OF FIGURES ......................................................................................................... xi

1. INTRODUCTION .................................................................................................... 1

2. HISTORY AND BACKGROUND ........................................................................... 5
   2.1 Evaluation of Rocket Performance .............................................................. 5
   2.2 Non-linear Optimization Techniques ........................................................... 7

3. THEORETICAL ROCKET PERFORMANCE .................................................... 15
   3.1 Thermochemical Properties ....................................................................... 15
   3.2 Chemical Equilibrium and Combustion Gas Properties ............................ 16
   3.3 Infinite Area Combustion Chamber Performance ........................................ 21
   3.4 Finite Area Combustion Chamber Performance ......................................... 25

4. PARTICLE SWARM OPTIMIZATION ............................................................... 27
   4.1 Unified Particle Swarm Optimization ....................................................... 27
   4.2 Application to the Performance Problem ................................................. 29

5. VALIDATION AND RESULTS ............................................................................ 32
   5.1 Thermodynamic Properties and Equilibrium Validation ......................... 32
   5.2 Rocket Performance Validation ................................................................. 34
5.2 Particle Swarm Optimization of a LOX/LH2 System ..................35

5.3 Additional Test Cases ..................................................................40

6. CONCLUSIONS.................................................................................46

6.1 Conclusions .................................................................................46

6.2 Recommendations for Future Work .............................................47

REFERENCES ..................................................................................49

APPENDIX A: LIST OF ASSUMPTIONS .............................................53

APPENDIX B: MATLAB CODE .........................................................54
# LIST OF TABLES

1. Various PSO Modifications .............................................................. 11-13
2. Values of PSO Constants ................................................................. 29
4. Equilibrium Mole Fractions of LOX/LH2 Combustion ...................... 33
5. HP Combustion Gas Properties ......................................................... 34
6. SP Combustion Gas Properties ......................................................... 34
7. Sample IAC Frozen Flow Performance Analysis ............................... 35
8. Sample FAC Equilibrium Flow Performance Analysis ...................... 35
9. Optimization Case Definition Variable Parameters......................... 36
10. Optimization Case Definition Constant Parameters ....................... 36
11. 10-Run Results for Case 1 ............................................................ 37
12. 10-Run Results for Case 2 ............................................................ 41
13. 10-Run Results for Case 3 ............................................................ 42
14. 10-Run Results for Case 4 ............................................................ 42
15. 10-Run Results for Case 5 ............................................................ 43
16. 10-Run Results for Case 6 ............................................................ 43
17. 10-Run Results for Case 7 ............................................................ 44
18. 10-Run Results for Case 8 ............................................................ 44
19. 10-Run Results for Case 9 ............................................................ 45
LIST OF FIGURES

1. Frozen Flow Nozzle Zones .................................................................22

2. Equilibrium Flow Nozzle Zones .........................................................23

3. Subfunction Interaction Flow Diagram .................................................31

4.1 Swarm Progression and Convergence, Iteration 0 .......................37

4.2 Swarm Progression and Convergence, Iteration 12 .......................38

4.3 Swarm Progression and Convergence, Iteration 25 .......................38

5. Isp Contours and Particle Swarm Result, Entire Search Domain .......39

6. Isp Contours and Particle Swarm Result, Optimum Region .............40
CHAPTER 1
INTRODUCTION

At the end of the 20th century and beginning of the 21st, a relatively unnoticed trend has developed within the technological underpinnings of Western culture: heavy reliance upon satellite technology. Though few realize it, space assets have become absolutely critical to daily life, both for the military and for average civilians.

Beginning in the 1960’s, the United States started to employ satellites for reconnaissance purposes, primarily providing the locations of nuclear weapon launch sites in the U.S.S.R. and estimating the potential threat of their use. Satellite surveillance continues today, enabling the identification of foreign threats and aiding nearly all tactical forces in the field by providing real-time observation of locations that are out of direct line-of-sight. Modern Predator drones have seen extensive use in combat throughout recent conflicts in the Middle East, and are used for reconnaissance and precision strikes in environments considered too hostile for human pilots. Despite the fact that they remain in the atmosphere, their entire existence is owed to satellites. Once a drone leaves the immediate area of its control center, all of the remote pilot’s instructions are relayed to the aircraft via a satellite data link that can be maintained for the duration of its mission. The rest of the military also relies on the extraordinary capabilities of satellites to supplement operations, most notably the Global Positioning System (GPS). According to army-technology.com [1], “GPS is used to aid land, sea and airborne navigation, geographical
exploration, cartography and geodesy, surveying, vehicle location systems, search and rescue operations, aerial refueling and rendezvous and a wide range of additional applications.”

Though military applications of space assets have become essential tools of defense, the daily experience of the average American citizen is also strongly influenced by satellites. Weather forecast simulations are heavily dependent on the radiometer data from polar-orbiting spacecraft. Navigation tools like Google Maps are reliant upon both satellite imagery of the Earth and the GPS system. Cell phone carriers employ satellites to relay calls over long distances to prevent excessive communication lag. Radio, television, and Internet are streamed from satellites to consumers throughout the world. Surprisingly, all of the aforementioned satellites are even reliant upon solar observation satellites for advanced warning of solar flares, which can damage or disable active electronics in space.

Unfortunately, the process of reliably inserting such satellites into orbit is notoriously difficult and cost-intensive. Currently, the expense associated with putting a one-pound mass into space is $10,000, with a large portion of the cost originating in vehicle design. The rocket’s propulsion system is among the most complex and time-consuming components in the development phase, with most engines taking around seven years to complete. This is because the engine’s reliability and efficiency are imperative to every mission’s primary directive of delivering a payload to space, and each new design must prove itself through a variety of tests to ensure that it will be able to meet that objective. The work presented within this thesis is an attempt to show that the design and optimization process of conventional liquid rocket engines could potentially be automated to reduce design duration while maximizing mission-critical engine performance and ultimately provide more affordable access to space.
The method by which this is achieved is the utilization of a variant of the Particle Swarm Optimization technique that mimics the swarm intelligence of animal colonies found throughout nature. Examples of such collectives are ant and termite colonies, flocks of birds, and schools of fish. Though each individual within the collective may lack intelligence, the simple act of sharing information provides a type of decentralized decision making that allows the unified group to conquer difficult problems that none of the individuals would have been able to solve on their own. A good example of this is given by a bee hive searching for new sources of food. Initially, scout bees are sent in random directions away from the hive, to explore the general lay of the land and note any promising areas. One bee may encounter a shopping mall parking lot, and another might stumble upon a sprawling meadow full of blossoming flowers. Each of the scouts report back to the hive and shares the details of its discovery. The bee that found the parking lot will encourage others not to waste their time flying in that direction, while the bee that stumbled upon the meadow will highly recommend that the other bees explore it for themselves. With that new information, the hive sends more bees in the direction of the flowers and less in the direction of the parking lot. This process of exploration and communication repeats until all of the bees are at one location that they all consider to be the most promising food source for their hive.

To demonstrate how this methodology can be applied to rocket performance, the analogy of the bee hive will be continued. Instead of exploring the three-dimensional physical world with which all are familiar, each bee would be traveling in a spatial domain defined by key rocket parameters. Rather than searching for areas with the most food, the bees now try to find regions with the best rocket performance.
In this thesis, the application of PSO to maximize a liquid rocket engine’s specific impulse is discussed. Chapter 2 provides the history and background of both liquid rocket engine performance and nonlinear optimization techniques. The governing equations for rocket performance are described in Chapter 3. PSO and its application to the performance problem are outlined in Chapter 4. Chapter 5 presents the validation of the rocket performance and optimization codes. The conclusions of this study are discussed in Chapter 6.
CHAPTER 2
HISTORY AND BACKGROUND

This chapter discusses the conventional methods of analysis used to predict the performance of liquid rocket engines and provides a brief overview of the optimization techniques that have historically been applied to aerospace propulsion systems.

2.1 Evaluation of Rocket Performance

The most basic theoretical investigation of liquid rocket engine performance is given by the quasi-one-dimensional isentropic relations of ideal compressible flow [2], [3]. Instead of considering the actual propellants burning within the combustion chamber, this method typically assumes a fixed ratio of specific heats of the exhaust gas and a chamber pressure and temperature to provide a rudimentary back-of-the-napkin estimate of performance. Though it is able to calculate thrust and specific impulse in addition to the pressure and temperature variations throughout the rocket nozzle, this method is severely limited by its underlying assumptions, namely that the exhaust gas is calorically perfect and non-reacting [3].

A significant improvement to this was provided in 1971 with NASA’s Chemical Equilibrium with Applications (CEA) program, developed by Gordon and McBride [4]. This code is able to handle many types of problems, including shock tubes, rocket performance, and Chapman-Jouget detonations. At the heart of its power lies the ability to calculate the chemical
equilibrium of combustion products, which is accomplished by referencing a large thermodynamic database. It is able to provide all relevant properties of a particular species of gas using high-order polynomial curve fits [4]. This enables the code to move beyond the several inhibiting assumptions made in ideal nozzle flows and provides theoretical upper and lower bounds on engine performance that only differ from real hardware by 1-4% [5]. Instead of simply assigning the chamber temperature and obtaining temperature contours by allowing the gas to expand isentropically, the model is now able to obtain that data based on the chemical reactions of the propellants, which occur throughout the system, not just in the combustion chamber. Though it can provide very good estimates of performance, CEA is still limited by three major assumptions. It considers the flow throughout the nozzle to be quasi-one-dimensional, meaning that components of nozzle geometry beyond expansion and contraction ratio (wall contours, roughness, etc.) are neglected and no realistic flow fields can be visualized since all real nozzles have three spatial components rather than one. No viscous effects are considered, resulting in an overestimate of performance and slight underestimation of optimal area ratios. Finally, the flow is considered to be steady, meaning that engine start-up and combustion instability are beyond the capabilities of this program.

In an attempt to address the aforementioned limitations of the quasi-one-dimensional models, a mathematical technique used to solve partial differential equations called the method of characteristics (MOC) was applied to the problem of rocket performance by Prandtl and Busemann in 1929 and provide analysis of axisymmetric two-dimensional flows [3]. Initially, the MOC served as a flow-field analysis tool constrained to gases of fixed composition, but has more recently been coupled with finite-rate chemical kinetics to provide estimates of both engine performance and nozzle flow-field [6]. Sierra Engineering’s industry-standard nozzle design
software Two-Dimensional Kinetics (TDK) has expanded this idea even further by employing additional modules that calculate boundary layer and transonic effects within the nozzle [7].

Though some variations of the MOC are able to provide nearly-perfect estimates of performance for steady, two-dimensional, supersonic flows, computational fluid dynamics (CFD) can be applied to almost any fluids problem. It is able to solve the full viscous Navier-Stokes equations and is able to provide realistic results for a variety of conditions, including unsteady, reactive, viscous, three-dimensional, and subsonic flows [8]. This allows it to investigate the engine start and cutoff sequences, the nozzle plume, thermal loads, and practically any other operating condition of interest.

Though significant advances have been made over the quasi-one-dimensional models, NASA’s CEA model was chosen for this research as it is able to provide solutions accurate enough for a systems-level performance analysis at low computational expense and complexity compared to more advanced models.

2.2 Non-linear Optimization Techniques

When considering the optimization of non-linear systems, the intuitive approach is the brute force search (also known as generate-and-test search). The implementation of this algorithm is very simple. Each contributing variable is independently incremented by a given amount and the fitness function is evaluated on a grid over the entirety of the search space, then the resulting data is processed to find the optimal solution. Brute force search is extremely slow for large systems because it requires an inordinate number of function evaluations, but it is guaranteed to find the global optimum given adequately small parameter incrementations. Due to
its hefty computational expense, it is rarely applied to complex problems except to validate other optimizers [9].

Gradient based methods (GBMs) have a long history of use dating back to the origins of calculus, when Sir Isaac Newton outlined their use to solve for roots of polynomial equations in his 1669 publication “De Analyisi per Aequationes Numerorum Infinitas” [10]. Beginning in 1960 with the work of Professor Lucien Schmit of the University of California, Los Angeles, GBMs were applied to structural optimization problems by adapting them to computational algorithms [11]. The main advantage presented by the GBMs is that they are able to utilize knowledge of a function’s partial derivatives to significantly reduce the number of function evaluations required for optimization [12]. This directly translates into a lower CPU clock time compared to the brute force method and makes the GBM family of algorithms very popular within industry. However, there are several drawbacks associated with them. GBMs are not guaranteed to converge because they are strongly dependent on good initial guesses of the optimal solution [12]. In addition, it is possible for them to converge to a local optimum rather than a global optimum. GBMs also require that the fitness function and its derivatives are continuous and that the partial derivatives are known. This can complicate their implementation or even make it impossible.

Simulated Annealing (SA) is an optimization algorithm that imitates the annealing of metals, devised by Kirkpatrick, Gelatt, and Vecchi in 1983 [14]. When the “temperature” of the simulation is high, potential solutions are able to move quickly toward an optimal solution. As the temperature decreases, the solutions gradually lose their mobility within the search space and only explore their respective local areas. Because it is a metaheuristic algorithm and does not rely on the fitness function’s continuity or computations of its derivatives, SA is able to handle
complex systems that GBMs are incapable of optimizing and is more likely to find the global optimum than GBMs [12]. However, the downside is that its solutions are not continually pointed in the direction of the optimal solution, meaning that many function evaluations are required [12].

Rechenberg and Schwefel developed Genetic Algorithms (GAs) in the 1960’s in an attempt to solve complex engineering problems by mimicking natural selection’s propagation of the specific traits that allow an organism to be successful within its environment, [13]. This was accomplished by including several biological processes in a simulation of an evolving population of candidate solutions. Genetic inheritance is accomplished by “mating” two potential solutions and producing offspring which then form a successive generation. Mutation is applied by randomly scrambling the search space location of a given percentage of candidate solutions and keeps the gene pool of the population from becoming stagnant. From the perspective of the optimization, this helps to keep the algorithm from converging to a local optimum. Natural selection is enforced by allowing the worst candidate solutions to die off before reproducing, ensuring that their poor genes are not carried on to successive generations [15]. From a randomly distributed initial population, each consecutive generation ideally moves closer to the fitness function’s optimum as the best solutions survive and the worst expire. GAs are metaheuristic in nature, require no knowledge of the fitness function’s derivatives, are simple to implement, and provide robust convergence to the global optimum [16]. As such, many variations have seen wide use within the science and engineering communities and have successfully been used within the aerospace field to optimize turbines, spacecraft and helicopter controls, wings and airfoils, rockets, missiles, and propellers, according to Badyrka, Jenkins, and Hartfield [17].
Nevertheless, they require a large number of function evaluations since the populations required for proper mating of the solutions tend to be rather large.

In 1986, Craig Reynolds attempted to understand how flocks of birds are able to make nearly-instantaneous, coordinated, cohesive movements without an identifiable leader [18]. He attributed this capability to three simple behaviors exhibited by each individual in the flock: separation, alignment, and cohesion. Essentially, each bird simultaneously avoids collisions with its neighbors and balances flying in the average direction of its neighbors with flying toward the average location of the flock. The result was an artificial life program entitled “Boids” which has been used in numerous movies and video games to produce realistic swarm behavior [18].

Realizing that the algorithm developed by Reynolds could potentially perform optimization, James Kennedy and Russell Eberhart [19] adapted it and proposed the initial form of Particle Swarm Optimization in 1995. Rather than defining the objectives to be separation, cohesion, and alignment, the individual “particles” were instructed to “fly” throughout the search space by combining their accumulated personal knowledge of the domain and the aggregate knowledge of the collective to find the most promising regions. (It is important to note that the word “particle” does not mean a molecule in this context; rather, it serves as a general designation of an individual solution within the simulated collective.) As with their biological analogs, sharing information in this manner allows the collective to accomplish tasks beyond the abilities of its individual members.

Computationally, PSO provides many advantages over other optimization techniques. Like GAs and SA, it does not place the restriction of differentiability on the fitness function since no gradient information is used. However, PSO has been shown to require a significantly reduced number of function evaluations compared to GAs, which directly translates into
decreased clock times [17]. Additionally, convergence to an optimal solution is nearly completely independent of the initial guesses [20].

Given these advantages, simple implementation requirements, and the novelty of using a technique not commonly employed within the aerospace community, PSO was selected as a suitable optimizer for the rocket engine performance problem. However, in the 18 years since its inception in 1995, many modifications have arisen from the optimization community that each have distinct advantages and disadvantages. In the interest of concision, a large portion of these features have been collected in Table 1. The specific intent of each feature, the actual result of its inclusion into the PSO algorithm, examples of its use in literature, and examples of applications have also been included.

<table>
<thead>
<tr>
<th>PSO Feature</th>
<th>Intent</th>
<th>Actual Result</th>
<th>Examples in Literature</th>
<th>Applications</th>
</tr>
</thead>
</table>
| Constant Trust Parameters | Simulates sharing of knowledge between members of the collective and the head of the collective. | Enables global optimization scheme.                      | • Kennedy & Eberhart [19]  
• Parsopoulos & Vrahatis [21]  
• Talbi & Bartouche [22]  | • Nearly all PSO applications.       |
| Variable Trust Parameters | Balances local and global search directives by simulating particles gaining self-confidence through time. | Improves convergence rate, compromises reliability. [23]  | • Petriciolet &Hernandez [23]  
• Schutte & Groenwold [24]  | • Chemical Equilibrium and Phase Stability [23]       |
| Neighborhood | Balances local and global search directives by simulating dissemination of information between neighbors in the array. | Provides tighter convergence grouping, explores local optima more thoroughly. [21]  | • Deepa & Sugumaran [25]  
• Parsopoulos & Vrahatis [21]  | • MIMO LTI Systems [25]  
• Pressure Vessels, Gear Trains, Beams [21] |

Table 1. Various PSO Modifications
<table>
<thead>
<tr>
<th>PSO Feature</th>
<th>Intent</th>
<th>Actual Result</th>
<th>Examples in Literature</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tabli &amp; Bartouche [22]</td>
<td>Image Registration [22]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Amaya, Cruz, Correa [27]</td>
<td>Nonlinear Systems Root Finding [27]</td>
</tr>
<tr>
<td>Variable Inertia</td>
<td>Balances local and global search directives by allowing particles to change directions quicker toward the end of the optimization.</td>
<td>Inhibits convergence. [23], [28]</td>
<td>Petriciolet &amp; Hernandez [23]</td>
<td>Benchmarks [23], [28]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zhao, Guo, Cao [32]</td>
<td>Neural Networks, Metal Milling, Electric Vehicle Batteries [33]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Eberhart &amp; Shi [33]</td>
<td></td>
</tr>
<tr>
<td>Craziness/Randomness Factor</td>
<td>Prevents convergence to local optima.</td>
<td>Slows convergence or is ineffective [34].</td>
<td>Venter &amp; Sobieski [34]</td>
<td>Structural Design [34]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Badyrka, Jenkins, &amp; Hartfield [17]</td>
<td>Solid Rocket Motor Propellant Grains [17]</td>
</tr>
</tbody>
</table>

Table 1 (Continued). Various PSO Modifications
<table>
<thead>
<tr>
<th>PSO Feature</th>
<th>Intent</th>
<th>Actual Result</th>
<th>Examples in Literature</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraints by Search Space</td>
<td>Does not allow particles to move beyond a certain region.</td>
<td>Constraints cannot be violated.</td>
<td>• Venter &amp; Sobieski [34]</td>
<td>• Structural Design [34]</td>
</tr>
<tr>
<td>Enclosure</td>
<td>Simulates particles being punished for going beyond boundaries.</td>
<td>Discourages particles from moving beyond desired variable boundaries.</td>
<td>• Parsopoulos &amp; Vrahatis [21]</td>
<td>• Pressure Vessels, Gear Trains, Beams [21]</td>
</tr>
<tr>
<td>Constraints by Penalty Functions</td>
<td>Enables high-dimensional optimization.</td>
<td>Prevents particles from becoming trapped in local optima in high-dimensional problem.</td>
<td>• Mo, Liu, &amp; Wang [36]</td>
<td>• Benchmarks [36]</td>
</tr>
<tr>
<td>Combination with Conjugate</td>
<td>To enable multidisciplinary/multiple-objective optimization</td>
<td>Provides Pareto-Dominant Solutions [37]</td>
<td>• Reddy &amp; Kumar [37]</td>
<td>• Water Resource Management and Hydrology Problems [37]</td>
</tr>
<tr>
<td>Direction Method</td>
<td></td>
<td></td>
<td>• Sierra &amp; Coello [38]</td>
<td></td>
</tr>
<tr>
<td>Consideration of Pareto-Optimal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fronts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 (Continued). Various PSO Modifications

The desired optimizer characteristics for application to the rocket problem were as follows:

- Single objective, multiple parameter optimization
- Robust convergence
- Absolute enforcement of constraints
• Ability to handle unknown or indirect variable interactions

• Simple implementation.

