J58 (JT11D-20) Preliminary Design and Analysis

Christian J. Lagares,[[1]](#footnote-1) Edwin R. Aponte[[2]](#footnote-2) and Joel Quijano [[3]](#footnote-3)

*Department of Mechanical Engineering, University of Puerto Rico at Mayaguez*

**As part of the INME 4707 course offered at the Department of Mechanical Engineering, University of Puerto Rico at Mayaguez, we are required to model and analyze the thermal performance of a J58 Turbojet engine.**

1. **Nomenclature**

*T0* = ambient temperature

*T1* = inlet temperature

*T2*= compressor inlet temperature

*TD* = fourth stage compressor temperature

*T3* = burner inlet temperature

T4 = turbine inlet temperature

T5 = turbine exit temperature

*T6* = afterburner flame-holder temperature

*T8* = nozzle temperature

NEGT = Nominal Exhaust Gas Temperature

*z* = altitude

*M* = Mach Number

= Thrust

*QLHV* = Fuel Lower Heating Value

1. **Introduction**

This project is composed of an analysis of a turbojet engine and its components using the equations derived in class and parameters found in literature. The results obtained were compared to experimental values found in literature. The turbojet engine is the basic jet engine [1]. These engines are used extensively in military aircraft due to their supersonic flight capability. They were also used in commercial flight in airplanes such as the BAC Concorde. Turbojet engines are composed of an inlet, a compressor, a combustion chamber, a turbine, and a nozzle. The inlet reduces the velocity of the air mass before entering the engine. The air then enters the compressor, where its pressure is increased. A turbojet engine may have a single compressor or two compressors in series, a low pressure compressor (LPC) and a high pressure compressor (HPC). Then the air enters the combustion chamber where it is mixed with fuel and ignited. This causes the temperature of the mixture to increase. Next, the hot mixture enters the turbine where work is extracted from the fluid to power the compressor and the pressure of air is reduced. As with the compressor, the turbine might be a single one or it may be high pressure turbine (HPT) and a low pressure turbine (LPT) connected in series. The air-fuel mixture finally exits through the nozzle, where it is accelerated and thrust is generated. Some turbojets, specifically those used in military aircraft, may have an afterburner between the turbine and the nozzle that further increases the temperature and velocity of the air mixture. It is used to improve characteristics such as thrust at takeoff, climb, and the combat performance [ref?]. All the engine components play a part in the net thrust achievable by the engine and the efficiency of the engine.

## Problem Statement

To gain a better understanding of turbojet engines it is important to analyze the engine characteristics over a range of condition to fully grasp the capabilities of the engine. To that end, an analytical model will be developed that describes the impact of changes in component characterization on the overall performance of a turbojet engine. This will be done for a range of conditions to survey the design space.

## Background Information

The engine chosen to be analyzed for this project is the Pratt and Whittney J-58JT11D20 jet engine, commonly referred to as just the J58. This is a single spool (one compressor, one turbine) engine with an afterburner, developed from 1958 and produced from 1964 [Smithsonian ref]. It was used as the engine for the various “Blackbird” variants, the A12; YF-12; and SR-71, which were a series of stealth reconnaissance military aircraft used by the CIA and later by NASA as a research aircraft. This engine is famous for being the first with Mach 3 capabilities and broke a number of speed and altitude records [ref?].



Figure 1: Standard J11D-20 Station Nomenclature [1]

Table 1: Maximum Operating Temperatures [1] [2]

|  |  |  |
| --- | --- | --- |
| COMPONENT/STAGE | TEMP (ºF) | TEMP (ºC) |
| Inlet T1 | 800+ | 426+ |
| COMPRESSOR Inlet T2 | 800+ | 426+ |
| COMPRESSOR 4th Stage TD | 1050 | 565.56 |
| COMBUSTOR Inlet T3 | 1300 | 704.44 |
| TURBINE Inlet T4 | 2000 | 1093.33 |
| TURBINE Exit T5 | 1450 | 787.78 |
| AB T6 | 3200 | 1760 |
| Exhaust NOZZLE T8 | 1500 | 815.15 |

The JT11D-20 variant of the P&W J58 engine has several components that merit some explanation. For instance, Figure 1 depicts a Bypass Air and Secondary Air Flow; the engine behaved as a traditional afterburning turbojet from subsonic to Mach 2.2, but transitioned to a turboramjet at Mach 2.2 . Above Mach 2.2, 6 valves bypass air from the fourth compressor stage (Station D) to the afterburner thereby combining a turbojet with a compressor assisted ramjet. However, this report will limit the analysis to conditions below Mach 2.2 in order to consider the turbojet nature of the JT11D-20. The secondary airflow depicted in Figure 1 allows “descent at low airflow, low power, without unstarting the inlet.“ [3] (It is also shared with the cowl shock trap bleed as per [3].)

The JT11D-20 was designed for a wide range of operational requirements which included sub- and supersonic flight conditions and a wide range of altitudes. This versatility requires the designed to be evaluated at several conditions which are listed in Table 3. The engine must be capable of performing Buddy Missions, Recon Missions, Long Range Flight Deployments plus the typical Takeoff/Landing conditions. Additionally, the aircraft usually performed high altitude, high Mach flights, but these will not be evaluated due to the Turbo-Ramjet limitation after Mach 2.2. The majority of the flight conditions closely resemble an actual flight condition possibly experienced by an SR-71.

