Assignment #2: Engine Analysis

Daniel Card*, Torrence Gue†, and Marc Chattrabhuti‡
University of Michigan, Ann Arbor, MI, 48109

Due: Wednesday, November 4 at 5:00 P.M. ET

I. Nomenclature

I_{sp}	=	Specific impulse (s)
T	=	Thrust (N)
g_0	=	Gravity constant (m/s^2)
P	=	Pressure (Pa)
T_0	=	Adiabatic Flame Temperature (K)
ζ_F	=	Thrust Correction Factor
ζ_d	=	Discharge Correction Factor
n	=	Molar Concentration
H	=	Enthalpy (kJ)
H°	=	Enthalpy of Species (kJ/mol)
ΔH_f	=	Specific Heat of Formation (kJ/mol)
h	=	Specific Enthalpy (kJ/kg)
S	=	Entropy (J/K)
s	=	Specific Entropy (J/mol-K)
γ	=	Ratio of Specific Heats
$\bar{M}$	=	Molecular Mass (g/mol)
$\bar{R}$	=	Universal Gas Constant (J K^-1 mol^-1)
ρ	=	Density (kg/m^3)
A	=	Cross-Sectional Area (m^2)
A_t	=	Cross-Sectional Area of Throat (m^2)
c_F	=	Coefficient of Thrust
c^*	=	Characteristic Speed (km/s)
C_p	=	Specific Heat at Constant Pressure (J/mol-K)
C_v	=	Specific Heat at Constant Volume (J/mol-K)
f	=	Oxidizer-Fuel Ratio

II. Introduction

NASA's road map for crewed exploration is contingent on the ability to develop a heavy launch vehicle to replace the Saturn V. Key mission level requirements include the ability to launch human-rated payloads to cislunar and Mars injection. For known values for delta-v and payload, there are established methodologies for generating optimal launch vehicle design (see "Trade Study for Crewed Mission to Moon"). This analysis is contingent, however, on accurate assessments of the performance metrics for the engines of each stage. As a critical step then for designing the overall vehicle, it is necessary to establish the trade space of predicted performance for different engine options.

To this end, NASA has solicitated designs from two contractors for upperstage engines, the J2 engine and the Merlin 1D vacuum. The contractors have provided design parameters for each of these systems as well as notional fuel and

*Graduate Aerospace Engineering, AIAA Student Member, Aerospace Honors Society.
†Undergraduate Aerospace Engineering, AIAA Student Member, Aerospace Honors Society, External Vice President.
‡Graduate Aerospace Engineering, Aerospace Honors Society

oxidizer combinations. They similarly have quoted to NASA estimates for performance for each of these systems. As an independent check, we need to evaluate whether the promised performance metrics are physically plausible and consistent with known principles of operation for rockets. This is the goal of this report.

This work is organized in the following way. We first outline the key assumptions for our analysis, the design parameters for the engines, and the properties of the combustion reactants and products. We then perform a 1D analysis of the J2 engine in both the frozen flow and shifting equilibrium cases. This is followed by an analysis of the Merlin 1D vacuum employing the numerical tool, CEA.

III. Key Assumptions for Analysis

We outline in the following the key assumptions we use in our evaluation of the J2 and Merlin 1D engines. These include estimates for the chemical properties of the fuel, oxidizer, and reaction products as well as a list of assumptions we employ to analyze the evolution of the mixture in the nozzle.

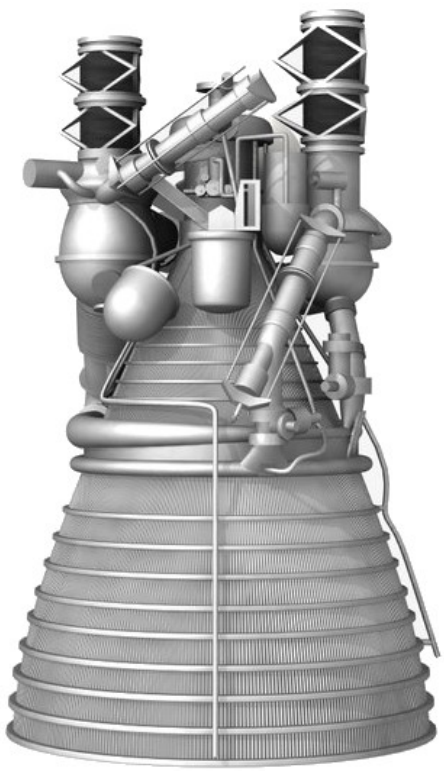

Fig. 1 Aerojet Rocketdyne's J-2 engine.

Fig. 2 SpaceX's Merlin 1D engine.

A. General Nozzle Characteristics

Table 1 gives overall relevant dimensions and characteristics of the J-2 and Merlin engines as reported by the contractors.The overall dimensions, expansion ratio, and combustion pressure are given for both systems. The fuel-oxidizer ratio was provided for the J2, but it is a free parameter we will examine parametrically in our analysis for the Merlin. We also assume, consistent with the need to reduce pressure losses in the combustion chamber, that the combustion chamber area is 4 times the throat area.

Table 1 Design Parameters for the J-2 and Merlin 1D Vacuum Systems.

	Propellant	Combustion Pressure, MPa	Expansion Ratio	Throat Area, m^2	Fuel-Oxidizer Ratio (mass)
J-2	LOX/LH2	5.3	27.5	0.1092	5.5
Merlin 1D vac	LOX/RP1	9.7	117	0.042	TBD

B. Assumptions for Analysis and Empirical Correction Factors

For our analysis, we will apply both the assumption of local equilibrium combustion as well as the following assumptions about the flow through the nozzle:

- Gas is homogenously mixed
- Combustion occurs with negligible drift speed
- No heat lost to walls
- Nozzle is adiabatic downstream of combustion chamber
- No shocks or friction downstream of combustion chamber
- Both engines are discharging into vacuum

C. Empirical Correction Factors

For performance estimates, we use the following empirical correction factors for thrust and specific impulse. These account for a number of non-ideal effects such as flow divergence, boundary layer effects, unsteady combustion, nozzle erosion, and many others. The factors include

- A thrust correction factor of $\zeta_F = 0.95$,which is the ratio of real to ideal thrust
- A discharge correction of $\zeta_d = 1.1$,which is ratio of actual mass flow rate to the theoretical value.

D. Key Chemical Properties

We enumerate here the chemical properties for the anticipated reactants and productions of the combustion processes in the two engines.

1. LOX/LH2 Reaction

a. Specific enthalpy, specific heat at constant pressure, and specific entropy
For combustion of hydrogen(H_2) and oxygen (O_2), we assume the possible combustion products are H_2O, H, O, OH, O_2, H_2:

$$n_o O_2 + n_F H_2 \rightarrow n_{O_2} O_2 + n_{H_2} H_2 + n_H H + n_{OH} OH + n_o O + n_{H_2O} H_2O \tag{1}$$

where n_i denotes the molar concentration of the ith species. There are two other possible products, ozone (O_3) and peroxide (H_2O_2); however, these are highly unstable and quickly break down into the other products. On the time scales of interest, they thus can be ignored in the reaction.

The thermodynamic properties of each species including the specific heat, enthalpy, and entropy are determined as a function of temperature from the NIST database. These are typically given in the form of analytical expressions with best fit coefficients, e.g.

$$C_p{}^\circ = A + Bt + Ct^2 + Dt^3 + E/t^2 \tag{2}$$

$$H^\circ - H^\circ{}_{298.15} = At + Bt^2/2 + Ct^3/3 + Dt^4/4 - E/t + F - H \tag{3}$$

$$S^\circ = A\ln(t) + Bt + Ct^2/2 + Dt^3/3 - E/\left(2t^2\right) + G \tag{4}$$

where t = Temperature (K)/1000 and the governing equations for the specific heat, enthalpy, and entropy for the different species are given by units of $J/mol - K$, kJ/mol, and $J/mol - k$ respectively.

b. Heats of formation for each species
We show in Table 2 the specific heats of formation for the reactants and products of the LOX/LH2 combination. These are expressed in units of kJ/mol and are given at the reference temperature of 298 K.

Table 2 Specific Heats of Formation at 298 K [5pts].

Species	H_2	O_2	H_2O	O	H	OH
$\Delta H_f, kJ/mol$	0	0	-241.8	249	218	38.99

2. LOX/RP-1 reaction

LOX/RP-1 (a form of kerosene) is a more complex reaction than LOX/LH2 giving rise to more combustion products. These include $H_2O, CO_2, CO, H, O, O_3, HO_2, COOH$, and HCO, which are governed by the reaction equation:

$$n_o O_2 + n_F CH_{1.953} \rightarrow n_{O_2} O_2 + n_{H_2} H_2 + n_H H + n_{OH} OH + n_O O + n_{H_2O} H_2O + n_{CO_2} CO_2 + n_{CO} CO + n_O O + n_H H + n_{COOH} COOH + n_{HCO} HCO \quad (5)$$

The use of $CH_{1.953}$ is an approximation for the significantly more complex chemical formula for RP-1. This effectively represents the fact that on average, the chemical composition of RP-1 has 1.953 mols of hydrogen per mol of carbon. This is a useful truncation for calculating the resulting mixture content. With that said, as the reaction in Equation 5 is substantially more complex process, we will use the Chemical Equilibrium with Applications (CEA) tool for our analysis of the Merlin 1D. CEA and the chemical properties of each species is handled internally by the solver, and as such, we do not list the propellant properties here.

E. Reaction Equilibrium through Nozzle

In our analysis, we will consider two situations for the chemical species as they traverse the nozzle: frozen flow and shifting equilibrium. For the frozen flow case,we will assume that the specific heat and relative mole fractions of the combustion products will remain constant through the nozzle.This composition is given by the result of the equilibrium calculation in the combustion chamber.In the shifting equilibrium case,we will propagate the thermodynamic quantities with changing area, and the relative molar concentrations (and specifics heats of the mixture) will be recalculated at each location.

IV. J-2 Analysis

We present in the following section our analysis of the J-2 rocket engine and how we arrived to our conclusions.

A. Combustion Chamber Product Composition [10pts]

We calculate here the final molar concentrations of each species and the final combustion temperature of the reaction.The process we adopt is based on the summation of heats of formation to calculate the resulting temperature for combustion and the Gibbs-free energy method to estimate the relative molar contributions of different species.

Implementation of CEA Solver

In our analysis of the J-2 analysis, we will use design parameters for the J-2 from Table 1 in order to determine the performance of the rocket engine. Using the combustion chamber P_0, and the fuel-oxidizer ratio f, we will be able to approximate the combustion products through numerically solving and approximation. In order to solve for the molecular composition we will use governing equations for the combustion reaction employing the minimizing free energy equations to arrive at the governing expression to approximate the molar values. These minimizing energy equations are as follows,

$$O_2: \quad \left(H^0_{O_2}(T_0) - H^0_{O_2}(298)\right) + \Delta_f H^0(O_2) - T_0 s^0_{O_2}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{O_2}}{\sum_j n_j}\right) + 2\lambda_O = 0 \tag{6}$$

$$H_2: \quad \left(H^0_{H_2}(T_0) - H^0_{H_2}(298)\right) + \Delta_f H^0(H_2) - T_0 s^0_{H_2}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{H_2}}{\sum_j n_j}\right) + 2\lambda_{H_2} = 0 \tag{7}$$

$$H: \quad \left(H^0_{H}(T_0) - H^0_{H}(298)\right) + \Delta_f H^0(H) - T_0 s^0_{H}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{H}}{\sum_j n_j}\right) + \lambda_H = 0 \tag{8}$$

$$OH: \left(H^0_{OH}(T_0) - H^0_{OH}(298)\right) + \Delta_f H^0(OH) - T_0 s^0_{OH}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{OH}}{\sum_j n_j}\right) + \lambda_O + \lambda_H = 0 \tag{9}$$

$$O: \quad \left(H^0_{O}(T_0) - H^0_{O}(298)\right) + \Delta_f H^0(O) - T_0 s^0_{O}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{O}}{\sum_j n_j}\right) + \lambda_O = 0 \tag{10}$$

$$\left(H^0_{H_2O}(T_0) - H^0_{H_2O}(298)\right) + \Delta_f H^0(H_2O) - T_0 s^0_{H_2O}(T_0) + \bar{R}T_0 \ln\left(\frac{P_0}{P_{stp}}\right) + \bar{R}T_0 \ln\left(\frac{n_{H_2O}}{\sum_j n_j}\right) + \lambda_O + 2\lambda_H = 0 \tag{11}$$

Using the equations above, with the mass conservation relationship in Eqn. 1 we can expand it to be a relationship that's entirely dependant upon Oxygen and Hydrogen. Writing this expression gives,

$$O: \quad 2n_{oxi} = 2n_{O_2} + n_{OH} + n_O + n_{H_2O} \tag{12}$$

$$H: \quad 2n_f = 2n_{H_2} + n_H + n_{OH} + 2n_{H_2O} \tag{13}$$

With these two relationships expressed we will implement this into Matlab with an initial guess for Adiabatic flame temperature T_0, molar concentration of products, and lagrange multipliers. MATLAB will iterate with the solver "`vpasolve`" until we can arrive to the approximate solution to find the molecular composition of products. Based on this information we will now feed the molar concentrations into the energy conservation equation, shown below:

$$E_{react} = n_{O_2}\left(H^0_{O_2}(T_0) - H^0_{O_2}(298)\right) + n_{H_2}\left(H^0_{H_2}(T_0) - H^0_{H_2}(298)\right) + n_H\left(H^0_{H}(T_0) - H^0_{H}(298)\right) + n_O\left(H^0_{O}(T_0) - H^0_{O}(298)\right) + n_{H_2O}\left(H^0_{H_2O}(T_0) - H^0_{H_2O}(298)\right) \tag{14}$$

$$E_{combustion} = n_{oxi}\Delta_f H^0(O_2) + n_f \Delta_f H^0(H_2) - n_{O_2}\Delta_f H^0(O_2) - n_H \Delta_f H^0(H) - n_{OH}\Delta_f H^0(OH) - n_O \Delta_f H^0(O) - n_H \Delta_f H^0(H) + n_{oxi}\left(H^0_{O_2}(T_{in}) - H^0_{O_2}(298)\right) + n_f\left(H^0_{H_2}(T_{in}) - H^0_{H_2}(298)\right) \quad (15)$$

$$E_{react} = E_{combustion} \quad (16)$$

Note that the values of where the values of $\left(H^0_{H_2}(T_{in}) - H^0_{H_2}(298)\right)$ are determined by the following equations because of the use of cryogenic oxidizers and fuels.

$$\left(H^0_{H_2}(T_{in}) - H^0_{H_2}(298)\right) = H^o_o(T_{boil_i}) - H^o_o(298K) + \Delta H_{vap}(i) \quad (17)$$

where ΔH_{vap} is the latent heat of vaporization for the oxidizer or fuel respectively. Additionally $H^o_o(T_{boil_i})$ is given by coefficients from CEA in the following equation:

$$H(T_{boil}) = H_{298.15} - (H_{298.15} - H_{bp}) - (H_{bp} - H_{bp,re}) - \Delta H_{vap} \quad (18)$$

where values of $H_{298.15}, (H_{298.15} - H_{bp}), (H_{bp} - H_{bp,re})$, and ΔH_{vap} for the O_2 and H_2 propellants are given in the CEA user guide.

In order to approximate the performance metrics of the J-2 engine. This CEA solver will return the molar composition in which our calculated molar concentrations and change in enthalpy are shown below in Table 3.

Table 3 Calculated Molar Concentrations and Change in Enthalpy.