Since the only objective considered was the maximization of specific impulse, Pareto-Optimal solutions that balance multiple objectives were not needed. Only three variables were controlled by PSO, rendering combination with other methods for high-dimensional problems similarly unnecessary. Constraints were required to prevent particles from leaving the search space so that particles would not spend clock time on physically infeasible solutions. This made enclosure of the search space more suitable for constraint enforcement than penalty functions. The free variables in question (contraction, expansion, and oxidizer-to-fuel ratios) did not have explicit interactions with each other or the specific impulse, which excluded probabilistic methods from the realm of implementation [35]. The mutation, creep, and crossover inspired by genetic algorithms [29], [30] violated the directive of simplicity, and the efficacy of craziness, variable inertia, and variable trust parameters were questioned within literature [23], [28], [34]. This left a basic particle swarm with constant trust parameters, constant inertia, velocity constriction, and a choice between repulsion and particle neighborhoods to enhance convergence rates.

Ultimately, the variant devised by Parsopoulos and Vrahatis entitled “Unified Particle Swarm Optimization” [21] was specifically chosen as it most closely resembled the desired PSO characteristics and was able to balance the local and global search directives without sacrificing efficiency. This feature enables the algorithm to thoroughly search the entire domain at the beginning of the run, but also to locate the precise optimum once a smaller region of interest has been established [21]. Constraints were handled by enclosing the search space so that invalid cases (ex: expansion ratio less than one) would not be considered.
CHAPTER 3
THEORETICAL ROCKET PERFORMANCE

3.1 Thermochemical Properties

In order to calculate the equilibrium composition of combustion products, it is first necessary to have access to the properties of the different species at varying temperatures and pressures. This can be achieved by interpolating between values of a standard thermodynamic database (ex: JANAF [39]) or by using a curve fit. Since NASA's thermodynamic curve fit equations and long list of considered species were freely available on the Internet and easily implemented, they were chosen for use in this program. Equations 1-3 are taken directly from NASA Reference Publication 1311 [4] and provide equations for non-dimensional molar specific heats, enthalpies, and entropies as functions of temperature. The constants are the same for each equation and are read into MATLAB from a slightly modified version of NASA CEA's thermodynamic library file.

\[
\frac{C_v}{R} = a_1 T^{-2} + a_2 T^{-1} + a_3 + a_4 T + a_5 T^2 + a_6 T^3 + a_7 T^4 \quad (1)
\]

\[
\frac{H^o}{RT} = -a_1 T^{-2} + a_2 T^{-1} \ln(T) + a_3 + a_4 \left(\frac{T}{2}\right) + a_5 \left(\frac{T^2}{3}\right) + a_6 \left(\frac{T^3}{4}\right) + a_7 \left(\frac{T^4}{5}\right) + a_8 \left(\frac{1}{T}\right) \quad (2)
\]

\[
\frac{S^o}{R} = -a_1 T^{-2} - a_2 T^{-1} + a_3 \ln(T) + a_4 T + a_5 \left(\frac{T^2}{2}\right) + a_6 \left(\frac{T^3}{3}\right) + a_7 \left(\frac{T^4}{4}\right) + a_9 \quad (3)
\]
After these values are obtained, it is possible to calculate the Gibbs free energy, chemical potential, and entropy of a species in the combustion mixture with equations 4-6, using a reference pressure of 1 bar. It should also be noted that the chemical potential and entropy are functions of the partial pressure of the species within the mixture, but the mole fraction and pressure terms have been split for convenience since the pressure ratio will remain constant.

\[
\frac{G_j^0}{RT} = \left( \frac{H_j^0}{RT} - \frac{S_j^0}{R} \right)
\]  
\[
\frac{\mu_j}{RT} = \frac{G_j^0}{RT} + \ln(x_j) + \ln\left(\frac{p}{P_{ref}}\right)
\]  
\[
\frac{S_j}{R} = \frac{S_j^0}{R} - \ln(x_j) - \ln\left(\frac{p}{P_{ref}}\right)
\]

Once the above variables have been determined, the specific entropy and specific enthalpy of the collective mixture can be attained using equations 7 and 8.

\[
s = \sum_{j=1}^{n_{SP}} n_j S_j
\]  
\[
h = \sum_{j=1}^{n_{SP}} n_j H_j^0
\]

3.2 Chemical Equilibrium and Combustion Gas Properties

The MATLAB script determines the equilibrium composition of combustion products by using the minimization of Gibbs free energy method, as described in NASA Reference Publication 1311 by Gordon and McBride [4]. All equations in Chapter 3 of this document come from that source unless otherwise specified. The chemical reactions that provide equilibrium
flow composition for evaluation of rocket performance are simulated using two assumptions: constant enthalpy and pressure conditions (HP) and constant entropy and pressure conditions (SP). The equations governing equilibrium conditions were originally obtained by Gordon and McBride using Lagrangian multipliers to enforce conservation of mass and combined with a Taylor series expansion of each equation, neglecting higher order terms. A descent Newton-Raphson method is then applied, and the solution of the resulting linear system provides corrections to initial estimates of moles numbers, temperature, and Lagrangian multipliers. Taken directly from Gordon and McBride, the iteration equations are

\[
\sum_{i=1}^{n_{EL}} \left( \sum_{j=1}^{n_{SP}} a_{kj} a_{ij} n_j \right) \pi_i + \left( \sum_{j=1}^{n_{SP}} a_{kj} n_j \right) \Delta \ln n + \left( \sum_{j=1}^{n_{SP}} a_{kj} n_j \frac{H_j^o}{RT} \right) \Delta \ln T
\]

\[= b_{0k} - \sum_{j=1}^{n_{SP}} a_{ij} n_j + \sum_{j=1}^{n_{SP}} a_{kj} n_j \frac{\mu_j}{RT} \]

\[\text{(k = 1, ..., nEL)} \tag{9}\]

\[
\sum_{i=1}^{n_{EL}} \left( \sum_{j=1}^{n_{SP}} a_{ij} n_j \right) \pi_i + \left( \sum_{j=1}^{n_{SP}} n_j - n \right) \Delta \ln n + \left( \sum_{j=1}^{n_{SP}} n_j \frac{H_j^o}{RT} \right) \Delta \ln T = n - \sum_{j=1}^{n_{SP}} n_j + \sum_{j=1}^{n_{SP}} n_j \frac{\mu_j}{RT}
\]

\[\left[ \sum_{i=1}^{n_{EL}} \left( \sum_{j=1}^{n_{SP}} a_{ij} n_j \frac{H_j^o}{RT} \right) \pi_i + \left( \sum_{j=1}^{n_{SP}} n_j \frac{H_j^o}{RT} \right) \Delta \ln n + \left( \sum_{j=1}^{n_{SP}} n_j \frac{C_{pj}^o}{R} + \sum_{j=1}^{n_{SP}} n_j \left( \frac{H_j^o}{RT} \right)^2 \right) \Delta \ln T \right]^T
\]

\[= \frac{h_o}{R} - \left[ \frac{h}{RT} + \sum_{j=1}^{n_{SP}} n_j \left( \frac{H_j^o}{RT} \right) \left( \frac{\mu_j}{RT} \right) \right]^T \tag{11}\]
where $b_0$ is a matrix containing the initial number of kilomoles of each element. These equations can be assembled into a matrix equation of $n_{EL}+2$ linear equations and $n_{EL}+2$ correction variables. Equations 9-11 are used for the HP problem, while 9, 10, and 12 are used for the SP problem. It should be noted that equation 9 provides $n_{EL}$ separate linear equations. New estimates for individual species’ mole numbers can then be found from equation 13. As suggested by Gordon and McBride, initial estimates for the first iteration are taken as a temperature of 3800K, a total mole number of 0.1, and species mole numbers of $0.1/n_{SP}$. To avoid numerical divergence, a control factor is introduced to prevent overcorrection. This is shown in equation 14 and employed in the generic update equation 15, where $Y$ is simply an arbitrary variable used for concision. In addition, corrections to temperature and total mole number are limited to a factor of $e^{0.4}$, and corrections to species mole numbers are limited to a factor of $e^{2}$.

$$
\sum_{i=1}^{n_{EL}} \left( \sum_{j=1}^{n_{SP}} a_{i,j} n_j \frac{S_j}{R} \right) \pi_i + \left( \sum_{j=1}^{n_{SP}} n_j \frac{S_j}{R} \right) \Delta \ln n + \left( \sum_{j=1}^{n_{SP}} n_j \frac{C_{pj}}{R} + \sum_{j=1}^{n_{SP}} n_j \frac{H_{f,j}^o}{RT} \right) \Delta \ln T
$$

$$
= \frac{s_0}{R} - \frac{s}{R} + n - \sum_{j=1}^{n_{SP}} n_j + \sum_{j=1}^{n_{SP}} n_j \left( \frac{S_j}{R} \right) \left( \frac{\mu_j}{RT} \right)
$$

(12)

$$
\Delta \ln n_j = -\frac{\mu_j}{RT} + \sum_{i=1}^{n_{EL}} a_{i,j} \pi_i + \Delta \ln n + \frac{H_{f,j}^o}{RT} \Delta \ln T
$$

(13)

$$
C. F. = \min \left( 1, \frac{2}{\max(5|\Delta \ln T|, 5|\Delta \ln n|, |\Delta \ln n_j|)} \right) \cdot \min \left( \frac{-\ln x - 9.2103404}{\Delta \ln n_j - \Delta \ln n} \right)
$$

(14)
The convergence criteria used to stop the iteration are as follows:

\[
\frac{n_j |\Delta \ln n_j|}{\sum_{j=1}^{n_{SP}} n_j} \leq 0.5 \times 10^{-8}
\]  (16)

\[
\frac{n |\Delta \ln n|}{\sum_{j=1}^{n_{SP}} n_j} \leq 0.5 \times 10^{-8}
\]  (17)

\[
\left| b_{0i} - \sum_{j=1}^{n_{SP}} a_{ij}n_j \right| \leq \max(b_{0i}) \times 10^{-8}
\]  (18)

\[
|\Delta \ln T| \leq 10^{-4}
\]  (19)

\[
\frac{s_0}{R} - \frac{s}{R} \leq 0.5 \times 10^{-4}
\]  (20)

Once the equilibrium composition and temperature have been determined using the above procedure, the thermodynamic properties of the mixture are found through several more systems of equations, given by equations 21-24. Equations 21 and 22 are solved simultaneously to provide the various constant pressure derivatives, while equations 23 and 24 are solved for the constant temperature derivatives. It should be noted that each system should contain \(n_{EL}+1\) individual equations. After obtaining all of the derivatives, the values of \(c_p, c_v, \gamma, \gamma_s\), and speed of sound for equilibrium flow can be computed from equations 27-30.

\[
\sum_{i=1}^{n_{EL}} \left( \sum_{j=1}^{n_{SP}} a_{kj}a_{ij}n_j \right) \left( \frac{\partial \pi_i}{\partial \ln T} \right)_p + \left( \sum_{j=1}^{n_{SP}} a_{kj}n_j \right) \left( \frac{\partial \ln n_i}{\partial \ln T} \right)_p = \sum_{j=1}^{n_{SP}} a_{kj}n_j \frac{H_j^o}{RT}
\]  (21)

\(( k = 1, \ldots, n_{EL})\)
\begin{align}
\sum_{i=1}^{nEL} \left( \sum_{j=1}^{nSP} a_{ij} n_j \right) \left( \frac{\partial \pi_i}{\partial \ln T} \right)_P &= - \sum_{j=1}^{nSP} n_j \frac{H^0_j}{RT} \\
\sum_{i=1}^{nEL} \left( \sum_{j=1}^{nSP} a_{ij} a_{ij} n_j \right) \left( \frac{\partial \pi_i}{\partial \ln P} \right)_T + \left( \sum_{j=1}^{nSP} a_{kj} n_j \right) \left( \frac{\partial \ln n_j}{\partial \ln P} \right)_T &= \sum_{j=1}^{nSP} a_{kj} n_j \\
\left( k = 1, \ldots, nEL \right) \\
\sum_{i=1}^{nEL} \left( \sum_{j=1}^{nSP} a_{ij} n_j \right) \left( \frac{\partial \pi_i}{\partial \ln P} \right)_T &= - \sum_{j=1}^{nSP} n_j \\
\left( \frac{\partial \ln V}{\partial \ln T} \right)_P &= 1 + \left( \frac{\partial \ln n}{\partial \ln T} \right)_P \\
\left( \frac{\partial \ln V}{\partial \ln P} \right)_T &= -1 + \left( \frac{\partial \ln n}{\partial \ln P} \right)_T \\
c_{p_{\text{equilibrium}}} &= \sum_{i=1}^{nEL} \left( \sum_{j=1}^{nSP} a_{ij} n_j \right) \frac{H^0_j}{RT} \left( \frac{\partial \pi_i}{\partial \ln T} \right)_P + \left( \sum_{j=1}^{nSP} n_j \frac{H^0_j}{RT} \right) \left( \frac{\partial \ln n}{\partial \ln T} \right)_P \\
&+ \sum_{j=1}^{nSP} n_j \frac{c_p^0}{R} + \sum_{j=1}^{nSP} n_j \left( \frac{H^0_j}{RT} \right)^2 \\
c_{v_{\text{equilibrium}}} &= c_{p_{\text{equilibrium}}} + nR \left( \frac{\partial \ln V}{\partial \ln T} \right)_P^2 \\
\gamma &= \frac{c_{p_{\text{equilibrium}}}}{c_{v_{\text{equilibrium}}}} \\
\gamma_s &= \frac{-\gamma}{\left( \frac{\partial \ln V}{\partial \ln P} \right)_T} \\
\alpha &= \sqrt{\gamma_s (R/M)} T
\end{align}
The values for equilibrium flow are used for the HP and SP conditions, but the assumptions made in frozen flow produce a distinct set of the same parameters. The calculation of the speed of sound remains the same and the rest of the values are given in equations 32-34.

\[
C_{P_{frozen}} = \sum_{j=1}^{n_{SP}} \frac{n_j C_{Pj}}{R} 
\]

(32)

\[
C_{v_{frozen}} = C_{p_{frozen}} - \frac{nR}{M}
\]

(33)

\[
\gamma_S = \gamma
\]

(34)

3.3 Infinite Area Combustion Chamber Performance

Throat conditions in de Laval nozzles have historically been computed either by assuming isentropic compression through an infinite area combustion chamber (IAC) or by calculating multiple combustion points for a finite area combustion chamber (FAC). The IAC assumption does not account for the entropy generated by the contraction of the flow from the combustion chamber to the nozzle’s throat, and provides a slightly higher estimate of performance than its FAC counterpart. With the IAC assumption, the properties at the nozzle throat can be found using the SP equilibrium code as discussed above. However, the throat pressure is unknown and must be found iteratively. The HP code provides the conditions at the \textit{inf} station for the given combustion chamber pressure. An initial estimate for throat pressure is given by equation 35. Using the enthalpies of the throat and combustor in equation 36 provides the throat flow velocity, which can subsequently be employed in equation 37 to find the throat Mach number. Corrections to the initial estimate are given by equation 38, and iteration is stopped when the convergence condition described by equation 39 is met.
After the flow properties are found for the throat, the remainder of the nozzle can be treated in two different ways. The first is called frozen flow and assumes that the chemical reactions completely cease beyond the throat, leaving the composition of the exhaust gas fixed. Since no chemical reactions release energy into the flow, this provides a lower bound on engine performance. The second is equilibrium flow, which asserts that the reactions beyond the throat occur instantaneously and release the maximum amount of energy into the flow. This provides an upper bound on performance. The two systems are shown in figure 1 and 2.

\[
\frac{P_{inf}}{P_t} = \left( \frac{\gamma_s+1}{2} \right)^{\frac{\gamma_s}{\gamma_s-1}}
\]

\[
u_t = \sqrt{2(h_{inf} - h_t)}
\]

\[
Mach_t = \frac{u_t}{a_t}
\]

\[
(P_t)_{i+1} = (P_t)_i \times \left( \frac{1+\gamma_sMach_t^2}{1+\gamma_s} \right)
\]

\[
\left| \frac{\nu_t^2 - a_t^2}{\nu_t^2} \right| < 0.4 \times 10^{-4}
\]

Figure 1. Frozen Flow Nozzle Zones
For the frozen flow model, the temperature and pressure are unknown at the exit of the de Laval nozzle and both must be obtained iteratively. The temperature iteration will be discussed first since it is nested within each iteration of pressure. An initial estimate for exit temperature is provided by equation 40, which comes from the isentropic relation for a calorically perfect gas. The estimate acquired is equivalent to the temperature of a calorically perfect gas exiting the nozzle at Mach 2. Equation 41 is then used to find corrections until the convergence criterion described by equation 42 is satisfied.

\[
T_e = \frac{T_{inf}}{1 + 2(\gamma_t - 1)}
\]

\[
(\ln T_e)_{i+1} = (\ln T_e)_i + \frac{s_{inf} - s_e}{c_p e}
\]

\[
\left|\frac{s_{inf} - s_e}{c_p e}\right| \leq .5 \times 10^{-4}
\]
The iteration procedure for pressure is nearly identical, consisting of an initial estimate provided by either equation 43 or equation 44. Corrections are made iteratively using equation 45 until the convergence criterion of equation 46 is met.

\[ \ln \frac{P_{\text{inf}}}{P_t} = \ln \frac{P_{\text{inf}}}{P_t} + \sqrt{3.294 \left( \ln \frac{A_e}{A_e} \right)^2 + 1.535 \ln \frac{A_e}{A_e}} \]  

(43)

If \( \frac{A_e}{A_t} \geq 2 \),

\[ \ln \frac{P_{\text{inf}}}{P_t} = \gamma_s + 1.4 \ln \frac{A_e}{A_t} \]  