Table 2: Engine Specs

|  |  |  |
| --- | --- | --- |
| SPECIFICATION | VALUE RANGE [EN] | VALUE RANGE [SI] |
| **Altitude** [4] | **25K-90K ft** | **7.62 – 27.43 km** |
| **Speed** [5] | **Mach 0.75 – 3.2** | |
| **Dry TSFC @ Max Thrust** [6] | **0.8 lb/lbf hr** | **81.6 kg/kN hr** |
| **Wet TSFC @ Max Thrust** [6] | **1.9 lb/lbf hr** | **164 kg/kN hr** |
| **Fuel** [7] | **JP-7** | |
| **Fuel Storage** [8] | **80,285 lb** | **36,416 kg** |
| **Fuel Lower Heating Value** [9] | **5.48 kWh/lb** | **43,682 kJ/kg** |
| **Thrust** [7] | **32,500 lbf** | **144,567 N** |
| **Air Volume Flow @ Cruise** [10] | **100K ft3/s** | **2831.68 m3/s** |
| **Compression Ratio < Mach 2.2** [8] | **8.8:1** | |
| **Compressor** [11] | **8-Stage Axial** | |
| **Turbine** [11] | **2-Stage** | |
| **Weight** [11] | **6,500 lb** | **2,948 kg** |
| **Air Mass Flow** [8] | **326-450 lb/s** | **147 – 204 kg/s** |
| **Dry Fuel Mass Flow @ Max** | **5.55 lb/s** | **2.52 kg/s** |
| **Wet Fuel Mass Flow @ Max** | **17.94 lb/s** | **8.14 kg/s** |
| **Dry Fuel to Air Ratio** | **0.012-0.017** | |
| **Wet Fuel to Air Ratio** | **0.0398-0.055** | |

# Methodology: Model Description

Modelling a JT11D-20 requires a non-linear approach; in other terms, the engine requires the coupled equations be solved simultaneously front-to-back and back-to-front in order to better approximate the engine’s actual functioning. For instance, the nominal EGT (T8) is provided by [2] as a function of compressor inlet temperature; therefore, this parameter is fixed once T2 is determined. The inlet design is also a major factor affecting the overall model. Given the supersonic nature of the SR-71 plane, the inlet was designed to minimize the losses incurred by shock waves. The recovery factor is then nonlinear and less than one for a typical flight.

### Ambient

Atmospheric condition will be modelled using [12] based on [13] and [14] with an offset temperature approximating typical aircraft temperatures.

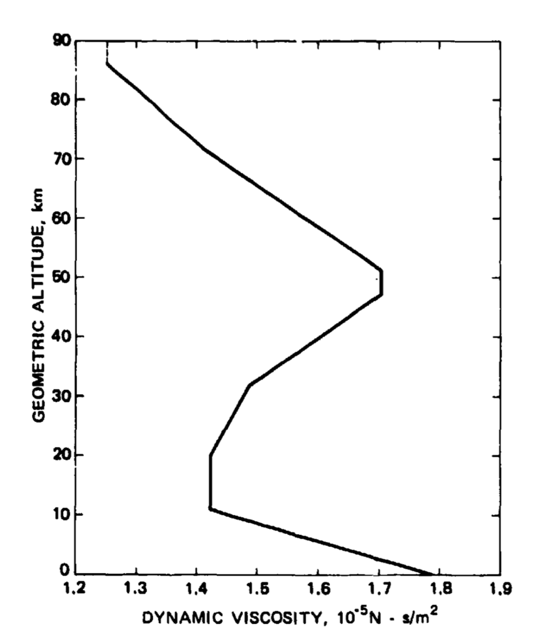


Figure 2: Standard Atmosphere Dynamic Viscosity [14]

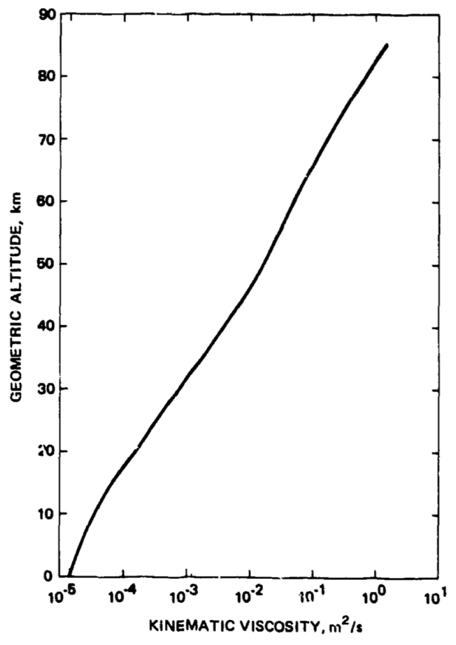


Figure 3: Standard Atmosphere Kinematic Viscosity [14]

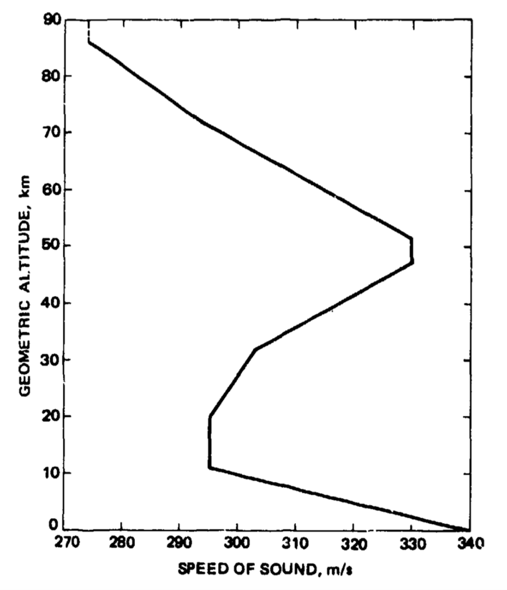


Figure 4: Standard Atmosphere Speed of Sound [14]