Species	H_2	O_2	H_2O	OH	H	O
Molar Concentration	0.317	0.0017	0.647	0.0345	0.0361	0.0020
Change in Enthalpy [kJ/mol]	103.27	113.59	148.35	103.91	64.21	64.12

Furthermore, using the CEA solver we can calculate the temperature, the ratio of specific heats, molecular mass, specific entropy, and specific enthalpy. After implementation of the CEA solver we determine that these properties are as follows shown below in Table 4.

Values of the ratio of specific heat, γ, as well as Molecular Mass, Specific Entropy, and Specific Enthalpy were also calculated using the following equations and previous mole fraction calculations:

$$C_p = \frac{1}{\sum_i n_i} \sum_i n_i C_{pi}(T) \quad (19)$$

Where n_i and C_{pi} are the product concentration and specific heat of the ith product. Based on this equation for specific heat at constant pressure, we calculate the ratio of specific heats in the following equations:

$$\gamma = \frac{C_p(T)}{C_p(T) - \bar{R}} \quad (20)$$

Additionally molecular mass $\bar{M}$, specific enthalpy $h(T)$, and entropy $s(T, P)$ for the composition can be calculated with the following equations:

$$\bar{M} = \frac{1}{\sum_i n_i} \sum_i n_i \bar{M}_i \tag{21}$$

$$h(T_0) = \frac{1}{\sum_i n_i \bar{M}} \sum_i n_i H_i^o(T_0) \tag{22}$$

$$s(T_0, P_0) = \sum_i n_{i_0} \left[s_i^0(T_o) - \bar{R} ln \left(\frac{n_i(0)}{\sum_i n_i(0)} \right) - \bar{R} ln \left(\frac{P_0}{P_{Stp}} \right) \right] \tag{23}$$

Using these equations the following values were calculated using the CEA solver:

Table 4 Calculated Values for Combustion Chamber Mixture.

Temperature, K	3387
Ratio of Specific Heats, γ	1.204
Molecular Mass, $\bar{M}$ $\frac{g}{mol}$	12.52
Specific Entropy, $J/mol/K$	245.31
Specific Enthalpy, kJ/kg	10.37

The following table from Suttons* shows the flame temperatures and molecular composition of the J2 engine based on measured data:

Table 5 Suttons, J-2 Molecular Composition Values and Temperature at End of Combustor

Temperature, K	3389
Ratio of Specific Heats, γ	1.202
Molecular Mass, $\bar{M}$ $\frac{g}{mol}$	12.72

Table 6 Molar Concentrations for J-2 Engine from Suttons.

Species	H_2	O_2	H_2O	OH	H	O
Molar Concentration	0.2941	0.00179	0.63643	0.03162	0.03390	0.00214

In conclusion, the measured results of the actual J-2 engine values of flame Temperature, ratio of specific heats, and molar masses compared to the implemented CEA solver are quite similar. There are slight variations between the molar concentrations of the actual J-2 engine and the CEA solver, but they are quite small. Using results from Table 3, 4 we determine the ideal, frozen, and equilibrium flow performance for the J-2 rocket engine.

*Rocket Propulsion Elements 9th Ed. - George P. Sutton, Oscar Biblarz

B. Rocket Performance [35pts]

We consider here the performance of the rocket in two limiting cases—for frozen flow and shifting equilibrium flow—where we use the results from the previous sections as initial conditions for the flow's expansion through the nozzle. We include plots of the change in Mach number, ratio of specific heats, temperature, pressure, density, and molecular weight as functions of cross-sectional area of the rocket. We also estimate the performance metrics at the exit plane including thrust, specific impulse, and mass flow rate.

In our analysis of the J-2 rocket engine, we imposed a multitude of techniques to arrive at the performance of how the engine performance varies from the given inputs into the combustion chamber. These input parameters include the mass flow rate $\dot{m}$, the combustion chamber pressure P_0, and the fuel-propellant ratio f. The steps to solve for the performance for frozen flow versus shifting equilibrium flow is as follows:

Frozen Flow

This flow occurs when the fuel-propellant ratio will result in a non-caloric perfect combustion resulting in several combustion products. In the frozen flow case these products will be "*frozen*" into that combustion composition and remain as such throughout the contraction and expansion of the nozzle. In this frozen flow case, the specific heat ratio γ will vary. However as the molecular mass $\bar{M}$ will remain constant.

Shifting Equilibrium

This flow is similar to the frozen flow in a sense, as there will be a non-caloric combustion and will result in many products after the combustion. How the shifting equilibrium case differs is that as the flow makes its way downstream its composition will continually change as it travels throughout the length of the rocket nozzle. This results in both the specific heat ratio γ and the molecular mass $\bar{M}$ varying as it makes its way down the nozzle.

Determining Performance

Employing the methods from part A. with isentropic flow relationships we can determine how the J-2 performs at differing conditions. Firstly for the frozen flow case we run our custom CEA solver to determine the molecular composition and properties such as the combustion temperature, and the ratio of specific heats γ before employing the relationships for isentropic flow. These relationships are as follows,

$$\frac{P}{P_0} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-\frac{\gamma}{\gamma-1}} \tag{24}$$

$$\frac{\rho}{\rho_0} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-\frac{1}{\gamma-1}} \tag{25}$$

$$\frac{T}{T_0} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-1} \tag{26}$$

$$\frac{A}{A_t} = \left(\frac{\gamma + 1}{2}\right)^{-\frac{\gamma+1}{2(\gamma-1)}} \frac{\left(1 + \frac{\gamma-1}{2} M^2\right)^{\frac{\gamma+1}{2(\gamma-1)}}}{M} \tag{27}$$

For the ideal case, we will vary the A/A_t ratio through the contracting part of the combustion chamber until reaching the throat and reaching sonic conditions. After this we will continue varying A/A_t through the expanding section of the nozzle until reaching the exit. In this case we will use the isentropic flow relations above to calculate their ratios from the mach number at each corresponding mach number.

For the frozen flow, we will use the expressions above but will first use our CEA solver to determine the molecular composition to determine the ratio of specific heats. Through each iteration we will decrease pressure by 2.5% each iteration and solve for the temperature via entropy conservation shown below in Eqn. 28.

$$\sum_i n_{i,(0)} \left(s_i^0(T_0) - R \ln \frac{n_{i(0)}}{\sum_i n_{i(0)}} - R \ln \frac{P_0}{P_{stp}} \right) = \sum_i n_i \left(s_i^0(T_0) - R \ln \frac{n_i}{\sum_i n_i} - R \ln \frac{P_0}{P_{stp}} \right) \tag{28}$$

Shifting equilibrium will be similar in practice however, requires an iterative solver to do so. In this case we will decrease the pressure by 2.5% each iteration and solve for the new composition and other gas properties to determine the mach number M at each iteration. We will employ this method iterating until we reach the mach number that is consistent with the expansion ratio for the J-2. Using Eqn. 28 above, and implementing a Gibbs minimizing free energy equation shown below in Equations 29, 30 below.

$$\frac{\partial G_{sys}}{\partial n_i} + \sum_i^L \lambda_j a_{ij} = 0 \tag{29}$$

$$\sum_j^{N_p} a_{ij} n_j - \sum_k^{N_r} b_{ik} n_k = 0 \tag{30}$$

Using Equations 28, 29, 30 above and using our CEA solver from part A. we can compare and contrast the performance parameters between frozen flow and shifting equilibrium flow. Tabulating the result gives that the differences between Ideal, Frozen Flow, and Shifting Equilibrium are below in Table 7.

Table 7 Performance Metrics for the J2 for frozen and shifting equilibrium flow.

	T, kN	I_{sp}, s	$\dot{m}$, kg/s	c^*, km/s	c_F
Ideal Flow	1015.05	376.02	275.17	2313.57	1.59
Frozen Flow	973.96	351.35	282.58	2252.98	1.53
Shifting Equilibrium Flow	990.73	352.22	286.73	2220.36	1.56

The performance between the ideal case, frozen flow, and shifting equilibrium arises from the molecular compositions or in addition the thermodynamic properties of the flow changing as it travels about the nozzle. In the ideal case, the gas is assumed to not change as it travels down the the nozzle resulting in the best performance metrics of the three cases as a result. Then the next best case would be shifting equilibrium flow since as the gas makes its way downstream it will change both its molecular composition n and the specific heat at constant pressure, as a result varying the ratio of specific heats resulting in differing mach numbers from the ideal case. Then for the frozen flow all that changes is the specific heat at constant pressure c_p since the temperature will vary and this only vary the ratio of specific heats and in return the mach number.

Going further to show how the different combustion processes differ along thermodynamic processes can be found on the following page in Fig. 3. In this Fig. we highlight the differences in the performance along varying A/A_t and ultimately show how the different combustion processes arrive to the performance metrics above in Table 7.

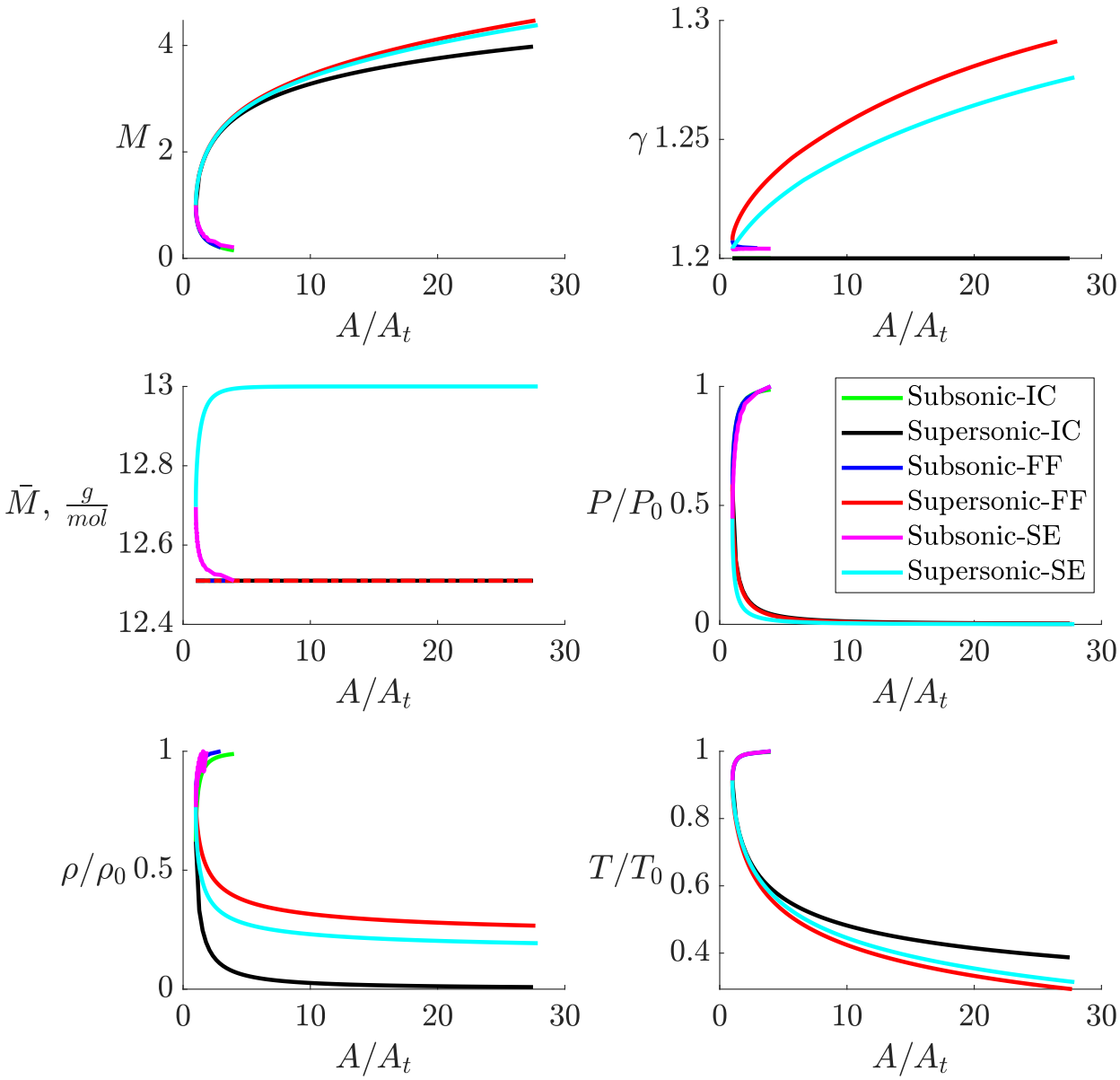

Fig. 3 Plots of key quantities as a function of area ratio for both frozen and equilibrium flow for the J2 engine.

Above in Fig. 3 is the performance quantities for both the frozen flow and the shifting equilibrium flow. Biggest takeaway is that there are differences arising from the different combustion processes which result in variances to the mach number, the ratio of specific heats, and the molecular weight. These three are the only variables in play but result in large differences in the performance metrics. These three have a crucial role in the performance of the J-2 engine and also determine the other ratio as a result. However, across all three configurations the one relation that still gives the same overall trends is the P/P_0 relationship.

In conclusion, looking to Fig. 3 we can come to the conclusion that the shifting equilibrium case is a more idealized case with it being biased towards the ideal case and takes more real world effects into account when solving for these performance metrics. Since in real-world combustion processes, there is constant shifts around the molecular compositions and the thermodynamic properties of the flow.

V. Merlin 1D Vacuum Engine [30 pts]

For the Merlin 1 engine, we use CEA to analyze both the frozen flow and shifting equilibrium cases. We also used some of the results from CEA to find the properties of the engine for ideal flow. From Eq. 5 we know that the reactants are liquid O_2 and RP-1. In CEA, we first set the "Chemical Equilibrium Problem Type" to the "rocket" option. Next, we set the pressure to be the combustion chamber pressure of the merlin engine P_0, which is 9.7 MPa. In CEA we input this pressure in units of atm, which is 95.73 atm. We then define our fuel as RP-1 and our oxidizer as liquid O_2. Then, we must input values for fuel-oxidizer ratio. Because we want to determine how the fuel-oxidizer ratio affects the thrust, I_{sp}, and T_0 of the engine, we can set a range of values for the oxidizer-fuel ratio in CEA. For our testing, we input a minimum O/F ratio of 2 and maximum O/F ratio of 10, with an interval of 0.25 between 2 and 5 and an interval of 0.5 between 5 and 10. This gave us 23 total data points. The reason we have more data points for the lower O/F ratio values is because we expect the most change to happen at that region.