(44)

\[ \left( \ln \frac{P_{\text{inf}}}{P_e} \right)_{i+1} = \ln \frac{P_{\text{inf}}}{P_e} + \left( \gamma_s \frac{u_s^2}{u_e^2 - u_e^2} \right) \left[ \ln \frac{A_e}{A_t} - \left( \frac{P_t}{P_e} \frac{T_e}{T_t} \right) \left( \frac{T_e}{P_e u_e} \right) \right] \]  

(45)

\[ \left| \left( \ln \frac{P_{\text{inf}}}{P_e} \right)_i - \left( \ln \frac{P_{\text{inf}}}{P_e} \right)_{i-1} \right| \leq 0.4 \times 10^{-5} \]  

(46)

The equilibrium flow model is essentially the exact same as the frozen flow model except that the exit temperature, ratio of specific heats, enthalpy, and speed of sound at the current estimate of exit pressure are provided by the SP equilibrium function. Once the conditions at the nozzle exit have been established using either method, it is a simple matter to calculate the specific impulse, thrust coefficient, characteristic velocity, and mass flow rate per unit volume. The following equations for each of these parameters are algebraically rearranged versions of equations found in Space Propulsion Analysis and Design, by Ronald Humble [43]. Equation 50 produces a characteristic velocity with a temporal dimension rather than dimensions of length.
per time. This was done to enable direct comparison between the output of the MATLAB script and NASA’s CEA program.

\[
\frac{\dot{m}}{A} = \rho u \tag{47}
\]

\[
\frac{F}{A} = \lambda \left( \frac{\dot{m}}{A} u_e + \frac{A}{m} (P_e - P_a) \right) \tag{48}
\]

\[
I_{SP} = \frac{F}{A} \left( \frac{A}{\dot{m}} \right) \left( \frac{1}{g_0} \right) \tag{49}
\]

\[
c^* \left( \frac{p}{g_0} \right) = \frac{p_{inf}}{g_0} \left( \frac{m}{A} \right) \tag{50}
\]

\[
C_F = \frac{u_e}{c^* g} \tag{51}
\]

3.4 Finite Area Combustion Chamber Performance

The major distinction of the FAC model from the IAC model is that the combustion chamber is treated as a constant-area duct, which enables the rocket to be analyzed using four points along the nozzle in addition to the imaginary infinite area point used for the IAC flow. These are the injector, the initial contraction point, the throat, and the exit plane. The injector point combustion properties are found using the HP equilibrium function given the chamber pressure and oxidizer-to-fuel ratio. The pressure of the infinite area point must be found iteratively to satisfy conservation of momentum as given by equation 52, with the contraction point and throat properties being nested within the loop. The throat properties are calculated in the same manner as for the IAC flow, and the contraction point properties are calculated as the exit plane properties were for IAC, except that the contraction ratio is used instead of the expansion ratio. The initial estimate for the infinite point pressure is provided by the empirical relation expressed by equation 53. The injector pressure as given by equation 52 and infinite
point pressure as given by equation 53 are employed in equation 54 to give a new estimate of the infinite point pressure. This iteration to find the infinite point pressure continues until the convergence criterion in equation 55 is satisfied. Once the convergence criterion has been met, the exit conditions and performance parameters are calculated using the aforementioned techniques.

\[
P_{\text{inj}} = (P + \rho u^2)_{c} = \left(P + \frac{m\bar{u}}{A}\right)_{c}
\]  

(52)

\[
P_{\text{inf}} = P_{\text{inf}} \left(\frac{1.0257 - 1.2318 \frac{A_e}{A_t}}{1 - 1.26505 \frac{A_e}{A_t}}\right)
\]  

(53)

\[
P_{\text{inf,new}} = P_{\text{inf}} \left(\frac{P_{\text{inj,assigned}}}{P_{\text{inj}}}\right)
\]  

(54)

\[
\left|\frac{P_{\text{inj}} - (P + \rho u^2)_{c}}{P_{\text{inj}}}\right| \leq 2 \times 10^{-5}
\]  

(55)

The FAC equilibrium flow model was chosen for optimization since it is able find estimates of performance for realistic engine geometries (i.e. finite nozzles). However, at low contraction ratios and high expansion ratios, the exit-plane equilibrium composition calculations can fail, and the program defaults to the frozen flow performance of the nozzle.
4.1 Unified Particle Swarm Optimization

As mentioned in Chapter 2, a variant of Kennedy and Eberhart’s original PSO algorithm [13] was employed to solve the rocket performance problem. UPSO was devised by Parsopoulos and Vrahatis in 2005 to augment the ability of PSO to find the precise optimum in a region of interest. All concepts and equations within this chapter have been obtained from their paper, *Unified Particle Swarm Optimization for Solving Constrained Engineering Optimization Problems* [21]. The basic premise is as follows: A swarm of solution particles is initially randomly distributed throughout a problem’s search space. It is important to remember that the term ‘particle’ does not represent a physical entity like a molecule, but rather describes a potential solution within the search space. The fitness function (function that is being optimized) is evaluated for each particle, and the particle positions in the search space are compared to one another to determine what area of the search space seems most promising. This simulates the social act of sharing information, and is executed in two ways. First, each particle is compared to all other particles in the collective to determine the best solution ever achieved by any individual within the group. Secondly, each particle is compared to its “neighborhood” within the array to find the best solution currently occupied by one of the neighbors. In addition, each particle remembers the best solution it has personally ever obtained. When the particles are set in motion
and allowed to explore the domain, their velocities are based on a weighted random combination of the experience they gain by sharing information. Each particle compares its current position within the search space to its best-ever position, the collective’s best-ever position, and its neighborhood’s current-best position then randomly combines them to obtain a speed and heading within the search space. This is done independently for each dimension within the search space, and the velocity is limited by multiplying the new speed value by a given constriction factor. The group’s social culture is also taken into account by adding coefficients to each of the velocity components so that the particles are allowed to be more or less reliant upon the collective’s accumulated wisdom, depending on what the PSO operator desires. The positions are updated with the new velocities at each iteration, and the process is repeated until all particles end up gathered around a specific location within the search space that has provided the best solution. This recursion ceases after a specified number of iterations.

Mathematically, the aforementioned algorithm is provided by equations 56-59. Equation 56 calculates a velocity based on the global search directive, and equation 57 calculates a velocity based on the local search directive. The two objectives are balanced by equation 58 and used to update particle position according to equation 59. The self-trust, global-trust, constriction, and local-global balance parameters used by the algorithm were taken as provided within the literature. These values are shown in Table 1. A parametric study was initially considered to determine the most efficient settings for the rocket performance problem, but was deemed unnecessary given the efficacy of the given values.

\[
V_{Global_i}(t+1) = \chi \left[ V_i(t) + c_1 r (X_{iPB}(t) - X_i(t)) + c_2 r (X_{CB}(t) - X_i(t)) \right]
\]  
(56)
\begin{equation}
V_{Local_i}(t + 1) = \chi \left[ V_i(t) + c_1 r \left( X_i^{PB}(t) - X_i(t) \right) + c_2 r \left( X_i^{NB}(t) - X_i(t) \right) \right] \tag{57}
\end{equation}

\begin{equation}
V_{Balanced_i} = (1 - \eta)V_{Local_i}(t + 1) + \eta V_{Global_i}(t + 1) \tag{58}
\end{equation}

\begin{equation}
X_i(t + 1) = X_i(t) + V_{Balanced_i}(t + 1) \tag{59}
\end{equation}

<table>
<thead>
<tr>
<th><strong>Constant</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\chi)</td>
<td>0.729</td>
</tr>
<tr>
<td>(c_1)</td>
<td>2.05</td>
</tr>
<tr>
<td>(c_2)</td>
<td>2.05</td>
</tr>
<tr>
<td>(\eta)</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Table 2. Values of PSO Constants**

4.2 Application to the Performance Problem

Before applying PSO to the rocket performance problem, it was first necessary to separate the input variables into those which were fixed by the user and those which were controlled by the optimization code. Since the combustion chamber pressure, propellant combination, and ambient conditions of any particular rocket engine would likely be determined by the demands of its mission, they were treated as user-defined constants by the program. Nozzle area expansion ratio and oxidizer-to-fuel ratio were allowed to be adjusted by PSO to find an optimal combination. The nozzle area contraction ratio was also included as a free variable to serve as a check that the optimization was working properly, as an increase in contraction ratio will always lead to less entropy generation and therefore greater specific impulse (though this difference is not substantial beyond a ratio of 3.5) [5].

The only modification to the algorithm that was necessary to ensure the PSO code could handle the problem was the enforcement of upper and lower limits on each variable to define the search space. An example of the importance of these boundaries is given by the case of an engine operating in interplanetary space. The ambient pressure is essentially a vacuum and the specific
impulse of any engine is maximized by matching the pressure at the exit plane to the ambient pressure. As such, the PSO code would attempt to give the nozzle an infinite expansion ratio, which is clearly not feasible.

A full map of the interactions between the PSO and rocket performance subfunctions is given in Figure 3. It should be noted that the straight arrows indicate a higher function passing input to a lower function, and the curved arrows indicate a lower function providing its output to a higher function.
Figure 3. Subfunction Interaction Flow Diagram
5.1 Thermodynamic Properties and Equilibrium Validation

To ensure proper function of the rocket performance algorithm, it was necessary to validate that the equilibrium composition and thermodynamic properties of the combustion gas were being calculated correctly. In addition to the LOX/LH2 combination focused on in the PSO example cases, LNH3/LF2 was also employed to test the programs. All results from the MATLAB code were compared to those from CEA and the commercially available RPA written by Alexander Ponomarenko [40]. The results from typical LNH3/LF2 and LOX/LH2 combustions are displayed in Tables 3, 4, and 5, respectively. Table 3 is representative of the combustion chamber conditions of the experimental Russian RD-301 [41], and Table 4 shows the theoretical combustion chamber conditions in the Space Shuttle’s RS-25 [42]. MATLAB and CEA agree perfectly to at least five significant digits for all cases, and RPA is slightly off with a maximum error of around six percent. The small discrepancies in composition between RPA and CEA are attributed to CEA’s consideration of more product species, like H3F3, NH2F, N2H2, and O3. Tables 5 and 6 show example calculations of the composite gas properties under constant entropy/constant pressure and constant enthalpy/constant pressure processes, respectively. The MATLAB code, CEA, and RPA are all in precise agreement to at least five significant digits for all cases. (The combustion products that RPA neglects are ones
with extraordinarily small mole fractions, so their exclusion makes no measurable difference in parameters like specific heat ratio and molecular weight.)

<table>
<thead>
<tr>
<th>Conditions:</th>
<th>T = 4516.7 K</th>
<th>P = 11.76 MPa</th>
<th>O/F = 2.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>MATLAB</td>
<td>CEA</td>
<td>% Error</td>
</tr>
<tr>
<td>F</td>
<td>3.31E-02</td>
<td>3.31E-02</td>
<td>0</td>
</tr>
<tr>
<td>F2</td>
<td>1.89E-06</td>
<td>1.89E-06</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>8.12E-02</td>
<td>8.12E-02</td>
<td>0</td>
</tr>
<tr>
<td>HF</td>
<td>6.66E-01</td>
<td>6.66E-01</td>
<td>0</td>
</tr>
<tr>
<td>H2</td>
<td>6.18E-02</td>
<td>6.18E-02</td>
<td>0</td>
</tr>
<tr>
<td>H2F2</td>
<td>9.09E-03</td>
<td>9.07E-03</td>
<td>0</td>
</tr>
<tr>
<td>H3F3</td>
<td>5.56E-08</td>
<td>5.69E-08</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>3.35E-04</td>
<td>3.35E-04</td>
<td>0</td>
</tr>
<tr>
<td>NF</td>
<td>8.91E-06</td>
<td>8.91E-06</td>
<td>0</td>
</tr>
<tr>
<td>NH</td>
<td>8.17E-05</td>
<td>8.17E-05</td>
<td>0</td>
</tr>
<tr>
<td>NHF</td>
<td>7.12E-07</td>
<td>7.12E-07</td>
<td>0</td>
</tr>
<tr>
<td>NH2</td>
<td>1.55E-05</td>
<td>1.55E-05</td>
<td>0</td>
</tr>
<tr>
<td>NH2F</td>
<td>3.83E-08</td>
<td>3.83E-08</td>
<td>0</td>
</tr>
<tr>
<td>NH3</td>
<td>2.48E-06</td>
<td>2.48E-06</td>
<td>0</td>
</tr>
<tr>
<td>N2</td>
<td>1.48E-01</td>
<td>1.48E-01</td>
<td>0</td>
</tr>
<tr>
<td>N2H2</td>
<td>6.42E-09</td>
<td>6.42E-09</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3. Equilibrium Mole Fractions of LNH3/LF2 Combustion

<table>
<thead>
<tr>
<th>Conditions:</th>
<th>T = 3603.91 K</th>
<th>P = 20.6294 MPa</th>
<th>O/F = 6.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>MATLAB</td>
<td>CEA</td>
<td>% Error</td>
</tr>
<tr>
<td>O</td>
<td>2.11E-03</td>
<td>2.11E-03</td>
<td>0</td>
</tr>
<tr>
<td>OH</td>
<td>3.75E-02</td>
<td>3.75E-02</td>
<td>0</td>
</tr>
<tr>
<td>H2O</td>
<td>6.88E-01</td>
<td>6.88E-01</td>
<td>0</td>
</tr>
<tr>
<td>O2</td>
<td>2.28E-03</td>
<td>2.28E-03</td>
<td>0</td>
</tr>
<tr>
<td>HO2</td>
<td>3.61E-05</td>
<td>3.61E-05</td>
<td>0</td>
</tr>
<tr>
<td>H2O2</td>
<td>1.76E-05</td>
<td>1.76E-05</td>
<td>0</td>
</tr>
<tr>
<td>O3</td>
<td>7.01E-09</td>
<td>7.01E-09</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>2.56E-02</td>
<td>2.56E-02</td>
<td>0</td>
</tr>
<tr>
<td>H2</td>
<td>2.44E-01</td>
<td>2.44E-01</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Equilibrium Mole Fractions of LOX/LH2 Combustion
### Conditions:
P = 10 MPa  
O/F = 5  
Propellants: LNH3/LF2

<table>
<thead>
<tr>
<th>Property</th>
<th>MATLAB</th>
<th>CEA</th>
<th>% Error</th>
<th>RPA</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>20.573</td>
<td>20.573</td>
<td>0</td>
<td>20.573</td>
<td>0</td>
</tr>
<tr>
<td>Cp (J/kg-K)</td>
<td>1991.5</td>
<td>1991.5</td>
<td>0</td>
<td>1991.5</td>
<td>0</td>
</tr>
<tr>
<td>Cv(J/kg-K)</td>
<td>1544.7</td>
<td>-</td>
<td>-</td>
<td>1544.7</td>
<td>0</td>
</tr>
<tr>
<td>γ</td>
<td>1.2774</td>
<td>1.2774</td>
<td>0</td>
<td>1.2774</td>
<td>0</td>
</tr>
<tr>
<td>T (K)</td>
<td>4214.5</td>
<td>4214.5</td>
<td>0</td>
<td>4214.5</td>
<td>0</td>
</tr>
<tr>
<td>a (m/s)</td>
<td>1475.1</td>
<td>1475.1</td>
<td>0</td>
<td>1475.1</td>
<td>0</td>
</tr>
<tr>
<td>h (J/kg)</td>
<td>-987370</td>
<td>-987370</td>
<td>0</td>
<td>-987370</td>
<td>0</td>
</tr>
<tr>
<td>s (J/kg-K)</td>
<td>10603</td>
<td>10603</td>
<td>0</td>
<td>10603</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5. HP Combustion Gas Properties

### Conditions:
P = 20.6429 MPa  
O/F = 3.15  
s = 11384 J/kg-K  
Propellants: LNH3/LF2

<table>
<thead>
<tr>
<th>Property</th>
<th>MATLAB</th>
<th>CEA</th>
<th>% Error</th>
<th>RPA</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>M (kg/kmol)</td>
<td>19.701</td>
<td>19.701</td>
<td>0</td>
<td>19.701</td>
<td>0</td>
</tr>
<tr>
<td>Cp (J/kg-K)</td>
<td>5108.8</td>
<td>5108.8</td>
<td>0</td>
<td>5108.8</td>
<td>0</td>
</tr>
<tr>
<td>Cv(J/kg-K)</td>
<td>4177.5</td>
<td>-</td>
<td>-</td>
<td>4177.5</td>
<td>0</td>
</tr>
<tr>
<td>γ</td>
<td>1.1622</td>
<td>1.1622</td>
<td>0</td>
<td>1.1622</td>
<td>0</td>
</tr>
<tr>
<td>T (K)</td>
<td>4816.6</td>
<td>4816.6</td>
<td>0</td>
<td>4816.6</td>
<td>0</td>
</tr>
<tr>
<td>a (m/s)</td>
<td>1537</td>
<td>1537</td>
<td>0</td>
<td>1537</td>
<td>0</td>
</tr>
<tr>
<td>h (J/kg)</td>
<td>-1273900</td>
<td>-1273900</td>
<td>0</td>
<td>-1273900</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6. SP Combustion Gas Properties

### 5.2 Rocket Performance Validation

Both the FAC and IAC rocket performance codes tested against CEA and RPA to validate they provided accurate estimates of performance. Each performance code was run with three combustion chamber pressures, propellants (LOX/RP-1, LOX/LH2, LNH3/LF2), and area ratios. It should be noted that only momentum thrust was used in the calculation of the specific impulse during validation to enable direct comparison to CEA’s results, which do not consider ambient pressure. For all calculations beyond this validation section, pressure thrust was included in Isp, as given in equation 49. Representative comparisons are shown in Tables 7 and
8, each using a LOX/LH2 propellant combination. The disparities between all three codes were less than a tenth of a percent for all test cases.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>MATLAB</th>
<th>CEA</th>
<th>% Error</th>
<th>RPA</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_e$</td>
<td>6.0476</td>
<td>6.048</td>
<td>0.006613757</td>
<td>6.0476</td>
<td>0</td>
</tr>
<tr>
<td>$P_e$ (Pa)</td>
<td>482060</td>
<td>482060</td>
<td>0</td>
<td>482400</td>
<td>-0.070480929</td>
</tr>
<tr>
<td>$T_e$ (K)</td>
<td>725.71</td>
<td>725.71</td>
<td>0</td>
<td>725.8411</td>
<td>-0.018061804</td>
</tr>
<tr>
<td>Mach_e</td>
<td>3.0106</td>
<td>3.011</td>
<td>-0.013284623</td>
<td>3.0102</td>
<td>0.013288154</td>
</tr>
<tr>
<td>Isp (s)</td>
<td>357.26</td>
<td>357.3</td>
<td>-0.011195074</td>
<td>357.24</td>
<td>0.005598477</td>
</tr>
<tr>
<td>$c^*$ (s)</td>
<td>240.96</td>
<td>240.797865</td>
<td>0.068196155</td>
<td>240.7957865</td>
<td>0.068196155</td>
</tr>
<tr>
<td>Cf</td>
<td>1.4827</td>
<td>1.4837</td>
<td>0.06739907</td>
<td>1.4826</td>
<td>0.006744908</td>
</tr>
</tbody>
</table>

Table 7. Sample IAC Frozen Flow Performance Analysis

<table>
<thead>
<tr>
<th>Parameter</th>
<th>MATLAB</th>
<th>CEA</th>
<th>% Error</th>
<th>RPA</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_e$</td>
<td>12.091</td>
<td>12.091</td>
<td>0</td>
<td>12.0908</td>
<td>0.00165415</td>
</tr>
<tr>
<td>$P_e$ (Pa)</td>
<td>154130</td>
<td>154130</td>
<td>0</td>
<td>154300</td>
<td>-0.110174984</td>
</tr>
<tr>
<td>$T_e$ (K)</td>
<td>1966.3</td>
<td>1966.3</td>
<td>0</td>
<td>1966.5202</td>
<td>-0.011197444</td>
</tr>
<tr>
<td>Mach_e</td>
<td>2.7404</td>
<td>2.74</td>
<td>0.01459854</td>
<td>2.74</td>
<td>0.01459854</td>
</tr>
<tr>
<td>Isp (s)</td>
<td>359.07</td>
<td>359.0624729</td>
<td>0.002096317</td>
<td>359.04</td>
<td>0.008355615</td>
</tr>
<tr>
<td>$c^*$ (s)</td>
<td>241.4</td>
<td>241.3974191</td>
<td>0.001069151</td>
<td>241.4035374</td>
<td>-0.001465345</td>
</tr>
<tr>
<td>Cf</td>
<td>1.4874</td>
<td>1.4874</td>
<td>0</td>
<td>1.4873</td>
<td>0.006723593</td>
</tr>
</tbody>
</table>

Table 8. Sample FAC Equilibrium Flow Performance Analysis

5.3 Particle Swarm Optimization of a LOX/LH2 System

Unlike the rocket performance code, the optimization of the performance problem could not be directly compared to pre-existing software or results found in literature. Instead, nine different LOX/LH2 test cases were executed ten times each to test the consistency of convergence of the algorithm. The case definitions are outlined in Tables 9 and 10, with Table 9
providing the inputs that varied between cases and Table 10 giving the invariant parameters. This section will focus on Case 1. The results of Case 2 through Case 9 are presented and discussed in section 5.3. The ten runs from Case 1 as well as the averages and standard deviations for each parameter are put forth in Table 11. As expected, the contraction ratio was selected to be the maximum allowed in each run. The largest difference between the specific impulses was 0.0006 seconds. In addition to the quantitative convergence study, the location of each particle was plotted throughout the runs to help visualize convergence patterns. Figure 4 demonstrates the swarm’s time history for the first run of Case 1. The particles start randomly distributed and clearly converge to one point in the search space as time goes on.