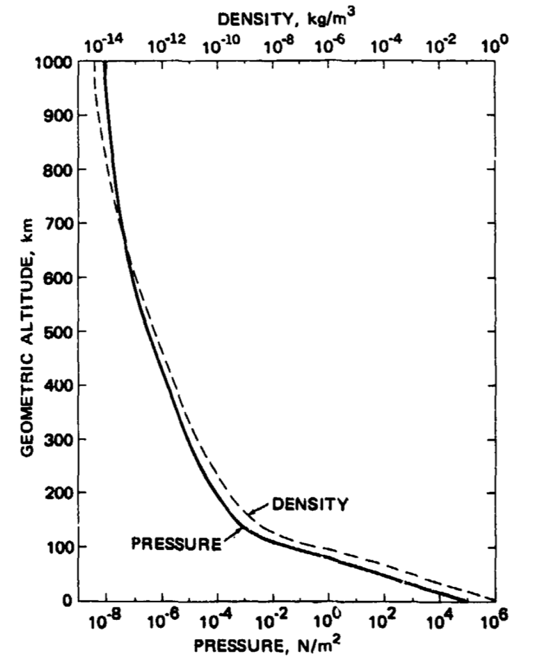


Figure 5: Standard Atmosphere Pressure [14]

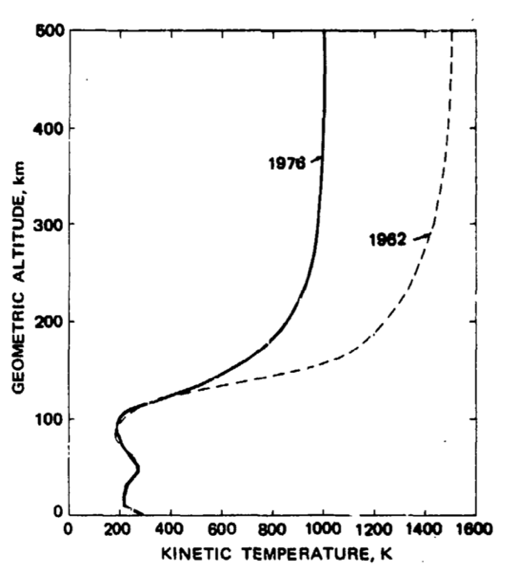


Figure 6: Standard Atmosphere Kinetic Temperature [14]

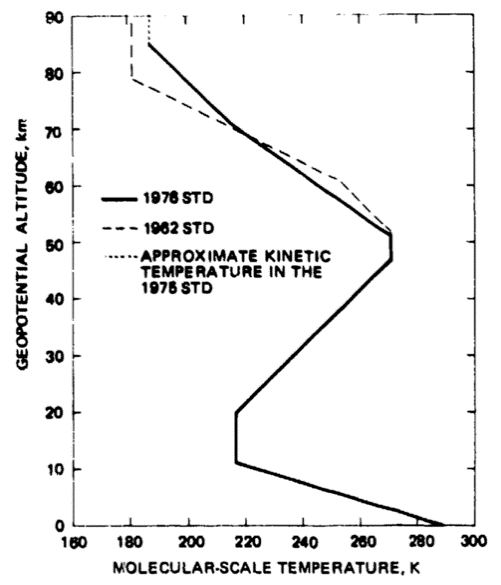


Figure 7: Standard Atmosphere Molecular-Scale Temperature [14]

### Inlet

The inlet’s recovery factor will be modelled after the more conservative curve in Figure 2.

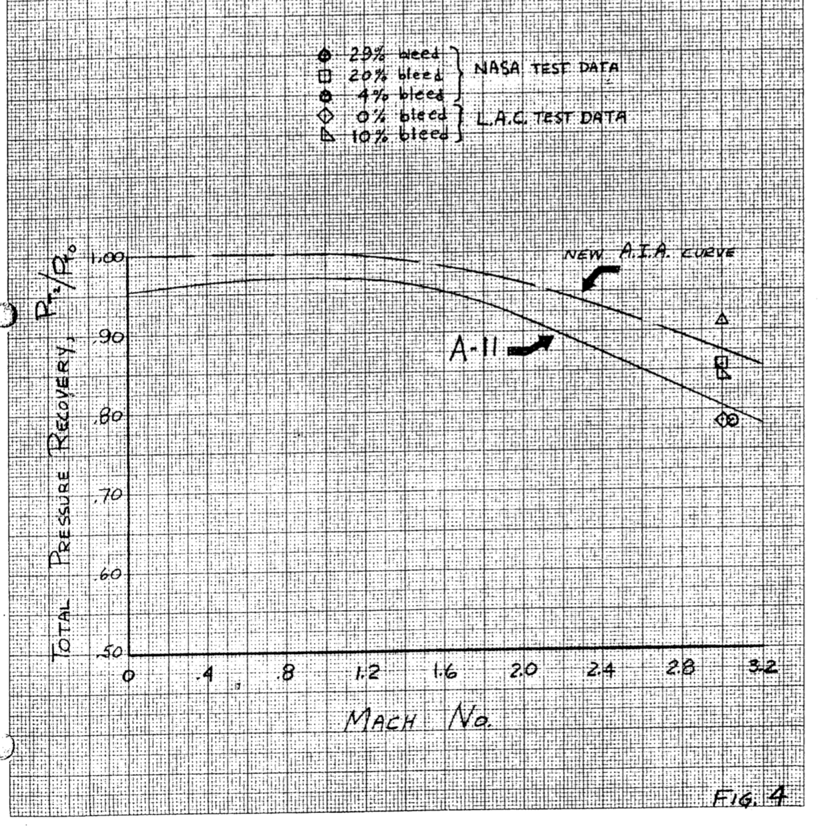


Figure 8: Expected Inlet Performance [15]

