

MAE 563 Project: Propulsion System Analysis Tool

Anushka Subedi

ASU ID: 1225812200

Contents

1	Introduction	3
2	Part A: T-S diagrams	4
3	Part B: Standard and Isentropic Atmosphere	5
4	Part C: Study of M1 Variation	6
4.1	Overall efficiency	6
4.2	Thrust	7
4.3	TSFC	7
5	Part D: Study of altitude Variation	8
5.1	Overall efficiency	8
5.2	Thrust	8
5.3	TSFC	9
6	Part E: Study of M1 and altitude Variation	10
6.1	Optimized Overall efficiency	10
6.2	Optimized TSFC	10
7	Part F: Study of diffuser efficiency Variation	11
7.1	Overall efficiency	11
7.2	Thrust	11
7.3	TSFC	12

8 Part G: Study of nozzle efficiency Variation	13
8.1 Overall efficiency	13
8.2 Thrust	13
8.3 TSFC	14
9 Part H: Study of M2 Variation	14
9.1 Overall efficiency	14
9.2 Thrust	15
9.3 TSFC	15
10 Part I: Design of Ramjet/Scramjet	16
11 Conclusion	16
Appendix	17

1 Introduction

This project aims to analyze ramjet and scramjet engines using a MATLAB-based parametric tool. The tool models non-ideal flow conditions for engines with a fixed-geometry converging-only nozzle, enabling systematic variation of input parameters such as flight altitude, Mach number, diffuser and nozzle efficiencies, and combustor temperature limits. The propulsion system is as shown in 1 with all the states 1-2-3-e-4.

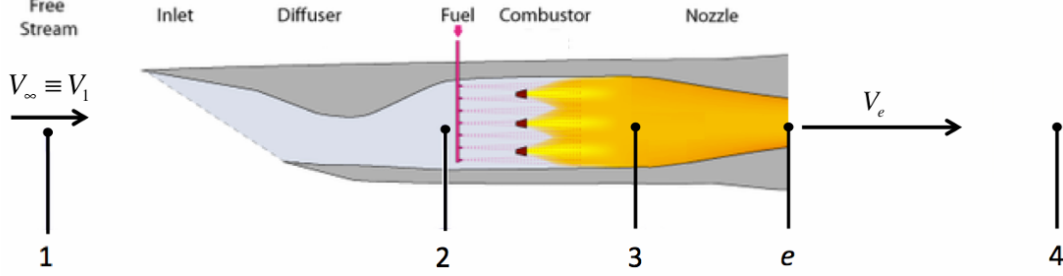


Figure 1: Propulsion System

The following inputs are used in the analysis tool:

Input Parameter	Symbol	Units
Flight altitude	z	meters
Flight Mach number	M_1	—
Inlet/diffuser efficiency	η_d	—
Mach number at diffuser exit	M_2	—
Maximum allowable total temperature at combustor exit	$(T_{t3})_{max}$	K
Fuel heating value	q_f	J/kg
Nozzle efficiency	η_n	—
Nozzle exit area	A_e	m ²

These inputs are used to calculate key outputs at each engine state, such as temperature, pressure, Mach number, and velocity. The outputs Calculated at each state 1-2-3-e-4:

Output Parameter	Symbol
Total and static pressure and temperature	P_t, T_t, P, T
Relative entropy change	$\Delta s = s_i - s_{i-1}$
Flow speed	V
Mach number	M

After getting the outputs at each step, they are used to find the thrust, power and efficiencies of the propulsion system. The outputs calculated at the end of the Propulsion System:

Output Parameter	Symbol	Units
Thrust	T	N
Propulsive power	P	W
Thermal efficiency	η_{th}	—
Propulsive efficiency	η_p	—
Overall efficiency	η_o	—
Exit mass flow rate	\dot{m}_e	kg/s
Fuel mass flow rate	\dot{m}_f	kg/s
Thrust specific fuel consumption	$TSFC$	(kg/hr)/N
Specific impulse	I_{sp}	s

The modules in the analysis tool:

Module Number	Description
Module 1	Isentropic Atmosphere
Module 2	Inlet/Diffuser
Module 3	Combustor
Module 4	Nozzle
Module 5	External Nozzle
Module 6	Outputs

Two validation cases: Non-thermally choked case and Thermally choked case were used to validate the developed modules before going into the analysis parts A-I.

2 Part A: T-S diagrams

The T-s diagrams for first validating case (non-thermally choked) and second validating (thermally choked) are presented below. These diagrams are crucial for evaluating the performance of Brayton Cycle engines, as the area enclosed by the T-s curve represents the net work produced by the cycle. In Figure2, corresponding to the non-thermally choked case, the significantly larger enclosed area highlights that thermal choking limits performance by capping the maximum achievable combustor temperature.

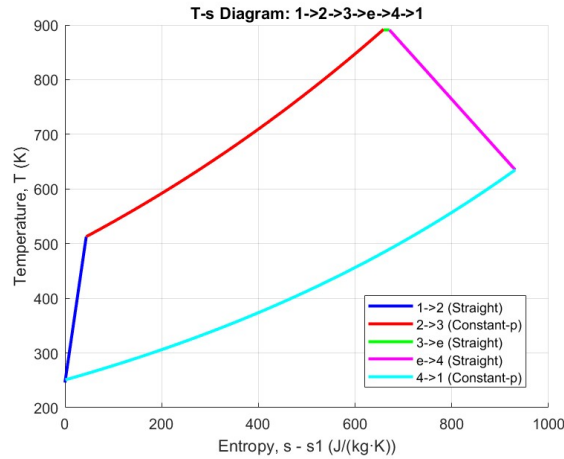


Figure 2: T-S diagram for thermally un-choked case

The gaps in the T-s diagrams arise due to the method used to calculate the entropy changes in the project. Specifically, the integrated form of the second law of thermodynamics was applied, but the specific heat capacity (C_p) was not properly accounted for as a function of temperature (T). Instead of using the mean value theorem (MVT) value of C_p , which adjusts for the variation of C_p with temperature, the project instructions used $C_p(T)=a+bT$ corresponding to a single temperature (T_3). This simplification fails to fully capture the actual entropy changes across each step. At this point of the project, the T-S diagrams are presented as they are. However, the correct approach which involves integrating $C_p(T)$ over the temperature range to calculate entropy changes and the MVT value of C_p can be used in future, which ensure that the temperature dependence is properly accounted for, eliminating discrepancies.

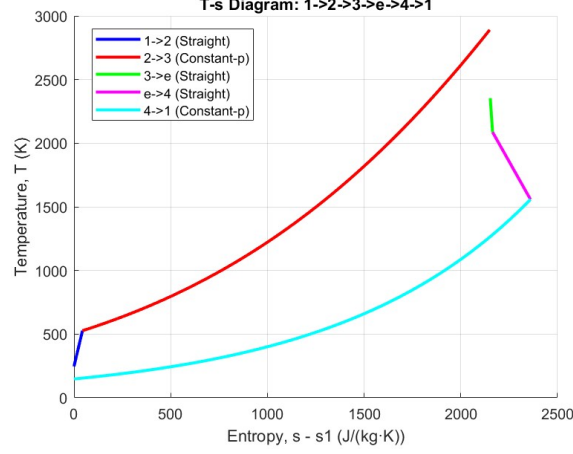


Figure 3: T-S diagram for thermally choked case

3 Part B: Standard and Isentropic Atmosphere

The flow path analysis starts by determining atmospheric conditions at the engine's operating flight envelope. This analysis uses the isentropic atmosphere model to calculate temperatures and pressures at various altitudes, as represented by Equations (1)-(4). A comparison with International Standard Atmosphere (ISA) data shows that pressure values are nearly identical, while the isentropic model slightly underestimates temperature at most altitudes. Despite these differences, the isentropic model is reliable and suitable for this analysis, as it aligns well with regional temperature variations for the altitudes of interest.

For flight altitudes $z < 7958$ m:

$$\frac{T(z)}{T_s} = \left[1 - \frac{\gamma - 1}{\gamma} \left(\frac{z}{z^*} \right) \right], \quad (1)$$

$$\frac{P(z)}{P_s} = \left[1 - \frac{\gamma - 1}{\gamma} \left(\frac{z}{z^*} \right) \right]^{\frac{\gamma}{\gamma - 1}}. \quad (2)$$

For flight altitudes $z > 7958$ m:

$$T(z) = 210 \text{ K} \quad (3)$$

$$P(z) = 33.6 e^{-\frac{z - 7958}{6605}}. \quad (4)$$

$$T_s = 288 \text{ K}, \quad P_s = 101.3 \times 10^3 \text{ Pa}, \quad \gamma = 1.4, \quad z^* = 8404 \text{ m}.$$

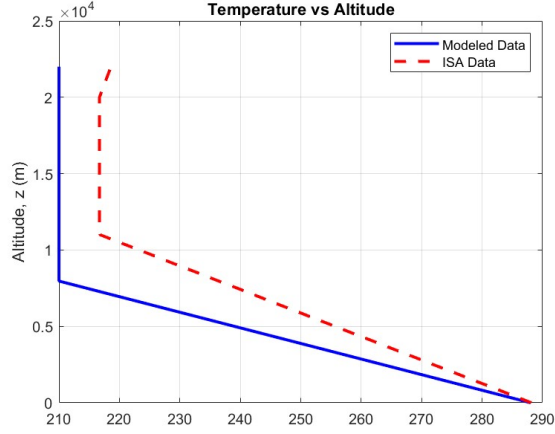


Figure 4: Temperature VS altitude

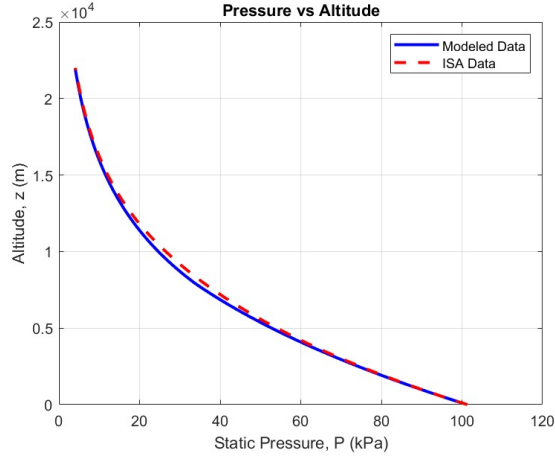


Figure 5: Pressure VS altitude

4 Part C: Study of M1 Variation

In part C, the inputs from validation case (a) are utilized, with the flight Mach number, M_1 varied incrementally from 0.8 to 5 to examine its impact on overall efficiency, thrust, and thrust-specific fuel consumption.

4.1 Overall efficiency

The overall efficiency with Mach number M_1 is as shown in Figure 6. The overall efficiency is low to begin with because the diffuser struggles to generate sufficient "ram compression" at lower flight speeds. It slowly goes up with it being the highest when the flight Mach number is between 3 and 3.5. Again, the overall efficiency drops sharply at flight speeds exceeding Mach 4. This behavior is influenced by the condition that the combustor inlet Mach number is fixed at 0.15 (non-thermally choked), requiring air traveling at supersonic speeds to decelerate significantly to Mach 0.15 at the combustor inlet.

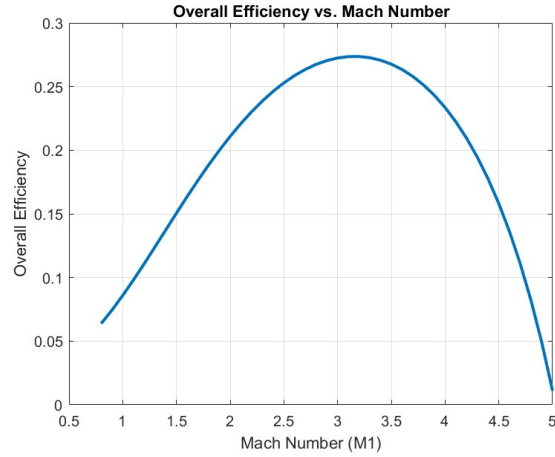


Figure 6: Overall efficiency as a function of M1

4.2 Thrust

The thrust with Mach number M1 is as shown in Figure 7. The thrust increases with increasing M1, peaking at Mach 4 to 4.5. After Mach 4.5, it drops sharply.