<table>
<thead>
<tr>
<th>Case</th>
<th>P_injector (MPa)</th>
<th>P_ambient (kPa)</th>
<th>Max Expansion Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>70</td>
</tr>
</tbody>
</table>

**Table 9. Optimization Case Definitions, Variable Parameters**

<table>
<thead>
<tr>
<th>Other Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min o/f: 3.5</td>
</tr>
<tr>
<td>Min Expansion Ratio: 2</td>
</tr>
<tr>
<td>Max o/f: 7.5</td>
</tr>
<tr>
<td>SwarmSize: 12</td>
</tr>
<tr>
<td>Min Contraction Ratio: 1.5</td>
</tr>
<tr>
<td>LocalRadius: 2</td>
</tr>
<tr>
<td>Max Contraction Ratio: 3.5</td>
</tr>
<tr>
<td>Maximum Iterations: 25</td>
</tr>
</tbody>
</table>

**Table 10. Optimization Case Definitions, Constant Parameters**
## Case 1

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.7696</td>
<td>31.0938</td>
<td>3.5000</td>
<td>431.5100</td>
</tr>
<tr>
<td>2</td>
<td>4.7601</td>
<td>31.0586</td>
<td>3.5000</td>
<td>431.5098</td>
</tr>
<tr>
<td>3</td>
<td>4.7797</td>
<td>31.0915</td>
<td>3.5000</td>
<td>431.5099</td>
</tr>
<tr>
<td>4</td>
<td>4.7720</td>
<td>31.0367</td>
<td>3.5000</td>
<td>431.5100</td>
</tr>
<tr>
<td>5</td>
<td>4.7685</td>
<td>31.0643</td>
<td>3.5000</td>
<td>431.5100</td>
</tr>
<tr>
<td>6</td>
<td>4.7686</td>
<td>31.1556</td>
<td>3.5000</td>
<td>431.5098</td>
</tr>
<tr>
<td>7</td>
<td>4.7591</td>
<td>30.8399</td>
<td>3.5000</td>
<td>431.5094</td>
</tr>
<tr>
<td>8</td>
<td>4.7721</td>
<td>31.2117</td>
<td>3.5000</td>
<td>431.5096</td>
</tr>
<tr>
<td>9</td>
<td>4.7750</td>
<td>31.0521</td>
<td>3.5000</td>
<td>431.5100</td>
</tr>
<tr>
<td>10</td>
<td>4.7702</td>
<td>31.0099</td>
<td>3.5000</td>
<td>431.5100</td>
</tr>
</tbody>
</table>

Average: 4.76949 31.06141 3.5 431.50985
Standard Deviation: 0.006201514 0.097665432 0.000206828

### Table 11. 10-Run Results for Case 1

![Figure 4.1 Swarm Progression and Convergence, Iteration 0](image-url)
Figure 4.2 Swarm Progression and Convergence, Iteration 12

Figure 4.3 Swarm Progression and Convergence, Iteration 25
To ensure that the global optimum was accurate as well as precise, an Isp contour plot was generated for Case 1 at the maximum contraction ratio, and the PSO result was plotted on top of it. (The test runs for this case were essentially identical, producing only one result.) Figure 5 shows the contours over the entire search domain and Figure 6 zooms in to provide a closer look at the contours around the precise optimal point. A measure of the disparity of the two scales can be obtained from the graphs’ axis labels.

Figure 5. Isp Contours and Particle Swarm Result, Entire Search Domain
5.4 Additional Test Cases

Tables 12-19 show the results from Cases 2-9. Including the first case discussed above, about 9% of test runs failed to recognize that the maximum contraction ratio provides maximum specific impulse. However, the disparity in Isp values between the test runs with optimal and non-optimal contraction ratio was exceptionally low, never exceeding 0.01 seconds. This is because the algorithm compensated by adjusting the expansion ratio in such a way that the performance was maximized with the lower contraction ratio. The standard deviation of the specific impulse of the test cases was extremely low as well, with the highest value being 0.003 seconds, in Case 3. Since the difference in performance of the theoretically non-optimal cases was essentially negligible, they were still considered to provide the global optimum.
The expansion ratio trends also behave as expected. In all cases with vacuum ambient conditions, the optimal expansion ratio is found to be the maximum allowable. Similarly, in the cases with increased ambient pressure the expansion ratio was found to be at neither extreme the allowable range, and the individual runs within each case agreed very well, with a maximum standard deviation of specific impulse on the order of 0.001%.

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.5032</td>
<td>18.2022</td>
<td>3.5</td>
<td>417.0091</td>
</tr>
<tr>
<td>2</td>
<td>4.5208</td>
<td>18.1308</td>
<td>3.4982</td>
<td>417.0076</td>
</tr>
<tr>
<td>3</td>
<td>4.4973</td>
<td>18.3087</td>
<td>3.5</td>
<td>417.0085</td>
</tr>
<tr>
<td>4</td>
<td>4.4897</td>
<td>18.2954</td>
<td>3.4988</td>
<td>417.0078</td>
</tr>
<tr>
<td>5</td>
<td>4.4822</td>
<td>18.1044</td>
<td>3.5</td>
<td>417.0077</td>
</tr>
<tr>
<td>6</td>
<td>4.5012</td>
<td>18.2558</td>
<td>3.5</td>
<td>417.009</td>
</tr>
<tr>
<td>7</td>
<td>4.5052</td>
<td>18.1779</td>
<td>3.5</td>
<td>417.009</td>
</tr>
<tr>
<td>8</td>
<td>4.5078</td>
<td>18.1986</td>
<td>3.5</td>
<td>417.0091</td>
</tr>
<tr>
<td>9</td>
<td>4.5033</td>
<td>18.19</td>
<td>3.5</td>
<td>417.0091</td>
</tr>
<tr>
<td>10</td>
<td>4.5165</td>
<td>18.1818</td>
<td>3.5</td>
<td>417.0087</td>
</tr>
<tr>
<td>Average</td>
<td>4.50272</td>
<td>18.20456</td>
<td>3.4997</td>
<td>417.00856</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.011409431</td>
<td>0.0655549</td>
<td>0.000648074</td>
<td>0.000625744</td>
</tr>
</tbody>
</table>

Table 12. 10-Run Results for Case 2
## Case 3

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.2806</td>
<td>100</td>
<td>3.5</td>
<td>469.69</td>
</tr>
<tr>
<td>2</td>
<td>5.2792</td>
<td>100</td>
<td>3.5</td>
<td>469.69</td>
</tr>
<tr>
<td>3</td>
<td>5.28</td>
<td>100</td>
<td>3.5</td>
<td>469.69</td>
</tr>
<tr>
<td>4</td>
<td>5.2595</td>
<td>100</td>
<td>2.7699</td>
<td>469.68</td>
</tr>
<tr>
<td>5</td>
<td>5.2803</td>
<td>100</td>
<td>3.5</td>
<td>469.69</td>
</tr>
<tr>
<td>6</td>
<td>5.2802</td>
<td>100</td>
<td>3.5</td>
<td>469.6877</td>
</tr>
<tr>
<td>7</td>
<td>5.2822</td>
<td>100</td>
<td>3.1274</td>
<td>469.6862</td>
</tr>
<tr>
<td>8</td>
<td>5.279</td>
<td>100</td>
<td>3.5</td>
<td>469.6877</td>
</tr>
<tr>
<td>9</td>
<td>5.2794</td>
<td>100</td>
<td>3.5</td>
<td>469.6877</td>
</tr>
<tr>
<td>10</td>
<td>5.2802</td>
<td>100</td>
<td>3.5</td>
<td>469.6877</td>
</tr>
</tbody>
</table>

Average 5.27806 100 3.38973 469.6877

| Standard Deviation | 0.006582671 | 0 | 0.247269992 | 0.003031318 |

Table 13. 10-Run Results for Case 3

## Case 4

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.2162</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>2</td>
<td>5.2153</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>3</td>
<td>5.2149</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>4</td>
<td>5.2156</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>5</td>
<td>5.2153</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>6</td>
<td>5.2153</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>7</td>
<td>5.2161</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>8</td>
<td>5.215</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>9</td>
<td>5.2155</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
<tr>
<td>10</td>
<td>5.2155</td>
<td>100</td>
<td>3.5</td>
<td>469.42</td>
</tr>
</tbody>
</table>

Average 5.21547 100 3.5 469.42

| Standard Deviation | 0.000419126 | 0 | 0 | 0 |

Table 14. 10-Run Results for Case 4
### Case 5

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.4336</td>
<td>18.1354</td>
<td>3.5</td>
<td>416.8371</td>
</tr>
<tr>
<td>2</td>
<td>4.4567</td>
<td>18.1028</td>
<td>3.5</td>
<td>416.8379</td>
</tr>
<tr>
<td>3</td>
<td>4.4645</td>
<td>18.2517</td>
<td>3.5</td>
<td>416.8382</td>
</tr>
<tr>
<td>4</td>
<td>4.4698</td>
<td>18.3218</td>
<td>3.5</td>
<td>416.8376</td>
</tr>
<tr>
<td>5</td>
<td>4.4825</td>
<td>18.2767</td>
<td>3.5</td>
<td>416.8369</td>
</tr>
<tr>
<td>6</td>
<td>4.461</td>
<td>18.2086</td>
<td>3.5</td>
<td>416.8384</td>
</tr>
<tr>
<td>7</td>
<td>4.4666</td>
<td>18.2704</td>
<td>3.5</td>
<td>416.8381</td>
</tr>
<tr>
<td>8</td>
<td>4.4568</td>
<td>18.1268</td>
<td>3.5</td>
<td>416.8381</td>
</tr>
<tr>
<td>9</td>
<td>4.4531</td>
<td>18.2013</td>
<td>3.5</td>
<td>416.8384</td>
</tr>
<tr>
<td>10</td>
<td>4.4933</td>
<td>18.3444</td>
<td>3.5</td>
<td>416.8352</td>
</tr>
</tbody>
</table>

|          | Average   |                |                  |          |
|          | 4.46379   | 18.22399       | 3.5              | 416.83759 |

|          | Standard Deviation |          |                  |          |
|          | 0.016323293       | 0.083332153 | 0                | 0.000984829 |

Table 15. 10-Run Results for Case 5

### Case 6

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.2153</td>
<td>10.8079</td>
<td>3.5</td>
<td>399.68</td>
</tr>
<tr>
<td>2</td>
<td>4.1713</td>
<td>10.7639</td>
<td>3.493</td>
<td>399.6752</td>
</tr>
<tr>
<td>3</td>
<td>4.2118</td>
<td>10.801</td>
<td>3.5</td>
<td>399.6801</td>
</tr>
<tr>
<td>4</td>
<td>4.2069</td>
<td>10.811</td>
<td>3.5</td>
<td>399.6801</td>
</tr>
<tr>
<td>5</td>
<td>4.232</td>
<td>10.7963</td>
<td>3.4998</td>
<td>399.6786</td>
</tr>
<tr>
<td>6</td>
<td>4.2045</td>
<td>10.7803</td>
<td>3.5</td>
<td>399.6802</td>
</tr>
<tr>
<td>7</td>
<td>4.2098</td>
<td>10.7775</td>
<td>3.5</td>
<td>399.6801</td>
</tr>
<tr>
<td>8</td>
<td>4.1973</td>
<td>10.7545</td>
<td>3.5</td>
<td>399.6798</td>
</tr>
<tr>
<td>9</td>
<td>4.1886</td>
<td>10.7723</td>
<td>3.5</td>
<td>399.6794</td>
</tr>
<tr>
<td>10</td>
<td>4.2149</td>
<td>10.7567</td>
<td>3.5</td>
<td>399.6798</td>
</tr>
</tbody>
</table>

|          | Average   |                |                  |          |
|          | 4.20524   | 10.78214       | 3.49928          | 399.67933 |

|          | Standard Deviation |          |                  |          |
|          | 0.016565037       | 0.020865399 | 0.002207462      | 0.001528289 |

Table 16. 10-Run Results for Case 6
<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.571</td>
<td>3.3834</td>
<td>3.5</td>
<td>344.6335</td>
</tr>
<tr>
<td>2</td>
<td>3.5633</td>
<td>3.4203</td>
<td>3.5</td>
<td>344.6311</td>
</tr>
<tr>
<td>3</td>
<td>3.5681</td>
<td>3.3902</td>
<td>3.5</td>
<td>344.6337</td>
</tr>
<tr>
<td>4</td>
<td>3.5647</td>
<td>3.3851</td>
<td>3.4996</td>
<td>344.6335</td>
</tr>
<tr>
<td>5</td>
<td>3.543</td>
<td>3.3469</td>
<td>3.5</td>
<td>344.6279</td>
</tr>
<tr>
<td>6</td>
<td>3.5633</td>
<td>3.3843</td>
<td>3.5</td>
<td>344.6337</td>
</tr>
<tr>
<td>7</td>
<td>3.5711</td>
<td>3.3968</td>
<td>3.5</td>
<td>344.6336</td>
</tr>
<tr>
<td>8</td>
<td>3.565</td>
<td>3.3844</td>
<td>3.5</td>
<td>344.6337</td>
</tr>
<tr>
<td>9</td>
<td>3.5768</td>
<td>3.4034</td>
<td>3.5</td>
<td>344.633</td>
</tr>
<tr>
<td>10</td>
<td>3.5676</td>
<td>3.3832</td>
<td>3.5</td>
<td>344.6336</td>
</tr>
</tbody>
</table>

Average: 3.56539  3.3878  3.49996  344.63273

Standard Deviation: 0.00892667  0.018651184  0.000126491  0.001873233

Table 17. 10-Run Results for Case 7

<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.7804</td>
<td>5.5162</td>
<td>3.5</td>
<td>371.2375</td>
</tr>
<tr>
<td>2</td>
<td>3.7834</td>
<td>5.5126</td>
<td>3.5</td>
<td>371.2375</td>
</tr>
<tr>
<td>3</td>
<td>3.7974</td>
<td>5.5185</td>
<td>3.5</td>
<td>371.2368</td>
</tr>
<tr>
<td>4</td>
<td>3.7869</td>
<td>5.5286</td>
<td>3.5</td>
<td>371.2372</td>
</tr>
<tr>
<td>5</td>
<td>3.7815</td>
<td>5.4994</td>
<td>3.5</td>
<td>371.2374</td>
</tr>
<tr>
<td>6</td>
<td>3.7784</td>
<td>5.4784</td>
<td>3.5</td>
<td>371.2367</td>
</tr>
<tr>
<td>7</td>
<td>3.7802</td>
<td>5.5074</td>
<td>3.5</td>
<td>371.2375</td>
</tr>
<tr>
<td>8</td>
<td>3.772</td>
<td>5.4685</td>
<td>3.5</td>
<td>371.2361</td>
</tr>
<tr>
<td>9</td>
<td>3.7774</td>
<td>5.5109</td>
<td>3.5</td>
<td>371.2374</td>
</tr>
<tr>
<td>10</td>
<td>3.7831</td>
<td>5.5055</td>
<td>3.5</td>
<td>371.2375</td>
</tr>
</tbody>
</table>

Average: 3.78207  5.5046  3.5  371.23716

Standard Deviation: 0.00670374  0.018356954  0  0.000476562

Table 18. 10-Run Results for Case 8
<table>
<thead>
<tr>
<th>Run Number</th>
<th>o/f Ratio</th>
<th>Expansion Ratio</th>
<th>Contraction Ratio</th>
<th>Isp (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.8778</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>2</td>
<td>4.8768</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>3</td>
<td>4.8761</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>4</td>
<td>4.8723</td>
<td>70</td>
<td>3.0871</td>
<td>463.5722</td>
</tr>
<tr>
<td>5</td>
<td>4.8764</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>6</td>
<td>4.8766</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>7</td>
<td>4.8763</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>8</td>
<td>4.8819</td>
<td>70</td>
<td>3.4904</td>
<td>463.5747</td>
</tr>
<tr>
<td>9</td>
<td>4.8742</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
<tr>
<td>10</td>
<td>4.8766</td>
<td>70</td>
<td>3.5</td>
<td>463.5748</td>
</tr>
</tbody>
</table>

| Average | 4.8765 | 70 | 3.45775 | 463.57453 |
| Standard Deviation | 0.00245176 | 0 | 0.130268076 | 0.000819282 |

Table 19. 10-Run Results for Case 9
6.1 Conclusions

In this study, Unified Particle Swarm Optimization has been applied to a quasi-one-dimensional, equilibrium flow liquid rocket engine performance model to maximize the specific impulse of the system using the MATLAB computational engine. Constraints were handled by complete enclosure of the search domain, rendering the particles unable to travel beyond established boundaries. Calculations of equilibrium composition, combustion gas properties, and rocket performance were tested against NASA’s industry-standard Chemical Equilibrium with Applications program [4] and the commercially available Rocket Propulsion Analysis tool [40]. The chemical equilibrium composition and combustion gas properties functions were able to produce essentially flawless results when compared to CEA and RPA. The rocket performance code was shown to consistently produce results for key performance parameters accurate to within 0.07%. Nine distinct LOX/LH2 cases with varying combustion chamber and ambient pressures were employed to test the validity of the UPSO code. Run-to-run agreement was markedly high, with very slightly non-optimal solutions occurring in approximately 9% of test cases. Differences in optimal and non-optimal cases resulted in Isp disparities on the order of hundredths of a percent, which were considered to be negligible. UPSO was also proven to find the global maximum Isp by comparison to the brute force optimization method. In conclusion,
UPS0 was shown to reliably provide optimal engine specific impulse in all test cases by adjusting nozzle expansion ratio, contraction ratio, and oxidizer-to-fuel ratio.