CEA then asks for the supersonic area ratio of our engine, which to equal the expansion ratio of 117. On the final tab, we will run an analysis first for equilibrium flow, then for frozen flow. For shifting equilibrium flow, we first checked the "Equilibrium" box in the final tab of the CEA analyzer. Underneath "Rocket Problem Options" we select a Finite Area Combustor, and input the contraction ratio of 4 (which is the ratio of the chamber area to the throat area). We then performed the CEA analysis, and recorded the output vacuum I_{sp}, coefficient of thrust c_F, and adiabatic flame temperature T_0 for each oxidizer fuel ratio. For frozen flow, the only difference in our process was that we instead checked the "Frozen" box, chose and NFZ value of 1 (which indicates frozen flow in the combustor), and checked the Infinite Area Combustor box. Since we are given the throat area A_t, the thrust correction factor ζ_F and the combustion chamber pressure P_0 of the Merlin engine, we can convert the coefficient of thrust c_F to actual thrust T_{actual} using Eq. 31. We can also calculate the actual I_{sp} from the CEA value using the discharge correction factor ζ_d and equation 32.

$$T_{actual} = \zeta_F c_F P_0 A_t \tag{31}$$

$$I_{sp(actual)} = \frac{\zeta_F}{\zeta_d} I_{sp(ideal)} \tag{32}$$

To find the engine properties in an idealized flow, we used the values of the ratio of specific heats γ, the adiabatic flame temperature T_0 and the molecular weight $\bar{M}$ found for each OF ratio in the frozen case. Since we know the expansion ratio $\frac{A_e}{A_t}$ of the Merlin engine, using MATLAB we can solve numerically for the Mach number at the engine exit M_e at each OF ratio using Eq. 33.

$$\frac{A_e}{A_t} = \frac{1}{M_e}\left[\frac{2}{\gamma+1}(1+\frac{\gamma-1}{2}M_e^2)\right]^{\frac{\gamma+1}{2(\gamma-1)}} \tag{33}$$

From these Mach number values, we can find the exit pressure P_e using Eq. 34 and the mass flow rate $\dot{m}$ using Eq. 35 for each OF ratio. Note that $\bar{R}$ is the universal gas constant.

$$P_e = P_0\left[1+\frac{\gamma-1}{2}M_e^2\right]^{-\frac{\gamma}{\gamma-1}} \tag{34}$$

$$\dot{m} = \left[\frac{\sqrt{\gamma}}{\sqrt{\bar{R}T_0/\bar{M}}}\right]^{\frac{\gamma+1}{2(\gamma-1)}} \tag{35}$$

Assuming that the atmospheric pressure in a vacuum is negligible, we can plug in our calculated values to find the total thrust of the engine using Eq. 36, which can then be used to find the I_{sp} of the engine in Eq. 37.

$$T = \zeta_F\left(P_e A_e + \dot{m}\sqrt{2\frac{\gamma\bar{R}}{(\gamma-1)\bar{M}}T_0\left(1-\left(\frac{P_e}{P_0}\right)^{\frac{\gamma-1}{\gamma}}\right)}\right) \tag{36}$$

$$I_{sp} = \frac{\zeta_F}{\zeta_d} \frac{1}{g_0} \frac{T}{\dot{m}} \tag{37}$$

Note that g_0 is gravity constant 9.8 m/s. Using the data from CEA as well as the calculations above, we then plotted the output vacuum specific impulse I_{sp}, actual thrust T, and adiabatic flame temperature T_0 as a function of oxidizer fuel ratio for ideal, frozen and shifting equilibrium flow, shown in Fig. 4.

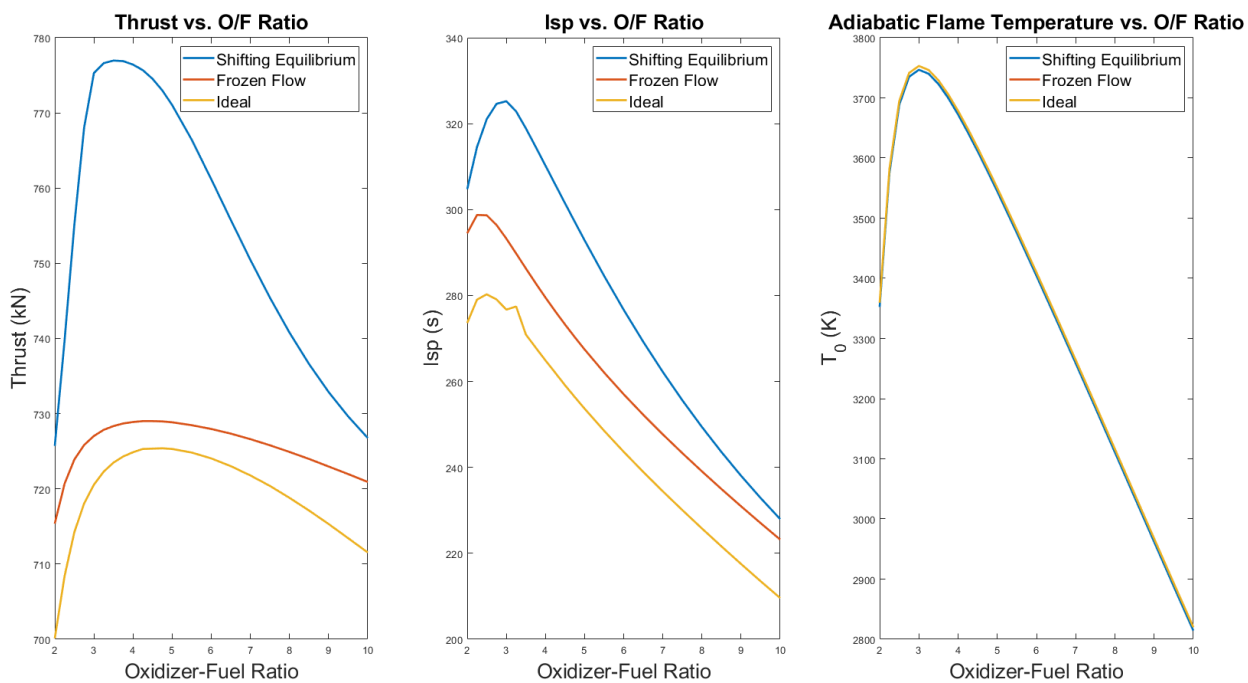

Fig. 4 Thrust, specific impulse, and adiabatic flame temperature for the Merlin 1D as a function of fuel-oxidizer ratio.

From our analysis, we found that the optimal O-F ratio for the greatest I_{sp} was 3.0 for the shifting equilibrium case and 2.25 for the frozen flow case. Because the Merlin engine's vacuum nozzle is very long, we can assume that the flow will behave much more closely to the shifting equilibrium. Even though RP-1 has longer characteristic equilibrium times, we believe that the transit time through the nozzle is sufficiently long enough that the flow behaves like equilibrium flow. Therefore, we decided that our optimal O-F ratio would be set at 3.0.

Next, we varied the area ratio for all cases to see how certain properties were affected. For each case we set the O-F ratio to be the optimal value, and kept it constant. In CEA, we can input the area ratio for both the subsonic and supersonic portions of the nozzle. In the subsonic region, we varied the ratio between the cross sectional area and the throat area from 1 to 4, in increments of 0.25. For the supersonic region, we varied the area ratio for the supersonic region between 1 and 120, with increments of 1 from 1 to 10 and increments of 10 from 10 to 120.

Plugging in these values into CEA, we can find some key quantities as a function of the area ratio for both frozen and equilibrium flow. A sample output from CEA using a varying area ratio is shown in Fig. 5.

Some of the important properties calculated from CEA include the Mach Number M, the ratio of specific heats γ, the average molecular weight $\bar{A}$, the pressure ratio P/P_0, the density ratio ρ/ρ_0 and the temperature ratio T/T_0, where P_0, ρ_0, and T_0 are the pressure, density, and temperature inside the combustion chamber.

It should be noted that CEA does not automatically output the density ratio and temperature ratio. Instead, it simply returns the values for temperature and density at a certain area ratio, and therefore manually divided these values by the

O/F= 2.25000 %FUEL= 30.769231 R,EQ.RATIO= 1.513629 PHI,EQ.RATIO= 1.513629

	CHAMBER	THROAT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT
Pinf/P	1.0000	1.7840	88.180	227.19	393.54	580.69	785.14	1004.62
P, BAR	96.998	54.370	1.1000	0.42694	0.24648	0.16704	0.12354	0.09655
T, K	3582.19	3228.83	1538.61	1264.61	1124.01	1032.03	964.84	912.51
RHO, KG/CU M	7.2191 0	4.4894 0	1.9060-1	9.0009-2	5.8462-2	4.3152-2	3.4138-2	2.8210-2
H, KJ/KG	-824.98	-1563.82	-4923.70	-5420.01	-5665.90	-5823.13	-5936.06	-6022.83
U, KJ/KG	-2168.61	-2774.91	-5500.82	-5894.35	-6087.50	-6210.23	-6297.96	-6365.10
G, KJ/KG	-41928.2	-38612.4	-22578.2	-19930.5	-18563.2	-17665.0	-17006.9	-16493.2
S, KJ/(KG)(K)	11.4743	11.4743	11.4743	11.4743	11.4743	11.4743	11.4743	11.4743
M, (1/n)	22.167	22.167	22.167	22.167	22.167	22.167	22.167	22.167
Cp, KJ/(KG)(K)	2.1021	2.0791	1.8482	1.7716	1.7255	1.6930	1.6681	1.6482
GAMMAs	1.2172	1.2201	1.2546	1.2686	1.2778	1.2846	1.2901	1.2946
SON VEL,M/SEC	1278.9	1215.6	850.9	775.7	734.0	705.2	683.3	665.7
MACH NUMBER	0.000	1.000	3.365	3.908	4.239	4.484	4.679	4.844

PERFORMANCE PARAMETERS

Ae/At	1.0000	10.000	20.000	30.000	40.000	50.000	60.000
CSTAR, M/SEC	1777.4	1777.4	1777.4	1777.4	1777.4	1777.4	1777.4
CF	0.6839	1.6108	1.7056	1.7506	1.7788	1.7988	1.8140
Ivac, M/SEC	2211.9	3064.7	3188.0	3247.1	3284.1	3310.4	3330.4
Isp, M/SEC	1215.6	2863.1	3031.5	3111.6	3161.7	3197.2	3224.2

Fig. 5 Sample CEA output for frozen flow at an O-F ratio of 2.25 with varying area ratio

combustor temperature and density respectively, which are constant for the given O-F ratio.

For ideal flow, we used MATLAB to calculate the values mentioned previously. First, since we assumed that the molecular weight does not change throughout the engine in an ideal case, we simply set the value of $\bar{M}$ for an O-F ratio of 3. This value is equal to that for the frozen flow. We also used the same value for the ratio of specific heats from the frozen flow, and set it as constant for the ideal case.

We can then use the same equations previously to determine the key properties. To calculate how the exit Mach number M_e varies with area ratio, we can again use MATLAB as well as Eq. 33 to numerically solve for M_e for both the subsonic and supersonic cases. Next, using the exit mach numbers for each area ratio we can solve for the pressure ratio P/P_0 using Eq. 34 from earlier, using P instead of P_e. We can then find the density ratio $\frac{\rho}{\rho_0}$ using Eq. 38 below.

$$\frac{\rho}{\rho_0} = \left[1 + \frac{\gamma - 1}{2} M_e^2\right]^{-\frac{1}{\gamma-1}} \tag{38}$$

Finally, to calculate the temperature ratio T/T_0 we can use Eq. 39 below. Note that these equations for the ideal case can be used for both the supersonic and subsonic regions.

$$\frac{T}{T_0} = \left[1 + \frac{\gamma - 1}{2} M_e^2\right]^{-1} \tag{39}$$

Fig. 6 shows these values plotted as a function of the area ratio, for the three cases and in both regions (supersonic and subsonic). Because the plot partially overlap in some cases, it was necessary to differentiate between the ideal, equilibrium and frozen flows using markers.

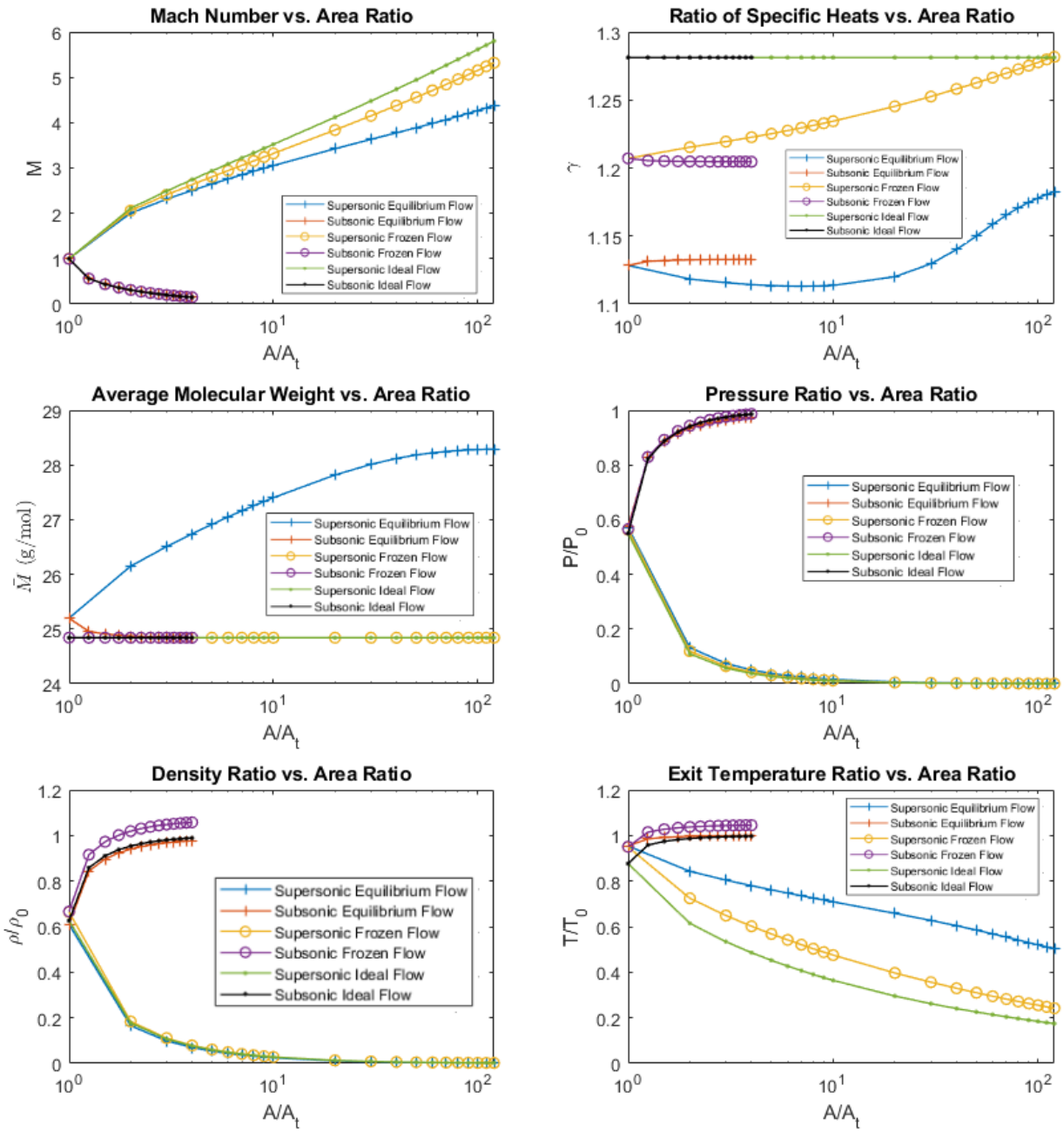

Fig. 6 Plots of key quantities as a function of area ratio for both frozen and equilibrium flow for the Merlin 1D engine at the optimal fuel/oxidizer combination.

For supersonic region, Mach number increases with area ratio. Ideal flow has the highest Mach number at any given area ratio, followed by frozen flow then equilibrium. The ratio of specific heats for ideal flow is constant, while it increases with area for frozen and equilibrium flow. The average molecular stays practically constant for every case except for supersonic equilibrium flow, which increases with area until eventually reaching a limit at an area ratio of ~100. The density ratio and temperature ratio both decrease with the area ratio. The density ratios values three cases are basically the same. For the temperature ratio, equilibrium flow has the highest values followed by frozen flow and then ideal flow.

For the supersonic region, the values for Mach number are the same for all flow types, and decrease as the area ratio increases. The ratio of specific heats is is the greatest for ideal flow, followed by frozen flow and then equilibrium flow. These values all barely change with a change in area ratio. Except for at very low values of area ratio, the average molecular weight in the subsonic region is constant and the same for all three cases. At an area ratio of less than 2, the molecular weight for equilibrium flow is slightly greater, but it rapidly decreases to match the value of the other two cases. The pressure ratio for all flows are basically equal, and increases as the area ratio increases. The density ratio follows a similar trend to the pressure ratio, though the values for subsonic frozen flow are slightly higher. Finally, frozen flow has the greatest exit temperature ratio, followed by equilibrium flow and then ideal flow.