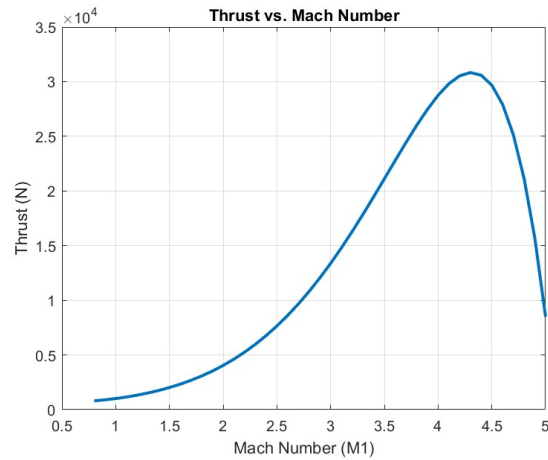


Figure 7: Thrust as a function of M1

4.3 TSFC

The TSFC with Mach number M1 is as shown in Figure 8. We can see that the TSFC remains mostly stable up to a flight speed of about Mach 4, after which it rises sharply. This behavior makes sense because flying at very high supersonic speeds requires a significant increase in fuel flow to maintain performance. This trend is also reflected in the overall efficiency plot, highlighting the challenges of operating at such high velocities. There is no clear "best" flight speed to operate a ramjet engine, as every speed involves trade-offs. The choice of flight speed should depend on the specific requirements of the mission, balancing efficiency, thrust, and fuel consumption to meet operational goals effectively.

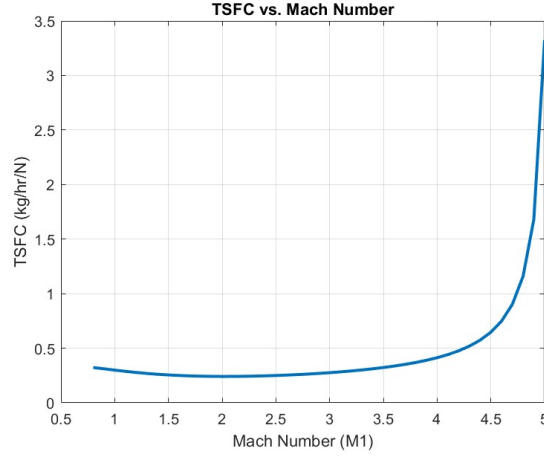


Figure 8: TSFC as a function of M1

5 Part D: Study of altitude Variation

In part D, the impact of flight altitude on performance parameters is analyzed by gradually increasing altitude, z from 2,000 to 30,000 meters.

5.1 Overall efficiency

The overall efficiency with altitude z is as shown in Figure 9. The overall efficiency decreases almost linearly up to an altitude of 8,000 meters, after which it remains constant up to 30,000 meters. This trend can be explained by temperature plot in standard vs isentropic atmosphere study, which illustrates that beyond 8,000 meters, the temperature stays constant in the isentropic atmosphere.

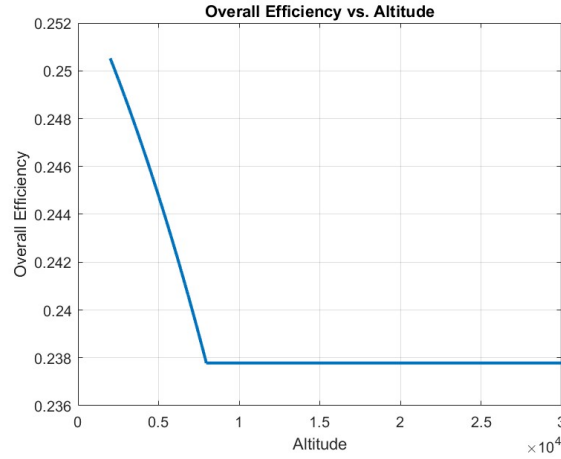


Figure 9: Overall efficiency vs altitude

5.2 Thrust

The thrust with altitude z is as shown in Figure 10. The thrust decreases with increasing altitude. This happens because higher altitudes have lower air density, reducing the mass flow rate into the engine.

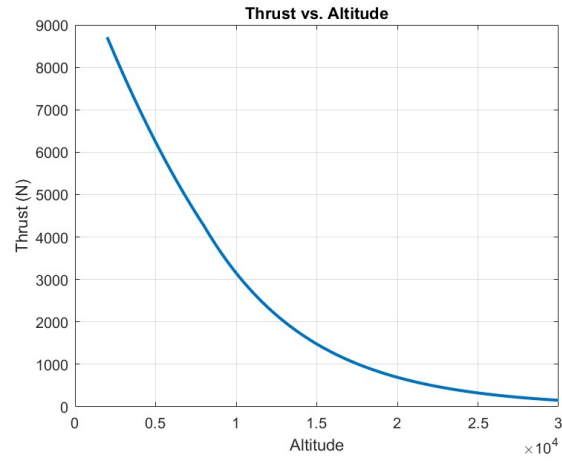


Figure 10: Thrust vs altitude

5.3 TSFC

The TSFC with altitude z is as shown in Figure 11. The TSFC has the same pattern as that for the overall efficiency and for the same reasons.

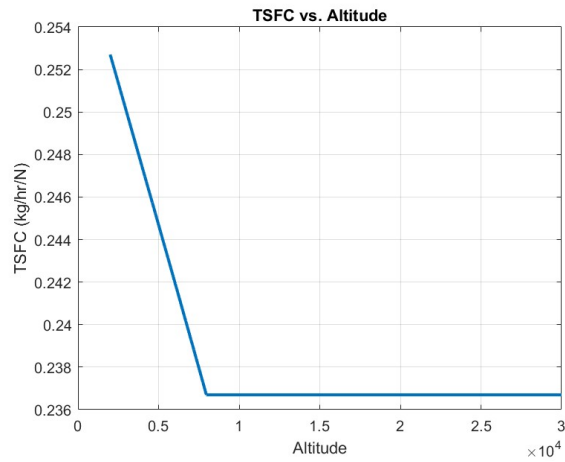


Figure 11: TSFC vs altitude

6 Part E: Study of M1 and altitude Variation

In part E, the flight altitude was varied from 2,000 to 20,000 meters in 500-meter increments. For each altitude, the flight Mach number was adjusted to find the value that maximized overall efficiency and this optimal Mach number was recorded. A similar approach was taken to identify the flight Mach number that minimized Thrust Specific Fuel Consumption (TSFC).

6.1 Optimized Overall efficiency

The overall efficiency optimizing (maximum) Mach number for each altitude are as shown in Figure 12. From 2,000 to 8,000 meters, the optimal flight Mach number increases linearly with altitude. Beyond 8,000 meters, it levels off, stabilizing at approximately 3.35 for the remainder of the altitude range.

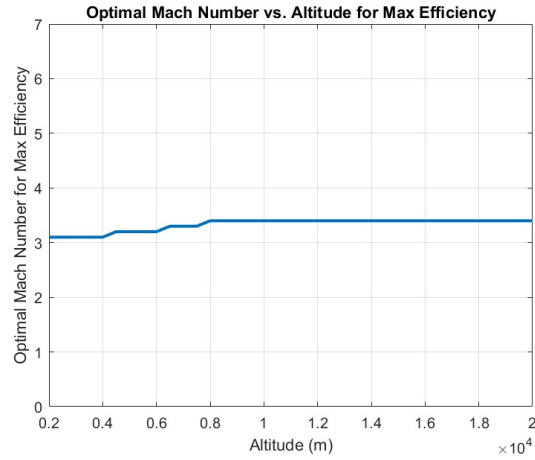


Figure 12: M1 VS Altitude for optimized thrust

6.2 Optimized TSFC

The TSFC optimizing (minimum) Mach number for each altitude are as shown in Figure 13. The trend for TSFC is different than that for overall efficiency, showing a sharp drop in the optimal flight Mach number between 2,000 and 4,000 meters. At lower altitudes, the denser atmosphere allows for higher thrust due to increased air mass flow through the engine. However, as altitude increases, achieving higher thrust becomes more fuel-intensive, causing the optimal flight Mach number to stabilize around 2 for the rest of the range.

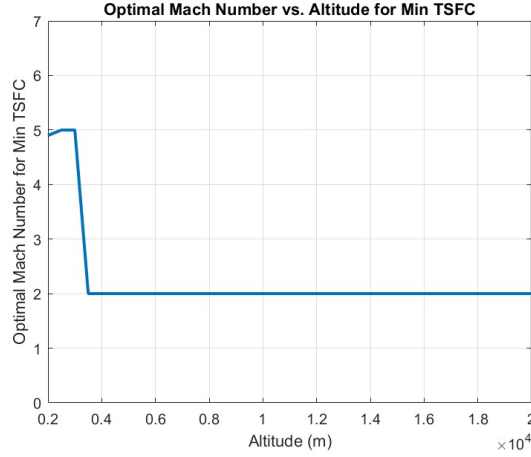


Figure 13: M1 VS Altitude for optimized thrust

7 Part F: Study of diffuser efficiency Variation

Part F highlights the significant impact of irreversibility by incrementally increasing diffuser efficiency and observing its effect on various performance parameters. The diffuser efficiency is varied from 0.5 to 1, representing the isentropic condition.

7.1 Overall efficiency

The overall efficiency with diffuser efficiency is as shown in Figure 14. The overall efficiency improves as ram compression becomes more efficient. This is because with higher diffuser efficiency, total pressure loss through the diffuser decreases, contributing to better overall performance.

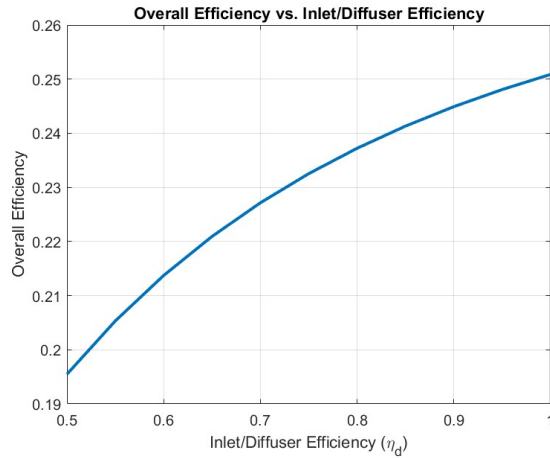


Figure 14: Overall efficiency vs diffuser efficiency

7.2 Thrust

The thrust with diffuser efficiency is as shown in Figure 15. The thrust increase greatly with increasing diffuser efficiency. The thrust produced with an ideal diffuser is nearly four times greater than that of a diffuser with 0.5 efficiency. This is because the ideal diffuser retains a much higher pressure ratio, significantly enhancing performance.

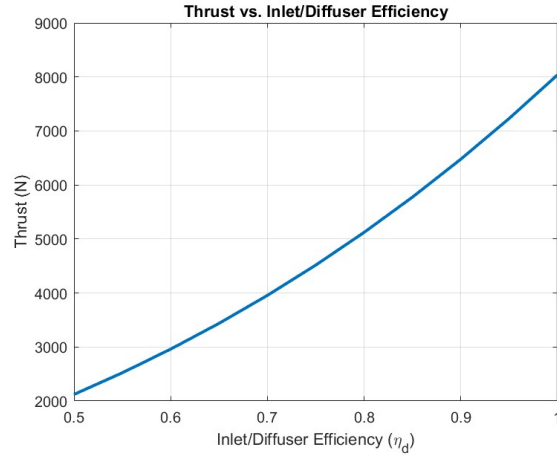


Figure 15: Thrust vs diffuser efficiency

7.3 TSFC

The TSFC with diffuser efficiency is as shown in Figure 16. TSFC decreases significantly as the diffuser becomes more efficient, approaching ideal performance. This is because TSFC is inversely proportional to thrust and has opposite effects to that for thrust.