6.2 Recommendations for Future Work

Though the work presented in this document serves as a proof-of-concept for design automation of liquid rocket engines, there are still many additions, modifications, and expansions to be made before it will become a viable alternative to historical methods of design. For a full systems-level optimization, a number of modules representing different configurations could be employed in the rocket performance analysis. Examples of such modules would be theoretical models of various engine cycles (expander, gas generator, pressure-fed, staged combustion, etc.), turbomachinery, and nozzle geometries (thrust optimized parabolic, true thrust optimized contour, conical, annular and linear aerospike, etc.). With so many variables in consideration, numerous local optima would be inherent to such a model. The UPSO variant would no longer be a feasible optimizer because it is incapable of handling such a high dimensional seach space, and would likely need to be augmented with another method, such as the Conjugate Direction Method scheme mentioned in Table 1. In addition, the optimizer would have to be tuned to the specific problem to make it as computationally efficient as possible. This would entail not only a parametric study of the PSO characteristic variables, but also a new criterion for convergence to be able to compare the optimizer’s efficiency between runs.

Though a full system-level optimization would be the next logical step in automation, PSO’s potential reaches far beyond that. Theoretically, it could control every aspect of a rocket engine’s design, from the component layout and piping to the nozzle contour and turbopump blades’ airfoil shape. PSO would again need to be augmented to consider multiple conflicting
objectives (thrust level vs. engine size and weight) and hundreds of design variables. It would be a monumental task with today’s technology, requiring detailed reactive CFD models to be generated and evaluated tens thousands of times, if not more.
REFERENCES


APPENDIX A: LIST OF ASSUMPTIONS

- All gases are ideal.
- No condensed or ionized species exist in the exhaust gases.
- Homogeneous mixing of fuel and oxidizer.
- Combustion is complete and adiabatic.
- Zero velocity at the combustion chamber inlet.
- Axisymmetric quasi-one-dimensional flow through the combustion chamber and nozzle.
- Isentropic expansion in the nozzle.
- Steady-state conditions.
- Combustion chamber has a constant area cross-section.
- Viscous effects are negligible.
- Relativistic effects are negligible.
APPENDIX B: MATLAB CODE

FACEquilibriumFlow.m

function [Performance Thermo Fractions]=FACEquilibriumFlow(ChamberPressure, AmbientPressure, OxidizerToFuelRatio, ReactionNumber, ExpansionRatio, ContractionRatio)

UniversalGasConstant=8314.51;
g=9.80665;

% FAC Injector Combustion
[InjectorFractions,InjectorProperties]=ReducedCEA_HP(ChamberPressure,OxidizerToFuelRatio,ReactionNumber);
M_inj=InjectorProperties(1);
Cp_inj=InjectorProperties(2);
Cv_inj=InjectorProperties(3);
gamma_inj=InjectorProperties(5);
a_inj=InjectorProperties(6);
T_inj=InjectorProperties(7);
h_inj=InjectorProperties(8);
s_inj=InjectorProperties(9);

% Initial P_inf estimate
P_inf=ChamberPressure*(1.0257-1.2318*ContractionRatio)/(1-1.26505*ContractionRatio);
% CEA 6.30
% Iteration for P_inf
outerconvergence=0;
outeriterations=0;
while (outerconvergence==0) && (outeriterations<=3)

%Recalculation of Inf Station Combustion w/ new P_inf Estimate
[~,InfProperties]=ReducedCEA_HP(P_inf,OxidizerToFuelRatio,ReactionNumber);
M_inf=InfProperties(1);
Cp_inf=InfProperties(2);
Cv_inf=InfProperties(3);
gamma_inf=InfProperties(5);
a_inf=InfProperties(6);
T_inf=InfProperties(7);
h_inf=InfProperties(8);
s_inf=InfProperties(9);

% Iteration to find throat pressure
P_infOverP_t=((gamma_inf+1)/2)^(gamma_inf/(gamma_inf-1)); %CEA 6.15
P_t=P_inf/P_infOverP_t;
innerconvergence=0;
inneriterations=0;
while (innerconvergence==0) && (inneriterations<=3)
\[\text{gamma}_t = \text{ThetaProperties}(4);\]
\[a_t = \text{ThetaProperties}(6);\]
\[h_t = \text{ThetaProperties}(8);\]
\[u_t = \sqrt{2(h_{\text{inj}} - h_t)};\]
\[\text{Mach}_t = \frac{u_t}{a_t};\]
\[\text{if } \text{abs}(\frac{u_t^2 - a_t^2}{u_t^2}) \leq 0.4 \times 10^{-8}\]
\[\text{innerconvergence} = 1;\]
\[\text{else}\]
\[\text{P}_t = P_t \times \left(\frac{1 + \gamma_t \text{Mach}_t^2}{1 + \gamma_t}\right);\]
\[\text{end}\]
\[\text{inneriterations} = \text{inneriterations} + 1;\]
\[\text{end}\]
\[\% \text{Calculation of full throat properties}\]

\[\text{[ThroatFractions, ThroatProperties]} = \text{ReducedCEA_SP}(P_t, \text{OxidizerToFuelRatio}, \text{ReactionNumber}, s_{\text{inf}});\]
\[M_t = \text{ThroatProperties}(1);\]
\[C_p_t = \text{ThroatProperties}(2);\]
\[C_v_t = \text{ThroatProperties}(3);\]
\[\text{gamma}_t = \text{ThroatProperties}(5);\]
\[a_t = \text{ThroatProperties}(6);\]
\[T_t = \text{ThroatProperties}(7);\]
\[h_t = \text{ThroatProperties}(8);\]
\[s_t = \text{ThroatProperties}(9);\]
\[u_t = \sqrt{2(h_{\text{inj}} - h_t)};\]
\[\text{Mach}_t = \frac{u_t}{a_t};\]
\[\% \text{Initial Estimate Of Contraction Point Pressure}\]
\[\text{if } (\text{ContractionRatio} < 1.09) \&\& (\text{ContractionRatio} > 1)\]
\[\text{P}_{\text{infOverP_c}} = \exp\left(\frac{0.9 \times \log(P_{\text{inf}}/P_t)}{(\text{ContractionRatio} + 10.587 \times (\log(\text{ContractionRatio}))^3 + 9.454 \times \log(\text{ContractionRatio}))}\right);\]
\[\% \text{CEA 6.19}\]
\[\text{elseif} \text{ContractionRatio} \geq 1.09;\]
\[\text{P}_{\text{infOverP_c}} = \exp\left(\frac{\log(P_{\text{inf}}/P_t)}{(\text{ContractionRatio} + 10.587 \times (\log(\text{ContractionRatio}))^3 + 9.454 \times \log(\text{ContractionRatio}))}\right);\]
\[\% \text{CEA 6.20}\]
\[\text{else}\]
\[\text{disp(‘Error in FACEquilibriumFlow.m. Contraction Ratio must be greater than one.’);}\]
\[\text{end}\]
\[\text{P_c} = P_{\text{inf}}/P_{\text{infOverP_c}};\]
\[\% \text{Iteration To Find Contraction Point Pressure}\]
\[\text{innerconvergence} = 0;\]
\[\text{inneriterations} = 0;\]
\[\text{while } (\text{innerconvergence} == 0) \&\& (\text{inneriterations} <= 10)\]
\[\% \text{Initial Estimates for other contraction point properties given pressure}\]
\[ \text{CurrentRatio} = \frac{P_t \cdot u_t}{T_t} \cdot \frac{M_t}{M_c} \]

\[ P_{\text{infOverP_c}} = \exp(\log(P_{\text{inf}}/P_c) + \frac{\gamma_c \cdot u_c^2}{u_c^2 - a_c^2} \cdot (\log(\text{ContractionRatio}) - \log(\text{CurrentRatio}))) \]

\[
\text{innerconvergence} = 1 \\
\text{inneriterations} = \text{inneriterations} + 1 \\
\]

\[
\text{if abs}(\log(P_{\text{inf}}/P_c) - \log(P_{\text{inf}}/\text{Previous})) \leq 0.4 \times 10^{-4} \\
\text{end} \\
\]

\[
\text{if abs}(\text{ChamberPressure} - (P_c + \rho_c \cdot u_c^2))/\text{ChamberPressure} \leq 2 \times 10^{-5} \\
\text{outerconvergence} = 1 \\
\text{else} \\
\text{P}_{\text{inf}} = P_{\text{inf}} \cdot \text{ChamberPressure}/P_{\text{inj}} \\
\text{end} \\
\text{outeriterations} = \text{outeriterations} + 1 \\
\]

\[
\text{if} \ (\text{ExpansionRatio} < 2) \ \&\& \ (\text{ExpansionRatio} > 1) \\
\text{P}_{\text{infOverP_e}} = P_{\text{infOverP_t}} + \exp(3.294 \cdot (\log(\text{ExpansionRatio}))^2 + 1.535 \cdot \log(\text{ExpansionRatio})) \\
\]

\[
\text{%CEA 6.21} \\
\]
elseif ExpansionRatio>=2;
P_infOverP_e=exp(gamma_t+1.4*log(ExpansionRatio)); % CEA 6.22
else
    disp('Error in IACEquilibriumFlow.m. Expansion Ratio must be greater than one.')
end
P_e=P_inf/P_infOverP_e;

% Iteration To Find Exit Pressure
convergence=0;
iterations=0;
while (convergence==0) && (iterations<=10)
    % Initial Estimates for other exit properties given pressure
    [~, ExitProperties]=ReducedCEA_SP(P_e,OxidizerToFuelRatio,ReactionNumber,s_t);
    M_e=ExitProperties(1);
    gamma_e=ExitProperties(5);
    a_e=ExitProperties(6);
    T_e=ExitProperties(7);
    h_e=ExitProperties(8);
    ExitProperties;
    u_e=sqrt(2*(h_inj-h_e));
    nocomplexity=isreal(u_e);
    if nocomplexity==0
        disp('Error in FACEquilibriumFlow.m. Combustion cannot be calculated at the specified combination of o/f and Ae/At.')
        return
    end
    % Pressure Correction
    CurrentRatio=(P_t*u_t/T_t)/(P_e*u_e/T_e)*(M_t/M_e); % CEA 6.12, 6.14
    P_infOverP_e=exp(log(P_infOverP_e)+(gamma_e*u_e^2/(u_e^2-a_e^2))*(log(ExpansionRatio)-log(CurrentRatio)));
    Previous=P_e;
    P_e=P_inf/P_infOverP_e;
    % Convergence Check
    if abs(log(P_inf/P_e)-log(P_inf/Previous))<=.5*10^-4
        convergence=1;
    end
    iterations=iterations+1;
end

% Final calculation of properties given last pressure correction
[ExitFractions, ExitProperties]=ReducedCEA_SP(P_e,OxidizerToFuelRatio,ReactionNumber,s_t);
M_e=ExitProperties(1);
Cp_e=ExitProperties(2);
Cv_e=ExitProperties(3);
gamma_e=ExitProperties(5);
a_e=ExitProperties(6);
h_e=ExitProperties(8);
s_e=ExitProperties(9);
u_e=sqrt(2*(h_inj-h_e));
Mach_e=u_e/a_e;
% Assembling output matrices
P_inj=ChamberPressure;
Thermo=[M_inj     M_c     M_t     M_e
Cp_inj    Cp_c    Cp_t    Cp_e
Cv_inj    Cv_c    Cv_t    Cv_e
gamma_inj gamma_c gamma_t gamma_e
a_inj     a_c     a_t     a_e
h_inj     h_c     h_t     h_e
s_inj     s_c     s_t     s_e
T_inj     T_c     T_t     T_e
P_inj     P_c     P_t     P_e
0         Mach_c  Mach_t  Mach_e];
Fractions=[InjectorFractions ContractionFractions ThroatFractions
ExitFractions];

%Calculation of Performance
rho=P_t/(UniversalGasConstant/M_t*T_t);
mdotOverA=rho*u_t;
rho=P_e/(UniversalGasConstant/M_e*T_e);
Isp=(u_e+(P_e-AmbientPressure)/(u_e*rho))/g;
FOverAe=Isp*mdotOverA*g;
cstar=P_inf/mdotOverA/g;
Cf=u_e/cstar/g;
Performance=[mdotOverA; FOverAe; Isp; cstar; Cf];

FrozenProperties.m

function [2]=FrozenProperties(Pressure, OxidizerToFuelRatio, ReactionNumber, n, Temperature)
n(length(n)+1)=1;
UniversalGasConstant=8314.51;

% Supported Reactions
% 1) H2(L)+O2(L)
% 2) RP-1(L)+O2(L)
% 3) NH3(L)+F2(L)

% Initialization
[~,~,SpeciesIndex,NumberOfSpecies,~,~,~]=ReactionProperties(ReactionNumber,OxidizerToFuelRatio);

for i=1:NumberOfSpecies

Properties(i,:)=ThermochemicalProperties(SpeciesIndex(i),Temperature,"Gas");
Gibbs(i)=(Properties(i,4)-Properties(i,5))+log(n(i)/n(NumberOfSpecies+1))+log(Pressure/100000);
end

Cpf=0;
MolecularWeight=0;
for i=1:NumberOfSpecies
Fractions(i,1)=n(i)/n(NumberOfSpecies+1);
MolecularWeight=MolecularWeight+Fractions(i)*Properties(i,1);
Cpf=Cpf+n(i)*Properties(i,3);
end
SpecificGasConstant=UniversalGasConstant/MolecularWeight;
% Calculation of the ratio of specific heats and speed of sound for
% frozen flow
Cp=Cpf*SpecificGasConstant;
Cv=Cp-n(NumberOfSpecies+1)*SpecificGasConstant;
Gamma=Cp/Cv;
Gamma_S=Gamma;
SpeedOfSound=sqrt(Gamma*SpecificGasConstant*Temperature);

Enthalpy=0;
Entropy=0;
for i=1:NumberOfSpecies
    Enthalpy=Enthalpy+n(i)*Properties(i,4)*SpecificGasConstant*Temperature;
    S=(Properties(i,5)-log(Fractions(i))-log(Pressure/100000))*SpecificGasConstant;
    Entropy=Entropy+n(i)*S;
end
x=[MolecularWeight,Cp,Cv,Gamma,Gamma_S,SpeedOfSound,Temperature, Enthalpy, Entropy];

IACEquilibriumFlow.m

function [Performance Thermo Fractions]=IACEquilibriumFlow(ChamberPressure, AmbientPressure, OxidizerToFuelRatio, ReactionNumber, ExpansionRatio)

UniversalGasConstant=8314.51;
g=9.80665;
%Combustion
[ChamberFractions,ChamberProperties]=ReducedCEA_HP(ChamberPressure,OxidizerToFuelRatio,ReactionNumber);
M_inf=ChamberProperties(1);
Cp_inf=ChamberProperties(2);
Cv_inf=ChamberProperties(3);
gamma_inf=ChamberProperties(5);
a_inf=ChamberProperties(6);
T_inf=ChamberProperties(7);
h_inf=ChamberProperties(8);
s_inf=ChamberProperties(9);
P_inf=ChamberPressure;
% Iteration to find throat pressure
P_infOverP_t=((gamma_inf+1)/2)^(gamma_inf/(gamma_inf-1)); %CEA 6.15
P_t=ChamberPressure/P_infOverP_t;
convergence=0;
iterations=0;
while (convergence==0) && (iterations<=3)
    [~,ThroatProperties]=ReducedCEA_SP(P_t,OxidizerToFuelRatio,ReactionNumber,s_inf);
    gamma_t=ThroatProperties(4);
end

59
\begin{verbatim}
a_t=ThroatProperties(6);
h_t=ThroatProperties(8);

u_t=sqrt(2*(h_inf-h_t));
Mach_t=u_t/a_t;
if abs((u_t^2-a_t^2)/(u_t^2))<=.4*10^-8
convergence=1;
else
P_t=P_t*((1+gamma_t*Mach_t^2)/(1+gamma_t));
end
iterations=iterations+1;
end

%Calculation of full throat properties
[ThroatFractions,ThroatProperties]=ReducedCEA_SP(P_t,OxidizerToFuelRatio,ReactionNumber,s_inf);
M_t=ThroatProperties(1);
Cp_t=ThroatProperties(2);
Cv_t=ThroatProperties(3);
gamma_t=ThroatProperties(5);
a_t=ThroatProperties(6);
T_t=ThroatProperties(7);
h_t=ThroatProperties(8);
s_t=ThroatProperties(9);

% Initial Estimate Of Exit Pressure
if (ExpansionRatio<2) && (ExpansionRatio>1)
P_infOverP_e=exp(log(P_infOverP_t)+sqrt(3.294*(log(ExpansionRatio))^2+1.535*log(ExpansionRatio)))); %CEA 6.21
else ExpansionRatio>=2;
P_infOverP_e=exp(gamma_t+1.4*log(ExpansionRatio)); % CEA 6.22
else
disp('Error in IACEquilibriumFlow.m. Expansion Ratio must be greater than one.');
end
P_e=P_inf/P_infOverP_e;

% Iteration To Find Exit Pressure
convergence=0;
iterations=0;
while (convergence==0) && (iterations<=10)

% Initial Estimates for other exit properties given pressure
[~,ExitProperties]=ReducedCEA_SP(P_e,OxidizerToFuelRatio,ReactionNumber,s_t);
gamma_e=ExitProperties(5);
M_e=ExitProperties(1);
a_e=ExitProperties(6);
T_e=ExitProperties(7);
h_e=ExitProperties(8);

u_e=sqrt(2*(h_inf-h_e));
nocomplexity=isreal(u_e);
\end{verbatim}
if nocomplexity==0
    break
end

% Pressure Correction
CurrentRatio=(P_t*u_t/T_t)/(P_e*u_e/T_e)*(M_t/M_e); % CEA 6.12, 6.14
P_infOverP_e=exp(log(P_inf/P_e)+(gamma_e*u_e^2/(u_e^2-a_e^2)))*(log(ExpansionRatio)-log(CurrentRatio));
Previous=P_e;
P_e=P_inf/P_infOverP_e;