VI. Discussion and Conclusion [10 pts]

A. Analysis of Results

For the J-2 engine, we analyzed the ideal, shifting equilibrium and frozen flow cases through varying A/A_t ratios using a custom CEA solver that would iterate through the values and determine the gas properties throughout the nozzle. We found that, the idealized case would return the most "ideal" performance values followed by shifting equilibrium, then by frozen flow. However, the shifting equilibrium should be the most accurate approximation of the thrust.

For the Merlin engine, we analyzed the ideal, shifting equilibrium and frozen flow cases over a range of oxidizer-fuel ratios using CEA solver and isentropic flow relations. We found that, in general the shifting equilibrium assumption had the most optimistic results, with the largest predicted values for thrust and I_{sp} at all O-F ratios. Frozen flow had the next best results, while the ideal flow analysis returned the most conservative values. Focusing on engine efficiency, we found that the equilibrium, frozen, and ideal flows had the greatest I_{sp} at an O-F ratio of 3, 2.25, and 2.5 respectively. Because we concluded that the flow inside the Merlin engine would like an equilibrium flow, we chose our "optimal" O-F ratio to be 3. Using this optimal O-F ratio, we again used CEA as well as MATLAB to determine different flow properties at any point along the engine for both the subsonic and supersonic regions, again using equilibrium, frozen, or ideal flow analysis.

B. Engine Comparisons

Comparing our performance values for J-2 engine and the Merlin engine, we found that the J-2 engine produces more thrust and has a higher I_{sp} compared to the Merlin engine. The J-2 engine has a maximum thrust value of just below 1000 kN, while the Merlin has a max thrust of about 775 kN. The J-2 engine has a maximum I_{sp} value of around 352 seconds, while the Merlin has a max thrust of about 325 I_{sp}. This is most likely due to the choice of propellant, since a LOX/LH2 propellant gives much better performance and is more efficient than LOX/RP1. However, as we know LOX/LH2 takes up much more volume and presents its own challenges, which is why most modern engines (such as the Merlin) use LOX/RP-1.

Below in Figs. 7 and 8 are the performance tables for the actual J-2 and Merlin 1D engines.

Liquid-fuel engine	
Propellant	Liquid oxygen / Liquid hydrogen
Mixture ratio	5.5:1
Cycle	Gas generator
Configuration	
Nozzle ratio	27.5:1
Performance	
Thrust (vac.)	1,033.1 kN (232,250 lb_f)
Thrust (SL)	486.2 kN (109,302 lb_f)
Thrust-to-weight ratio	73.18
Chamber pressure	5,260 kilopascals (763 psi)
I_{sp} (vac.)	421 seconds (4.13 km/s)
I_{sp} (SL)	200 seconds (2.0 km/s)
Burn time	500 seconds

Fig. 7 Actual J-2 engine performance.

Engine	**Merlin-1CV**	**Merlin-1DV**
Use	Falcon-9 (B1)	Falcon-9 (v1.1)
	 without nozzle extension	
Propellant mix	2.17	2.36
Net flow rate (kg)	157.95	236.56
Thrust sea level	-	-
Thrust vacuum	117,000 lbf (?)	181,000 lbf
Isp s.l. (sec)	-	-
Isp vac (sec)	336	347
Chamber pressure	6.14 MPa	9.72 MPa
Pressure exp. rate	?	3240
Throat area (m^2)	0.042	0.042
Nozzle area (m^2)	4.90	4.90
= Area exp. ratio	117	117
Nozzle L/D (m)	2.70 x 2.50	2.70 x 2.50

Fig. 8 Actual Merlin 1D vacuum engine performance.

For the J-2 engine, the actual value for max thrust is 1033.1 kN and the I_{sp} is 421 seconds. Our thrust values are somewhat smaller than the actual values (between 973 to 1015 kN depending on the flow type), and our specific impulse values are also smaller (between 351 and 376 seconds). This leads us to conclude that the cause of this discrepancy would most likely be caused from the correction factors, which might be over-conservative. Our analysis could also be off because of the information that we were given (such as throat area, contraction ratio, etc.)

For the Merlin-1D Vacuum engine, it has a reported thrust value of 181000 lbf, which is around 805 kN. Compared to our maximum calculated thrust of 775.3 kN, the actual value is a bit larger than what we calculated. This could be due to our given thrust correction factor, which may have been too conservative. This could also be due to the fact that our given contraction ratio is different than that of the actual Merlin engine, which could have given us lower performance estimates. SpaceX's Merlin engine also has a reported I_{sp} of 347 seconds, which differs from our calculated optimum I_{sp} of 376.6 seconds. This could be due to the discharge correction factor being inaccurate, or due to engine inefficiencies that were not accounted for in our analysis. Finally, it is important to note that the Merlin engine does not behave totally in equilibrium flow as we have assumed. Therefore, there will always be an inherent discrepancy between our analysis and the actual values.

VII. Appendix

A. Thermodynamic Properties

We list in the following tables the coefficients we used for each of the reactants and products for the LOX/LH2 combustion process.

Table 8 Coefficients for O2

Temperature (K)	100. - 700.	700. - 2000.	2000. - 6000.
A	31.32234	30.03235	20.91111
B	-20.23531	8.772972	10.72071
C	57.86644	-3.988133	-2.020498
D	-36.50624	0.788313	0.146449
E	-0.007374	-0.741599	9.245722
F	-8.903471	-11.32468	5.337651
G	246.7945	236.1663	237.6185
H	0.0	0.0	0.0
Reference	Chase, 1998	Chase, 1998	Chase, 1998
Comment	Data last reviewed in March, 1977; New parameter fit January 2009	Data last reviewed in March, 1977; New parameter fit January 2009	Data last reviewed in March, 1977; New parameter fit January 2009

Table 9 Coefficients for H2

Temperature (K)	298. - 1000.	1000. - 2500.	2500. - 6000.
A	33.066178	18.563083	43.413560
B	-11.363417	12.257357	-4.293079
C	11.432816	-2.859786	1.272428
D	-2.772874	0.268238	-0.096876
E	-0.158558	1.977990	-20.533862
F	-9.980797	-1.147438	-38.515158
G	172.707974	156.288133	162.081354
H	0.0	0.0	0.0
Reference	Chase, 1998	Chase, 1998	Chase, 1998
Comment	Data last reviewed in March, 1977; New parameter fit October 2001	Data last reviewed in March, 1977; New parameter fit October 2001	Data last reviewed in March, 1977; New parameter fit October 2001

Table 10 Coefficients for H

Temperature (K)	298. - 6000.
A	20.78603
B	4.850638×10^{-10}
C	-1.582916×10^{-10}
D	1.525102×10^{-11}
E	3.196347×10^{-11}
F	211.8020
G	139.8711
H	217.9994
Reference	Chase, 1998
Comment	Data last reviewed in March, 1982

Table 11 Coefficients for H2O

Temperature (K)	500. - 1700.	1700. - 6000.
A	30.09200	41.96426
B	6.832514	8.622053
C	6.793435	-1.499780
D	-2.534480	0.098119
E	0.082139	-11.15764
F	-250.8810	-272.1797
G	223.3967	219.7809
H	-241.8264	-241.8264
Reference	Chase, 1998	Chase, 1998
Comment	Data last reviewed in March, 1979	Data last reviewed in March, 1979

Table 12 Coefficients for OH

Temperature (K)	298. - 1300.	1300. - 6000.
A	32.27768	28.74701
B	-11.36291	4.714489
C	13.60545	-0.814725
D	-3.846486	0.054748
E	-0.001335	-2.747829
F	29.75113	26.41439
G	225.5783	214.1166
H	38.98706	38.98706
Reference	Chase, 1998	Chase, 1998
Comment	Data last reviewed in June, 1977	Data last reviewed in June, 1977

These tables are to be used and implemented in our CEA solver to approximate combustion analysis and is shown below in our supporting code Appendix section.

B. Atomic Oxygen Shomate's Coefficients

Because there is limited knowledge known about the performance of atomic oxygen. In our solver we treat the product atomic oxygen as having three translational degrees of freedom, keeping the value of specific heat at constant pressure C_p constant as a function of temperature. Additional information from the NIST database also assumes that the entropy of gas as standard condition, S_o, is equal to 161 J/mol-K.

In order to derive the Shomate Coefficients A through H, we first start with the definition of enthalpy in the following equation:

$$C_p \equiv \frac{dH}{dT} \tag{40}$$

Where C_p is the specific heat at constant pressure of a atom capable of three degrees of translational motion. C_p of an atom or molecule is determined by the following equation:

$$C_p = \left(n\frac{\bar{R}}{2} + \bar{R}\right) \tag{41}$$

Where n = 3 degrees of freedom and $\bar{R} = 8.314\frac{J}{molK}$. Evaluating this equation we get that $C_p = 21.9\frac{J}{molK}$ for atomic oxygen. Taking this knowledge, we then integrate both sides of Equation VII.B with respect to enthalpy and temperature. Solving with the initial state being at standard temperature and pressure, we find the following equation:

$$H - H_{298.15K} = C_p(T - T_{stp}) \tag{42}$$

This matches Equation 2. However because there is only one value dependent on temperature raised to the first power and a constant independent of temperature, the coefficient values of B, C, D, E are equal to zero. Given This makes the following simplification of Equation 2 possible:

$$H - H_{298.15K} = A(t) + F - H \tag{43}$$

There is only one term linearly dependent on temperature t when comparing Equation 2 and Equation 43. Therefore coefficient A is equal to molecular specific heat at constant pressure of oxygen C_{pO}, and the difference of coefficient F and H are equivalent to the product of C_{pO} and the Temperature at STP.

In the CEA solver we also use an entropy derivation from the first law of thermodynamics to calculate entropy for atomic oxygen:

$$ds = \frac{dh}{T} - \bar{R}\frac{dP}{P} \tag{44}$$

After several substitutions and integrating with respect to the initial condition at standard temperature and pressure we get the following:

$$s_i(T, P) = s_i^0(T) - \bar{R}ln\left(\frac{n_i}{\sum_i n_i}\right) - \bar{R}ln\left(\frac{P}{P_{stp}}\right) \tag{45}$$

This equation is similar to that of the previous Shomate equation seen in Equation 3. However there is one term dependent on temperature T and two that are constant and functions of molar concentration and Pressure evaluated at stagnation pressure P_o. Though this form is ideal to use in the CEA solver code, a more practical form is required to solve for the Shomate's coefficients that have yet to be defined. In order to get a more usable form of the equation, we solve Equation 3 at the value of t = 298.15/1000 K at STP. Along with the coefficient values B, C, D, and E being equal to zero we find the following simplification:

$$S_o(T = 295.15) = 161.059 = A\left(\frac{295.15}{1000}\right) + G \tag{46}$$

Solving for G we find that G = 186.17. Based on our calculations, the values for our final Shomates coefficients are tabulated below:

Table 13 Calculated Shomate's Coefficients for Atomic Oxygen.