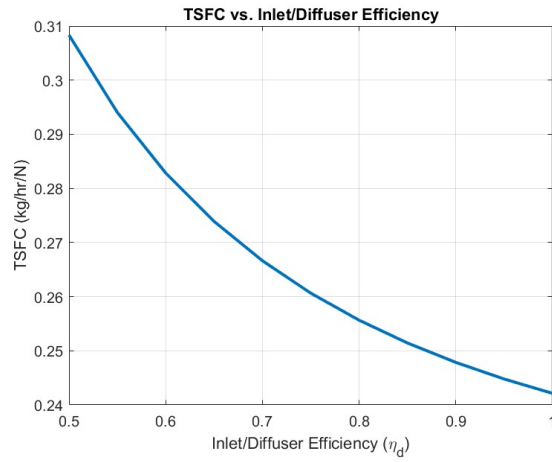


Figure 16: TSFC vs diffuser efficiency

8 Part G: Study of nozzle efficiency Variation

The part G examines the effect of varying the efficiency of the converging nozzle from 0.5 to 1 (isentropic).

8.1 Overall efficiency

The overall efficiency with nozzle efficiency is as shown in Figure 17. The overall efficiency improves as compression becomes more efficient. This is because with higher nozzle efficiency, total pressure loss through the nozzle decreases, contributing to better overall performance.

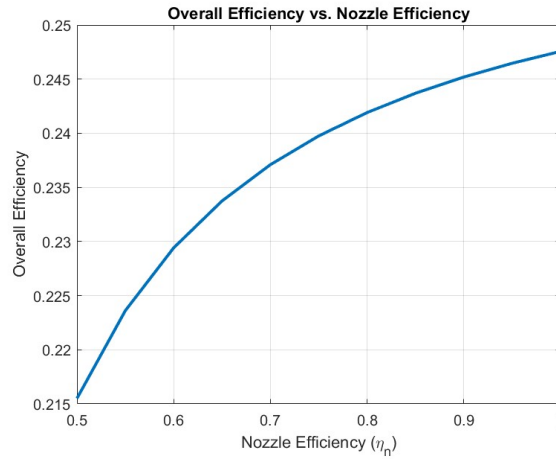


Figure 17: Overall efficiency vs nozzle efficiency

8.2 Thrust

The thrust with nozzle efficiency is as shown in Figure 18. The thrust increases with increasing nozzle efficiency. The thrust produced with an ideal nozzle is nearly 2.5 times greater than that of a nozzle with 0.5 efficiency. This is because the ideal nozzle retains a much higher pressure ratio, enhancing performance but is lower than the results from diffuser efficiency.

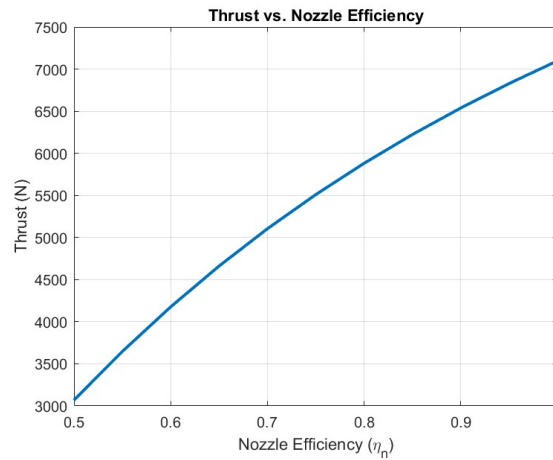


Figure 18: Thrust vs nozzle efficiency

8.3 TSFC

The TSFC with nozzle efficiency is as shown in Figure 19. TSFC decreases as the nozzle becomes more efficient, approaching ideal performance. This is because TSFC is inversely proportional to thrust and has opposite effects to that for thrust

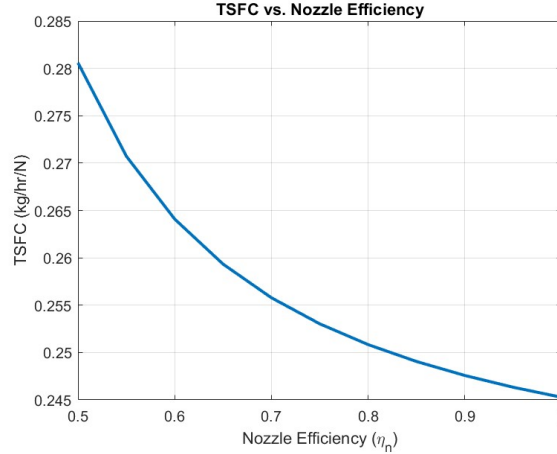


Figure 19: TSFC vs nozzle efficiency

9 Part H: Study of M2 Variation

In part H, the combustor inlet Mach number, M2 is varied from 0.1 to 2.5, and its impact on overall efficiency, thrust, and thrust-specific fuel consumption is analyzed.

9.1 Overall efficiency

The overall efficiency with M2 is as shown in Figure 20. The combustor inlet Mach number beyond about 0.6 until 1.3 results in a negative overall efficiency which is not useful.

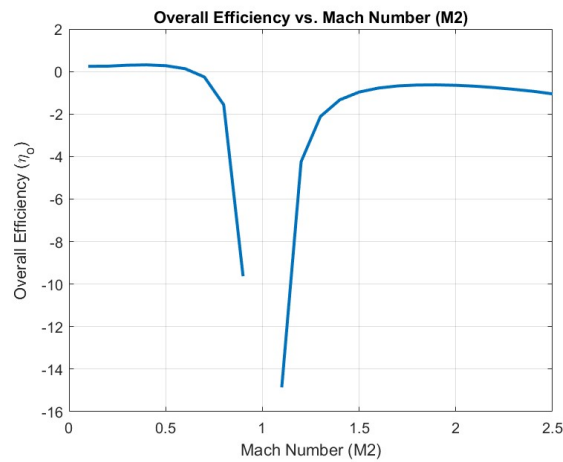


Figure 20: Overall efficiency vs M2

9.2 Thrust

The thrust with M_2 is as shown in Figure 21. Thrust performance declines sharply when the combustor inlet Mach number exceeds approximately 0.3. The unfavorable behavior of performance parameters closely tied to the combustor becoming thermally choked around $M_2=0.3$, as evident from the validation case 2 with $M_2=0.4$. At this point, additional heat input no longer raises combustion temperatures, limiting the performance.

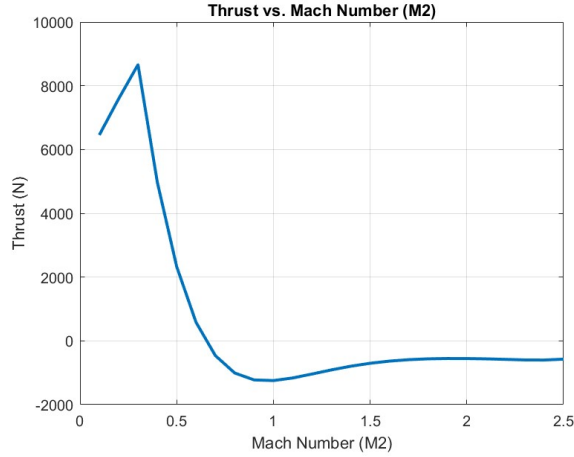


Figure 21: Thrust vs M_2

9.3 TSFC

The TSFC with M_2 is as shown in Figure 22. Thrust Specific Fuel Consumption (TSFC) displays unrealistic and erratic values when the combustor inlet Mach number, M_2 exceeds approximately 0.6. This behavior indicates a deviation from efficient operation at higher Mach numbers. But TSFC does show an optimal range that closely aligns with the thrust performance (valley in the thrust peak) trends observed in Figure 21.

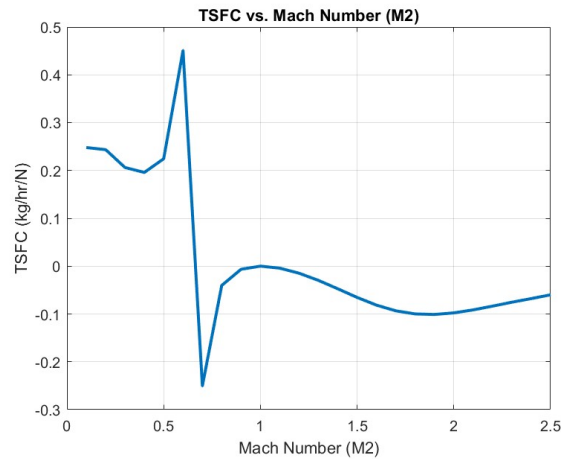


Figure 22: TSFC vs M_2

10 Part I: Design of Ramjet/Scramjet

Using the parametric analysis tool, a ram/scramjet propulsion system was designed for hypersonic flight at $M_1=5$ and $z=90,000\text{ft}$ (27,400m), adhering to the specified constraints. The design maintained the same fuel heating value (q_f) and inlet/diffuser and nozzle efficiencies as in the validation cases. The combustor exit total temperature (T_{t3}) was constrained to a maximum of 2400K, ensuring the diffuser exit Mach number (M_2) met this requirement. By systematically varying M_2 and T_{t3} , two optimal designs were identified: one that maximized positive thrust and another that maximized overall efficiency. The results for maximized positive thrust are presented in Table(1) and the results for maximized overall efficiency are listed in Table(2).

Maximum Thrust	M_2	T_{t3}
1970.42 N	0.41	2360K

Table 1: Optimized Thrust

Maximum Overall Efficiency	M_2	T_{t3}
0.1305	0.40	2400K

Table 2: Optimized Overall Efficiency

From the results, we can note that the design parameters for maximizing thrust and overall efficiency—specifically the combustor exit temperature and combustor inlet Mach number—are nearly identical. This outcome is logical, as the combustor exit temperature is capped at 2400 K, reflecting the realistic material limitations of modern engines. This temperature constraint significantly influences the extent to which the engine’s performance and efficiency can be optimized.

11 Conclusion

This project provided a detailed analysis of ramjet and scramjet propulsion systems using a MATLAB-based parametric tool. By modeling non-ideal flow conditions and systematically varying key input parameters, the study explored the impact on performance metrics such as thrust, overall efficiency, and thrust-specific fuel consumption.

The analysis showed that increasing flight Mach number improves overall efficiency and thrust up to a peak between Mach 3 and 3.5. However, beyond Mach 4, efficiency drops due to limitations in the diffuser and combustor. Similarly, altitude variations revealed that overall efficiency decreases linearly up to 8,000 meters before stabilizing, reflecting the isentropic atmosphere’s behavior at higher altitudes.

Improving diffuser and nozzle efficiencies significantly enhanced thrust and overall efficiency by reducing pressure losses. This also lowered thrust-specific fuel consumption (TSFC), highlighting the importance of minimizing irreversibility in propulsion systems to achieve better performance.

Varying the combustor inlet Mach number revealed important trade-offs. While performance improved within an optimal range, significant limitations occurred beyond $M_2=0.3$ due to thermal choking, which restricted the combustor’s ability to use additional heat effectively.

Finally, a ram/scramjet propulsion system was designed for hypersonic flight at $M_1=5$ and $z=27,400\text{m}$. The optimal designs for maximum thrust and maximum overall efficiency showed that combustor exit temperature constraints play a key role in determining performance. Future advances in materials (by the whitecoat lab

people :)) capable of handling higher temperatures could further enhance the performance of ramjet and scramjet engines, enabling more efficient designs for hypersonic applications.