The air density will be modelled using Equation (1). The speed of sound inside the inlet is then determined as shown in Equation (2). From this change in density and the total pressure recovery, the inlet temperature (T1 or T2 for typical operations) can be determined as seen in Equation (3).

(1)

(2)

(3)

### Nominal EGT

The EGT is given by Figure 3 extracted from [2].



Figure 9: Indicated EGT vs Compressor Inlet Temperature [2]

### Compressor

As per [11], the compression ratio is typically 8.8 and will be assumed constant throughout the model.

### Burner

The JT11D-20’s burner is another source of complexity in the overall design. Albeit the main fuel consumed is JP-7, it is typically mixed with a nitrogen-based additive to promote the ignition of the stable JP-7 [8] [2]. The model assumes JP-7 to be the only fuel present; thereby treating the additive as a neglectable component per unit volume of fuel. Another major assumption presumes the turbine inlet temperature (T4) to remain constant at a maximum value.

The afterburner will be modeled as the burner, however the JP-7 additive assumption is relaxed as the fuel added to the AB is exclusively JP-7.

### Nozzle

The Nozzle’s Area is variable and is a major limiting factor in the numerical modelling still being researched. However, a density-based model was used to determine the volumetric flow at the outlet from which, knowing the speed and mass flow, one could approximate the nozzle’s area.

### Model Validation

The model will be validated at standby with maximum afterburner where a 34000 lbf is expected at 1.9 pounds of JP-7 per hour per pound of thrust generated.

### JT11D-20 Conditions

Table 3: Flight Conditions

|  |  |  |  |
| --- | --- | --- | --- |
| Condition ID | Altitude [ft] | Mach | Afterburner |
| Takeoff [2] | 0 (@ Sea level) | 0.3542 | ON |
| Refueling/Buddy Mission [2] | 25000 | 0.75 | OFF |
| Climbing [2] | 30000 | 1.25 | ON |
| Concorde [16] | 60000 | 2.00 | ON |
| YF12A (03/18/65) [17] | 65000 | 2.2 | ON |
| A12 Max Altitude at Mach 2.2 [18] | 75000 | 2.2 | ON |
| Lake County Airport [18] | 9928 | 0.3545 | ON |
| Lowest Altitude at Mach 1.0 [18] | 15000 | 1.0 | ON |
| MA139-XAA[] | 40000 | 1.9 | ON |
| French Griffon II [] | 61000 | 2.1 | ON |
| Constant Climb[5] | 33000 | 0.9 | ON |
| Supersonic Transport flight [19] | 70000 | 2.5 | OFF |

### Implementation

The model’s implementation language is Matlab and the code is being maintained in GitHub for source control facilitation.

# Results and Observations

## Pressure and Temperature Profiles

## Efficiencies

## TSFC

## Fuel Consumption

## Impulse

## Range

## Optimum Determination

## Summary

# Appendix

## Group Meetings

INME 4707 Team 1: Group Meeting|Minutes 

## Meeting date | time 04/02/18 | 10:30 | Meeting location Lucchetti

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Meeting called by | Christian Lagares | | Type of meeting | Work Division | | Facilitator | Christian Lagares | | Note taker | Christian Lagares | | Timekeeper | Edwin Aponte | | Edwin Aponte  Joel Quijano (Excused from meeting) |

# Agenda topics

## Time allotted | 15 | Agenda topic Source Control | Presenter Christian Lagares

Discuss the importance of Source Control for Code and Document Management and sharing.

Team agreed to a platform

|  |  |  |
| --- | --- | --- |
| Action items | Person responsible | Deadline |
| Create GitHub Account | Joel/Edwin | 04/03/2018 |

## Time allotted | 15 | Agenda topic Code Division | Presenter Christian Lagares

Created basic source tree

Allowed team members to choose desired code sections

|  |  |  |
| --- | --- | --- |
| Action items | Person responsible | Deadline |
| Vote on code section | All | ASAP |