% Convergence Check
if abs(log(P_inf/P_e)-log(P_inf/Previous))<=.4*10^-5
    convergence=1;
end
iterations=iterations+1;
end

if nocomplexity==1

    % Final calculation of properties given last pressure correction
    [ExitFractions, ExitProperties]=ReducedCEA_SP(P_e, OxidizerToFuelRatio, ReactionNumber, s_t);
        M_e=ExitProperties(1);
        Cp_e=ExitProperties(2);
        Cv_e=ExitProperties(3);
        gamma_e=ExitProperties(5);
        a_e=ExitProperties(6);
        h_e=ExitProperties(8);
        s_e=ExitProperties(9);
        u_e=sqrt(2*(h_inf-h_e));
        Mach_e=u_e/a_e;
        Fractions=[ChamberFractions ThroatFractions ExitFractions];
end

% If combustion can't be calculated due to low O/F, frozen flow is assumed.
if nocomplexity==0
    disp('Reverted to Frozen Flow')
    % Iteration to find Exit Pressure
    if (ExpansionRatio<2) && (ExpansionRatio>1)

        P_infOverP_e=P_infOverP_t+exp(sqrt(3.294*(log(ExpansionRatio))^2+1.535*log(ExpansionRatio)))); % CEA 6.21
    elseif ExpansionRatio>=2;
        P_infOverP_e=exp(gamma_t+1.4*log(ExpansionRatio)); % CEA 6.22
    end
    P_e=ChamberPressure/P_infOverP_e;
    T_e=T_inf/(1+(gamma_t-1)/2*2^2);

    [ExitProperties]=FrozenProperties(P_e, OxidizerToFuelRatio, ReactionNumber, ThroatFractions, T_e);
        s_e=ExitProperties(9);
        Cp_e=ExitProperties(2);
        convergence=0;
        iterations=0;
        while (convergence==0) && (iterations<=10)
            % Temperature Correction
            iterations=iterations+1;
        end

end
\[ i = 0; \]
\[ \text{while} \ (\text{abs}((s_{\text{inf}}-s_e)/Cp_e) > 10^{-4}) \land (i < 8) \]
\[ T_e = \exp(\log(T_e) + (s_{\text{inf}}-s_e)/Cp_e); \]

\[ [\text{ExitProperties}] = \text{FrozenProperties}(P_e, \text{OxidizerToFuelRatio}, \text{ReactionNumber}, \text{ThroatFractions}, T_e); \]
\[ s_e = \text{ExitProperties}(9); \]
\[ Cp_e = \text{ExitProperties}(2); \]
\[ i = i + 1; \]
\[ \text{end} \]

\[ [\text{ExitProperties}] = \text{FrozenProperties}(P_e, \text{OxidizerToFuelRatio}, \text{ReactionNumber}, \text{ThroatFractions}, T_e); \]
\[ a_e = \text{ExitProperties}(6); \]
\[ h_e = \text{ExitProperties}(8); \]
\[ s_e = \text{ExitProperties}(9); \]
\[ u_e = \sqrt{2*(h_{\text{inf}}-h_e)}; \]
\[ \text{CurrentRatio} = (P_t*u_t/T_t)/(P_e*u_e/T_e); \]
\[ \% \text{Pressure Correction} \]
\[ P_{\text{infOverP_e}} = \exp(\log(P_{\text{infOverP_e}}) + (\gamma_t*u_e^2/(u_e^2-a_e^2))*(\log(\text{ExpansionRatio}) - \log(\text{CurrentRatio}))); \]
\[ \text{Previous} = P_e; \]
\[ P_e = \text{ChamberPressure}/P_{\text{infOverP_e}}; \]
\[ \text{if abs} (\log(\text{ChamberPressure}/P_e) - \log(\text{ChamberPressure}/\text{Previous})) <= 10^{-4} \]
\[ \text{convergence} = 1; \]
\[ \text{end} \]
\[ \text{iterations} = \text{iterations} + 1; \]
\[ \text{end} \]
\[ \% \text{Temperature Correction} \]
\[ i = 0; \]
\[ \text{while} \ (\text{abs}((s_{\text{inf}}-s_e)/Cp_e) > 10^{-4}) \land (i < 8) \]
\[ T_e = \exp(\log(T_e) + (s_{\text{inf}}-s_e)/Cp_e); \]

\[ [\text{ExitProperties}] = \text{FrozenProperties}(P_e, \text{OxidizerToFuelRatio}, \text{ReactionNumber}, \text{ThroatFractions}, T_e); \]
\[ s_e = \text{ExitProperties}(8); \]
\[ Cp_e = \text{ExitProperties}(2); \]
\[ i = i + 1; \]
\[ \text{end} \]
\[ \% \text{Final property evaluation at exit} \]

\[ [\text{ExitProperties}] = \text{FrozenProperties}(P_e, \text{OxidizerToFuelRatio}, \text{ReactionNumber}, \text{ThroatFractions}, T_e); \]
\[ M_e = \text{ExitProperties}(1); \]
\[ Cp_e = \text{ExitProperties}(2); \]
\[ Cv_e = \text{ExitProperties}(3); \]
\[ \gamma_e = \text{ExitProperties}(5); \]
\[ a_e = \text{ExitProperties}(6); \]
\[ h_e = \text{ExitProperties}(8); \]
\[ s_e = \text{ExitProperties}(9); \]
\[ u_e = \sqrt{2*(h_{\text{inf}}-h_e)}; \]
\[ \text{Mach}_e = u_e/a_e; \]
\[ \text{Fractions} = [\text{ChamberFractions ThroatFractions ThroatFractions}]; \]
% Assembling output matrices
P_inf=ChamberPressure;
Thermo=[M_inf M_t M_e;
        Cp_inf Cp_t Cp_e;
        Cv_inf Cv_t Cv_e;
        gamma_inf gamma_t gamma_e;
        a_inf a_t a_e;
        h_inf h_t h_e;
        s_inf s_t s_e;
        T_inf T_t T_e;
        P_inf P_t P_e;
0        Mach_t Mach_e];

% Calculation of Performance
rho=P_t/(UniversalGasConstant/M_t*T_t);
mdotOverA=rho*u_t;
FOverAe=mdotOverA*u_e+(P_e-AmbientPressure);
Isp=FOverAe/mdotOverA/g;
cstar=ChamberPressure/mdotOverA/g;
Cf=u_e/cstar/g;
Performance=[mdotOverA; FOverAe; Isp; cstar; Cf];

IACFrozenFlow.m
function [Performance Thermo Fractions]=IACFrozenFlow(ChamberPressure, AmbientPressure, OxidizerToFuelRatio, ReactionNumber, ExpansionRatio)

UniversalGasConstant=8314.51;
g=9.80665;

% Combustion
[ChamberFractions,ChamberProperties]=ReducedCEA_HP(ChamberPressure,OxidizerToFuelRatio,ReactionNumber);
M_inf=ChamberProperties(1);
Cp_inf=ChamberProperties(2);
Cv_inf=ChamberProperties(3);
gamma_inf=ChamberProperties(5);
a_inf=ChamberProperties(6);
T_inf=ChamberProperties(7);
h_inf=ChamberProperties(8);
s_inf=ChamberProperties(9);
% Iteration to find throat pressure
P_infOverP_t=((gamma_inf+1)/2)^(gamma_inf/(gamma_inf-1)); % CEA 6.15
P_t=ChamberPressure/P_infOverP_t;
convergence=0;
iterations=0;
while (convergence==0) && (iterations<=4)

[~,ThroatProperties]=ReducedCEA_SP(P_t,OxidizerToFuelRatio,ReactionNumber,s_inf);
gamma_t=ThroatProperties(4);
a_t=ThroatProperties(6);
h_t=ThroatProperties(8);

u_t=sqrt(2*(h_inf-h_t));
Mach_t=u_t/a_t;
if abs((u_t^2-a_t^2)/(u_t^2))<=.4*10^-8
    convergence=1;
else
    P_t=P_t*((1+gamma_t*Mach_t^2)/(1+gamma_t));
end
iterations=iterations+1;
end

%Calculation of full throat properties
[ThroatFractions,ThroatProperties]=ReducedCEA_SP(P_t,OxidizerToFuelRatio,ReactionNumber,s_inf);
M_t=ThroatProperties(1);
Cp_t=ThroatProperties(2);
Cv_t=ThroatProperties(3);
gamma_t=ThroatProperties(5);
a_t=ThroatProperties(6);
T_t=ThroatProperties(7);
h_t=ThroatProperties(8);
s_t=ThroatProperties(9);

% Iteration to find Exit Pressure
if (ExpansionRatio<2) && (ExpansionRatio>1)
    P_infOverP_e=P_infOverP_t+exp(sqrt(3.294*(log(ExpansionRatio))^2+1.535*log(ExpansionRatio))));  % CEA 6.21
elseif ExpansionRatio>=2;
    P_infOverP_e=exp(gamma_t+1.4*log(ExpansionRatio));  % CEA 6.22
else
    disp('Error in FrozenFlow.m. Expansion Ratio must be greater than one.')
end
P_e=ChamberPressure/P_infOverP_e;
T_e=T_inf/(1+(gamma_t-1)/2*2^2);
[ExitProperties]=FrozenProperties(P_e,OxidizerToFuelRatio,ReactionNumber,ThroatFractions,T_e);
s_e=ExitProperties(9);
Cp_e=ExitProperties(2);
convergence=0;
iterations=0;
while (convergence==0) && (iterations<=10)
    % Temperature Correction
    i=0;
    while (abs((s_inf-s_e)/Cp_e)>.5*10^-4) && (i<8)
        T_e=exp(log(T_e)+(s_inf-s_e)/Cp_e);
    end
    [ExitProperties]=FrozenProperties(P_e,OxidizerToFuelRatio,ReactionNumber,ThroatFractions,T_e);
s_e=ExitProperties(9);
    Cp_e=ExitProperties(2);
i=i+1;
end

a_e=ExitProperties(6);
h_e=ExitProperties(8);
s_e=ExitProperties(9);
u_e = sqrt(2*(h_inf - h_e));
CurrentRatio = (P_t*u_t/T_t)/(P_e*u_e/T_e); % CEA 6.12, 6.14

% Pressure Correction
P_infOverP_e = exp(log(P_infOverP_e) + (gamma_t*u_e^2/(u_e^2 - a_e^2))*(log(ExpansionRatio) - log(CurrentRatio)));
Previous = P_e;
P_e = ChamberPressure/P_infOverP_e;
if abs(log(ChamberPressure/P_e) - log(ChamberPressure/Previous)) <= .5*10^-4
  convergence = 1;
end
iterations = iterations + 1;
end

% Temperature Correction
i = 0;
while (abs(((s_inf - s_e)/Cp_e) > .5*10^-4) && (i < 8))
  T_e = exp(log(T_e) + (s_inf - s_e)/Cp_e);
end

[ExitProperties] = FrozenProperties(P_e, OxidizerToFuelRatio, ReactionNumber, ThroatFractions, T_e);
s_e = ExitProperties(8);
Cp_e = ExitProperties(2);
i = i + 1;
end

% Final property evaluation at exit
[ExitProperties] = FrozenProperties(P_e, OxidizerToFuelRatio, ReactionNumber, ThroatFractions, T_e);
M_e = ExitProperties(1);
Cp_e = ExitProperties(2);
Cv_e = ExitProperties(3);
gamma_e = ExitProperties(5);
a_e = ExitProperties(6);
h_e = ExitProperties(8);
s_e = ExitProperties(9);
u_e = sqrt(2*(h_inf - h_e));
Mach_e = u_e/a_e;

% Assembling output matrices
Thermo = [M_inf M_t M_e;
          Cp_inf Cp_t Cp_e;
          Cv_inf Cv_t Cv_e;
          gamma_inf gamma_t gamma_e;
          a_inf a_t a_e;
          h_inf h_t h_e;
          s_inf s_t s_e;
          T_inf T_t T_e;
          ChamberPressure P_t P_e
          0 Mach_t Mach_e];
Fractions = [ChamberFractions ThroatFractions ThroatFractions];

% Calculation of Performance
rho = P_t/(UniversalGasConstant/M_t*T_t);
mdotOverA = rho*u_t;
\[ F_{\text{OverAe}} = \dot{m}_{\text{OverA}} u_e + (P_e - \text{Ambient Pressure}); \]
\[ I_{\text{sp}} = \frac{F_{\text{OverAe}}}{\dot{m}_{\text{OverA}}}/g; \]
\[ c_{\text{star}} = \frac{\text{Chamber Pressure}}{\dot{m}_{\text{OverA}}}/g; \]
\[ C_f = \frac{u_e}{c_{\text{star}}}/g; \]
\[ \text{Performance} = \begin{bmatrix} \dot{m}_{\text{OverA}}; F_{\text{OverAe}}; I_{\text{sp}}; c_{\text{star}}; C_f \end{bmatrix}; \]

**InitialPosition.m**

```matlab
function [Position] = InitialPosition(xhigh, xlow)
    rng('shuffle');
    Position = xlow + rand*(xhigh - xlow);
```

**InitialVelocities.m**

```matlab
function [Velocities] = InitialVelocities
    Velocities = zeros(3,1);
```

**LocalBestPositions.m**

```matlab
function [LocalBestPositions] = LocalBestPositions(Positions, PSOVector)
    FitnessIndex = PSOVector(1);
    SwarmSize = PSOVector(2);
    LocalRadius = PSOVector(7);
    LocalBestPositions = zeros(FitnessIndex, SwarmSize);
    for i = 1:SwarmSize
        if (i > (LocalRadius)) && (i <= (SwarmSize - LocalRadius))
            for j = (i - LocalRadius): (i + LocalRadius)
                if Positions(FitnessIndex, j) > LocalBestPositions(FitnessIndex, i)
                    LocalBestPositions(:, i) = Positions(:, j);
                end
            end
        elseif i <= LocalRadius
            for j = 1: (i + LocalRadius)
                if Positions(FitnessIndex, j) > LocalBestPositions(FitnessIndex, i)
                    LocalBestPositions(:, i) = Positions(:, j);
                end
            end
        elseif i > (SwarmSize - LocalRadius)
            for j = (i - LocalRadius): SwarmSize
                if Positions(FitnessIndex, j) > LocalBestPositions(FitnessIndex, i)
                    LocalBestPositions(:, i) = Positions(:, j);
                end
            end
        end
    end
```

**PositionVelocityUpdate.m**

```matlab
function [Positions, Velocities] = PositionVelocityUpdate(OldVelocities, OldPositions, ParticleBestPositions, LocalBestPositions, GlobalBestPositions, PSOVector, Constraints)
    Positions(:, i) = [MixtureRatio; ExpansionRatio; ContractionRatio; Isp];
    PSOVector = [FitnessIndex; SwarmSize; GlobalTrust; SelfTrust; Inertia; Constriction; LocalRadius; LocalGlobalBalance];
    Constraints = [LowestMixtureRatio, HighestMixtureRatio; LowestExpansionRatio, HighestExpansionRatio; LowestContractionRatio, HighestContractionRatio];
```
FitnessIndex=PSOVector(1);
SwarmSize=PSOVector(2);
GlobalTrust=PSOVector(3);
SelfTrust=PSOVector(4);
Inertia=PSOVector(5);
Constriction=PSOVector(6);
LocalGlobalBalance=PSOVector(8);
UnifiedPSOVelocities=zeros(FitnessIndex-1,SwarmSize);
Positions=zeros(FitnessIndex,SwarmSize);
for i=1:SwarmSize
  for j=1:FitnessIndex-1
    GlobalPSOVelocity=Constriction*(OldVelocities(j,i)+SelfTrust*rand*(ParticleBestPositions(j,i)-OldPositions(j,i))+GlobalTrust*rand*(GlobalBestPositions(j)-OldPositions(j,i)));
    LocalPSOVelocity=Constriction*(OldVelocities(j,i)+SelfTrust*rand*(ParticleBestPositions(j,i)-OldPositions(j,i))+GlobalTrust*rand*(LocalBestPositions(j,i)-OldPositions(j,i)));
    UnifiedPSOVelocities(j,i)=(1-
LocalGlobalBalance)*LocalPSOVelocity+LocalGlobalBalance*GlobalPSOVelocity;
    Positions(j,i)=OldPositions(j,i)+UnifiedPSOVelocities(j,i);
    if Positions(j,i)<Constraints(j,1)
      Positions(j,i)=Constraints(j,1);
    elseif Positions(j,i)>Constraints(j,2)
      Positions(j,i)=Constraints(j,2);
    end
  end
end
Velocities=UnifiedPSOVelocities;

% clear all
close all
tic
% Given 
ChamberPressure=2*10^6;
AmbientPressure=0;
ReactionNumber=1;
HighestMixtureRatio=7.5;
LowestMixtureRatio=3.5;
HighestExpansionRatio=70;
LowestExpansionRatio=2;
HighestContractionRatio=3.5;
LowestContractionRatio=1.5;
% PSO Parameters 
GlobalTrust=2.05;
SelfTrust=2.05;
Inertia=1;  % Not currently in use.
Constriction=0.729;
LocalRadius=2;
LocalGlobalBalance=.5;
SwarmSize=12;
MaximumIterations=25;
FitnessIndex=4;

%%%%%%%%%%%%%%%%%%
% Initialization 
%%%%%%%%%%%%%%%%%%
PSOVector=[FitnessIndex;SwarmSize;GlobalTrust;SelfTrust;Inertia;Constriction;
LocalRadius;LocalGlobalBalance];
Constraints=[LowestMixtureRatio, HighestMixtureRatio; LowestExpansionRatio, HighestExpansionRatio;
LowestContractionRatio, HighestContractionRatio];
Positions=zeros(FitnessIndex,SwarmSize);
Velocities=zeros(FitnessIndex-1,SwarmSize);
ParticleBestPositions=zeros(FitnessIndex,SwarmSize);
GlobalBestPositions=zeros(FitnessIndex,1);
time=.85*SwarmSize*MaximumIterations/60;
string=['Estimated run time: ',num2str(time),' hours'];
disp(string)
h=waitbar(0,'Initializing Particle Swarm...');
for i=1:SwarmSize
    MixtureRatio=InitialPosition(HighestMixtureRatio,LowestMixtureRatio);
    ExpansionRatio=InitialPosition(HighestExpansionRatio,LowestExpansionRatio);
    ContractionRatio=InitialPosition(HighestContractionRatio,LowestContractionRatio);
    Performance=FACEquilibriumFlow(ChamberPressure,AmbientPressure,MixtureRatio,
                                 ReactionNumber,ExpansionRatio,ContractionRatio);
    Isp=Performance(3);
    Positions(:,i)=[MixtureRatio;ExpansionRatio;ContractionRatio;Isp];
    Velocities(:,i)=[0;0;0];
    ParticleBestPositions(:,i)=Positions(:,i);
    if Positions(FitnessIndex,i)>GlobalBestPositions(FitnessIndex)
        GlobalBestPositions=Positions(:,i);
    end
    waitbar(i/SwarmSize)
end
close(h)
LocalBestPositions=LocalBestPositions(Positions, PSOVector);

string=['Iteration 0.png'];
w=figure;
hold on
scatter3(Positions(1,:),Positions(2,:),Positions(3,:),'*');
xlabel('Mixture Ratio')
ylabel('Expansion Ratio')
zlabel('Contraction Ratio')
grid on
view(135,45)
axis([LowestMixtureRatio HighestMixtureRatio LowestExpansionRatio
     HighestExpansionRatio LowestContractionRatio HighestContractionRatio])
saveas(w,string,'png');
close(w);
convergence=0;
iterations=0;
Completion=0;
Positions;
h=waitbar(0,'Swarming in Progress...');
while (iterations<MaximumIterations)
    OldVelocities=Velocities;
    OldPositions=Positions;
    [Positions,Velocities]=PositionVelocityUpdate(OldVelocities,OldPositions,
                                                ParticleBestPositions,LocalBestPositions,
                                                GlobalBestPositions,PSOVector,Constraints);
    for i=1:SwarmSize
        Completion=(iterations+i/SwarmSize)/MaximumIterations;
        waitbar(Completion)
        Performance=FACEquilibriumFlow(ChamberPressure,AmbientPressure,
                                           Positions(1,i),ReactionNumber,
                                           Positions(2,i),Positions(3,i));
        Isp=Performance(3);
        Positions(FitnessIndex,i)=Isp;
        if Positions(FitnessIndex,i)>ParticleBestPositions(FitnessIndex,i)
            ParticleBestPositions(:,i)=Positions(:,i);
        end
        clear LocalBestPositions
        LocalBestPositions=LocalBestPositions(Positions, PSOVector);
        if Positions(FitnessIndex,i)>GlobalBestPositions(FitnessIndex)
            GlobalBestPositions=Positions(:,i);
        end
    end
end
Positions;
iterations=iterations+1;
string=['Iteration ',num2str(iterations),'.png'];
w=figure;
hold on
scatter3(Positions(1,:),Positions(2,:),Positions(3,:),'*');
xlabel('Mixture Ratio')
ylabel('Expansion Ratio')
zlabel('Contraction Ratio')
grid on
view(135,45)
axis([LowestMixtureRatio HighestMixtureRatio LowestExpansionRatio
     HighestExpansionRatio LowestContractionRatio HighestContractionRatio])
saveas(w,string,'png');
close(w);
end
close(h)
Time=toc;
TotalTime=Time/3600;
string=['Total time elapsed : ',num2str(TotalTime),' hours'];
disp(string)
string=['Time per iteration : ',num2str(Time/MaximumIterations/60),' minutes'];
disp(string)
string=['Time per evaluation: ','num2str(Time/MaximumIterations/SwarmSize/60),' minutes'];
disp(string)