Appendix

Code for all modules development:

```

1  clc;clear;clear all;
2  %% module 1
3
4  % Constants
5  gamma = 1.4;           % Specific heat ratio
6  R = 286.9;             % Specific gas constant in J/(kg*K)
7  z_star = 8404;         % Scale height in meters
8  T_s = 288.0;           % Standard temperature at sea level in Kelvin
9  p_s = 101.3;           % Standard pressure at sea level in kPa
10
11 % Input Parameters
12 z = 4300;
13 M1=2.4;
14
15 % Calculate T and P for each altitude
16 if z < 7958 % Within the troposphere
17     T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
18     p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
        gamma - 1)));
19 else % In the tropopause
20     T1 = 210.0; % Constant temperature in tropopause
21     p1 = 33.6 * exp(-(z - 7958) / 6605);
22 end
23
24 % Total-to-static relations
25 Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
26 pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
27
28 % Sound speed and velocity
29 a1 = sqrt(gamma * R * T1);
30 V1 = M1 * a1;
31 %% module 2
32
33 % Constants
34 gamma = 1.4;           % Specific heat ratio
35 R = 286.9;             % Specific gas constant in J/(kg*K)
36
37 % Inputs from Module 1 or given data
38 M2 = 0.15;             % Mach number at State 2
39 M2=0.40;               %vary for case 2
40 eta_d = 0.92;          % Inlet/diffuser efficiency
41
42 % Module 2 Calculations
43 % Total temperature remains constant (no work or heat transfer)
44 Tt2 = Tt1;

```

```

45
46 % Compute static temperature at State 2
47 T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
48
49 % Compute total and static pressures
50 pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
51 p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
52
53 % Compute entropy change across the diffuser
54 cp2 = 1004; % Specific heat at constant pressure (J/(kg*K)) for air
55 Delta_s_12 = cp2 * log(Tt2 / Tt1) - R * log(pt2 / pt1); % Entropy change (J/(kg
    *K))
56
57 % Compute speed of sound and velocity at State 2
58 a2 = sqrt(gamma * R * T2); % Speed of sound at State 2
59 V2 = M2 * a2; % Velocity at State 2
60
61 %% module 3
62
63 % Constants
64 gamma = 1.3; % Specific heat ratio changed to 1.3 from 1.4
65 Tt3_max = 2400; % Maximum allowable total temperature at
    combustor exit (K)
66
67
68 % Module 3 Calculations
69 % Step 1: Check if the combustor is thermally choked
70 Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
71     ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
72     ((1 + ((gamma - 1) / 2) * M2^2))^(-1);
73
74 % If thermally choked, adjust the maximum temperature
75 if Tt3_choked < Tt3_max
76     Tt3 = Tt3_choked;
77     M3=1;
78 else
79     Tt3 = Tt3_max;
80     % Solve for M3 in the non-choked case
81     % Use quadratic equation to solve for M3
82     C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)^2))
        * M2^2;
83
84
85 % Quadratic coefficients
86 a = C * (gamma^2) - ((gamma - 1)/2);
87 b = 2 * C * gamma - 1;
88 c = C;
89
90 % Solve the quadratic equation
91 M3_roots = roots([a, b, c]);
92
93 if M2<1
94     M3_squared = M3_roots(M3_roots > 0 & M3_roots <= 1);
95 else

```

```

96         M3_squared = M3_roots( M3_roots >=1);
97     end
98
99
100     % Take the square root to find M3
101     M3 = sqrt(M3_squared);
102 end
103
104 % Step 2: Compute heat added (q23)
105 q23 = 986 * (Tt3 - Tt2) + 0.5*0.179*(Tt3^2 - Tt2^2);
106
107 % Step 3: Compute static properties at combustor exit
108 T3 = Tt3 / (1 + (gamma - 1) / 2 * M3.^2);
109 p3 = p2; % Static pressure remains the same in constant-pressure combustion
110 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma/(gamma-1)));
111
112
113 % Step 4: Compute speed of sound and velocity at combustor exit
114 a3 = sqrt(gamma * R * T3); % Speed of sound at State 3
115 V3 = M3 * a3; % Velocity at State 3
116
117 %need cp3 from T3
118 cp3=986+0.179*T3;
119
120 % Step 5: Compute entropy increase across the combustor
121 Delta_s_23 = cp3* log(Tt3 / Tt2) - R * log(pt3 / pt2);
122
123 %entropy chnage 1-3
124 Delta_s_31 = cp3* log(Tt3 / Tt1) - R * log(pt3 / pt1);
125
126 %% module 4
127
128 % Inputs
129 Ae=0.015;
130 eta_n=0.94;
131 Tte = Tt3; % Total temperature at nozzle entrance
132
133
134 % Step 1: Compute test Mach number
135 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
136     ((1 - (p1 / pt3)^(gamma - 1) / gamma))) / ...
137     (1 - eta_n * (1 - (p1 / pt3)^(gamma - 1) / gamma))));
138
139
140 % Step 2: Check choking condition
141 if test_M < 1
142     % Nozzle is not choked
143     Me = test_M;
144     pe = p1; % Exit pressure equals ambient pressure
145 else
146     % Nozzle is choked
147     Me = 1;
148     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (gamma -
149         1));

```

```

149 end
150
151 % Step 3: Compute static properties at nozzle exit
152 Te = Tte / (1 + (gamma - 1) / 2 * Me^2); % Static temperature
153 pte= pe*((1 + (gamma - 1) / 2 * Me^2)^(gamma / (gamma - 1)));
154
155 %Compute speed of sound and velocity at nozzle exit
156 ae = sqrt(gamma * R * Te); % Speed of sound
157 Ve = Me * ae; % Velocity
158
159 % Step 4: Compute entropy increase
160 cpe = 986 + 0.179 * Te; % Specific heat at exit (J/(kg*K))
161 Delta_s_3e = cpe * log(Tte / Tt3) - R * log(pte / pt3);
162
163
164 % Mass flux
165 rho_e = pe *1000/ (R * Te); % Convert Pe to Pascals if needed
166 m_dot_e = rho_e * Ve * Ae;
167
168 %% module 5
169 Tt4=Tte;
170
171 % Step 2: Apply exit criterion for eta_n_ext
172 if test_M < 1
173     eta_n_ext = 1;
174 else
175     eta_n_ext = test_M.^ (-0.3);
176 end
177
178 % Step 3: Compute T4 (Static Temperature at State 4)
179 T4 = Tte * (1 - eta_n_ext * (1 - (p1 / pte)^((gamma - 1) / gamma)));
180
181 % Step 4: Compute M4 (Mach Number at State 4)
182 M4 = sqrt((2 / (gamma - 1)) * ((Tt4 / T4) - 1));
183
184
185 % Step 6: Compute p4 (Static Pressure at State 4)
186 p4 = p1; % Assume static pressure matches ambient pressure
187
188 % Step 7: Compute pt4 (Total Pressure at State 4)
189 pt4 = p4 * (1 + (gamma - 1) / 2 * M4^2)^(gamma / (gamma - 1));
190
191 % Step 8: Compute velocity at State 4
192 a4 = sqrt(gamma * R * T4); % Speed of sound at State 4
193 V4 = M4 * a4; % Velocity at State 4
194
195 % Step 9: Compute entropy increase across the nozzle
196 cp4 = 986 + 0.179 * T4;
197 Delta_s_4e = cp4 * log(Tt4 / Tte) - R * log(pt4 / pte);
198 Delta_s_4l = Delta_s_12+Delta_s_23+Delta_s_3e+Delta_s_4e;
199
200 %% module 6
201
202 % Constants

```

```

203 g0 = 9.81; % Gravitational acceleration (m/s^2)
204 q_f = 43.2e6; % Heating value of fuel (J/kg)
205
206
207 % Step 1: Compute air mass flow rate (mi)
208 m_dot_i = m_dot_e / (1 + q23 / q_f);
209
210 % Step 2: Compute fuel mass flow rate (mf)
211 m_dot_f = m_dot_e - m_dot_i;
212
213 % Step 3: Compute fuel-to-air ratio (f)
214 f = m_dot_f / m_dot_i;
215
216 % Step 4: Compute thrust
217 jet_thrust = m_dot_i * (1 + f) * Ve - m_dot_i * V1; % Jet thrust
218 pressure_thrust = (pe - p1)*1000 * Ae; % Pressure thrust
219 total_thrust = jet_thrust + pressure_thrust; % Total thrust
220
221 % Step 5: Compute equivalent velocity (Veq)
222 Veq = Ve + ((pe - p1) * 1000*Ae / m_dot_e); % Equivalent velocity
223
224 % Step 6: Compute TSFC
225 TSFC = (m_dot_f / total_thrust) * 3600; % TSFC in (kg/hr)/N
226
227 % Step 7: Compute specific impulse
228 Isp = total_thrust / (m_dot_f * g0); % Specific impulse
229 (s)
230
231 % Step 8: Compute efficiencies
232 thermal_efficiency = (m_dot_e * Veq^2 / 2 - m_dot_i * V1^2 / 2) / (m_dot_i *
q23);
233 propulsive_efficiency = 2 / (1+(Veq / V1));
234 overall_efficiency = thermal_efficiency * propulsive_efficiency;
235
236 %Propulsive power
237 Prop_power=total_thrust*V1;

```

Code for part A:

```

1 clc; clear; close all;
2
3 % Constants
4 a = 986; % Coefficient for cp(T) fit
5 b = 0.179; % Constant coefficient for cp(T) fit
6 cp_4to1 = 1004; % Constant cp for state 4->1 (J/kg-K)
7
8 % Static temperature (K) and entropy (J/(kg K)) at each state
9 %T = [245.9, 512.8, 891, 891, 635];
10 %s = [0, 43.95, 658.95, 670.95, 930.95];
11
12 T = [245.9, 526.8, 2354, 2087, 1558];
13 s = [0, 43.95, 2152, 2164, 2360]; %
14
15 % Extract points for states
16 T1 = T(1); T2 = T(2); T3 = T(3); Te = T(4); T4 = T(5);

```

```

17 s1 = s(1); s2 = s(2); s3 = s(3); se = s(4); s4 = s(5);
18
19 % Step sizes for entropy
20 ds_23 = 5; % Smaller step size for 2->3
21 ds_41 = 5; % Smaller step size for 4->1
22
23 % 1 -> 2 (straight line)
24 T_12 = linspace(T1, T2, 100);
25 s_12 = linspace(s1, s2, 100);
26
27 % Initialize entropy and temperature arrays
28 s_23 = s2:ds_23:s3; % Generate entropy steps
29 T_23 = zeros(size(s_23)); % Preallocate temperature array
30 T_23(1) = T2; % Set initial temperature
31
32 % Iteratively compute T values
33 for i = 2:length(s_23)
34     % Compute change in entropy
35     ds_actual = s_23(i) - s_23(i-1);
36
37     % Update temperature using the relation
38     dT = (T_23(i-1) * ds_actual) / (a + b * T_23(i-1));
39     T_23(i) = T_23(i-1) + dT;
40 end
41
42
43
44 % 3 -> e (straight line)
45 T_3e = linspace(T3, Te, 100);
46 s_3e = linspace(s3, se, 100);
47
48 % e -> 4 (straight line)
49 T_e4 = linspace(Te, T4, 100);
50 s_e4 = linspace(se, s4, 100);
51
52 % 4 -> 1 (constant-pressure curve)
53 T_41 = T4; % Initialize with T4
54 s_41 = s4:-ds_41:s1; % Entropy steps
55 for i = 2:length(s_41)
56     T_41(i) = T_41(i-1) - (T_41(i-1) * ds_41) / cp_4to1;
57 end
58
59 % Plot T-s diagram
60 figure;
61 hold on;
62
63 % Add lines and curves
64 plot(s_12, T_12, 'b', 'LineWidth', 2, 'DisplayName', '1->2 (Straight)');
65 plot(s_23, T_23, 'r', 'LineWidth', 2, 'DisplayName', '2->3 (Constant-p)');
66 plot(s_3e, T_3e, 'g', 'LineWidth', 2, 'DisplayName', '3->e (Straight)');
67 plot(s_e4, T_e4, 'm', 'LineWidth', 2, 'DisplayName', 'e->4 (Straight)');
68 plot(s_41, T_41, 'c', 'LineWidth', 2, 'DisplayName', '4->1 (Constant-p)');
69
70