Figure 10: Work distribution email

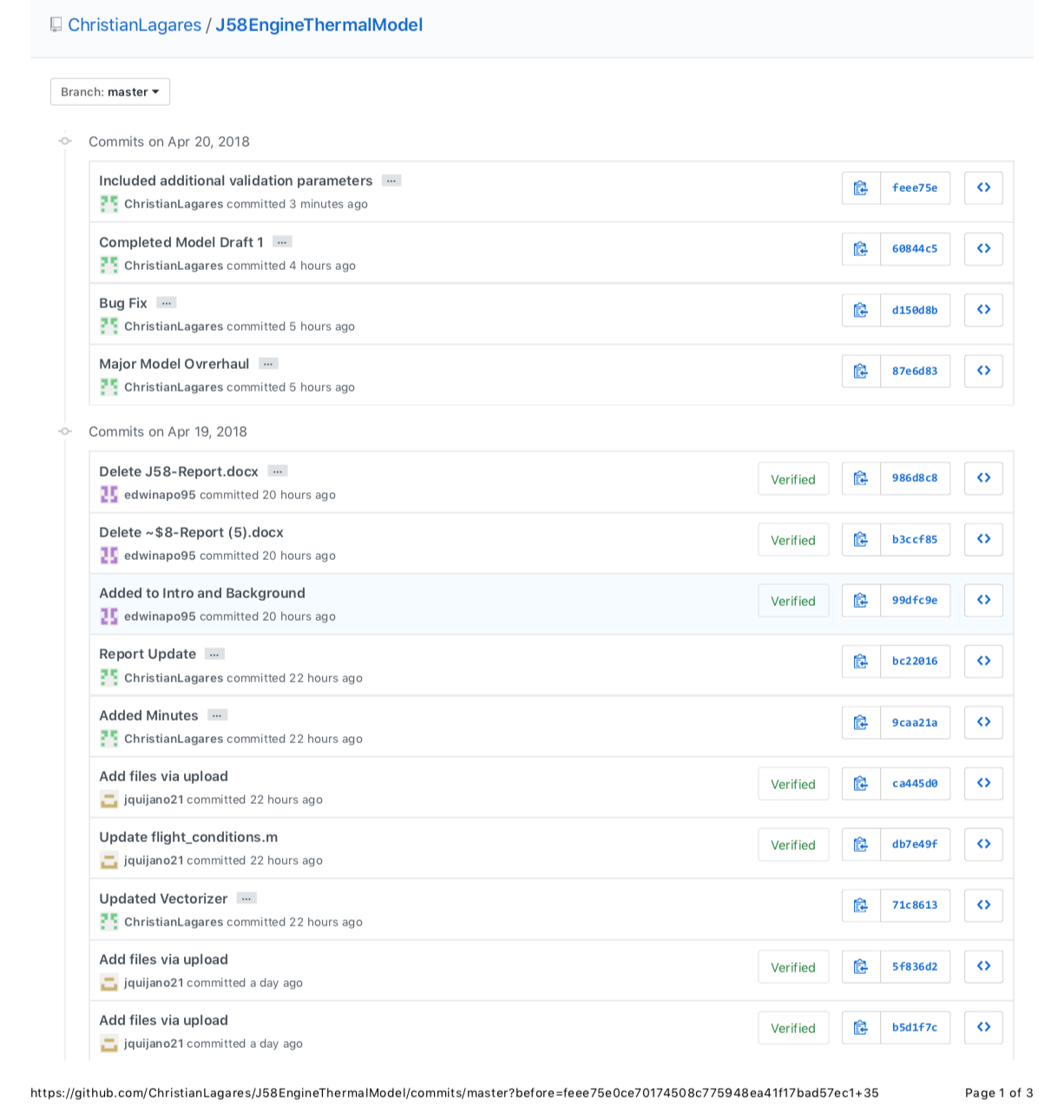


Figure 11: GitHub Commits; 1 of 5

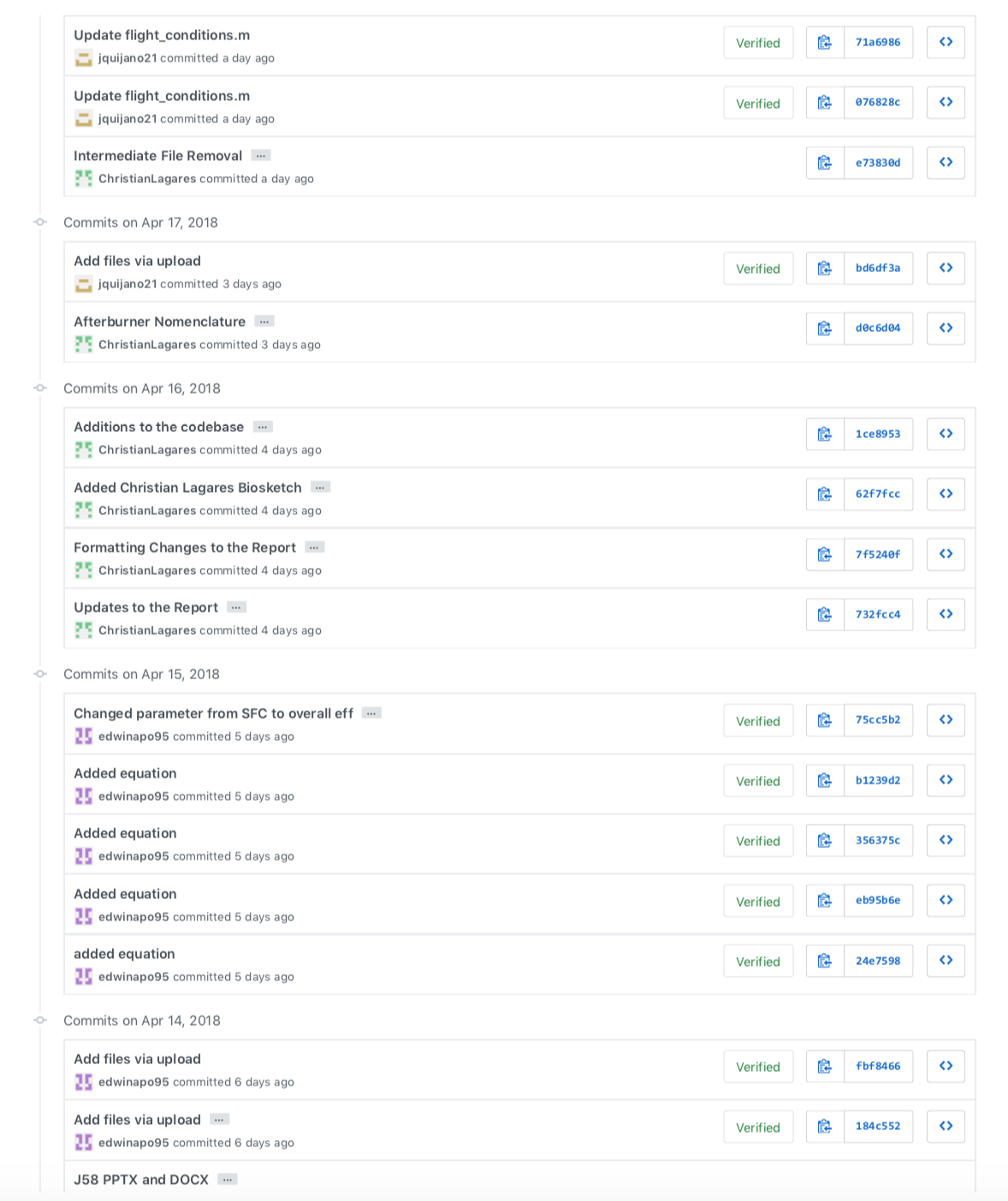