A	3387
B	0
C	0
D	0
E	0
G	186.17
Quantity (F-H)	6188

C. Supporting Code

Listing 1 Matlab implementation of CEA solver.

```
close all; clear all;
%% Part 1a
%  !!!!!!!!!!!!!! Note that t = Temp /1000!!!!!!!!!!!!!!

% Regime 1 from 500 to 700 K (Limited by o2 upper bound) (h2o on lower
% bound)



% Regime 2 from 700 to 1000 K (limited by h2)

%Regime 3 from 1000K to 1300K

% Regime 4 from 1300K to 1700K (limited to h2o)


% %Regime 5 from 1700K to 2000K (limited by O2)

% %Regime 6 from 2000K to 2500K


%Regime 7 from 2500K to 6000K



%Energy Conservation
% eqn9 = no2*H_delo2 + nh2*H_delh2 + nh*H_delh + noh*H_deloh + no*H_delo + nh2o*H_delh2o == noxi*
    hfor_o2 + nf*hfor_h2 - no2*hfor_o2 - nh2*hfor_h2 - nh*hfor_h - noh*hfor_oh - no*hfor_o - nh2o
    *hfor_h2o + noxi*H_delo2 + nf*H_delh2;


T0_guess = 3000;
% T0_guess = 3389;
%enthalpies of formation
global hfor_h2 hfor_o2 hfor_h2o hfor_o hfor_h hfor_oh

hfor_h2 = 0; %All in J/mol
hfor_o2 = 0;
hfor_h2o = -241.8e3;
hfor_o = 249e3;
hfor_h = 218e3;
% hfor_oh = 38.99e3;
hfor_oh = 37.49e3;
% hfor_oh = -139.060e3;

%initial guesses (close-ish to CEA)
lam_oguess = 10e3;
lam_hguess = 10e3;
no2g = 0.0021;
nh2g = 0.6;
nhg = 0.033;
nohg = 0.032;
nog = 0.002;
nh2og = 0.30;
enth_oxi_boil = -12.979e3; %j/mol
enth_f_boil = -9.012e3; %j/mol
latent_heat_oxi = 3.4099e3; %o2
latent_heat_f = 0.44936e3;%h2
X0 = [lam_oguess lam_hguess no2g nh2g nhg nohg nog nh2og];
%lam_o lam_h no2 nh2 nh noh no nh2o

T_e = 0.5; %assume initial error of 20%
while(T_e > 1e-4)

   [mols,enth,moxifuel, sanity] = J2engine(T0_guess,X0);
    mo2 = mols(3);
    mh2 = mols(4);
    mh = mols(5);
```

```
    moh = mols(6);
    mo = mols(7);
    mh2o = mols(8);
%     mo2 = 0.00179;
%     mh2 = 0.29410;
%     mh = 0.03390;
%     moh = 0.03162;
%     mo = 0.00214;
%     mh2o = 0.63643;
    moxi = moxifuel(1);
    mf = moxifuel(2); %no2 nh2 nh noh no nh2o
    h_delo2 = enth(1);
    h_delh2 = enth(2);
    h_delh = enth(3);
    h_deloh = enth(4);
    h_delo = enth(5);
    h_delh2o = enth(6);
%     e_combustion = mo2*h_delo2 + mh2*h_delh2 + mh*h_delh + moh*h_deloh + mo*h_delo + mh2o*h_delh2o;
%     e_reaction = moxi*hfor_o2 + mf*hfor_h2 - mo2*hfor_o2 - mh2*hfor_h2 - mh*hfor_h - moh*hfor_oh - mo*hfor_o - mh2o*hfor_h2o + moxi*h_delo2 + mf*h_delh2;
    e_combustion = mo2*h_delo2 + mh2*h_delh2 + mh*h_delh + moh*h_deloh + mo*h_delo + mh2o*h_delh2o;
    e_reaction = moxi*hfor_o2 + mf*hfor_h2 - mo2*hfor_o2 - mh2*hfor_h2 - mh*hfor_h - moh*hfor_oh - mo*hfor_o - mh2o*hfor_h2o + moxi*(enth_oxi_boil - hfor_o2 + latent_heat_oxi) + mf*(enth_f_boil - hfor_h2 + latent_heat_f);
    T_e = abs((e_combustion - e_reaction)./e_combustion);
    if e_combustion < e_reaction
        T0_guess_mult = 1 + T_e/4;
        T0_guess = T0_guess*T0_guess_mult;
    else
        T0_guess_mult2 = 1 - T_e/4;
        T0_guess = T0_guess*T0_guess_mult2;
    end



end

%Summary of Results
 Combuster_pressure = P0
 Fuel_oxi_ratio = 5.5
 Tflame_temp = T0_guess
 Molar_concentration_names = ['o2' 'h2' 'h' 'oh' 'o' 'h2o'];
 Molar_concentration_values = [mo2, mh2, mh, moh, mo, mh2o];

% order no2 nh2 nh noh no nh2o
% Notes: enthalpies check out, formations check out, mols check out


% TEST REGULAR FUNCTION USING THIS
% [mols,enth,moxifuel,sanity] = J2engine(T0_guess,X0);

% Calculate T0

function [X,enthalpies,noxifuel,check] = J2engine(T0, X0_guess)
syms lam_o lam_h no2 nh2 nh noh no nh2o

assume(no2,'Real');
assume(nh2,'Real');
% assume(nh,'Real');
% assume(noh,'Real');
% assume(no,'Real');
assumeAlso(nh2o,'Real');
assumeAlso(no2 >= 0);
assumeAlso(nh2 >= 0);
% assumeAlso(nh >= 0);
% assumeAlso(noh >= 0);
% assumeAlso(no >= 0);
assumeAlso(nh2o >= 0);

% Regime = 7;
```

```
R0 = 500;
R1 = 700;
R2 = 1000;
R3 = 1300;
R4 = 1700;
R5 = 2000;
R6 = 2500;
R7 = 6000;
t = T0/1000;
% Coefficients for H2 (diatomic)
% if Regime <= 2
if T0 >= R0 && T0 <= R2
    ah2 = 33.066178;
    bh2 = -11.363417;
    ch2 = 11.432816;
    dh2 = -2.772874;
    eh2 = -0.158558;
    fh2 = -9.980797;
    gh2 = 172.707974;
    hh2 = 0;
end
% if Regime > 2 && Regime <= 6
if T0 > R2 && T0 <= R6
    ah2 = 18.563083;
    bh2 = 12.257357;
    ch2 = -2.859786;
    dh2 = 0.268238;
    eh2 = 1.97799;
    fh2 = -1.147438;
    gh2 = 156.288133;
    hh2 = 0;
end

% if Regime == 7
if T0 > R6 && T0 <= R7

    ah2 = 43.41356;
    bh2 = -4.293079;
    ch2 = 1.272428;
    dh2 = -0.096876;
    eh2 = -20.533862;
    fh2 = -38.515158;
    gh2 = 162.081354;
    hh2 = 0;
end


[Cph2, H_delh2, Sh2] = basedOnCoeff(t,ah2,bh2,ch2,dh2,eh2,fh2,gh2,hh2);

% Coefficients for O2
% if Regime == 1
if T0 >= R0 && T0 <= R1

    ao2 =31.32234;
    bo2 =-20.23531;
    co2 =57.86644;
    do2 =-36.50624;
    eo2 =-0.007374;
    fo2 =-8.903471;
    go2 =246.7945;
    ho2 = 0;
end
% if Regime >= 2 && Regime <= 5
if T0 > R1 && T0 <= R5

    ao2 = 30.03235;
    bo2 = 8.772972;
    co2 = -3.988133;
    do2 = 0.788313;
    eo2 = -0.741599;
    fo2 = -11.32468;
    go2 = 236.1663;
```

```
    ho2 =0;
end

% if Regime == 6 || Regime == 7
if T0 > R5 && T0 <= R7

    ao2 = 20.91111;
    bo2 = 10.72071;
    co2 = -2.020498;
    do2 = 0.146449;
    eo2 = 9.245722;
    fo2 = 5.337651;
    go2 = 237.6185;
    ho2 = 0;
end


[Cpo2, H_delo2, So2] = basedOnCoeff(t,ao2,bo2,co2,do2,eo2,fo2,go2,ho2);




% Coefficients for h2o
% if Regime >= 1 && Regime <= 4
if T0 >= R0 && T0 <= R4

    ah2o = 30.09200;
    bh2o = 6.832514;
    ch2o = 6.793435;
    dh2o = -2.53448;
    eh2o = 0.082139;
    fh2o = -250.881;
    gh2o = 223.3967;
    hh2o = -241.8264;
end

% if Regime >= 5 && Regime <= 7
if T0 > R4 && T0 <= R7
    ah2o = 41.96426;
    bh2o = 8.622053;
    ch2o = -1.49978;
    dh2o = 0.098119;
    eh2o = -11.15764;
    fh2o = -272.1797;
    gh2o = 219.7809;
    hh2o = -241.8264;
end



[Cph2o, H_delh2o, Sh2o] = basedOnCoeff(t,ah2o,bh2o,ch2o,dh2o,eh2o,fh2o,gh2o,hh2o);

% Monotonic Hydrogen H (this does not change with regime)

ah = 20.78603;
bh = 4.850638*10.^-10;
ch = -1.5825102*10.^-10;
dh = 1.525102*10.^-11;
eh = 3.196347*10.^-11;
fh = 211.802;
gh = 139.8711;
hh = 217.9994;
[Cph, H_delh, Sh] = basedOnCoeff(t,ah,bh,ch,dh,eh,fh,gh,hh);

% Monotonic Hydrogen O (this does not change with regime)

ao = 20.78603;
bo = 4.850638*10.^-10;
co = -1.5825102*10.^-10;
do = 1.525102*10.^-11;
eo = 3.196347*10.^-11;
fo = 211.802;
```

```
go = 186.2131;
ho = 217.9994;
[Cpo, H_delo, So] = basedOnCoeff(t,ao,bo,co,do,eo,fo,go,ho);

%OH (Hydroxl radical ??)
% if Regime >= 1 && Regime <= 3
if T0 >= R0 && T0 <= R3

    aoh = 32.27768;
    boh = -11.36291;
    coh = 13.60545;
    doh = -3.846486;
    eoh = -0.001335;
    foh = 29.75113;
    goh = 225.5783;
    hoh = 38.98706;
end
% if Regime >= 4 && Regime <= 7
if T0 > R3 && T0 <= R7

    aoh = 28.74701;
    boh = 4.714489;
    coh = -0.814725;
    doh = 0.054748;
    eoh = -2.747829;
    foh = 26.41439;
    goh = 214.1166;
    hoh = 38.98706;
end

[Cpoh, H_deloh, Soh] = basedOnCoeff(t,aoh,boh,coh,doh,eoh,foh,goh,hoh);


% enthalpies = [H_delo2 H_delh2 H_delh H_deloh H_delo H_delh2o];

%Still need monotomic oxygen
% use enthalpy cp relations for monotomic oxygen. Slide 29 of combustion chamber analysis simple
      combustion. Need to figur out
% to incorporate energy per mol (or make it specific enthalpy
%Assign Hdel and S standard here

% !!!!!!!!!!!!!! Note that t = Temp /1000!!!!!!!!!!!!!!

% -------------------- Minimizzing free energy -------------------

%Things I need to check
        %Are the OH values correct?
        %Note: ALL P measurements will be in atm

global hfor_h2 hfor_o2 hfor_h2o hfor_o hfor_h hfor_oh

%Stagnation Pressure
P0 = 5.3e6; %Pa

%Note WE ARE SCALING EVERYTHING TO 1 MOL OF FUEL
nf = 1; %1 mol
nf_div_noxi = 5.5;

mbarh2 = 2;
mbaro2 = 32;

% mdotf = nf*mbarh2 ;
% mdotoxi = mdotf*nf_div_noxi;
% noxi = mdotoxi/mbaro2;
noxi = nf_div_noxi * (mbarh2/mbaro2);

nj = no2 + nh2 + nh + noh + no + nh2o;

R_uni = 8.314; %J?K^?1?mol^?1
P_stp = 101.325e3; %Pa

%Vars For atomic O
```

```
% Hfor_o = 249.18e3; %j/mol --> this is the heat of formation
% Cp_o = 21.9; %j/mol
Cp_o = 20.758;
S_stan_o = 161.1; %J K-1mol-1

% %Equations (Minizing Gibbs)
eqn1 = 0 == H_delo2 + hfor_o2 - T0*So2 + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(no2./nj) + 2*lam_o ; %O2

eqn2 = 0 == H_delh2 + hfor_h2 - T0*Sh2 + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(nh2./nj) + 2*lam_h; %H2

eqn3 = 0 == H_delh + hfor_h - T0*Sh + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(nh./nj) + lam_h; %H

eqn4 = 0 == H_deloh + hfor_oh - T0*Soh + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(noh./nj) + lam_o + lam_h ; %OH

%sub in equation for constant cp, and enthalpy and entropy equations for
%eqn 5
        % cp = dH/dT (partial)
        % (H - H_298.5) = Cp_o *(T0 - 298.5)
        %  T0*Si(T0) = T0*S_stan_o + T0*Cp_o*log(T0./298.5) - T0*Rbar*log(no./nj) -
        %  T0*R_uni*log(P0./P_stp)
        % eqn5 = 0 == Cp_o *(T0 - 298.5) - T0*S_stan_o - T0*Cp_o*log(T0./298.5) + T0*R_uni*log(no./nj) + T0*R_uni*log(P0./P_stp) + lam_o;
% eqn5 = 0 == H_delo + hfor_o - T0*So + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(no./nj) + lam_o; %O

H_delo_r = Cp_o *(T0 - 298.5);

eqn5 = 0 == H_delo_r + hfor_o - T0*S_stan_o - T0*Cp_o*log(T0./298.5)+ T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(no./nj) + lam_o; %O

eqn6 = 0 == H_delh2o + hfor_h2o - T0*Sh2o + T0*R_uni*log(P0./P_stp)+ T0*R_uni*log(nh2o./nj) + 2*lam_h + lam_o; %H2O

enthalpies = [H_delo2 H_delh2 H_delh H_deloh H_delo_r H_delh2o];


%Equations (Minizing Gibbs + Normalization)
% eqn1 = 0 == (H_delo2 + hfor_o2)/(T0*R_uni) - So2/R_uni + log(P0./P_stp)+ log(no2./nj) + 2*lam_o ; %O2
%
% eqn2 = 0 == (H_delh2 + hfor_h2)/(T0*R_uni) - Sh2/R_uni + (P0./P_stp)+ log(nh2./nj) + 2*lam_h; %H2
%
% eqn3 = 0 == (H_delh + hfor_h)/(T0*R_uni) - Sh/R_uni + log(P0./P_stp)+ log(nh./nj) + lam_h; %H
%
% eqn4 = 0 == (H_deloh + hfor_oh)/(T0*R_uni) - Soh/R_uni + log(P0./P_stp)+ log(noh./nj) + lam_o + lam_h ;  %OH
%
% eqn5 = 0 == (H_delo + hfor_o)/(T0*R_uni) - So/R_uni + log(P0./P_stp)+ log(no./nj) + lam_o; %O
%
% eqn6 = 0 == (H_delh2o + hfor_h2o)/(T0*R_uni) - Sh2o/R_uni + log(P0./P_stp)+ log(nh2o./nj) + 2*lam_h + lam_o; %H2O
%
% Mass Conservation
eqn7 = 0== -2*noxi + 2*no2 + noh + no + nh2o;
eqn8 = 0 == -2*nf +2*nh2 + nh + noh + 2*nh2o;

%Solver
sol = vpasolve([eqn1 eqn2 eqn3 eqn4 eqn5 eqn6 eqn7 eqn8],[lam_o lam_h no2 nh2 nh noh no nh2o],X0_guess);


% sum_j = double(sol.no2) + double(sol.nh2) + double(sol.nh) + double(sol.noh) + double(sol.no) + double(sol.nh2o);

X = [double(sol.lam_o) double(sol.lam_h) double(sol.no2) double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.no) double(sol.nh2o) ];
noxifuel = [noxi nf];
```

```
checkeqn1 = subs(eqn1, [lam_o, no2, nh2, nh, noh, no, nh2o],[double(sol.lam_o) double(sol.no2) double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.no) double(sol.nh2o)]);
checkeqn2 = subs(eqn2, [lam_h, no2, nh2, nh, noh, no, nh2o],[double(sol.lam_h) double(sol.no2) double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.no) double(sol.nh2o)]);
checkeqn3 = subs(eqn2, [lam_h, no2, nh2, nh, noh, no, nh2o],[double(sol.lam_h) double(sol.no2) double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.no) double(sol.nh2o)]);
checkeqn4 = subs(eqn4,[lam_o, lam_h, no2, nh2, nh, noh, no, nh2o],X);
checkeqn5 = subs(eqn5, [lam_o, no2, nh2, nh, noh, no, nh2o],[double(sol.lam_o) double(sol.no2) double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.no) double(sol.nh2o)]);
checkeqn6 = subs(eqn6,[lam_o, lam_h, no2, nh2, nh, noh, no, nh2o],X);
checkeqn7 = subs(eqn7,[no2, noh, no, nh2o],[double(sol.no2) double(sol.noh) double(sol.no) double(sol.nh2o)]);
checkeqn8 = subs(eqn8,[nh2, nh, noh, nh2o],[double(sol.nh2) double(sol.nh) double(sol.noh) double(sol.nh2o)]);


check(1) = vpa(checkeqn1);
check(2) = vpa(checkeqn2);
check(3) = vpa(checkeqn3);
check(4) = vpa(checkeqn4);
check(5) = vpa(checkeqn5);
check(6) = vpa(checkeqn6);
check(7) = vpa(checkeqn7);
check(8) = vpa(checkeqn8);
% check = 0;
% Note all values for energy are now in Joules !!!!!!!!!!!!!!


end

function [Cp, Hdel, S] = basedOnCoeff(t,a,b,c,d,e,f,g,h)

Cp = a + b*t + c*t.^2 + d*t.^3 + e./t.^2;
Hdel = a*t + (1./2)*b*t.^2 + (1./3)*c*t.^3 + (1./4)*d*t.^4 -e./t + f - h;
S = a*log(t) + b*t + (1./2)*c*t.^2 + (1./3)*d*t.^3 - e./(2*t.^2) + g;
Hdel = Hdel*10.^3; %NIST gave this in units of kJ for some reason
end
```