```

```

71 % Set axes and labels
72 xlabel('Entropy, s - s1 (J/(kg K))');
73 ylabel('Temperature, T (K)');
74 title('T-s Diagram: 1->2->3->e->4->1');
75 legend('show');
76 grid on;
77 % hold off;

```

Code for part B:

```

1  clc;clear;clear all;
2  %% module 1
3
4  % Constants
5  gamma = 1.4;           % Specific heat ratio
6  R = 286.9;             % Specific gas constant in J/(kg*K)
7  z_star = 8404;         % Scale height in meters
8  T_s = 288.0;           % Standard temperature at sea level in Kelvin
9  p_s = 101.3;           % Standard pressure at sea level in kPa
10
11 % Input Parameters
12 z = linspace(0, 22000, 500); % Altitude range from 0 to 25,000 meters
13
14 % Initialize arrays for temperature and pressure
15 T1 = zeros(size(z));
16 p1 = zeros(size(z));
17
18 % Calculate T and P for each altitude
19 for i = 1:length(z)
20     if z(i) < 7958 % Within the troposphere
21         T1(i) = T_s * (1 - (((gamma - 1) / gamma) * (z(i) / z_star)));
22         p1(i) = p_s * ((1 - (((gamma - 1) / gamma) * (z(i) / z_star)))^(gamma
23             / (gamma - 1)));
24     else % In the tropopause
25         T1(i) = 210.0; % Constant temperature in tropopause
26         p1(i) = 33.6 * exp(-(z(i) - 7958) / 6605);
27     end
28 end
29
30 % International Standard Atmosphere (ISA) data
31 z_isa = [0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, ...
32     5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000, ...
33     10500, 11000, 11500, 12000, 12500, 13000, 13500, 14000, ...
34     14500, 15000, 15500, 16000, 16500, 17000, 17500, 18000, ...
35     18500, 19000, 19500, 20000, 22000];
36 T_isa = [288.15, 284.9, 281.7, 278.4, 275.2, 271.9, 268.7, 265.4, ...
37     262.2, 258.9, 255.7, 252.4, 249.2, 245.9, 242.7, 239.5, ...
38     236.2, 233, 229.7, 226.5, 223.3, 220, 216.8, 216.7, ...
39     216.7, 216.7, 216.7, 216.7, 216.7, 216.7, 216.7, 216.7, ...
40     216.7, 216.7, 216.7, 216.7, 216.7, 216.7, 216.7, 216.7, 216.7 218.6];
41 p_isa = [101.325, 95.46, 89.88, 84.56, 79.5, 74.69, 70.12, 65.78, ...
42     61.66, 57.75, 54.05, 50.54, 47.22, 44.08, 41.11, 38.3, ...
43     35.65, 33.15, 30.8, 28.58, 26.5, 24.54, 22.7, 20.98, ...
44     19.4, 17.93, 16.58, 15.33, 14.17, 13.1, 12.11, 11.2, ...
45     10.35, 9.572, 8.85, 8.182, 7.565, 6.995, 6.467, 5.98, 5.529, 4.047];

```

```

45
46 % Plot T vs z
47 figure;
48 plot(T1, z, 'b-', 'LineWidth', 2); % Modeled data
49 hold on;
50 plot(T_isa, z_isa, 'r--', 'LineWidth', 2); % ISA data
51 ylabel('Altitude, z (m)');
52 xlabel('Static Temperature, T (K)');
53 title('Temperature vs Altitude');
54 legend('Modeled Data', 'ISA Data');
55 grid on;
56
57 % Plot P vs z
58 figure;
59 plot(p1, z, 'b-', 'LineWidth', 2); % Modeled data
60 hold on;
61 plot(p_isa, z_isa, 'r--', 'LineWidth', 2); % ISA data
62 ylabel('Altitude, z (m)');
63 xlabel('Static Pressure, P (kPa)');
64 title('Pressure vs Altitude');
65 legend('Modeled Data', 'ISA Data');
66 grid on;

```

Code for part C:

```

1  clc; clear; clear all;
2
3  % Initialize Mach number range
4  M1_range = 0.8:0.1:5.0;
5
6  % Preallocate arrays for results
7  overall_efficiency_results = zeros(size(M1_range));
8  thrust_results = zeros(size(M1_range));
9  TSFC_results = zeros(size(M1_range));
10
11 for idx = 1:length(M1_range)
12     % Set M1 for this iteration
13     M1 = M1_range(idx);
14
15     %% module 1
16     % Constants
17     gamma = 1.4; % Specific heat ratio
18     R = 286.9; % Specific gas constant in J/(kg*K)
19     z_star = 8404; % Scale height in meters
20     T_s = 288.0; % Standard temperature at sea level in Kelvin
21     p_s = 101.3; % Standard pressure at sea level in kPa
22
23     % Input Parameters
24     z = 4300;
25
26     % Calculate T and P for each altitude
27     if z < 7958 % Within the troposphere
28         T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
29         p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
            gamma - 1)));

```



```

30     else % In the tropopause
31         T1 = 210.0; % Constant temperature in tropopause
32         p1 = 33.6 * exp(-(z - 7958) / 6605);
33     end
34
35     % Total-to-static relations
36     Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
37     pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
38
39     % Sound speed and velocity
40     a1 = sqrt(gamma * R * T1);
41     V1 = M1 * a1;
42
43     %% module 2
44     gamma = 1.4; % Specific heat ratio
45     eta_d = 0.92; % Inlet/diffuser efficiency
46     M2 = 0.15;
47
48     Tt2 = Tt1;
49     T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
50     pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
51     p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
52
53     %% module 3
54     gamma = 1.3; % Changed after the combustor
55     Tt3_max = 2400;
56
57     % Choking check
58     Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
59         ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
60         ((1 + (((gamma - 1) / 2) * M2^2)))^(-1);
61
62     if Tt3_choked < Tt3_max
63         Tt3 = Tt3_choked;
64         M3 = 1;
65     else
66         Tt3 = Tt3_max;
67         C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)
68             ^2)) * M2^2;
69         a = C * (gamma^2) - ((gamma - 1) / 2);
70         b = 2 * C * gamma - 1;
71         c = C;
72         M3_roots = roots([a, b, c]);
73         M3_squared = M3_roots(M3_roots > 0 & M3_roots < 2);
74         M3 = sqrt(M3_squared);
75     end
76
77     q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
78     T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
79     p3 = p2;
80     pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2)^(gamma / (gamma - 1)));
81
82     %% module 4
83     Ae = 0.015;

```

```

83     eta_n = 0.94;
84     Tte = Tt3;
85
86     test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
87         ((1 - (p1 / pt3).^((gamma - 1) / gamma))) / ...
88         (1 - eta_n * (1 - (p1 / pt3).^((gamma - 1) / gamma))))));
89
90     if test_M < 1
91         Me = test_M;
92         pe = p1;
93     else
94         Me = 1;
95         pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (
            gamma - 1));
96     end
97
98     Te = Tte / (1 + (gamma - 1) / 2 * Me^2);
99     ae = sqrt(gamma * R * Te);
100    Ve = Me * ae;
101    rho_e = pe * 1000 / (R * Te);
102    m_dot_e = rho_e * Ve * Ae;
103
104    %% module 6
105    g0 = 9.81;
106    m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
107    m_dot_f = m_dot_e - m_dot_i;
108    f = m_dot_f / m_dot_i;
109    jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
110    pressure_thrust = (pe - p1) * 1000 * Ae;
111    total_thrust = jet_thrust + pressure_thrust;
112    Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
113    TSFC = (m_dot_f ./ total_thrust) * 3600;
114    thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./ (
        m_dot_i .* q23);
115    propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
116    overall_efficiency = thermal_efficiency * propulsive_efficiency;
117
118    %% Store results
119    overall_efficiency_results(idx) = overall_efficiency;
120    thrust_results(idx) = total_thrust;
121    TSFC_results(idx) = TSFC;
122 end
123
124 %% Plot the results
125 figure;
126 plot(M1_range, overall_efficiency_results, 'LineWidth', 2);
127 xlabel('Mach Number (M1)');
128 ylabel('Overall Efficiency');
129 title('Overall Efficiency vs. Mach Number');
130 grid on;
131
132 figure;
133 plot(M1_range, thrust_results, 'LineWidth', 2);
134 xlabel('Mach Number (M1)');

```

```

135 ylabel('Thrust (N)');
136 title('Thrust vs. Mach Number');
137 grid on;
138
139 figure;
140 plot(Ml_range, TSFC_results, 'LineWidth', 2);
141 xlabel('Mach Number (Ml)');
142 ylabel('TSFC (kg/hr/N)');
143 title('TSFC vs. Mach Number');
144 grid on;

```

Code for part D:

```

1
2 clc; clear; clear all;
3
4 % Initialize altitude number range
5 z_range=2000:1:30000;
6
7 % Preallocate arrays for results
8 overall_efficiency_results = zeros(size(z_range));
9 thrust_results = zeros(size(z_range));
10 TSFC_results = zeros(size(z_range));
11
12 for idx = 1:length(z_range)
13     % Set Ml for this iteration
14     % Ml = Ml_range(idx);
15
16     %% module 1
17     % Constants
18     gamma = 1.4; % Specific heat ratio
19     R = 286.9; % Specific gas constant in J/(kg*K)
20     z_star = 8404; % Scale height in meters
21     T_s = 288.0; % Standard temperature at sea level in Kelvin
22     p_s = 101.3; % Standard pressure at sea level in kPa
23
24     Ml=2.4;
25     % Input Parameters
26     z = z_range(idx);
27
28     % Calculate T and P for each altitude
29     if z < 7958 % Within the troposphere
30         T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
31         p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
32             gamma - 1)));
33     else % In the tropopause
34         T1 = 210.0; % Constant temperature in tropopause
35         p1 = 33.6 * exp(-(z - 7958) / 6605);
36     end
37
38     % Total-to-static relations
39     Tt1 = T1 * (1 + (gamma - 1) / 2 * Ml^2);
40     pt1 = p1 * (1 + (gamma - 1) / 2 * Ml^2)^(gamma / (gamma - 1));
41
42     % Sound speed and velocity

```

```

42     a1 = sqrt(gamma * R * T1);
43     V1 = M1 * a1;
44
45 %% module 2
46 gamma = 1.4; % Specific heat ratio
47 eta_d = 0.92; % Inlet/diffuser efficiency
48 M2 = 0.15;
49
50 Tt2 = Tt1;
51 T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
52 pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
53 p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
54
55 %% module 3
56 gamma = 1.3; % Changed after the combustor
57 Tt3_max = 2400;
58
59 % Choking check
60 Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
61     ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
62     ((1 + (((gamma - 1) / 2) * M2^2)))^(-1);
63
64 if Tt3_choked < Tt3_max
65     Tt3 = Tt3_choked;
66     M3 = 1;
67 else
68     Tt3 = Tt3_max;
69     C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)
70         ^2)) * M2^2;
71     a = C * (gamma^2) - ((gamma - 1) / 2);
72     b = 2 * C * gamma - 1;
73     c = C;
74     M3_roots = roots([a, b, c]);
75     M3_squared = M3_roots(M3_roots > 0 & M3_roots < 2);
76     M3 = sqrt(M3_squared);
77
78 end
79
80 q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
81 T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
82 p3 = p2;
83 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma / (gamma - 1)));
84
85 %% module 4
86 Ae = 0.015;
87 eta_n = 0.94;
88 Tte = Tt3;
89
90 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
91     ((1 - (p1 / pt3).^((gamma - 1) / gamma))) / ...
92     (1 - eta_n * (1 - (p1 / pt3).^((gamma - 1) / gamma)))));
93
94 if test_M < 1
95     Me = test_M;
96     pe = p1;