Figure 12: GitHub Commits; 2 of 5

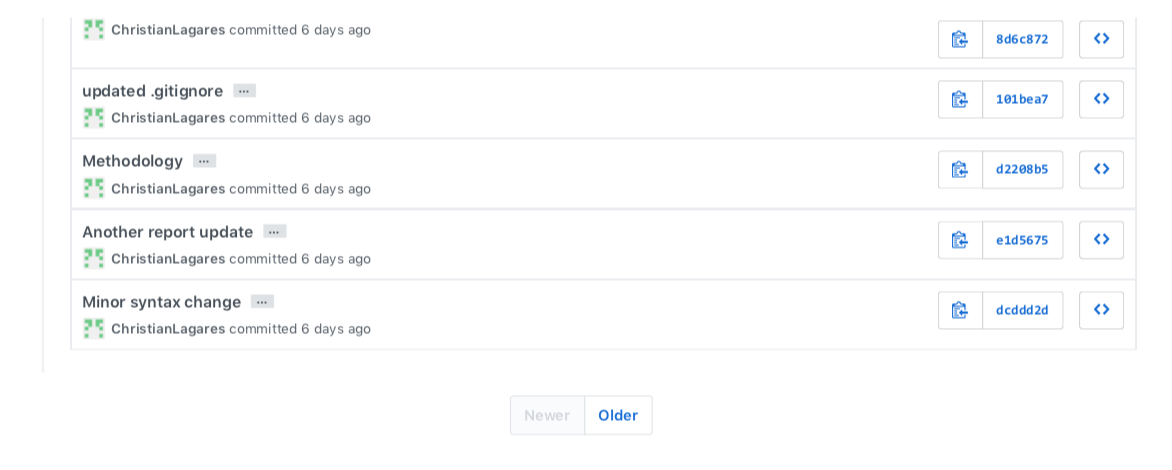


Figure 13: GitHub Commits; 3 of 5

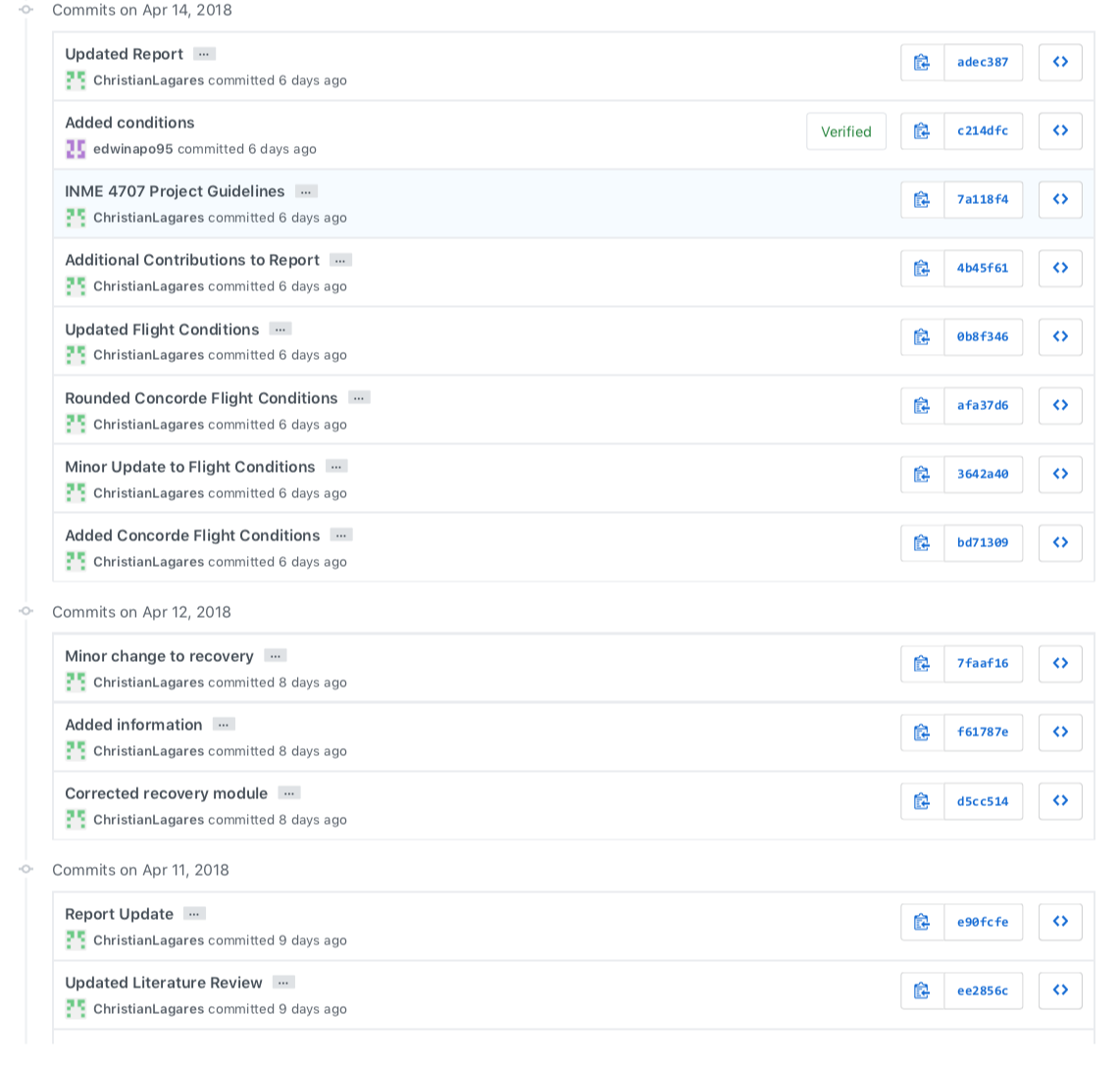