Listing 2 Matlab code to determine J-2 engine performance.

```matlab
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
set(0, 'defaultaxesfontsize', 16);
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Universal Constants / Variables
runi = 8.314; % Universal Gas Constant R, J/k-mol
g0 = 9.81; % Acceleration at sea level
at = 0.1092;  % J-2 Throat Area, m^2
epsi = 27.5;  % Epansion ratio
p0 = 5.3e6;   % Chamber pressure, Pa
etad = 1.1;   % Discharge Correction
etaf = 0.95;  % Thrust Correction
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%%
% Ideal Flow Analysis

% CEA for Idealized Flow
gam = 1.2;
molmass = 0.012510058715950;
t0 = 3.387478019e3;

num = 100;
aat = [linspace(4, 1, num), linspace(1, epsi, num)];
mlin = zeros(1, max(size(aat)));
syms mach
tic
for i = 1:num
   eqn = aat(i) == aastarratio(mach, gam);
   sol = vpasolve(eqn, mach, [0, 1]);
   mlin(i) = sol;
end
for i = num+1:2*num
   eqn = aat(i) == aastarratio(mach, gam);
   sol = vpasolve(eqn, mach, [1, Inf]);
   sol = sol(sol >= 1);

   mlin(i) = sol(1);
end
fprintf('\nJ-2 Ideal Case - Complete\n-------------------------\n')
toc

pp0 = 1./stagpratio(mlin, gam);
rhorho0 = 1./stagrhoratio(mlin, gam);
tt0 = 1./stagtratio(mlin, gam);
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% J-2 Ideal Flow Performance Analysis
mdot = sqrt(gam/(runi/molmass * t0)) * p0*at/((gam+1)/2)^((gam+1)/(2*(gam-1)));
T = (pp0(end)*p0 - 1e3)*aat(end)*at + mdot*sqrt(2*gam*runi/(molmass*(gam-1))*t0*(1 -
     pp0(end)^((gam-1)/gam)));
mdot = mdot*etad;
T = T*etaf;
isp = 1/g0 * (T/mdot);
cstar = ((gam+1)/2)^((gam+1)/(2*(gam-1)))*sqrt(runi*t0/(gam*molmass));
cf = T/(cstar*mdot);

fprintf('\n\nIdealized Flow Analysis\n--------------------');
fprintf('\nThrust = %.2f kN', T/1e3);
fprintf('\nIsp = %.2f sec', isp);
fprintf('\nmdot = %.2f kg/s', mdot);
fprintf('\nc* = %.2f m/s', cstar);
fprintf('\nCf = %.2f ', cf);

fid = fopen('j2_ic_performance','w');
fprintf(fid,'\n Ideal Flow & %.2f & %.2f & %.2f & %.2f & %.2f \\\\ \\hline',T/1e3,
    isp, mdot, cstar, cf);
fclose(fid);
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
%%
% Frozen Flow Analysis
names = [{'o2', 'h2', 'h', 'oh', 'o', 'h2o'}];
molsi = [0.0017, 0.3172, 0.0361, 0.0344, 0.0020, 0.6475];
masses = [32, 2, 1, 17, 16, 18].*1e-3;
t0 = 3.387478019e3;

tic
s0 = 0; h0 = 0;
for i = 1:length(molsi)
    [stp, htp, cp] = gassProp(t0, names(i));

    s0 = s0 + molsi(i)*(stp - runi*log(molsi(i)/sum(molsi)) - runi*log(p0/101.325e3));
    h0 = h0 + molsi(i)*htp;
end
h0 = h0/sum(molsi.*masses);

machff = []; gamff = []; mwff = []; pff = []; rhoff = []; tff = []; aeff = [];
pff(1) = p0; tff(1) = t0; aeff(1) = 4; mwff(1) = sum(molsi.*masses)/sum(molsi);
loop_true = true; i = 1;
while loop_true
    pff(i+1) = pff(i)*0.975;
    tff(i+1) = tff(i);
    [machff(i), gamff(i), mwff(i+1), rhoff(i), tff(i+1), ~] = calcTemp(pff(i+1), tff(i+1), names, s0, h0, p0, molsi, 'ff');

    aeff(i+1) = aastarratio(machff(i), gamff(i));
    if aastarratio(machff(i), gamff(i)) > epsi
      loop_true = false;
    end
    i = i + 1;
end
fprintf('\n\n\nJ-2 Frozen Flow Case - Complete\n-------------------------------\n')
toc
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% J-2 Frozen Flow Performance Analysis
mdot = sqrt(gamff(end)/(runi/mwff(end) * t0)) * p0*at/((gamff(end)+1)/2)^((gamff(end)+1)/(2*(gamff(end)-1)));
T = (pff(end) - 1e3)*epsi*at + mdot*sqrt(2*gamff(end)*runi/(mwff(end)*(gamff(end)-1))*t0*(1 - (pff(end)/p0)^((gamff(end)-1)/gamff(end))));
mdot = mdot*etad;
T = T*etaf;
isp = 1/g0 * (T/mdot);
cstar = ((gamff(end)+1)/2)^((gamff(end)+1)/(2*(gamff(end)-1)))*sqrt(runi*t0/(gamff(end)*mwff(end)));
cf = T/(cstar*mdot);

fprintf('\n\nFrozen Flow Analysis\n--------------------');
fprintf('\nThrust = %.2f kN', T/1e3);
fprintf('\nIsp = %.2f sec', isp);
fprintf('\nmdot = %.2f kg/s', mdot);
fprintf('\nc* = %.2f m/s', cstar);
fprintf('\nCf = %.2f ', cf);

fid = fopen('j2_ff_performance','w');
fprintf(fid,'\n Frozen Flow & %.2f & %.2f & %.2f & %.2f & %.2f \\\\ \\hline',T/1e3, isp, mdot, cstar, cf);
fclose(fid);
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%%
% J-2 Shifting Equilibrium Flow Analysis
names = [{'o2', 'h2', 'h', 'oh', 'o', 'h2o'}];
molsi = [0.0017, 0.3172, 0.0361, 0.0344, 0.0020, 0.6475];
masses = [32, 2, 1, 17, 16, 18].*1e-3;
t0 = 3.387478019e3;

tic
s0 = 0; h0 = 0;
for i = 1:length(molsi)
    [stp, htp, cp] = gassProp(t0, names(i));

```

```matlab
    s0 = s0 + molsi(i)*(stp - runi*log(molsi(i)/sum(molsi)) - runi*log(p0/101.325e3));
    h0 = h0 + molsi(i)*htp;
end
h0 = h0/sum(molsi.*masses);

machse = []; gamse = []; mwse = []; pse = []; rhose = []; tse = []; aese = [];
pse(1) = p0; tse(1) = t0; aese(1) = 4; mwse(1) = sum(molsi.*masses)/sum(molsi);
loop_true = true; i = 1;
while loop_true
    pse(i+1) = pse(i)*0.975;
    tse(i+1) = tse(i);
    [machse(i), gamse(i), mwse(i+1), rhose(i), tse(i+1), molsi] = calcTemp(pse(i+1), tse(i+1), names, s0, h0, p0, molsi, 'se');

    aese(i+1) = aastarratio(machse(i), gamse(i));
    if aastarratio(machse(i), gamse(i)) > epsi
      loop_true = false;
    end
    i = i + 1;
end
fprintf('\n\n\nShifting Equilibrium Case - Complete\n------------------------------------\n')
toc
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% J-2 Shifting Equilibrium Flow Performance Analysis
mdot = sqrt(gamse(end)/(runi/mwse(end) * t0)) * p0*at/((gamse(end)+1)/2)^((gamse(end)+1)/(2*(gamse(end)-1)));
T = (pse(end) - 1e3)*epsi*at + mdot*sqrt(2*gamse(end)*runi/(mwse(end)*(gamse(end)-1))*t0*(1 - (pse(end)/p0)^((gamse(end)-1)/gamse(end))));
mdot = mdot*etad;
T = T*etaf;
isp = 1/g0 * (T/mdot);
cstar = ((gamse(end)+1)/2)^((gamse(end)+1)/(2*(gamse(end)-1)))*sqrt(runi*t0/(gamse(end)*mwse(end)));
cf = T/(cstar*mdot);

fprintf('\n\nShifting Equilibrium Flow Analysis\n--------------------');
fprintf('\nThrust = %.2f kN', T/1e3);
fprintf('\nIsp = %.2f sec', isp);
fprintf('\nmdot = %.2f kg/s', mdot);
fprintf('\nc* = %.2f m/s', cstar);
fprintf('\nCf = %.2f ', cf);

fid = fopen('j2_se_performance','w');
fprintf(fid,'\n Shifting Equilibrium Flow & %.2f & %.2f & %.2f & %.2f & %.2f \\\\ \\hline',T/1e3, isp, mdot, cstar, cf);
fclose(fid);
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%%
% Figure and Post-Processing
close all

idx = find(abs(mlin - 1) <= 1e-3);
idxff = find(abs(machff - 1) <= 0.125e-1)-1;
idxse = find(abs(machse - 1) <= 0.025e-1);


figure()
subplot(3,2,1)
hold on
plot(aat(1:idx(1)), mlin(1:idx(1)), 'g-', 'linewidth', 2)
plot(aat(idx(2):end), mlin(idx(2):end), 'k-', 'linewidth', 2)
plot(aeff(2:idxff), machff(1:idxff-1), 'b-', 'linewidth', 2)
plot(aeff(idxff:end), machff(idxff-1:end), 'r-', 'linewidth', 2)
plot(aese(1:idxse), machse(1:idxse), 'm-', 'linewidth', 2)
plot(aese(idxse+1:end), machse(idxse:end), 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$M$')
legend({'Subsonic-IC', 'Supersonic-IC'}, 'fontsize', 16, 'location', 'southeast')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', '
```

```
    right')

subplot(3,2,2)
hold on
plot(aat(1:idx(1)), gam.*ones(1, max(size(aat(1:idx(1))))), 'g-', 'linewidth', 2, 'handlevisibility', 'off')
plot(aat(idx(2):end), gam.*ones(1, max(size(aat(idx(2):end)))), 'k-', 'linewidth', 2, 'handlevisibility', 'off')
plot(aeff(2:2:idxff), gamff(1:2:idxff-1), 'b-', 'linewidth', 2)
plot(aeff(idxff:4:end), gamff(idxff-1:4:end), 'r-', 'linewidth', 2)
plot(aese(1:idxse), gamse(1:idxse), 'm-', 'linewidth', 2)
plot(aese(idxse+1:end), gamse(idxse:end), 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$\gamma$')
legend({'Subsonic-FF', 'Supersonic-FF'}, 'fontsize', 16, 'location', 'southeast')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', 'right')

subplot(3,2,3)
hold on
plot(aat(1:idx(1)), molmass.*1e3.*ones(1, max(size(aat(1:idx(1))))), 'g-', 'linewidth', 2, 'handlevisibility', 'off')
plot(aat(idx(2):end), molmass.*1e3.*ones(1, max(size(aat(idx(2):end)))), 'k-', 'linewidth', 2, 'handlevisibility', 'off')
plot(aeff(1:idxff), mwff(1:idxff).*1e3, 'b-.', 'linewidth', 2, 'handlevisibility', 'off')
plot(aeff(idxff:end), mwff(idxff:end).*1e3, 'r-.', 'linewidth', 2, 'handlevisibility', 'off')
plot(aese(1:idxse), mwse(1:idxse).*1e3, 'm-', 'linewidth', 2)
plot(aese(idxse:end), mwse(idxse:end).*1e3, 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$\bar{M}$, $\frac{g}{mol}$')
legend({'Subsonic-SE', 'Supersonic-SE'}, 'fontsize', 16, 'location', 'east')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', 'right')

subplot(3,2,4)
hold on
plot(aat(1:idx(1)), pp0(1:idx(1)), 'g-', 'linewidth', 2)
plot(aat(idx(2):end), pp0(idx(2):end), 'k-', 'linewidth', 2)
plot(aeff(1:idxff), pff(1:idxff)./p0, 'b-', 'linewidth', 2)
plot(aeff(idxff:end), pff(idxff:end)./p0, 'r-', 'linewidth', 2)
plot(aese(1:idxse), pse(1:idxse)./p0, 'm-', 'linewidth', 2)
plot(aese(idxse:end), pse(idxse:end)./p0, 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$P/P_0$')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', 'right')

subplot(3,2,5)
hold on
plot(aat(1:idx(1)), rhorho0(1:idx(1)), 'g-', 'linewidth', 2)
plot(aat(idx(2):end), rhorho0(idx(2):end), 'k-', 'linewidth', 2)
plot(aeff(2:idxff), rhoff(1:idxff-1)./rhoff(1), 'b-', 'linewidth', 2)
plot(aeff(idxff:end), rhoff(idxff-1:end)./rhoff(1), 'r-', 'linewidth', 2)
plot(aese(5:idxse), rhose(4:idxse-1)./max(rhose), 'm-', 'linewidth', 2)
plot(aese(idxse:end), rhose(idxse-1:end)./max(rhose), 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$\rho/\rho_0$')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', 'right')

subplot(3,2,6)
hold on
plot(aat(1:idx(1)), tt0(1:idx(1)), 'g-', 'linewidth', 2)
plot(aat(idx(2):end), tt0(idx(2):end), 'k-', 'linewidth', 2)
plot(aeff(1:idxff), tff(1:idxff)./t0, 'b-', 'linewidth', 2)
plot(aeff(idxff:end), tff(idxff:end)./t0, 'r-', 'linewidth', 2)
```

```matlab
plot(aese(1:idxse), tse(1:idxse)./t0, 'm-', 'linewidth', 2)
plot(aese(idxse:end), tse(idxse:end)./t0, 'cy-', 'linewidth', 2)
xlabel('$A/A_t$')
ylabel('$T/T_0$')
ylab = get(gca, 'ylabel');
set(ylab, 'rotation', 0, 'verticalalignment', 'middle', 'horizontalalignment', 'right')
set(gcf, 'Color', 'w', 'Position', [0 0 800 800]);
% export_fig('j2_key_quantities.eps')

%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%   Isentropic Flow Relations
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function an = mdotaaratio(gam, r, p0, t0)
    an = p0 .* sqrt(gam./(r.*t0)).*(1./(1/2.*(gam+1))).^((gam+1)./(2.*(gam-1)));
end
function an = stagtratio(mach, gam)
    an = (1 + 1/2.*(gam-1).*mach.^2);
end
function an = stagpratio(mach, gam)
    an = (1 + 1/2.*(gam-1).*mach.^2).^(gam./(gam-1));
end
function an = stagrhoratio(mach, gam)
   an = (1 + 1/2.*(gam-1).*mach.^2).^(1./(gam-1));
end
function an = aastarratio(mach, gam)
   an = ((gam + 1)./2).^(-((gam+1)/(2*(gam-1))))*((1 + 1/2*(gam-1).*mach.^2).^((gam+1)/(2*(gam-1))))./mach;
end
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%   Frozen Flow, Shifting Equilibrim Equations
%   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [mach, gam, molmass, rho, t, molsi] = calcTemp(p, t, names, sstag, hstag, pstag, molsi, solver)
    err = 1; cplin = zeros(1, length(molsi)); hlin = cplin; stplin = cplin;
    masses = [32, 2, 1, 17, 16, 18].*1e-3;

    while abs(err) > 1e-5

        snew = 0; hnew = 0;
        for i = 1:length(molsi)
            [stplin(i), hlin(i), cplin(i)] = gassProp(t, names(i));

            snew = snew + molsi(i)*(stplin(i) - 8.314*log(molsi(i)/sum(molsi)) - 8.314*log(p/101.325e3));
            hnew = hnew + molsi(i)*hlin(i);
        end
        h0 = hnew/sum(molsi.*masses);

        err = sstag - snew;
        if sstag > snew
            t = t*(1 + abs((sstag - snew)/sstag));
        else
            t = t*(1 - abs((sstag - snew)/sstag));
        end
    end

    if strcmp(solver, 'se') % se - Shifting Equilibrium
        molsi = minGibbs(t, p, hlin, stplin, molsi);
    end
    cp = sum(molsi.*cplin)/sum(molsi);

    gam = cp/(cp - 8.314);
    molmass = sum(molsi.*masses)/sum(molsi);
    mach = sqrt(2*(hstag - h0))/sqrt(gam*8.314/molmass * t);
    rho = (pstag - p)/ (mach * sqrt(gam*8.314/molmass * t))^2;
end
function molsi = minGibbs(t, p, h, s, molsi)
    syms lam_o lam_h no2 nh2 nh noh no nh2o
    hfor_h2 = 0; hfor_o2 = 0;
    hfor_h2o = -241.8e3; hfor_o = 249e3;
    hfor_h = 218e3; hfor_oh = 37.49e3;
```

```
    nj = no2 + nh2 + nh + noh + no + nh2o;

    nf = 1;
    noxi = 5.5 * (2/32);
    assume(no2,'Real'); assumeAlso(no2 >= 0);
    assume(nh2,'Real'); assumeAlso(nh2 >= 0);
    assume(nh,'Real'); assumeAlso(nh >= 0);
    assume(noh,'Real'); assumeAlso(noh >= 0);
    assume(no,'Real'); assumeAlso(no >= 0);
    assume(nh2o,'Real'); assumeAlso(nh2o >= 0);

    eqn1 = 0 == h(1) + hfor_o2 - t*s(1) + t*8.314*log(p/101.325e3)+ t*8.314*log(no2/nj) + 2*lam_o ; %O2
    eqn2 = 0 == h(2) + hfor_h2 - t*s(2) + t*8.314*log(p/101.325e3)+ t*8.314*log(nh2/nj) + 2*lam_h; %H2
    eqn3 = 0 == h(3) + hfor_h - t*s(3) + t*8.314*log(p/101.325e3)+ t*8.314*log(nh/nj) + lam_h; %H
    eqn4 = 0 == h(4) + hfor_oh - t*s(4) + t*8.314*log(p/101.325e3)+ t*8.314*log(noh/nj) + lam_o + lam_h ; %OH
    eqn5 = 0 == h(5) + hfor_o - t*s(5) + t*8.314*log(p/101.325e3)+ t*8.314*log(no/nj) + lam_o; %O
    eqn6 = 0 == h(6) +hfor_h2o - t*s(6) + t*8.314*log(p/101.325e3)+ t*8.314*log(nh2o/nj) + 2*lam_h + lam_o; %H2O
    eqn7 = 0 == -2*noxi + 2*no2 + noh + no + nh2o;
    eqn8 = 0 == -2*nf +2*nh2 + nh + noh + 2*nh2o;
    sol = vpasolve([eqn1,eqn2,eqn3,eqn4,eqn5,eqn6,eqn7,eqn8],[lam_o,lam_h,no2,nh2,nh,noh,no,nh2o],[10e4, 10e4, molsi]);


    molsi = [sol.no2, sol.nh2, sol.nh, sol.noh, sol.no, sol.nh2o];
end
function [stp, htp, cp] = gassProp(T0, name)
    R0 = 500; R1 = 700;
    R2 = 1000; R3 = 1300;
    R4 = 1700; R5 = 2000;
    R6 = 2500; R7 = 6000;

    t = T0/1000;

    if strcmp(name, 'h2')
        if T0 >= R0 && T0 <= R2
            ah2 = 33.066178; bh2 = -11.363417;
            ch2 = 11.432816; dh2 = -2.772874;
            eh2 = -0.158558; fh2 = -9.980797;
            gh2 = 172.707974; hh2 = 0;
        end
        if T0 > R2 && T0 <= R6
            ah2 = 18.563083; bh2 = 12.257357;
            ch2 = -2.859786; dh2 = 0.268238;
            eh2 = 1.97799; fh2 = -1.147438;
            gh2 = 156.288133;hh2 = 0;
        end
        if T0 > R6 && T0 <= R7
            ah2 = 43.41356; bh2 = -4.293079;
            ch2 = 1.272428; dh2 = -0.096876;
            eh2 = -20.533862; fh2 = -38.515158;
            gh2 = 162.081354;hh2 = 0;
        end
        [cp, htp, stp] = basedOnCoeff(t,ah2,bh2,ch2,dh2,eh2,fh2,gh2,hh2);
    end
    if strcmp(name, 'o2')
        if T0 >= R0 && T0 <= R1
            ao2 =31.32234; bo2 =-20.23531;
            co2 =57.86644; do2 =-36.50624;
            eo2 =-0.007374; fo2 =-8.903471;
            go2 =246.7945; ho2 = 0;
        end
        if T0 > R1 && T0 <= R5
            ao2 = 30.03235; bo2 = 8.772972;
            co2 = -3.988133; do2 = 0.788313;
            eo2 = -0.741599; fo2 = -11.32468;
            go2 = 236.1663; ho2 =0;
```

```
        end
        if T0 > R5 && T0 <= R7
            ao2 = 20.91111; bo2 = 10.72071;
            co2 = -2.020498; do2 = 0.146449;
            eo2 = 9.245722; fo2 = 5.337651;
            go2 = 237.6185; ho2 = 0;
        end
        [cp, htp, stp] = basedOnCoeff(t,ao2,bo2,co2,do2,eo2,fo2,go2,ho2);
    end
    if strcmp(name, 'h2o')
        if T0 >= R0 && T0 <= R4
            ah2o = 30.09200; bh2o = 6.832514;
            ch2o = 6.793435; dh2o = -2.53448;
            eh2o = 0.082139; fh2o = -250.881;
            gh2o = 223.3967; hh2o = -241.8264;
        end
        if T0 > R4 && T0 <= R7
            ah2o = 41.96426; bh2o = 8.622053;
            ch2o = -1.49978; dh2o = 0.098119;
            eh2o = -11.15764; fh2o = -272.1797;
            gh2o = 219.7809; hh2o = -241.8264;
        end
        [cp, htp, stp] = basedOnCoeff(t,ah2o,bh2o,ch2o,dh2o,eh2o,fh2o,gh2o,hh2o);
    end
    if strcmp(name, 'h')
        ah = 20.78603;
        bh = 4.850638*10.^-10;
        ch = -1.5825102*10.^-10;
        dh = 1.525102*10.^-11;
        eh = 3.196347*10.^-11;
        fh = 211.802;
        gh = 139.8711;
        hh = 217.9994;
        [cp, htp, stp] = basedOnCoeff(t,ah,bh,ch,dh,eh,fh,gh,hh);
    end
    if strcmp(name, 'o')
        ao = 20.78603;
        bo = 4.850638*10.^-10;
        co = -1.5825102*10.^-10;
        do = 1.525102*10.^-11;
        eo = 3.196347*10.^-11;
        fo = 211.802;
        go = 186.2131;
        ho = 217.9994;
        [cp, htp, stp] = basedOnCoeff(t,ao,bo,co,do,eo,fo,go,ho);
    end
    if strcmp(name, 'oh')
        if T0 >= R0 && T0 <= R3
            aoh = 32.27768; boh = -11.36291;
            coh = 13.60545; doh = -3.846486;
            eoh = -0.001335; foh    = 29.75113;
            goh = 225.5783; hoh = 38.98706;
        end
        if T0 > R3 && T0 <= R7
            aoh = 28.74701; boh = 4.714489;
            coh = -0.814725; doh    = 0.054748;
            eoh = -2.747829; foh    = 26.41439;
            goh = 214.1166; hoh = 38.98706;
        end
        [cp, htp, stp] = basedOnCoeff(t,aoh,boh,coh,doh,eoh,foh,goh,hoh);
    end
end
function [Cp, Hdel, S] = basedOnCoeff(t,a,b,c,d,e,f,g,h)
    Cp = a + b*t + c*t.^2 + d*t.^3 + e./t.^2;
    Hdel = a*t + (1./2)*b*t.^2 + (1./3)*c*t.^3 + (1./4)*d*t.^4 -e./t + f - h;
    S = a*log(t) + b*t + (1./2)*c*t.^2 + (1./3)*d*t.^3 - e./(2*t.^2) + g;
    Hdel = Hdel*10.^3;
end
```