GlobalBestPositions

**ReactionProperties.m**

*Function [a b_0 SpeciesIndex NumberOfSpecies NumberOfElements h_0 Enthalpy]=ReactionProperties(ReactionNumber,OxidizerToFuelRatio)*

%ReactionNumber=1;
%OxidizerToFuelRatio=5.5;

UniversalGasConstant=8314.51;
% Considering 100 kg of total reactant,
FuelMass=1000/(1+OxidizerToFuelRatio);
OxidizerMass=FuelMass*OxidizerToFuelRatio;

if ReactionNumber==1
  %order = [O OH H2O H2O2 O3 H H2]
  NumberOfSpecies=9;
  NumberOfElements=2;
  a=[0 1 2 0 1 2 0 1 2; %H
    1 1 2 2 2 3 0 0]; %O
  %Reactants
  FuelProperties=ThermochemicalProperties('H2(L)',20.27,'Liquid');
  OxidizerProperties=ThermochemicalProperties('O2(L)',90.17,'Liquid');
  nFuel=FuelMass/FuelProperties(1);
  nOxidizer=OxidizerMass/OxidizerProperties(1);
  NumberOfH=2*nFuel;
  NumberOfO=2*nOxidizer;
  %Products
  Order=['O   '; 'OH  '; 'H2O ';
        'O2  '; 'H2O2'; 'H2O2';
        'O3  '; 'H   '; 'H2  '];
  SpeciesIndex=cellstr(Order);
  b_0(1,1)=NumberOfH;
  b_0(2,1)=NumberOfO;
elseif ReactionNumber==2
  %order = [C CO CO2 COOH H HCO HO2 H2 HCHO HCOOH H2O H2O2 O OH O2 O3]
  NumberOfSpecies=12;
  NumberOfElements=3;
  a=[1 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0; %C
     0 1 2 2 0 1 2 0 1 2 1 2 1 1 2 3; %O
     0 0 1 1 1 1 2 2 2 2 0 1 0 0]; %H
end
%Reactants
FuelProperties=ThermochemicalProperties('RP-1',298.15,'Liquid');
OxidizerProperties=ThermochemicalProperties('O2(L)',90.17,'Liquid');
nFuel=FuelMass/FuelProperties(1);
nOxidizer=OxidizerMass/OxidizerProperties(1);
NumberOfC=nFuel;
NumberOfH=1.95*nFuel;
NumberOfO=2*nOxidizer;
%Products
Order=['C ','CO ','CO2 ','COOH ',
'H ',
'HCO ','HO2 ','H2 ','HCHO ',
'H2O ',
'O ',
'O2 ','O3 '];
SpeciesIndex=cellstr(Order);
b_0(1,1)=NumberOfC;
b_0(2,1)=NumberOfH;
b_0(3,1)=NumberOfO;
elseif ReactionNumber==3
%order = [F F2 H HF H2 H2F2 H3F3 N NF NH NHF NH2 NH2F NH3 N2 N2H2]
NumberOfSpecies=16;
NumberOfElements=3;
a= [1 2 0 1 0 2 3 0 1 0 1 0 1 0 0 0; 
0 0 1 1 2 2 3 0 0 1 1 2 2 3 0 2; 
0 0 0 0 0 0 0 1 1 1 1 1 1 2 2 2];
%Reactants
OxidizerProperties=ThermochemicalProperties('F2(L)',85.02,'Liquid');
FuelProperties=ThermochemicalProperties('NH3(L)',239.720,'Liquid');
nFuel=FuelMass/FuelProperties(1);
nOxidizer=OxidizerMass/OxidizerProperties(1);
NumberOfF=2*nOxidizer;
NumberOfN=nFuel;
NumberOfH=3*nFuel;
Order=['F ',
'F2 ',
'H ',
'HF ',
'H2 ',
'H2F2 ',
'H3F3 ',
'N ',
'NF ',
'NH '];
'NHF';
'NH2';
'NH2F';
'NH3';
'N2';
'N2H2'];
SpeciesIndex=cellstr(Order);
b_0(1,1)=NumberOfOfF;
b_0(2,1)=NumberOfOfH;
b_0(3,1)=NumberOfOfN;
else
    disp('ERROR! Chemical reaction not supported.'); 
end

HF=FuelProperties(2)/(UniversalGasConstant*FuelProperties(1));
HO=OxidizerProperties(2)/(UniversalGasConstant*OxidizerProperties(1));
h_0=(HF+OxidizerToFuelRatio*HO)/(1+OxidizerToFuelRatio)*1000000;
Enthalpy=(nFuel*FuelProperties(2)+nOxidizer*OxidizerProperties(2));

ReducedCEA_HP.m

function [Fractions x]=ReducedCEA_HP(Pressure, OxidizerToFuelRatio, ReactionNumber)

Temperature=3800;
UniversalGasConstant=8314.51;
ZERO=10^-8;

% Supported Reactions
% 1) H2(L)+O2 (L)
% 2) RP-1(L)+O2 (L)
% 3) NH3(L)+F2(L)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[a,b_0,SpeciesIndex,NumberOfSpecies,NumberOfElements, h_0, Enthalpy]=ReactionProperties(ReactionNumber,OxidizerToFuelRatio);

n(NumberOfSpecies+1)=0;
for i=1:NumberOfSpecies
    n(i)=.1/NumberOfSpecies;
end
n(NumberOfSpecies+1)=0.1; %total number of moles

iterations=0;
convergence=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iteration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while convergence==0
    for i=1:NumberOfSpecies

Properties(i,:) = ThermochemicalProperties(SpeciesIndex(i), Temperature, 'Gas');

Gibbs(i) = (Properties(i,4) - Properties(i,5)) + log(n(i)/n(NumberOfSpecies+1)) + log(Pressure/100000);
end

b = zeros(NumberOfElements,1);
for i=1:NumberOfElements
    for j=1:NumberOfSpecies
        b(i) = b(i) + a(i,j)*n(j);
    end
end

% Construction of Matrices
A = zeros(NumberOfElements+2);

% CEA 2.24
for k=1:NumberOfElements
    for i=1:NumberOfElements
        for j=1:NumberOfSpecies
            A(k,i) = A(k,i) + a(k,j)*a(i,j)*n(j);  % pi terms
        end
    end
    SUM = 0;
    for j=1:NumberOfSpecies
        A(k,NumberOfElements+1) = A(k,NumberOfElements+1) + a(k,j)*n(j);
        % Delta ln (n) term
        SUM = SUM + a(k,j)*n(j)*Gibbs(j);
    end
    A(k,NumberOfElements+2) = A(k,NumberOfElements+2) + a(k,j)*n(j)*Properties(j,4);
end

B(k,1) = b_0(k) - b(k) + SUM;
end

% CEA 2.26
for i=1:NumberOfElements
    for j=1:NumberOfSpecies
        A(NumberOfElements+1,i) = A(NumberOfElements+1,i) + a(i,j)*n(j);
    end
end

ASUM1 = 0;
ASUM2 = 0;
BSUM1 = 0;
BSUM2 = 0;
for j=1:NumberOfSpecies
    ASUM1 = ASUM1 + n(j);
    ASUM2 = ASUM2 + n(j)*Properties(j,4);
    BSUM1 = BSUM1 + n(j);
    BSUM2 = BSUM2 + n(j)*Gibbs(j);
end

A(NumberOfElements+1,NumberOfElements+1) = ASUM1 - n(NumberOfSpecies+1);
A(NumberOfElements+1,NumberOfElements+2) = ASUM2;
B(NumberOfElements+1,1) = n(NumberOfSpecies+1) - BSUM1 + BSUM2;
% CEA 2.27
for i=1:NumberOfElements
    for j=1:NumberOfSpecies
        A(NumberOfElements+2,i)=A(NumberOfElements+2,i)+a(i,j)*n(j)*Properties(j,4);
    end
end

h=0;
BSUM=0;
for j=1:NumberOfSpecies
    A(NumberOfElements+2,NumberOfElements+1)=A(NumberOfElements+2,NumberOfElements+1)+n(j)*Properties(j,4);
    A(NumberOfElements+2,NumberOfElements+2)=A(NumberOfElements+2,NumberOfElements+2)+n(j)*(Properties(j,3)+(Properties(j,4))^2);
    h=h+n(j)*Properties(j,4);
    BSUM=BSUM+n(j)*Properties(j,4)*Gibbs(j);
end
B(NumberOfElements+2,1)=h_0-h*Temperature+BSUM*Temperature;
A(NumberOfElements+2,:) = A(NumberOfElements+2,:) * Temperature;

% Matrix Solution
Corrections=A\B;

CF1=0;
CF2=0;
for j=1:NumberOfSpecies
    SUM=0;
    for i=1:NumberOfElements
        SUM=SUM+a(i,j)*Corrections(i);
    end
    %Calculation of Delta ln (n_j)
    LogChange(j)=-Gibbs(j)+Corrections(NumberOfElements+1)+SUM+Properties(j,4)*Corrections(NumberOfElements+2); % CEA 2.18
    %Calculation of Control Factors
    CF1(j)=2/max([5*abs(Corrections(NumberOfElements+1)),5*abs(LogChange(j)))];
    if log(n(j)/n(NumberOfSpecies+1))<log(ZERO)
        CF2(j)=abs((-log(n(j)/n(NumberOfSpecies+1)))/LogChange(j)-Corrections(NumberOfElements+1)));
    else
        CF2(j)=1;
    end
end

LogChange(NumberOfSpecies+1)=Corrections(NumberOfElements+1);

ControlFactor1=min(CF1);
ControlFactor2=min(CF2);
ControlFactor=min([1,ControlFactor1,ControlFactor2]);
CF(iterations+1)=ControlFactor;

% Corrections
for j=1:NumberOfSpecies+1
    Previous(j)=n(j);  %#ok<*SAGROW>
    n(j)=exp(log(Previous(j))+ControlFactor*LogChange(j));

    if abs(n(j))/abs(Previous(j))>exp(2)
        n(j)=Previous(j)*exp(2);
    elseif abs(Previous(j))/abs(n(j))>exp(2)
        n(j)=Previous(j)*exp(-2);
    end

    if j==NumberOfSpecies+1 && abs(n(j))/abs(Previous(j))>exp(.4)
        n(j)=Previous(j)*exp(.4);
    elseif j==NumberOfSpecies+1 && abs(Previous(j))/abs(n(j))>exp(.4)
        n(j)=Previous(j)*exp(-.4);
    end
end

% Temperature Correction
PT=Temperature;
Temperature=exp(log(PT)+ControlFactor*Corrections(NumberOfElements+2));
if Temperature/PT>exp(.4)
    Temperature=PT*exp(.4);
elseif PT/Temperature>exp(.4)
    Temperature=PT*exp(-.4);
end

% Check for Convergence
SUM=0; CompositionConvergence=1; MassBalance=1; TemperatureConvergence=1;
for j=1:NumberOfSpecies
    SUM=SUM+n(j);
end
for j=1:NumberOfSpecies+1 % Composition convergence
    if n(j)*abs(LogChange(j))/SUM>0.5*10^-8
        CompositionConvergence=0;
    end
end

for i=1:NumberOfElements % Mass balance
    SUM=0;
    for j=1:NumberOfSpecies
        SUM=SUM+a(i,j)*n(j);
    end
    if abs(b_0(i)-SUM)>max(b_0)*10^-8
        MassBalance=0;
    end
end

if abs(Corrections(NumberOfElements+2))>10^-4
    TemperatureConvergence=0;
end
iterations=iterations+1;
if iterations>50
  convergence=1;
elseif CompositionConvergence==1 && MassBalance==1 &&
  TemperatureConvergence==1
  convergence=1;
end
end

% Calculation of Mole Fractions, Molecular Weight, and Cpf
Cpf=0;
MolecularWeight=0;
for i=1:NumberOfSpecies
  Fractions(i,1)=n(i)/n(NumberOfSpecies+1);
  MolecularWeight=MolecularWeight+Fractions(i)*Properties(i,1);
  Cpf=Cpf+n(i)*Properties(i,3);
end
SpecificGasConstant=UniversalGasConstant/MolecularWeight;

% Calculation of the ratio of specific heats and speed of sound for % equilibrium flow.

% Tempeature Derivatives
A=zeros(NumberOfElements+1);
B=zeros(NumberOfElements+1,1);
% CEA (2.56)
for k=1:NumberOfElements
  for i=1:NumberOfElements
    for j=1:NumberOfSpecies
      A(k,i)=A(k,i)+a(k,j)*a(i,j)*n(j);
    end
  end
  for j=1:NumberOfSpecies
    A(k,NumberOfElements+1)=A(k,NumberOfElements+1)+a(k,j)*n(j);
    B(k)=B(k)-a(k,j)*n(j)*Properties(j,4);
  end
end
% CEA (2.59)
k=NumberOfElements+1;
for i=1:NumberOfElements
  for j=1:NumberOfSpecies
    A(k,i)=A(k,i)+a(i,j)*n(j);
  end
end
for j=1:NumberOfSpecies
  B(k)=B(k)-n(j)*Properties(j,4);
end
%Solving system of equations
TemperatureDerivatives=A\B;
% CEA (2.59)
C=zeros(3,1);
for i=1:NumberOfElements
for j=1:NumberOfSpecies
    C(1)=C(1)+a(i,j)*n(j)*Properties(j,4)*TemperatureDerivatives(i);
end

for j=1:NumberOfSpecies
    C(2)=C(2)+n(j)*Properties(j,4)*TemperatureDerivatives(NumberOfElements+1);
    C(3)=C(3)+n(j)*(Properties(j,4))^2;
end

Cp=(Cpf+sum(C))*UniversalGasConstant/10^3;%divided by mass of 10^3 to give J/kg-K

% Pressure Derivatives
B=zeros(NumberOfElements+1,1);
%CEA (2.64)
for k=1:NumberOfElements
    for j=1:NumberOfSpecies
        B(k)=B(k)+a(k,j)*n(j);
    end
end
B(NumberOfElements+1)=n(NumberOfSpecies+1);
%Solving system of equations
PressureDerivatives=A\B;
%Finding Thermo properties
ConstantPressureDerivative=1+TemperatureDerivatives(NumberOfElements+1);
ConstantTemperatureDerivative=1-PressureDerivatives(NumberOfElements+1);
Cv=Cp-n(NumberOfSpecies+1)*UniversalGasConstant*(ConstantPressureDerivative)^2/ConstantTemperatureDerivative/10^3;%divided by mass of 10^3 to give J/kg-K

ConstantEntropyDerivative=Cp/(Cv*ConstantTemperatureDerivative);
Gamma=Cp/Cv;
Gamma_S=Gamma/ConstantTemperatureDerivative;
SpeedOfSound=sqrt(Gamma_S*SpecificGasConstant*Temperature);

Entropy=0;
for i=1:NumberOfSpecies
    S=(Properties(i,5)-log(Fractions(i))-log(Pressure/100000))*UniversalGasConstant;
    Entropy=Entropy+n(i)*S/1000;
end

% ReducedCEA_SP.m
function [Fractions x n]=ReducedCEA_SP(Pressure, OxidizerToFuelRatio, ReactionNumber, Entropy)
%Test values

%OxidizerToFuelRatio=5.5;
%Pressure=206.429*10^5;
%ReactionNumber=1;
%Frozen=0;
%Entropy=17853.4651435026;

Temperature=3800;
UniversalGasConstant=8314.51;
ZERO=10^-8;

% Supported Reactions
% 1) H2(L)+O2 (L)
% 2) RP-1 (L)+O2 (L)
% 3) NH3(L)+F2 (L)

[a,b_0,SpeciesIndex,NumberOfSpecies,NumberOfElements,~,~]=ReactionProperties(ReactionNumber,OxidizerToFuelRatio);

for i=1:NumberOfSpecies
    n(i)=.1/NumberOfSpecies;
end
n(NumberOfSpecies+1)=0.1; %total number of moles

s_0=Entropy*1000/UniversalGasConstant;

iterations=0;
convergence=0;

% Iteration
while convergence==0
    for i=1:NumberOfSpecies
        Properties(i,:)=ThermochemicalProperties(SpeciesIndex(i),Temperature,'Gas');
        Gibbs(i)=(Properties(i,4)-Properties(i,5))+log(n(i)/n(NumberOfSpecies+1))+log(Pressure/100000);
    end
    b=zeros(NumberOfElements,1);
    for i=1:NumberOfElements
        for j=1:NumberOfElements
            b(i)=b(i)+a(i,j)*n(j);
        end
    end

    % Construction of Matrices
    A=zeros(NumberOfElements+2);
    CEA 2.24
    for k=1:NumberOfElements
        for i=1:NumberOfElements
            for j=1:NumberOfSpecies

78
\[ A(k,i) = A(k,i) + a(k,j) \cdot a(i,j) \cdot n(j); \quad \% \text{pi terms} \]

end

end

SUM = 0;
for j = 1:NumberOfSpecies
    \[ A(k,\text{NumberOfElements}+1) = A(k,\text{NumberOfElements}+1) + a(k,j) \cdot n(j); \]
\% \delta ln (n) term
    SUM = SUM + a(k,j) \cdot n(j) \cdot \text{Gibbs}(j);
end

A(k,\text{NumberOfElements}+2) = A(k,\text{NumberOfElements}+2) + a(k,j) \cdot n(j) \cdot \text{Properties}(j,4);
end