```

```

95     else
96         Me = 1;
97         pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (
            gamma - 1));
98     end
99
100     Te = Tte / (1 + (gamma - 1) / 2 * Me^2);
101     ae = sqrt(gamma * R * Te);
102     Ve = Me * ae;
103     rho_e = pe * 1000 / (R * Te);
104     m_dot_e = rho_e * Ve * Ae;
105
106     %% module 6
107     g0 = 9.81;
108     m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
109     m_dot_f = m_dot_e - m_dot_i;
110     f = m_dot_f / m_dot_i;
111     jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
112     pressure_thrust = (pe - p1) * 1000 * Ae;
113     total_thrust = jet_thrust + pressure_thrust;
114     Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
115     TSFC = (m_dot_f ./ total_thrust) * 3600;
116     thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./ (
        m_dot_i .* q23);
117     propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
118     overall_efficiency = thermal_efficiency * propulsive_efficiency;
119
120     %% Store results
121     overall_efficiency_results(idx) = overall_efficiency;
122     thrust_results(idx) = total_thrust;
123     TSFC_results(idx) = TSFC;
124 end
125
126 %% Plot the results
127 figure;
128 plot(z_range, overall_efficiency_results, 'LineWidth', 2);
129 xlabel('Altitude');
130 ylabel('Overall Efficiency');
131 title('Overall Efficiency vs. Altitude');
132 grid on;
133
134 figure;
135 plot(z_range, thrust_results, 'LineWidth', 2);
136 xlabel('Altitude');
137 ylabel('Thrust (N)');
138 title('Thrust vs. Altitude');
139 grid on;
140
141 figure;
142 plot(z_range, TSFC_results, 'LineWidth', 2);
143 xlabel('Altitude');
144 ylabel('TSFC (kg/hr/N)');
145 title('TSFC vs. Altitude');
146 grid on;

```

Code for part E:

```

1  clc; clear; clear all;
2
3  % Initialize altitude and Mach number ranges
4  z_range = 2000:500:20000; % Altitudes in meters
5  M1_range = 0.8:0.1:5.0;    % Flight Mach numbers
6
7  % Preallocate arrays for results
8  optimal_M1_efficiency = zeros(size(z_range));
9  optimal_M1_TSFC = zeros(size(z_range));
10 max_efficiency_results = zeros(size(z_range));
11 min_TSFC_results = zeros(size(z_range));
12
13 for z_idx = 1:length(z_range)
14     z = z_range(z_idx); % Set altitude for this iteration
15
16     % Initialize temporary storage for results
17     efficiency_for_M1 = zeros(size(M1_range));
18     TSFC_for_M1 = zeros(size(M1_range));
19
20     for M1_idx = 1:length(M1_range)
21         M1 = M1_range(M1_idx); % Set Mach number for this iteration
22
23         %% module 1
24         % Constants
25         gamma = 1.4; % Specific heat ratio
26         R = 286.9; % Specific gas constant in J/(kg*K)
27         z_star = 8404; % Scale height in meters
28         T_s = 288.0; % Standard temperature at sea level in Kelvin
29         p_s = 101.3; % Standard pressure at sea level in kPa
30
31         % Calculate T and P for each altitude
32         if z < 7958 % Within the troposphere
33             T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
34             p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (gamma - 1)));
35         else % In the tropopause
36             T1 = 210.0; % Constant temperature in tropopause
37             p1 = 33.6 * exp(-(z - 7958) / 6605);
38         end
39
40         % Total-to-static relations
41         Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
42         pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
43
44         % Sound speed and velocity
45         a1 = sqrt(gamma * R * T1);
46         V1 = M1 * a1;
47
48         %% module 2
49         gamma = 1.4; % Specific heat ratio
50         eta_d = 0.92; % Inlet/diffuser efficiency
51         M2 = 0.15;
52

```

```

53 Tt2 = Tt1;
54 T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
55 pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1)
);
56 p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
57
58 %% module 3
59 gamma = 1.3; % Changed after the combustor
60 Tt3_max = 2400;
61
62 % Choking check
63 Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
64 ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
65 ((1 + ((gamma - 1) / 2) * M2^2))^(-1);
66
67 if Tt3_choked < Tt3_max
68     Tt3 = Tt3_choked;
69     M3 = 1;
70 else
71     Tt3 = Tt3_max;
72     C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2
^2)^2)) * M2^2;
73     a = C * (gamma^2) - ((gamma - 1) / 2);
74     b = 2 * C * gamma - 1;
75     c = C;
76     M3_roots = roots([a, b, c]);
77     M3_squared = M3_roots(M3_roots > 0 & M3_roots < 1);
78     M3 = sqrt(M3_squared);
79 end
80
81 q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
82 T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
83 p3 = p2;
84 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma / (gamma - 1)));
85
86 %% module 4
87 Ae = 0.015;
88 eta_n = 0.94;
89 Tte = Tt3;
90
91 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
92 ((1 - (p1 / pt3)^(gamma - 1) / gamma))) / ...
93 (1 - eta_n * (1 - (p1 / pt3)^(gamma - 1) / gamma))));
94
95 if test_M < 1
96     Me = test_M;
97     pe = p1;
98 else
99     Me = 1;
100     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma /
(gamma - 1));
101 end
102
103 Te = Tte / (1 + (gamma - 1) / 2 * Me^2);

```

```

104     ae = sqrt(gamma * R * Te);
105     Ve = Me * ae;
106     rho_e = pe * 1000 / (R * Te);
107     m_dot_e = rho_e * Ve * Ae;
108
109     %% module 6
110     g0 = 9.81;
111     m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
112     m_dot_f = m_dot_e - m_dot_i;
113     f = m_dot_f / m_dot_i;
114     jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
115     pressure_thrust = (pe - p1) * 1000 * Ae;
116     total_thrust = jet_thrust + pressure_thrust;
117     Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
118     TSFC = (m_dot_f ./ total_thrust) * 3600;
119     thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./
        (m_dot_i .* q23);
120     propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
121     overall_efficiency = thermal_efficiency * propulsive_efficiency;
122
123     %% Store temporary results
124     efficiency_for_M1(M1_idx) = overall_efficiency;
125     TSFC_for_M1(M1_idx) = TSFC;
126 end
127
128 %% Find optimal M1 for this altitude
129 [max_efficiency, max_eff_idx] = max(efficiency_for_M1);
130 [min_TSFC, min_TSFC_idx] = min(TSFC_for_M1);
131
132 % Store optimal results
133 optimal_M1_efficiency(z_idx) = M1_range(max_eff_idx);
134 optimal_M1_TSFC(z_idx) = M1_range(min_TSFC_idx);
135 max_efficiency_results(z_idx) = max_efficiency;
136 min_TSFC_results(z_idx) = min_TSFC;
137 end
138
139 %% Plot the results
140 figure;
141 plot(z_range, optimal_M1_efficiency, 'LineWidth', 2);
142 xlabel('Altitude (m)');
143 ylabel('Optimal Mach Number for Max Efficiency');
144 title('Optimal Mach Number vs. Altitude for Max Efficiency');
145 ylim([0 7]); % Set y-axis limits
146 grid on;
147
148 figure;
149 plot(z_range, optimal_M1_TSFC, 'LineWidth', 2);
150 xlabel('Altitude (m)');
151 ylabel('Optimal Mach Number for Min TSFC');
152 title('Optimal Mach Number vs. Altitude for Min TSFC');
153 ylim([0 7]); % Set y-axis limits
154 grid on;

```

Code for part F:


```

1  clc; clear; clear all;
2
3  % Initialize inlet/diffuser efficiency range
4  eta_d_range = 0.5:0.05:1.0;
5
6  % Preallocate arrays for results
7  overall_efficiency_results = zeros(size(eta_d_range));
8  thrust_results = zeros(size(eta_d_range));
9  TSFC_results = zeros(size(eta_d_range));
10
11 % Fixed parameters
12 z = 4300; % Altitude in meters
13 M1 = 2.4; % Mach number
14
15 for idx = 1:length(eta_d_range)
16     eta_d = eta_d_range(idx); % Current diffuser efficiency
17
18     %% module 1
19     % Constants
20     gamma = 1.4; % Specific heat ratio
21     R = 286.9; % Specific gas constant in J/(kg*K)
22     z_star = 8404; % Scale height in meters
23     T_s = 288.0; % Standard temperature at sea level in Kelvin
24     p_s = 101.3; % Standard pressure at sea level in kPa
25
26     % Calculate T and P for the given altitude
27     if z < 7958 % Within the troposphere
28         T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
29         p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
30             gamma - 1)));
31     else % In the tropopause
32         T1 = 210.0; % Constant temperature in tropopause
33         p1 = 33.6 * exp(-(z - 7958) / 6605);
34     end
35
36     % Total-to-static relations
37     Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
38     pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
39
40     % Sound speed and velocity
41     a1 = sqrt(gamma * R * T1);
42     V1 = M1 * a1;
43
44     %% module 2
45     M2 = 0.15;
46
47     Tt2 = Tt1;
48     T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
49     pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
50     p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
51
52     %% module 3
53     gamma = 1.3; % Changed after the combustor
54     Tt3_max = 2400;

```

```

54
55 % Choking check
56 Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
57 ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
58 ((1 + (((gamma - 1) / 2) * M2^2)))^(-1);
59
60 if Tt3_choked < Tt3_max
61     Tt3 = Tt3_choked;
62     M3 = 1;
63 else
64     Tt3 = Tt3_max;
65     C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)
66         ^2)) * M2^2;
67     a = C * (gamma^2) - ((gamma - 1) / 2);
68     b = 2 * C * gamma - 1;
69     c = C;
70     M3_roots = roots([a, b, c]);
71     M3_squared = M3_roots(M3_roots > 0 & M3_roots < 1);
72     M3 = sqrt(M3_squared);
73
74 end
75
76 q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
77 T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
78 p3 = p2;
79 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma / (gamma - 1)));
80
81 %% module 4
82 Ae = 0.015;
83 eta_n = 0.94;
84 Tte = Tt3;
85
86 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
87     ((1 - (p1 / pt3)^(gamma - 1) / gamma))) / ...
88     (1 - eta_n * (1 - (p1 / pt3)^(gamma - 1) / gamma))));
89
90 if test_M < 1
91     Me = test_M;
92     pe = p1;
93 else
94     Me = 1;
95     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (
96         gamma - 1));
97
98 end
99
100 Te = Tte / (1 + (gamma - 1) / 2 * Me^2);
101 ae = sqrt(gamma * R * Te);
102 Ve = Me * ae;
103 rho_e = pe * 1000 / (R * Te);
104 m_dot_e = rho_e * Ve * Ae;
105
106 %% module 6
107 g0 = 9.81;
108 m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
109 m_dot_f = m_dot_e - m_dot_i;