Listing 3 Matlab code to determine Merlin-1D engine performance.

```
close all
clear all
clc

OF = [2:0.25:5,5.5:.5:10]; %range of O/F values

% data from CEA for frozen flow
thrustcorrection = .95;
dischargecorrection = 1.1;
ispcorrection = thrustcorrection/dischargecorrection;
T0f = [3359.54 3582.19 3694.88 3741.39 3753.08 3746.29 3729.10 3705.70 3678.39
    3648.51 3616.88 3584.01 3550.23 3480.80 3409.66 3337.35 3264.18 3190.37 3116.14
    3041.71 2967.32 2893.28 2819.87];
IspVacf = ispcorrection.*[3345.2 3393.5 3392.5 3366.8 3330.9 3291.7 3251.9 3212.9
    3175.1 3138.8 3103.9 3070.3 3038.0 2976.8 2919.4 2865.2 2813.4 2763.7 2715.6
    2668.8 2623.3 2578.8 2535.3]/9.81;
cff = [1.8484 1.8621 1.8704 1.8754 1.8785 1.8806 1.8819 1.8828 1.8833 1.8836 1.8836
    1.8835 1.8832 1.8822 1.8809 1.8793 1.8774 1.8753 1.8730 1.8706 1.8680 1.8654
    1.8627];
P0 = 9.7e6;
At = .042;
Tidealf = cff.*P0.*At;
Tf = Tidealf.*thrustcorrection./1000;

%data from CEA for equilibrium flow
T0e = [3352.51,3575.48,3688.37,3734.97,3746.74,3740.02,3722.90,3699.57,3672.33,
    3642.52,3610.95,3578.15,3544.43, 3475.10, 3404.03, 3331.78, 3258.65, 3184.87,
    3110.64, 3036.19, 2961.79, 2887.71, 2814.28];
IspVace = ispcorrection.*[3461.7 3573.1 3647.1 3687.3 3694.2 3667.2 3622.3 3573.5
    3523.8 3474.0 3424.3 3375.1 3326.5 3232.3 3142.7 3058.1 2978.4 2903.4 2832.9
    2766.6 2704.0 2645.1 2589.4]./(9.81);
cfe = [1.8751,1.9113,1.9511,1.9845,2.0032,2.0066,2.0075,2.0073,2.0061,2.0042,2.0012,
     1.9972,1.9922,1.9803,1.9667,1.9527,1.9390,1.9260,
    1.9140,1.9032,1.8936,1.8852,1.8778];
P0 = 9.7e6; %combustor pressure
At = .042; %throat area
thrustcorrection = .95;
dischargecorrection = 1.1;
Tideale = cfe.*P0.*At; %Ideal Thrust
Te = Tideale.*thrustcorrection./1000; %Actual thrust in kN

%data from CEA for ideal flow
gammai = [1.3339 1.3115 1.2966 1.2872 1.2812 1.2771 1.2743 1.2724 1.2711 1.2701
    1.2700 1.2699 1.2701 1.2712 1.2730 1.2754 1.2783 1.2816 1.2854 1.2895 1.2939
    1.2986 1.3034];
MWi = [20.874 22.167 23.225 24.098 24.841 24.841 26.061 26.575 27.038 27.458 27.840
    28.188 28.507 29.064 29.532 29.925 30.255 30.531 30.760 30.950 31.107 31.236
    31.341];
Rbar = 8.314e3;
Ae = At.*117;
Mei = ones(1,size(gammai,2)); %solve for Me, Pe, mdot, and thrust for different values of gamma (
    and by extension O-F)
T0i = T0f; %same adiabatic flame temperature as frozen flow
for i = 1:size(gammai,2)
    syms Me
    gamma = gammai(i);
    eqn1 = Ae/At == (1./Me).*(((2./(gamma+1)).*(1 + (gamma-1).*(1/2).*Me^2)).^((gamma
        +1)./(2.*gamma - 2)));
    Mei(i) = vpasolve(eqn1,Me);
end
temp = ((gammai+1)./2).^((gammai+1)./(2.*(gammai-1)));
mdoti = ((sqrt(gammai))./(sqrt(Rbar.*T0i./MWi))).*P0.*At./temp;
Pei = P0.*((1 + (gammai -1).*(Mei.^2)./2).^(-gammai./(gammai -1)));
temp2 = (1 - (Pei./P0).^((gammai-1)./gammai));
Ti_1 = ((Pei.*Ae) + mdoti.*sqrt(2.*gammai.*Rbar.*T0i.*temp2./((gammai -1).*MWi)));
Ispi_1 = (1/9.81).*(Ti_1./mdoti);
Ti = thrustcorrection.*Ti_1;
Ispi = ispcorrection.*Ispi_1;