B(k,1) = b_0(k) - b(k) + SUM;
end

\% \text{CEA 2.26}
for i = 1:NumberOfElements
    for j = 1:NumberOfSpecies
        \[ A(\text{NumberOfElements}+1,i) = A(\text{NumberOfElements}+1,i) + a(i,j) \cdot n(j); \]
    end
end

ASUM1 = 0;
ASUM2 = 0;
BSUM1 = 0;
BSUM2 = 0;
for j = 1:NumberOfSpecies
    ASUM1 = ASUM1 + n(j);
    ASUM2 = ASUM2 + n(j) \cdot \text{Properties}(j,4);
    BSUM1 = BSUM1 + n(j);
    BSUM2 = BSUM2 + n(j) \cdot \text{Gibbs}(j);
end

A(\text{NumberOfElements}+1,\text{NumberOfElements}+1) = \text{ASUM1} - n(\text{NumberOfSpecies}+1);
A(\text{NumberOfElements}+1,\text{NumberOfElements}+2) = \text{ASUM2};
B(\text{NumberOfElements}+1,1) = n(\text{NumberOfSpecies}+1) - BSUM1 + BSUM2;

\% \text{CEA 2.28}
Entropy = 0;
s = 0;
for i = 1:NumberOfSpecies
    S(i) = (\text{Properties}(i,5) - \log(n(i)/n(\text{NumberOfSpecies}+1))) - \log(\text{Pressure}/100000));
    Entropy = Entropy + n(i) \cdot S(i) \cdot \text{UniversalGasConstant};
    s = s + n(i) \cdot S(i);
end
\% s = Entropy/\text{UniversalGasConstant};

for i = 1:NumberOfElements
    for j = 1:NumberOfSpecies
        \[ A(\text{NumberOfElements}+2,i) = A(\text{NumberOfElements}+2,i) + a(i,j) \cdot n(j) \cdot S(j); \]
    end
end

BSUM1 = 0;
BSUM2 = 0;
for j=1:NumberOfSpecies

A(NumberOfElements+2,NumberOfElements+1)=A(NumberOfElements+2,NumberOfElements+1)+n(j)*S(j);

A(NumberOfElements+2,NumberOfElements+2)=A(NumberOfElements+2,NumberOfElements+2)+n(j)*(Properties(j,3)+Properties(j,4)*S(j));

BSUM1=BSUM1+n(j);

BSUM2=BSUM2+n(j)*S(j)*Gibbs(j);
end

B(NumberOfElements+2,1)=s_0-s+n(NumberOfSpecies+1)-BSUM1+BSUM2;

% Matrix Solution
Corrections=A\B;

CF1=0;
CF2=0;
for j=1:NumberOfSpecies

SUM=0;
for i=1:NumberOfElements

SUM=SUM+a(i,j)*Corrections(i);
end

%Calculation of Delta ln (n_j)
LogChange(j)=-Gibbs(j)+Corrections(NumberOfElements+1)+SUM+Properties(j,4)*Corrections(NumberOfElements+2);

% CEA 2.18
%Calculation of Control Factors

CF1(j)=2/max([5*abs(Corrections(NumberOfElements+1)),5*abs(Corrections(NumberOfElements+2)),abs(LogChange(j))]);
if log(n(j)/n(NumberOfSpecies+1))<log(ZERO)

CF2(j)=abs((-log(n(j)/n(NumberOfSpecies+1))+log(ZERO))/(LogChange(j)-Corrections(NumberOfElements+1)));
else

CF2(j)=1;
end
end

LogChange(NumberOfSpecies+1)=Corrections(NumberOfElements+1);

ControlFactor1=min(CF1);
ControlFactor2=min(CF2);
ControlFactor=min([1,ControlFactor1,ControlFactor2]);
CF(iterations+1)=ControlFactor;
%Corrections
for j=1:NumberOfSpecies+1

Previous(j)=n(j); %#ok<*SAGROW>
n(j)=exp(log(Previous(j))+ControlFactor*LogChange(j));

if abs(n(j))/abs(Previous(j))>exp(2)

n(j)=Previous(j)*exp(2);
elseif abs(Previous(j))/abs(n(j))>exp(2)

n(j)=Previous(j)*exp(-2);
end
if j==NumberOfSpecies+1 && abs(n(j))/abs(Previous(j))>exp(.4)
n(j)=Previous(j)*exp(.4);
elseif j==NumberOfSpecies+1 && abs(Previous(j))/abs(n(j))>exp(.4)
n(j)=Previous(j)*exp(-.4);
end
end

% Temperature Correction
PT=Temperature;
Temperature=exp(log(PT)+ControlFactor*Corrections(NumberOfElements+2));
if Temperature/PT>exp(.4)
    Temperature=PT*exp(.4);
elseif PT/Temperature>exp(.4)
    Temperature=PT*exp(-.4);
end

% Check for Convergence
SUM=0; CompositionConvergence=1; MassBalance=1; TemperatureConvergence=1;
EntropyConvergence=1;
for j=1:NumberOfSpecies
    SUM=SUM+n(j);
end
for j=1:NumberOfSpecies+1 % Composition convergence
    if n(j)*abs(LogChange(j))/SUM>0.5*10^-8
        CompositionConvergence=0;
    end
end

for i=1:NumberOfElements % Mass balance
    SUM=0;
    for j=1:NumberOfSpecies
        SUM=SUM+a(i,j)*n(j);
    end
    if abs(b_0(i)-SUM)>max(b_0)*10^-16
        MassBalance=0;
    end
end
if abs(Corrections(NumberOfElements+2))>10^-4
    TemperatureConvergence=0;
end
if abs(s_0-s)>.5*10^-4
    EntropyConvergence=0;
end

iterations=iterations+1;
if iterations>100
    convergence=1;
elseif CompositionConvergence==1 && MassBalance==1 && TemperatureConvergence==1 && EntropyConvergence==1
    convergence=1;
else
    convergence=0;
end
% Calculation of Mole Fractions, Molecular Weight, and Cpf
Cpf=0;
MolecularWeight=0;
for i=1:NumberOfSpecies
    Fractions(i,1)=n(i)/n(NumberOfSpecies+1);
    MolecularWeight=MolecularWeight+Fractions(i)*Properties(i,1);
    Cpf=Cpf+n(i)*Properties(i,3);
end
SpecificGasConstant=UniversalGasConstant/MolecularWeight;

% Calculation of the ratio of specific heats and speed of sound for % equilibrium flow.

%Temperature Derivatives
A=zeros(NumberOfElements+1);
B=zeros(NumberOfElements+1,1);
%CEA (2.56)
for k=1:NumberOfElements
    for i=1:NumberOfElements
        for j=1:NumberOfSpecies
            A(k,i)=A(k,i)+a(k,j)*a(i,j)*n(j);
        end
    end
    for j=1:NumberOfSpecies
        A(k,NumberOfElements+1)=A(k,NumberOfElements+1)+a(k,j)*n(j);
        B(k)=B(k)-a(k,j)*n(j)*Properties(j,4);
    end
end
%CEA (2.58)
k=NumberOfElements+1;
for i=1:NumberOfElements
    for j=1:NumberOfSpecies
        A(k,i)=A(k,i)+a(i,j)*n(j);
    end
end
for j=1:NumberOfSpecies
    B(k)=B(k)-n(j)*Properties(j,4);
end
%Solving system of equations
TemperatureDerivatives=A\B;
%CEA (2.59)
C=zeros(3,1);
for i=1:NumberOfElements
    for j=1:NumberOfSpecies
        C(1)=C(1)+a(i,j)*n(j)*Properties(j,4)*TemperatureDerivatives(i);
    end
end
for j=1:NumberOfSpecies
    C(2)=C(2)+n(j)*Properties(j,4)*TemperatureDerivatives(NumberOfElements+1);
end
C(3)=C(3)+n(j)*(Properties(j,4))^2;
end
Cp=(Cpf+sum(C))*UniversalGasConstant/10^3;%divided by mass of 10^3 to give J/kg-K

% Pressure Derivatives
B=zeros(NumberOfElements+1,1);
%CEA (2.64)
for k=1:NumberOfElements
  for j=1:NumberOfSpecies
    B(k)=B(k)+a(k,j)*n(j);
  end
end
B(NumberOfElements+1)=n(NumberOfSpecies+1);
%Solving system of equations
PressureDerivatives=A\B;
%Finding Thermo properties
ConstantPressureDerivative=1+TemperatureDerivatives(NumberOfElements+1);
ConstantTemperatureDerivative=1-PressureDerivatives(NumberOfElements+1);
Cv=Cp-n(NumberOfSpecies+1)*UniversalGasConstant*(ConstantPressureDerivative)^2/ConstantTemperatureDerivative/10^3;%divided by mass of 10^3 to give J/kg-K
ConstantEntropyDerivative=Cp/(Cv*ConstantTemperatureDerivative);
Gamma=Cp/Cv;
Gamma_S=Gamma/ConstantTemperatureDerivative;
SpeedOfSound=sqrt(Gamma_S*SpecificGasConstant*Temperature);
Entropy=s*UniversalGasConstant/1000;
Enthalpy=0;
for i=1:NumberOfSpecies
  Enthalpy=Enthalpy+n(i)*Properties(i,4)*UniversalGasConstant*Temperature/1000;
end
%b_0-b
%Fractions
%MolecularWeight
x=[MolecularWeight,Cp,Cv,Gamma,Gamma_S,SpeedOfSound,Temperature,Enthalpy,Entropy];

ThermochemicalProperties.m
function [2]=ThermochemicalProperties(Chemical, Temperature, State)
%clear all
%Chemical='NH';
%Temperature=298.15;
%State='Gas';

% Opening the file
fileID= fopen('thermodata.txt');
% Finding the Chemical
stop=0;
number=0;
while stop==0;
  ID=fgetl(fileID);
  if strcmp(ID,Chemical)==1;
    stop=1;
  elseif feof(fileID)==1
    stop=1;
elseif feof(fileID)==1
  end
end

stop=1;
str = ['Species', Chemical, ' not found in thermodata.txt'];
disp(str)
end
end
if strcmp(State, 'Gas')==1
% Reading in chemical data
LINE1=fgetl(fileID);
LINE2=fgetl(fileID);
LINE3=fgetl(fileID);
LINE4=fgetl(fileID);
LINE5=fgetl(fileID);
LINE6=fgetl(fileID);
LINE7=fgetl(fileID);
% Extracting numbers from formatting
X=regexp(LINE1, ' ', 'split');
number=1;
for i = 1:length(X)
    v=X(i);
    w=cell2mat(v);
    x=str2num(w); % conversion from cell to matrix to number
    if (length(w)>7) && (number==1)
        MolecularWeight=x; % ST2NM
        number=2;
    elseif (length(w)>4) && (number==2)
        HeatOfFormation=x;
    end
end
if (Temperature>=200) && (Temperature<1000)
X=regexp(LINE3, ' ', 'split');
number=1;
i=1;
while number<=5
    v=X(i);
    w=cell2mat(v);
    x=str2num(w);
    if length(w)>3
        a(number)=x;
        number=number+1;
    end
    i=i+1;
end
X=regexp(LINE4, ' ', 'split');
number=1;
i=1;
while number<=4
    v=X(i);
    w=cell2mat(v);
    x=str2num(w);
    if length(w)>3
        a(5+number)=x;
        number=number+1;
    end
    i=i+1;
end
end
b1=a(8);
b2=a(9);

elseif (Temperature>=1000) && (Temperature<=6000)
X=regexp(LINE6,' ','split');
number=1;
i=1;
while number<=5
    v=X(i);
    w=cell2mat(v);
    x=str2num(w);
    if length(w)>3
        a(number)=x;
        number=number+1;
    end
    i=i+1;
end
i=i+1;
X=regexp(LINE7,' ','split');
number=1;
i=1;
while number<=4
    v=X(i);
    w=cell2mat(v);
    x=str2num(w);
    if length(w)>3
        a(5+number)=x;
        number=number+1;
    end
    i=i+1;
end
b1=a(8);
b2=a(9);

else
    disp ('Invalid temperature value. Must be between 200K and 600K')
end

CpoverR = a(1)*Temperature^(-2) + a(2)*Temperature^(-1) + a(3) + a(4)*Temperature + a(5)*Temperature^2 + a(6)*Temperature^3 + a(7)*Temperature^4; % [Ref.1]
HoverRT =-a(1)*Temperature^(-2) + a(2)*Temperature^(-1)*log(Temperature) + a(3) + a(4)*Temperature/2 + a(5)*(Temperature^2)/3 + a(6)*(Temperature^3)/4 + a(7)*(Temperature^4)/5 + b1/Temperature; % [Ref.1]
SoverR  =-a(1)*Temperature^(-2)/2 - a(2)*Temperature^(-1) + a(3)*log(Temperature) + a(4)*Temperature + a(5)*(Temperature^2)/2 + a(6)*(Temperature^3)/3 + a(7)*(Temperature^4)/4 + b2; % [Ref.1]
x=[MolecularWeight, HeatOfFormation, CpoverR, HoverRT, SoverR];

elseif strcmp(State,'Liquid')==1
% Reading in chemical data
LINE1=fgetl(fileID);
LINE2=fgetl(fileID);
% Extracting numbers from formatting
X = regexp(LINE1, ',,'split');
number = 1;
for i = 1:length(X)
    v = X(i);
    w = cell2mat(v);
    x = str2num(w); % conversion from cell to matrix to number
    if (length(w) > 5) && (number == 1)
        MolecularWeight = x; % ok<*ST2NM>
        number = 2;
    elseif (length(w) > 4) && (number == 2)
        HeatOfFormation = x;
    end
end
X = regexp(LINE2, ',,'split');
found = 0;
i = 1;
while (found == 0) && (i < 90)
    v = X(i);
    w = cell2mat(v);
    x = str2num(w); % conversion from cell to matrix to number
    if length(w) > 3
        ReferenceTemperature = x;
        found = 1;
    end
    i = i + 1;
end
x = [MolecularWeight, HeatOfFormation, ReferenceTemperature];
else
    disp('Invalid thermodynamic state. This program only supports liquid and gas phases. ')
end
fclose(fileID);

thermodata.txt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NOTE: Some minor edits are required when taking data from CEA's thermo file. %
% Thermo coefficients must have spaces in between them. a7 and b1 must be %
% separated by empty space, not 0.00000E+00. Also, tpisXX in the first line %
% of data may create problems if it is 8+ characters long. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% GAS DATA %

H
3 g 6/97 H 1.00 0.00 0.00 0.00 0.00 0.0079400     217998.828
200.000 1000.0007 -2.0 -1.0 0.0 1.0 2.0 3.0 4.0 0.0 6197.428
0.000000000E+00 0.000000000E+00 2.500000000E+00 0.000000000E+00 0.000000000E+00
0.000000000E+00 0.000000000E+00 2.547370801E+04 -4.466828530E-01
1000.000 6000.0007 -2.0 -1.0 0.0 1.0 2.0 3.0 4.0 0.0 6197.428
6.078774250E+01 -1.819354417E-01 2.500211817E+00 -1.226512864E-07 3.732876330E-11
-5.687744560E-15 3.410210197E-19 2.547486398E+04 -4.481917770E-01
HO2
2 g 4/02 H 1.000 2.00 0.00 0.00 0.00 0.00 33.0067400     12020.000

86
<table>
<thead>
<tr>
<th></th>
<th>200.000</th>
<th>1000.000</th>
<th>7</th>
<th>-2.0</th>
<th>-1.0</th>
<th>0.0</th>
<th>1.0</th>
<th>2.0</th>
<th>3.0</th>
<th>4.0</th>
<th>0.0</th>
<th>10002.162</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2</td>
<td>3 tpis78 H</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>2.01588000</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>1000.000</td>
<td>7</td>
<td>-2.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
<td>4.0</td>
<td>0.0</td>
<td>8468.102</td>
</tr>
</tbody>
</table>

| H2O | 2 g | 8/89 H | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.0152800 | -248182.000 |
|     | 100.000 | 1000.000 | 7 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 0.0 | 9904.092 |

| H2O2 | 2 g | 6/99 H | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 34.0146800 | -135880.000 |
|      | 200.000 | 1000.000 | 7 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 0.0 | 11158.835 |

| O   | 3 g | 5/97 O | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15.9994000 | 249175.003 |
|     | 100.000 | 1000.000 | 7 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 0.0 | 6725.403 |

| OH  | 3 g | 4/02 O | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.0073400 | 37278.206 |
|     | 200.000 | 1000.000 | 7 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 0.0 | 8813.106 |

| O3  | 2 g | 8/01 O | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 47.9982000 | 141800.000 |
|     | 200.000 | 1000.000 | 7 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 0.0 | 10366.305 |

---

87
1.359651320E+04 3.504927750E+04 6.674502990E+02 2 tpis89 N   1.00F   1.00    0.00    0.00    0.00 0   33.0051032     232990.500 0.000000000E+00 0.000000000E+00                 5.610463780E+04 4.193905030
3 g 5/97 N   1.00    0.00    0.00    0.00    0.00 0   14.0067000 5.259214710E+04 2 tpis89 H   2.00F   2.00    0.00    0.00    0.00 0   40.0126864 3.819312900E+02 2.725018980E+02 1.380212800E+03 8.703408870E+00 1.287394800E+03 3.418434160E+04 3.294904120E+00
100.000 6000.000    -2.0  0.0  1.0  2.0  3.0  4.0  0.0         6197.42830 16 1.0  0.0  1.0  2.0  3.0  4.0  0.0        13869.430 17 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 18 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 19 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 20 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 21 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 22 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 23 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 24 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 25 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 26 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 27 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 28 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 29 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 30 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 31 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 32 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 33 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 34 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 35 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 36 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 37 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 38 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106 39 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8599.103 40 1.0  0.0  1.0  2.0  3.0  4.0  0.0         8825.106
<table>
<thead>
<tr>
<th>Compound</th>
<th>Molecular Formula</th>
<th>C</th>
<th>H</th>
<th>O</th>
<th>Weight</th>
<th>Energy</th>
<th>Enthalpy</th>
<th>Entropy</th>
<th>Volume</th>
<th>Volume Pressure</th>
<th>Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>O₂(L)</td>
<td>0 g 6/96</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31.9988000</td>
<td>-12979.000</td>
<td>90.170</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>H₂(L)</td>
<td>0 g 6/96</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.0158800</td>
<td>-9012.000</td>
<td>20.270</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>Mol. Wt</td>
<td>Mol. %</td>
<td>Lat.</td>
<td>Hf</td>
<td>I</td>
<td>Enthalpy</td>
<td>Temp</td>
<td>Cal.</td>
<td>Lat.</td>
<td>Cal.</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>----</td>
<td>---</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>RP-1</td>
<td>0 g 1/00 C</td>
<td>1.00H 1.95 0.00 0.00 0.00</td>
<td>13.9761830</td>
<td>-24717.700</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>298.150</td>
<td>0.0000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NH3(L)</td>
<td>0 g 6/96 N</td>
<td>1.00H 3.00 0.00 0.00</td>
<td>17.0305200</td>
<td>-71555.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>239.720</td>
<td>0.0000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2(L)</td>
<td>0 g 6/96 F</td>
<td>2.00 0.00 0.00 0.00</td>
<td>37.9968064</td>
<td>-13091.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>85.020</td>
<td>0.0000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>