```

```

106     f = m_dot_f / m_dot_i;
107     jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
108     pressure_thrust = (pe - p1) * 1000 * Ae;
109     total_thrust = jet_thrust + pressure_thrust;
110     Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
111     TSFC = (m_dot_f ./ total_thrust) * 3600;
112     thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./ (
        m_dot_i .* q23);
113     propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
114     overall_efficiency = thermal_efficiency * propulsive_efficiency;
115
116     %% Store results
117     overall_efficiency_results(idx) = overall_efficiency;
118     thrust_results(idx) = total_thrust;
119     TSFC_results(idx) = TSFC;
120 end
121
122 %% Plot the results
123 figure;
124 plot(eta_d_range, overall_efficiency_results, 'LineWidth', 2);
125 xlabel('Inlet/Diffuser Efficiency (\eta_d)');
126 ylabel('Overall Efficiency');
127 title('Overall Efficiency vs. Inlet/Diffuser Efficiency');
128 grid on;
129
130 figure;
131 plot(eta_d_range, thrust_results, 'LineWidth', 2);
132 xlabel('Inlet/Diffuser Efficiency (\eta_d)');
133 ylabel('Thrust (N)');
134 title('Thrust vs. Inlet/Diffuser Efficiency');
135 grid on;
136
137 figure;
138 plot(eta_d_range, TSFC_results, 'LineWidth', 2);
139 xlabel('Inlet/Diffuser Efficiency (\eta_d)');
140 ylabel('TSFC (kg/hr/N)');
141 title('TSFC vs. Inlet/Diffuser Efficiency');
142 grid on;

```

Code for part G:

```

1  clc; clear; clear all;
2
3  % Initialize nozzle efficiency range
4  eta_n_range = 0.5:0.05:1.0;
5
6  % Preallocate arrays for results
7  overall_efficiency_results = zeros(size(eta_n_range));
8  thrust_results = zeros(size(eta_n_range));
9  TSFC_results = zeros(size(eta_n_range));
10
11 % Fixed parameters
12 z = 4300; % Altitude in meters
13 M1 = 2.4; % Mach number
14 eta_d = 0.92; % Fixed inlet/diffuser efficiency

```

```

15
16 for idx = 1:length(eta_n_range)
17     eta_n = eta_n_range(idx); % Current nozzle efficiency
18
19     %% module 1
20     % Constants
21     gamma = 1.4; % Specific heat ratio
22     R = 286.9; % Specific gas constant in J/(kg*K)
23     z_star = 8404; % Scale height in meters
24     T_s = 288.0; % Standard temperature at sea level in Kelvin
25     p_s = 101.3; % Standard pressure at sea level in kPa
26
27     % Calculate T and P for the given altitude
28     if z < 7958 % Within the troposphere
29         T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
30         p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
            gamma - 1)));
31     else % In the tropopause
32         T1 = 210.0; % Constant temperature in tropopause
33         p1 = 33.6 * exp(-(z - 7958) / 6605);
34     end
35
36     % Total-to-static relations
37     Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
38     pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
39
40     % Sound speed and velocity
41     a1 = sqrt(gamma * R * T1);
42     V1 = M1 * a1;
43
44     %% module 2
45     M2 = 0.15;
46
47     Tt2 = Tt1;
48     T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
49     pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
50     p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
51
52     %% module 3
53     gamma = 1.3; % Changed after the combustor
54     Tt3_max = 2400;
55
56     % Choking check
57     Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
58         ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
59         ((1 + (((gamma - 1) / 2) * M2^2)))^(-1);
60
61     if Tt3_choked < Tt3_max
62         Tt3 = Tt3_choked;
63         M3 = 1;
64     else
65         Tt3 = Tt3_max;
66         C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)
            ^2)) * M2^2;

```

```

67     a = C * (gamma^2) - ((gamma - 1) / 2);
68     b = 2 * C * gamma - 1;
69     c = C;
70     M3_roots = roots([a, b, c]);
71     M3_squared = M3_roots(M3_roots > 0 & M3_roots < 1);
72     M3 = sqrt(M3_squared);
73 end
74
75 q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
76 T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
77 p3 = p2;
78 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma / (gamma - 1)));
79
80 %% module 4
81 Ae = 0.015;
82 Tte = Tt3;
83
84 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
85     ((1 - (p1 / pt3)^(gamma - 1) / gamma))) / ...
86     (1 - eta_n * (1 - (p1 / pt3)^(gamma - 1) / gamma))));
87
88 if test_M < 1
89     Me = test_M;
90     pe = p1;
91 else
92     Me = 1;
93     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (
94         gamma - 1));
95 end
96
97 Te = Tte / (1 + (gamma - 1) / 2 * Me^2);
98 ae = sqrt(gamma * R * Te);
99 Ve = Me * ae;
100 rho_e = pe * 1000 / (R * Te);
101 m_dot_e = rho_e * Ve * Ae;
102
103 %% module 6
104 g0 = 9.81;
105 m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
106 m_dot_f = m_dot_e - m_dot_i;
107 f = m_dot_f / m_dot_i;
108 jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
109 pressure_thrust = (pe - p1) * 1000 * Ae;
110 total_thrust = jet_thrust + pressure_thrust;
111 Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
112 TSFC = (m_dot_f ./ total_thrust) * 3600;
113 thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./ (
114     m_dot_i .* q23);
115 propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
116 overall_efficiency = thermal_efficiency * propulsive_efficiency;
117
118 %% Store results
119 overall_efficiency_results(idx) = overall_efficiency;
120 thrust_results(idx) = total_thrust;

```

```

119     TSFC_results(idx) = TSFC;
120 end
121
122 %% Plot the results
123 figure;
124 plot(eta_n_range, overall_efficiency_results, 'LineWidth', 2);
125 xlabel('Nozzle Efficiency (\eta_n)');
126 ylabel('Overall Efficiency');
127 title('Overall Efficiency vs. Nozzle Efficiency');
128 grid on;
129
130 figure;
131 plot(eta_n_range, thrust_results, 'LineWidth', 2);
132 xlabel('Nozzle Efficiency (\eta_n)');
133 ylabel('Thrust (N)');
134 title('Thrust vs. Nozzle Efficiency');
135 grid on;
136
137 figure;
138 plot(eta_n_range, TSFC_results, 'LineWidth', 2);
139 xlabel('Nozzle Efficiency (\eta_n)');
140 ylabel('TSFC (kg/hr/N)');
141 title('TSFC vs. Nozzle Efficiency');
142 grid on;

```

Code for part H:

```

1  clc; clear; clear all;
2
3  % Initialize Mach number M2 range
4  M2_range = 0.1:0.1:2.5;
5
6  % Preallocate arrays for results
7  overall_efficiency_results = zeros(size(M2_range));
8  thrust_results = zeros(size(M2_range));
9  TSFC_results = zeros(size(M2_range));
10
11 % Fixed parameters
12 z = 4300; % Altitude in meters
13 M1 = 2.4; % Fixed Mach number at inlet
14 eta_d = 0.92; % Fixed inlet/diffuser efficiency
15 eta_n = 0.94; % Fixed nozzle efficiency
16
17 for idx = 1:length(M2_range)
18     M2 = M2_range(idx); % Current Mach number entering the combustor
19
20     %% module 1
21     % Constants
22     gamma = 1.4; % Specific heat ratio
23     R = 286.9; % Specific gas constant in J/(kg*K)
24     z_star = 8404; % Scale height in meters
25     T_s = 288.0; % Standard temperature at sea level in Kelvin
26     p_s = 101.3; % Standard pressure at sea level in kPa
27
28     % Calculate T and P for the given altitude

```

```

29     if z < 7958 % Within the troposphere
30         T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
31         p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
            gamma - 1)));
32     else % In the tropopause
33         T1 = 210.0; % Constant temperature in tropopause
34         p1 = 33.6 * exp(-(z - 7958) / 6605);
35     end
36
37     % Total-to-static relations
38     Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
39     pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
40
41     % Sound speed and velocity
42     a1 = sqrt(gamma * R * T1);
43     V1 = M1 * a1;
44
45     %% module 2
46     Tt2 = Tt1;
47     T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
48     pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
49     p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
50
51     %% module 3
52     gamma = 1.3; % Changed after the combustor
53     Tt3_max = 2400;
54
55     % Choking check
56     Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
57         ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
58         ((1 + (((gamma - 1) / 2) * M2^2)))^(-1);
59
60     if Tt3_choked < Tt3_max
61         Tt3 = Tt3_choked;
62         M3 = 1;
63     else
64         Tt3 = Tt3_max;
65         C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)
            ^2)) * M2^2;
66         a = C * (gamma^2) - ((gamma - 1) / 2);
67         b = 2 * C * gamma - 1;
68         c = C;
69         M3_roots = roots([a, b, c]);
70         if M2 < 1
71             M3_squared = M3_roots(M3_roots > 0 & M3_roots <= 1);
72         else
73             M3_squared = M3_roots(M3_roots >= 1);
74         end
75         M3 = sqrt(M3_squared);
76     end
77
78     q23 = 986 * (Tt3 - Tt2) + 0.5 * 0.179 * (Tt3^2 - Tt2^2);
79     T3 = Tt3 ./ (1 + (gamma - 1) ./ 2 * M3.^2);
80     p3 = p2;

```

```

81     pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma / (gamma - 1)));
82
83 %% module 4
84 Ae = 0.015;
85 Tte = Tt3;
86
87 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
88     ((1 - (p1 / pt3)^((gamma - 1) / gamma))) / ...
89     (1 - eta_n * (1 - (p1 / pt3)^((gamma - 1) / gamma)))));
90
91 if test_M < 1
92     Me = test_M;
93     pe = p1;
94 else
95     Me = 1;
96     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (
97         gamma - 1));
98
99 end
100 Te = Tte / (1 + (gamma - 1) / 2 * Me^2);
101 ae = sqrt(gamma * R * Te);
102 Ve = Me * ae;
103 rho_e = pe * 1000 / (R * Te);
104 m_dot_e = rho_e * Ve * Ae;
105
106 %% module 6
107 g0 = 9.81;
108 m_dot_i = m_dot_e / (1 + q23 / 43.2e6);
109 m_dot_f = m_dot_e - m_dot_i;
110 f = m_dot_f / m_dot_i;
111 jet_thrust = m_dot_i .* (1 + f) .* Ve - m_dot_i .* V1;
112 pressure_thrust = (pe - p1) * 1000 * Ae;
113 total_thrust = jet_thrust + pressure_thrust;
114 Veq = Ve + ((pe - p1) * 1000 * Ae / m_dot_e);
115 TSFC = (m_dot_f ./ total_thrust) * 3600;
116 thermal_efficiency = (m_dot_e .* Veq^2 ./ 2 - m_dot_i .* V1^2 ./ 2) ./ (
117     m_dot_i .* q23);
118 propulsive_efficiency = 2 ./ (1 + (Veq ./ V1));
119 overall_efficiency = thermal_efficiency * propulsive_efficiency;
120
121 %% Store results
122 overall_efficiency_results(idx) = overall_efficiency;
123 thrust_results(idx) = total_thrust;
124 TSFC_results(idx) = TSFC;
125
126 end
127
128 %% Plot the results
129 figure;
130 plot(M2_range, overall_efficiency_results, 'LineWidth', 2);
131 xlabel('Mach Number (M2)');
132 ylabel('Overall Efficiency (\eta_o)');
133 title('Overall Efficiency vs. Mach Number (M2)');
134 grid on;
135

```



```

133 figure;
134 plot(M2_range, thrust_results, 'LineWidth', 2);
135 xlabel('Mach Number (M2)');
136 ylabel('Thrust (N)');
137 title('Thrust vs. Mach Number (M2)');
138 grid on;
139
140 figure;
141 plot(M2_range, TSFC_results, 'LineWidth', 2);
142 xlabel('Mach Number (M2)');
143 ylabel('TSFC (kg/hr/N)');
144 title('TSFC vs. Mach Number (M2)');
145 grid on;