```

```
%plots with respect to oxi-fuel ratio
figure(1)
subplot(1,3,1);
plot(OF,Te,'LineWidth',2)
hold on
plot(OF,Tf,'LineWidth',2)
plot(OF,Ti./1000,'LineWidth',2)
xlabel('Oxidizer-Fuel Ratio','FontSize',20)
ylabel('Thrust (kN)','FontSize',20)
title('Thrust vs. O/F Ratio','FontSize',20)
legend('Shifting Equilibrium','Frozen Flow','Ideal')

subplot(1,3,2);
plot(OF,IspVace,'LineWidth',2)
hold on
plot(OF,IspVacf,'LineWidth',2)
plot(OF,Ispi,'LineWidth',2)
xlabel('Oxidizer-Fuel Ratio','FontSize',20)
ylabel('Isp (s)','FontSize',20)
title('Isp vs. O/F Ratio','FontSize',20)
legend('Shifting Equilibrium','Frozen Flow','Ideal')

subplot(1,3,3);
plot(OF,T0e,'LineWidth',2)
hold on
plot(OF,T0f,'LineWidth',2)
plot(OF,T0i,'LineWidth',2)
xlabel('Oxidizer-Fuel Ratio','FontSize',20)
ylabel('T_0 (K)','FontSize',20)
title('Adiabatic Flame Temperature vs. O/F Ratio','FontSize',20)
legend('Shifting Equilibrium','Frozen Flow','Ideal')

[Ispmaxe, temp1e] = max(IspVace);
[Ispmaxf, temp1f] = max(IspVacf);
T0maxe = T0e(temp1e);
T0maxf = T0f(temp1f);
rho0e = 7.7216;
rho0f = 7.2191;

%values for subsonic region from CEA eq
aratiosube = [1:.25:4];
gammasube = [1.1284 1.1314 1.1320 1.1323 1.1325 1.1326 1.1327 1.1328 1.1328 1.1328 1.1329 1.1329 1.1329];
machsube = [0.997 0.565 0.441 0.366 0.314 0.276 0.246 0.223 0.203 0.187 0.173 0.161 0.151];
%pratiosube = [1.7429 1.2106 1.1296 1.0920 1.0707 1.0572 1.0480 1.0414 1.0366 1.0329 1.0300 1.0277 1.0258];
pratiosube = [55.653 80.126 85.873 88.823 90.593 91.752 92.556 93.139 93.575 93.911 94.176 94.387 94.560].*100000./P0;
tratiosube = [3574.18 3691.33 3714.33 3725.64 3732.27 3736.55 3739.50 3741.62 3743.20 3744.41 3745.36 3746.12 3746.74]./T0maxe;
Mweightsube = [25.197 24.955 24.909 24.887 24.873 24.865 24.859 24.855 24.852 24.849 24.848 24.846 24.845];
rhoratiosube = [4.7187 6.5150 6.9262 7.1360 7.2614 7.3434 7.4002 7.4413 7.4721 7.4957 7.5144 7.5293 7.5414]./rho0e;

%values for supersonic region from CEA eq
aratioe = [1:1:9,10:10:120];
gammae = [1.1284 1.1184 1.1157 1.1143 1.1136 1.1132 1.1131 1.1132 1.1134 1.1137 1.1201 1.1296 1.1401 1.1500 1.1586 1.1654 1.1707 1.1748 1.1780 1.1805 1.1826];
mache = [1.000 2.006 2.313 2.507 2.648 2.759 2.850 2.928 2.995 3.054 3.425 3.632 3.776 3.888 3.983 4.065 4.139 4.207 4.269 4.327 4.381];
pratioe = [55.450 12.812 7.1949 4.8835 3.6446 2.8808 2.3667 1.9990 1.7239 1.5112 0.64245 0.39133 0.27520 0.20921 0.16706 0.13802 0.11693 0.10099 0.08857 0.07864 0.07056].*100000./P0;
tratioe = [3573.04 3157.76 3013.36 2920.91 2853.08 2799.59 2755.44 2717.85 2685.10 2656.06 2466.54 2351.20 2263.73 2191.47 2129.50 2075.39 2027.61 1985.06 1946.88 1912.39 1881.02]./T0maxe;
Mweighte = [25.200 26.149 26.507 26.741 26.913 27.049 27.160 27.253 27.333 27.403 27.817 28.011 28.119 28.183 28.222 28.247 28.263 28.274 28.281 28.287 28.291];
rhoratioe = [4.7035 1.2760 7.6120e-1 5.3772e-1 4.1349e-1 3.3476e-1 2.8057e-1 2.4108e-1 2.1107e-1 1.8752e-1 8.7141e-2 5.6072e-2 4.1113e-2 3.2358e-2 2.6628e-2 2.2594e
```

```
    -2 1.9603e-2 1.7300e-2 1.5474e-2 1.3991e-2 1.2763e-2]./rho0e;

%values for subsonic region from CEA frozen
aratiosubf = [1:.25:4];
gammasubf = [1.2071 1.2054 1.2051 1.2049 1.2048 1.2048 1.2047 1.2047 1.2047 1.2047 1.2047 1.2047 1.2046];
machsubf = [0.996 0.562 0.438 0.364 0.312 0.274 0.245 0.221 0.202 0.186 0.172 0.160 0.150];
pratiosubf = [54.825 80.436 86.500 89.620 91.494 92.721 93.573 94.191 94.654 95.010 95.290 95.514 95.697].*100000./P0;
tratiosubf = [3404.83 3635.42 3680.70 3702.96 3716.01 3724.43 3730.23 3734.40 3737.51 3739.90 3741.77 3743.27 3744.48]./T0maxf;
Mweightsubf = [24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841];
rhoratiosubf = [4.8107 6.6103 7.0213 7.2308 7.3561 7.4379 7.4945 7.5356 7.5663 7.5899 7.6085 7.6234 7.6355]./rho0f;

%values for supersonic region from CEA frozen
aratiof = [1:1:9,10:10:120]; %range of A/At
gammaf = [1.2071 1.2153 1.2194 1.2226 1.2252 1.2275 1.2295 1.2313 1.2329 1.2344 1.2453 1.2525 1.2581 1.2626 1.2664 1.2697 1.2726 1.2752 1.2776 1.2798 1.2817];
machf = [1.000 2.060 2.407 2.634 2.805 2.942 3.057 3.156 3.243 3.321 3.838 4.150 4.378 4.559 4.711 4.841 4.957 5.060 5.154 5.241 5.320];
pratiof = [54.615 11.540 6.2194 4.0987 2.9886 2.3173 1.8726 1.5590 1.3274 1.1502 0.45268 0.26372 0.17996 0.13384 0.10510 0.08566 0.07176 0.06138 0.05337 0.04702 0.04189].*100000./P0;
tratiof = [3402.59 2595.43 2324.26 2155.28 2034.27 1940.95 1865.55 1802.61 1748.83 1702.01 1421.26 1276.20 1180.77 1110.73 1055.95 1011.29 973.79 941.61 913.52 888.67 866.44]./T0maxf;
Mweightf = [24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841 24.841];
rhoratiof = [4.7955 1.3284 7.9945e-1 5.6816e-1 4.3893e-1 3.5669e-1 2.9990e-1 2.5839e-1 2.2677e-1 2.0190e-1 9.5160e-2 6.1739e-2 4.5535e-2 3.6001e-2 2.9735e-2 2.5308e-2 2.2016e-2 1.9474e-2 1.7453e-2 1.5809e-2 1.4445e-2]./rho0f;

%values for superosnic ideal flow
aratioi = [1:1:9,10:10:120];
aratiosubi = [1:.25:4];
Mweighti = 24.841*ones(1,size(aratioi,2)); %molecular weight for OF of 3
Mweightsubi = 24.841*ones(1,size(aratiosubi,2));

gammainum = gammai(5).*ones(1,size(aratioi,2));
gammasubinum = gammai(5).*ones(1,size(aratiosubi,2));

%% Solve for Msup using area ratio
ARatio = [2:1:9,10:10:120];
Machnumbersup = ones(1,size(ARatio,2));
% Define some paramters
g   = gammai(5);
gm1 = g-1;
gp1 = g+1;
% Define anonymous function with two inputs (M and ARatio)
% - Will be used in the methods below
% - Pass M and ARatio as arguments to AM_EQN to get function value
% -     funVal = AM_EQN(M,ARatio)
AM_EQN = @(M,ARatio) sqrt((1/M^2)*(((2+gm1*M^2)/gp1)^...
    (gp1/gm1)))-ARatio;
for k = 1:numel(ARatio)
    % Error tolerance
    errTol = 1e-4;
    % Flags for printing iterations to screen
    verboseBisection = 0;
    verboseIncremental = 0;
    %% SUBSONIC INCREMENTAL SEARCH
    % Initial values
    dM      = 0.1;                                                    % Initial M step size
    M       = 1e-6;                                                   % Initial M value
    iConvSub = 0;                                                     % Initial converge
```

```
index
if (verboseIncremental == 1)
    fprintf('Incremental Search Method: Subsonic\n');
    fprintf('-----------------------------------\n');
end
% Iterate to solve for root
iterMax = 100;                                                          % Maximum iterations
stepMax = 100;                                                          % Maximum step iterations
for i = 1:1:iterMax
    for j = 1:1:stepMax

        % Evaluate function at j and j+1
        fj   = AM_EQN(M,ARatio(k));
        fjp1 = AM_EQN(M+dM,ARatio(k));

        % Print iterations to command window
        if (verboseIncremental == 1)
            fprintf('fj | fjp1: %3.4f\t%3.4f\n',fj,fjp1);
        end

        % Update M depending on sign change or not
        % - If no sign change, then we are not bounding root yet
        % - If sign change, then we are bounding the root, update dM
        if (fj*fjp1 > 0)
            M = M + dM;                                                 % Update M
        elseif (fj*fjp1 < 0)
            dM = dM*0.1;                                                % Refine the M increment
            break;                                                      % Break out of j loop
        end

    end % END: j Loop

    % Check for convergence
    if (abs(fj-fjp1) <= errTol)                                         % If converged
        iConvSub = i;                                                   % Set converged index
        break;                                                          % Exit loop
    end

end % END: i Loop
% Set subsonic Mach number to final M from iterations
Msub = M;
%% SUPERSONIC INCREMENTAL SEARCH
% Initial values
dM      = 1;                                                            % Initial M step size
M       = 1+1e-6;                                                       % Initial M value
iConvSup = 0;                                                           % Initial converge index
if (verboseIncremental == 1)
    fprintf('\nIncremental Search Method: Supersonic\n');
    fprintf('-------------------------------------\n');
end
% Iterate to solve for root
iterMax = 100;                                                          % Maximum iterations
stepMax = 100;                                                          % Maximum step iterations
for i = 1:1:iterMax
    for j = 1:1:stepMax

        % Evaluate function at j and j+1
        fj   = AM_EQN(M,ARatio(k));
        fjp1 = AM_EQN(M+dM,ARatio(k));

        % Print iterations to command window
        if (verboseIncremental == 1)
            fprintf('fj | fjp1: %3.4f\t%3.4f\n',fj,fjp1);
```

```
            end

            % Update M depending on sign change or not
            % - If no sign change, then we are not bounding root yet
            % - If sign change, then we are bounding the root, update dM
            if (fj*fjp1 > 0)
                M = M + dM;                                         % Update M
            elseif (fj*fjp1 < 0)
                dM = dM*0.1;                                        % Refine the M increment
                break;                                              % Break out of j loop
            end

        end % END: j Loop

        % Check for convergence
        if (abs(fj-fjp1) <= errTol)                                 % If converged
            iConvSup = i;                                           % Set converged index
            break;                                                  % Exit loop
        end

    end % END: i Loop
    % Set supersonic Mach number to final M from iterations
    Msup = M;
    % Print solutions to command window
    Machnumbersup(k) = Msup;
end
%%
% Solve for Msub and Msup using this area
ARatio = [1.25:.25:4];
Machnumbersub = ones(1,size(ARatio,2));
% Define some paramters
g   = gammai(5);
gm1 = g-1;
gp1 = g+1;
% Define anonymous function with two inputs (M and ARatio)
% - Will be used in the methods below
% - Pass M and ARatio as arguments to AM_EQN to get function value
% -     funVal = AM_EQN(M,ARatio)
AM_EQN = @(M,ARatio) sqrt((1/M^2)*(((2+gm1*M^2)/gp1)^...
    (gp1/gm1)))-ARatio;
for k = 1:numel(ARatio)
    % Error tolerance
    errTol = 1e-4;
    % Flags for printing iterations to screen
    verboseBisection = 0;
    verboseIncremental = 0;
    %% SUBSONIC INCREMENTAL SEARCH
    % Initial values
    dM      = 0.1;                                                  % Initial M step size
    M       = 1e-6;                                                 % Initial M value
    iConvSub = 0;                                                   % Initial converge index
    if (verboseIncremental == 1)
        fprintf('Incremental Search Method: Subsonic\n');
        fprintf('-----------------------------------\n');
    end
    % Iterate to solve for root
    iterMax = 100;                                                  % Maximum iterations
    stepMax = 100;                                                  % Maximum step iterations
    for i = 1:1:iterMax
        for j = 1:1:stepMax

            % Evaluate function at j and j+1
            fj   = AM_EQN(M,ARatio(k));
            fjp1 = AM_EQN(M+dM,ARatio(k));

```

```
            % Print iterations to command window
            if (verboseIncremental == 1)
                fprintf('fj | fjp1: %3.4f\t%3.4f\n',fj,fjp1);
            end

            % Update M depending on sign change or not
            % - If no sign change, then we are not bounding root yet
            % - If sign change, then we are bounding the root, update dM
            if (fj*fjp1 > 0)
                M = M + dM;                                                 % Update M
            elseif (fj*fjp1 < 0)
                dM = dM*0.1;                                                % Refine the M increment
                break;                                                      % Break out of j loop
            end

        end % END: j Loop

        % Check for convergence
        if (abs(fj-fjp1) <= errTol)                                         % If converged
            iConvSub = i;                                                   % Set converged index
            break;                                                          % Exit loop
        end

    end % END: i Loop
    % Set subsonic Mach number to final M from iterations
    Msub = M;
    Machnumbersub(k) = Msub;
    %% SUPERSONIC INCREMENTAL SEARCH
    % Initial values
    dM      = 1;                                                            % Initial M step size
    M       = 1+1e-6;                                                       % Initial M value
    iConvSup = 0;                                                           % Initial converge index
    if (verboseIncremental == 1)
        fprintf('\nIncremental Search Method: Supersonic\n');
        fprintf('-------------------------------------\n');
    end
    % Iterate to solve for root
    iterMax = 100;                                                          % Maximum iterations
    stepMax = 100;                                                          % Maximum step iterations
    for i = 1:1:iterMax
        for j = 1:1:stepMax

            % Evaluate function at j and j+1
            fj  = AM_EQN(M,ARatio(k));
            fjp1 = AM_EQN(M+dM,ARatio(k));

            % Print iterations to command window
            if (verboseIncremental == 1)
                fprintf('fj | fjp1: %3.4f\t%3.4f\n',fj,fjp1);
            end

            % Update M depending on sign change or not
            % - If no sign change, then we are not bounding root yet
            % - If sign change, then we are bounding the root, update dM
            if (fj*fjp1 > 0)
                M = M + dM;                                                 % Update M
            elseif (fj*fjp1 < 0)
                dM = dM*0.1;                                                % Refine the M increment
                break;                                                      % Break out of j loop
            end

        end % END: j Loop

```

```
        % Check for convergence
        if (abs(fj-fjp1) <= errTol)                                    % If converged
            iConvSup = i;                                              % Set converged index
            break;                                                     % Exit loop
        end

    end % END: i Loop
    % Set supersonic Mach number to final M from iterations
    Msup = M;
    % Print solutions to command window
end
%%
machi = [1,Machnumbersup]; %do calculations for ideal flow using isentropic relations
machsubi = [1,Machnumbersub];
pratioi = ((1 + (gammai(5) -1).*(machi.^2)./2).^(-gammai(5)./(gammai(5) -1)));
pratiosubi = ((1 + (gammai(5) -1).*(machsubi.^2)./2).^(-gammai(5)./(gammai(5) -1)));
rhoratioi = (1 + (machi.^2)*(gammai(5)-1)./2).^(-1/(gammai(5)-1));
rhoratiosubi = (1 + (machsubi.^2)*(gammai(5)-1)./2).^(-1/(gammai(5)-1));
tratioi = (1 + (machi.^2).*(gammai(5) - 1)./2).^-1;
tratiosubi = (1 + (machsubi.^2).*(gammai(5) - 1)./2).^-1;
%%

%plots with respect to area ratio for eq flow
figure(2)
subplot(3,2,1)
semilogx(aratioe,mache,'+-','LineWidth',1)
hold on
semilogx(aratiosube,machsube,'+-','LineWidth',1)
semilogx(aratiof,machf,'o-','LineWidth',1)
semilogx(aratiosubf,machsubf,'o-','LineWidth',1)
semilogx(aratioi,machi,'.-','LineWidth',1)
semilogx(aratiosubi,machsubi,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('M')
title('Mach Number vs. Area Ratio')
% legend(' Supersonic Equilibrium Flow', ' Subsonic Equilibrium Flow',' Supersonic Frozen Flow', ' Subsonic Frozen Flow',' Location',' Southeast')

subplot(3,2,3)
semilogx(aratioe,Mweighte,'+-','LineWidth',1)
hold on
semilogx(aratiosube,Mweightsube,'+-','LineWidth',1)
semilogx(aratiof,Mweightf,'o-','LineWidth',1)
semilogx(aratiosubf,Mweightsubf,'o-','LineWidth',1)
semilogx(aratioi,Mweighti,'.-','LineWidth',1)
semilogx(aratiosubi,Mweightsubi,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('$\bar{M}$ (g/mol)','Interpreter','Latex')
% ylim([21 29]);
title('Average Molecular Weight vs. Area Ratio')

subplot(3,2,5)
semilogx(aratioe,rhoratioe,'+-','LineWidth',1)
hold on
semilogx(aratiosube,rhoratiosube,'+-','LineWidth',1)
semilogx(aratiof,rhoratiof,'o-','Linewidth',1)
semilogx(aratiosubf,rhoratiosubf,'o-','LineWidth',1)
semilogx(aratioi,rhoratioi,'.-','Linewidth',1)
semilogx(aratiosubi,rhoratiosubi,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('\rho/\rho_0')
title('Density Ratio vs. Area Ratio')
%legend(' Supersonic Equilibrium Flow', ' Subsonic Equilibrium Flow',' Supersonic Frozen Flow', ' Subsonic Frozen Flow',' Supersonic Ideal Flow', ' Subsonic Ideal Flow',' Location',' Southeast')

subplot(3,2,2)
semilogx(aratioe,gammae,'+-','LineWidth',1)
hold on
semilogx(aratiosube,gammasube,'+-','LineWidth',1)
semilogx(aratiof,gammaf,'o-','LineWidth',1)
semilogx(aratiosubf,gammasubf,'o-','LineWidth',1)
```

```
semilogx(aratioi,gammainum,'.-','LineWidth',1)
semilogx(aratiosubi,gammasubinum,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('\gamma')
title('Ratio of Specific Heats vs. Area Ratio')
%legend(' Supersonic  Equilibrium  Flow', ' Subsonic  Equilibrium  Flow',' Supersonic  Frozen  Flow', '
      Subsonic  Frozen  Flow','Location','Southeast')

subplot(3,2,4)
semilogx(aratioe,pratioe,'+-','LineWidth',1)
hold on
semilogx(aratiosube,pratiosube,'+-','LineWidth',1)
semilogx(aratiof,pratiof,'o-','LineWidth',1)
semilogx(aratiosubf,pratiosubf,'o-','LineWidth',1)
semilogx(aratioi,pratioi,'.-','LineWidth',1)
semilogx(aratiosubi,pratiosubi,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('P/P_0')
title('Pressure Ratio vs. Area Ratio')
%legend(' Supersonic  Equilibrium  Flow', ' Subsonic  Equilibrium  Flow',' Supersonic  Frozen  Flow', '
      Subsonic  Frozen  Flow','Location','Southeast')

subplot(3,2,6)
semilogx(aratioe,tratioe,'+-','LineWidth',1)
hold on
semilogx(aratiosube,tratiosube,'+-','LineWidth',1)
semilogx(aratiof,tratiof,'o-','LineWidth',1)
semilogx(aratiosubf,tratiosubf,'o-','LineWidth',1)
semilogx(aratioi,tratioi,'.-','LineWidth',1)
semilogx(aratiosubi,tratiosubi,'k.-','LineWidth',1)
xlabel('A/A_t')
ylabel('T/T_0')
title('Exit Temperature Ratio vs. Area Ratio')
```