```

Code for part I:

```

1  clc; clear; clear all;
2
3  % Define the range for parametric analysis
4  M2_range = 0.1:0.01:5; % Diffuser exit Mach number
5  Tt3_range = 1500:5:2400; % Combustor exit total temperature
6
7  % Preallocate results
8  Thrust = zeros(length(M2_range), length(Tt3_range));
9  Efficiency = zeros(length(M2_range), length(Tt3_range));
10
11 % Loop over M2 and Tt3 to calculate thrust and efficiency
12 for i = 1:length(M2_range)
13     M2 = M2_range(i);
14     for j = 1:length(Tt3_range)
15         Tt3 = Tt3_range(j);
16         %% module 1
17
18 % Constants
19 gamma = 1.4; % Specific heat ratio
20 R = 286.9; % Specific gas constant in J/(kg*K)
21 z_star = 8404; % Scale height in meters
22 T_s = 288.0; % Standard temperature at sea level in Kelvin
23 p_s = 101.3; % Standard pressure at sea level in kPa
24
25 % Input Parameters
26 z = 27400;
27 M1=5;
28
29 % Calculate T and P for each altitude
30 if z < 7958 % Within the troposphere
31     T1 = T_s * (1 - (((gamma - 1) / gamma) * (z / z_star)));
32     p1 = p_s * ((1 - (((gamma - 1) / gamma) * (z / z_star)))^(gamma / (
33         gamma - 1)));
34 else % In the tropopause
35     T1 = 210.0; % Constant temperature in tropopause
36     p1 = 33.6 * exp(-(z - 7958) / 6605);
37 end
38 % Total-to-static relations

```

```

39 Tt1 = T1 * (1 + (gamma - 1) / 2 * M1^2);
40 pt1 = p1 * (1 + (gamma - 1) / 2 * M1^2)^(gamma / (gamma - 1));
41
42 % Sound speed and velocity
43 a1 = sqrt(gamma * R * T1);
44 V1 = M1 * a1;
45 %% module 2
46
47 % Constants
48 gamma = 1.4;           % Specific heat ratio
49 R = 286.9;             % Specific gas constant in J/(kg*K)
50
51 % Inputs from Module 1 or given data
52 M2 = 0.15;             % Mach number at State 2
53 M2=0.40;
54 eta_d = 0.92;          % Inlet/diffuser efficiency
55
56 % Module 2 Calculations
57 % Total temperature remains constant (no work or heat transfer)
58 Tt2 = Tt1;
59
60 % Compute static temperature at State 2
61 T2 = Tt2 / (1 + (gamma - 1) / 2 * M2^2);
62
63 % Compute total and static pressures
64 pt2 = p1 * (1 + (eta_d * (gamma - 1) / 2) * M1^2)^(gamma / (gamma - 1));
65 p2 = pt2 / (1 + (gamma - 1) / 2 * M2^2)^(gamma / (gamma - 1));
66
67 % Compute entropy change across the diffuser
68 cp2 = 1004; % Specific heat at constant pressure (J/(kg*K)) for air
69 Delta_s_12 = cp2 * log(Tt2 / Tt1) - R * log(pt2 / pt1); % Entropy change (J/(kg
    *K))
70
71 % Compute speed of sound and velocity at State 2
72 a2 = sqrt(gamma * R * T2); % Speed of sound at State 2
73 V2 = M2 * a2;             % Velocity at State 2
74
75 %% module 3
76
77 % Constants
78 gamma = 1.3;             % Specific heat ratio changed to 1.3 from 1.4
79 Tt3_max = 2400;          % Maximum allowable total temperature at
    combustor exit (K)
80
81
82 % Module 3 Calculations
83 % Step 1: Check if the combustor is thermally choked
84 Tt3_choked = Tt2 * (1 / (2 * (gamma + 1))) * ...
    ((1 / (M2^2) * ((1 + gamma * M2^2)^2))) * ...
    ((1 + (((gamma - 1) / 2) * M2^2))^( -1));
85
86
87
88 % If thermally choked, adjust the maximum temperature
89 if Tt3_choked < Tt3
90     Tt3 = Tt3_choked;

```

```

91     M3=1;
92 else
93     %Tt3 = Tt3;
94     % Solve for M3 in the non-choked case
95     % Use quadratic equation to solve for M3
96     C = (Tt3 / Tt2) * ((1 + (gamma - 1) / 2 * M2^2) / ((1 + gamma * M2^2)^2))
          * M2^2;
97
98
99     % Quadratic coefficients
100    a = C * (gamma^2) - ((gamma - 1)/2);
101    b = 2 * C * gamma - 1;
102    c = C;
103
104    % Solve the quadratic equation
105    M3_roots = roots([a, b, c]);
106
107    if M2<1
108        M3_squared = M3_roots(M3_roots > 0 & M3_roots <= 1);
109    else
110        M3_squared = M3_roots( M3_roots >=1);
111    end
112
113
114    % Take the square root to find M3
115    M3 = sqrt(M3_squared);
116 end
117
118 % Step 2: Compute heat added (q23)
119 q23 = 986 * (Tt3 - Tt2)+0.5*0.179*(Tt3^2 - Tt2^2);
120
121 % Step 3: Compute static properties at combustor exit
122 T3 = Tt3 / (1 + (gamma - 1) / 2 * M3.^2);
123 p3 = p2; % Static pressure remains the same in constant-pressure combustion
124 pt3 = p3 * ((1 + (gamma - 1) / 2 * M3.^2).^(gamma/(gamma-1)));
125
126
127 % Step 4: Compute speed of sound and velocity at combustor exit
128 a3 = sqrt(gamma * R * T3); % Speed of sound at State 3
129 V3 = M3 * a3; % Velocity at State 3
130
131 %need cp3 from T3
132 cp3=986+0.179*T3;
133
134 % Step 5: Compute entropy increase across the combustor
135 Delta_s_23 = cp3* log(Tt3 / Tt2) - R * log(pt3 / pt2);
136
137 %entropy chnage 1-3
138 Delta_s_31 = cp3* log(Tt3 / Tt1) - R * log(pt3 / pt1);
139
140 %% module 4
141
142 % Inputs
143 Ae=0.015;

```

```

144 eta_n=0.94;
145 Tte = Tt3; % Total temperature at nozzle entrance
146
147
148 % Step 1: Compute test Mach number
149 test_M = sqrt(2 / (gamma - 1)) * sqrt((eta_n * ...
150      ((1 - (p1 / pt3)^((gamma - 1) / gamma))) / ...
151      (1 - eta_n * (1 - (p1 / pt3)^((gamma - 1) / gamma)))));
152
153
154 % Step 2: Check choking condition
155 if test_M < 1
156     % Nozzle is not choked
157     Me = test_M;
158     pe = p1; % Exit pressure equals ambient pressure
159 else
160     % Nozzle is choked
161     Me = 1;
162     pe = pt3 * (1 - (1 / eta_n) * (gamma - 1) / (gamma + 1))^(gamma / (gamma -
163         1));
164
165 % Step 3: Compute static properties at nozzle exit
166 Te = Tte / (1 + (gamma - 1) / 2 * Me^2); % Static temperature
167 pte= pe*((1 + (gamma - 1) / 2 * Me^2)^(gamma / (gamma - 1)));
168
169 %Compute speed of sound and velocity at nozzle exit
170 ae = sqrt(gamma * R * Te); % Speed of sound
171 Ve = Me * ae; % Velocity
172
173 % Step 4: Compute entropy increase
174 cpe = 986 + 0.179 * Te; % Specific heat at exit (J/(kg*K))
175 Delta_s_3e = cpe * log(Tte / Tt3) - R * log(pte / pt3);
176
177
178 % Mass flux
179 rho_e = pe *1000/ (R * Te); % Convert Pe to Pascals if needed
180 m_dot_e = rho_e * Ve * Ae;
181
182 %% module 5
183 Tt4=Tte;
184
185 % Step 2: Apply exit criterion for eta_n_ext
186 if test_M < 1
187     eta_n_ext = 1;
188 else
189     eta_n_ext = test_M.^ (-0.3);
190 end
191
192 % Step 3: Compute T4 (Static Temperature at State 4)
193 T4 = Tte * (1 - eta_n_ext * (1 - (p1 / pte)^((gamma - 1) / gamma)));
194
195 % Step 4: Compute M4 (Mach Number at State 4)
196 M4 = sqrt((2 / (gamma - 1)) * ((Tt4 / T4) - 1));

```

```

197
198
199 % Step 6: Compute p4 (Static Pressure at State 4)
200 p4 = p1; % Assume static pressure matches ambient pressure
201
202 % Step 7: Compute pt4 (Total Pressure at State 4)
203 pt4 = p4 * (1 + (gamma - 1) / 2 * M4^2)^(gamma / (gamma - 1));
204
205 % Step 8: Compute velocity at State 4
206 a4 = sqrt(gamma * R * T4); % Speed of sound at State 4
207 V4 = M4 * a4; % Velocity at State 4
208
209 % Step 9: Compute entropy increase across the nozzle
210 cp4 = 986 + 0.179 * T4;
211 Delta_s_4e = cp4 * log(Tt4 / Tte) - R * log(pt4 / pte);
212 Delta_s_4l = Delta_s_12+Delta_s_23+Delta_s_3e+Delta_s_4e;
213
214 %% module 6
215
216 % Constants
217 g0 = 9.81; % Gravitational acceleration (m/s^2)
218 q_f = 43.2e6; % Heating value of fuel (J/kg)
219
220
221 % Step 1: Compute air mass flow rate (mi)
222 m_dot_i = m_dot_e / (1 + q23 / q-f);
223
224 % Step 2: Compute fuel mass flow rate (mf)
225 m_dot_f = m_dot_e - m_dot_i;
226
227 % Step 3: Compute fuel-to-air ratio (f)
228 f = m_dot_f / m_dot_i;
229
230 % Step 4: Compute thrust
231 jet_thrust = m_dot_i * (1 + f) * Ve - m_dot_i * V1; % Jet thrust
232 pressure_thrust = (pe - p1)*1000 * Ae; % Pressure thrust
233 total_thrust = jet_thrust + pressure_thrust; % Total thrust
234
235 % Step 5: Compute equivalent velocity (Veq)
236 Veq = Ve + ((pe - p1) * 1000*Ae / m_dot_e); % Equivalent velocity
237
238 % Step 6: Compute TSFC
239 TSFC = (m_dot_f / total_thrust) * 3600; % TSFC in (kg/hr)/N
240
241 % Step 7: Compute specific impulse
242 Isp = total_thrust / (m_dot_f * g0); % Specific impulse
243 (s)
244
245 % Step 8: Compute efficiencies
246 thermal_efficiency = (m_dot_e * Veq^2 / 2 - m_dot_i * V1^2 / 2) / (m_dot_i *
q23);
247 propulsive_efficiency = 2 / (1+(Veq /V1));
248 overall_efficiency = thermal_efficiency * propulsive_efficiency;

```

```

249         Thrust(i, j) = total_thrust;
250         Efficiency(i, j) = overall_efficiency;
251     end
252 end
253
254 % Find optimal values
255 [max_thrust, idx_thrust] = max(Thrust(:));
256 [optimal_i_thrust, optimal_j_thrust] = ind2sub(size(Thrust), idx_thrust);
257 optimal_M2_thrust = M2_range(optimal_i_thrust);
258 optimal_Tt3_thrust = Tt3_range(optimal_j_thrust);
259
260 [max_efficiency, idx_efficiency] = max(Efficiency(:));
261 [optimal_i_efficiency, optimal_j_efficiency] = ind2sub(size(Efficiency),
    idx_efficiency);
262 optimal_M2_efficiency = M2_range(optimal_i_efficiency);
263 optimal_Tt3_efficiency = Tt3_range(optimal_j_efficiency);
264
265 % Display results
266 fprintf('Maximum Thrust: %.2f N at M2 = %.2f, Tt3 = %.2f K\n', max_thrust,
    optimal_M2_thrust, optimal_Tt3_thrust);
267 fprintf('Maximum Efficiency: %.4f at M2 = %.2f, Tt3 = %.2f K\n',
    max_efficiency, optimal_M2_efficiency, optimal_Tt3_efficiency);

```