Figure 14: GitHub Commits; 4 of 5

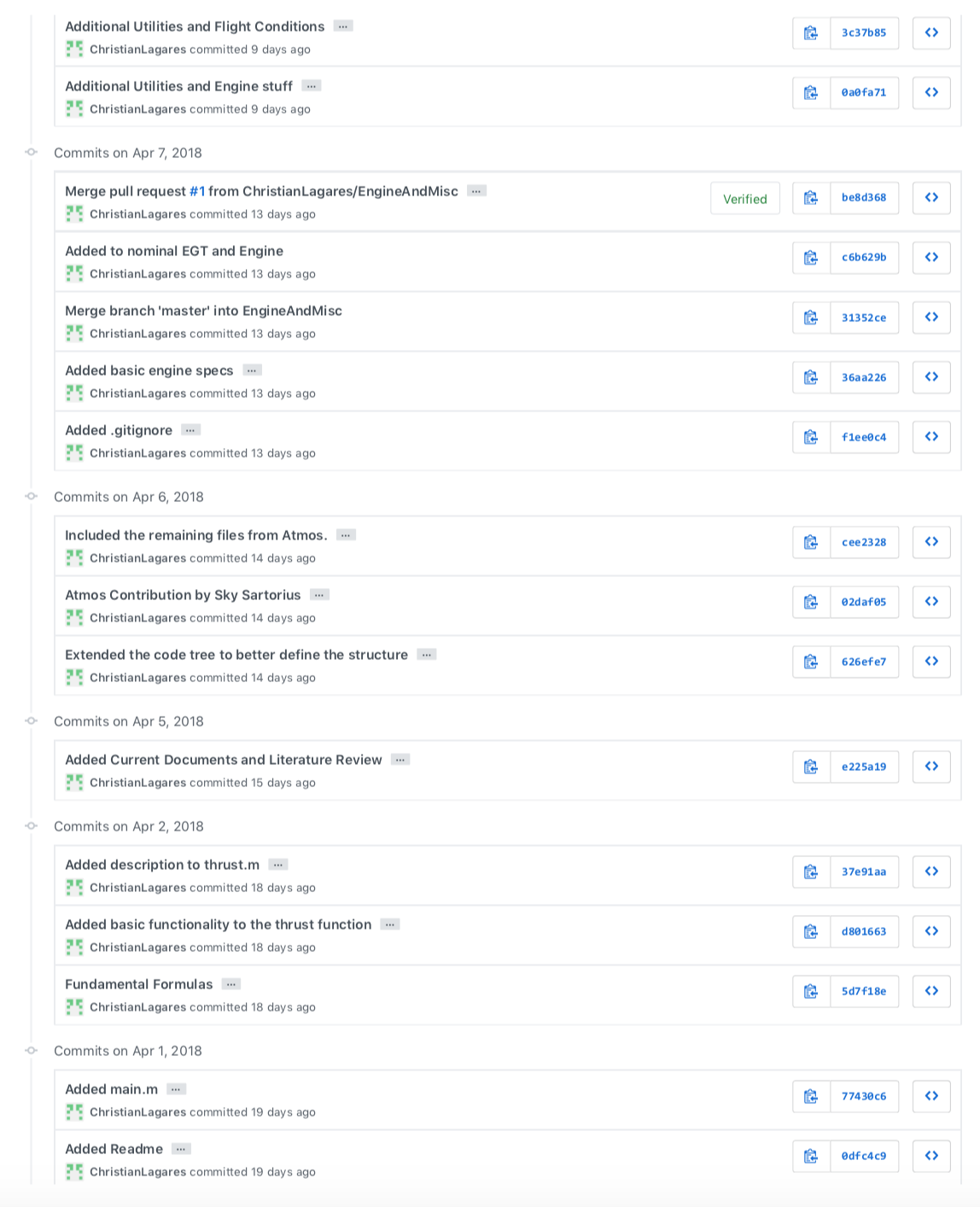


Figure 15: GitHub Commits; 5 of 5

## Model

The Codebase has been organized as follows:

* EngineModel
  + afterburner.m
  + burner.m
  + compressor.m
  + inlet.m
  + nozzle.m
  + shock\_trap.m
  + turbine.m
* PerfParameters
  + flight\_conditions.m
  + impulse.m
  + overall\_efficiency.m
  + propulsive\_efficiency.m
  + range.m
  + thermal\_efficiency.m
  + thrust.m
  + tsfc.m
* utils
  + atmos
    - atmos.m
    - densityalt.m
    - tropos.m
    - license.txt
  + EngineParameters
    - engine.m
    - recovery.m
  + FlightManualUtilities
    - knots.m
    - nominalEGT.m
* Main.m

#### Codebase - Engine Model – Inlet

%% Inlet

%

% INSERT DOC

%% CODE

function [P\_a, P\_0a, P2, T\_a, T2, mdot\_a, V\_inf] = inlet(altitude, Mach, Toffset)

% inlet - models the SR-71 supersonic inlet

eta\_d = recovery(Mach);

gamma\_c = 1.4;

air\_massflow = struct('ENGLISH',450.\*(exp(-4.27e-06.\*altitude)),...% lb\_s

'SI', 204.\*(exp(-4.34e-06.\*altitude))); % kg/s

mdot\_a = air\_massflow;

altitude = altitude\_converter(altitude, 'ft', 'm');

atmosphere = atmos(altitude, 'units', 'SI', 'structOutput', true);

P\_a = atmosphere.P;

T\_a = atmosphere.T +Toffset;

V\_inf = atmosphere.a .\* Mach;

P\_0a = P\_a + 0.5 .\* atmosphere.rho .\* (Mach .\* atmosphere.a).^2;

P2 = P\_0a.\*((1 + (((gamma\_c - 1)/2).\*(Mach.^2).\*(eta\_d))).^(gamma\_c/(gamma\_c-1)));

T2 = T\_a.\*((1 + (((gamma\_c - 1)/2).\*(Mach.^2))));

end

#### Codebase - Engine Model – Shock Trap

%% Shock Trap

%

% INSERT DOC

%% CODE

function [P2, T2, mdot\_a\_out] = shock\_trap(P2, T2, mdot\_a\_in, Flight\_Mach)

% shock\_trap

mdot\_a\_en\_out = mdot\_a\_in.ENGLISH;

mdot\_a\_si\_out = mdot\_a\_in.SI;

mdot\_a\_en\_in = mdot\_a\_in.ENGLISH;

mdot\_a\_si\_in = mdot\_a\_in.SI;

mdot\_a\_en\_out(Flight\_Mach >= 1.3) = mdot\_a\_en\_in(Flight\_Mach >= 1.3).\*0.85;

mdot\_a\_si\_out(Flight\_Mach >= 1.3) = mdot\_a\_si\_in(Flight\_Mach >= 1.3).\*0.85;

mdot\_a\_out = struct('ENGLISH', mdot\_a\_en\_out,...

'SI', mdot\_a\_si\_out);

end

#### Codebase - Engine Model – Compressor

%% Compressor

%

% INSERT DOC

%% CODE

function [P03, T03, mdot\_a, eta\_c] = compressor(P02, T02, mdot\_a)

% Compressor

eta\_c = 0.9;

gamma\_c = 1.4;

parameters = engine();

compression\_ratio = parameters.COMPRESSION\_RATIO;

P03 = compression\_ratio.\*P02;

T03 = T02.\*(1+(((compression\_ratio.^((gamma\_c-1)/gamma\_c))-1)/eta\_c));

end

#### Codebase - Engine Model – Burner

%% Burner

%

% INSERT DOC

%% CODE

function [P04, T04, fuel2air, mdot\_a, mdot\_e, mdot\_f, JP7LHV] = burner(P03, T03, mdot\_a)

% Docstring

Cph = 1.005; % [kJ/kg ∫K]

Cpc = 1.155; % [kJ/kg ∫K]

JP7LHV = 43682; % [kJ/kg]

T04 = ones(size(T03)).\*(1093.33+273.15); % [∫K] Assumption: Constant @ Nominal

eta\_b = 0.99; % Assumption: From Course Textbook

fuel2air = (Cph.\*T04 - Cpc.\*T03)./(eta\_b.\*JP7LHV - Cph.\*T04);

P04 = P03.\*(1-(1-eta\_b));

mdot\_e = struct('ENGLISH', mdot\_a.ENGLISH.\*(1+fuel2air),...

'SI', mdot\_a.SI.\*(1+fuel2air));

mdot\_f = struct('ENGLISH', mdot\_a.ENGLISH.\*(fuel2air),...

'SI', mdot\_a.SI.\*(fuel2air));

end

#### Codebase - Engine Model – Turbine

%% Turbine

%

% INSERT DOC

%% CODE

function [P05, T05] = turbine(T02, T03, P04, T04, fuel2air)

% Docstring

Cph = 1.005; % [kJ/kg ∫K]

Cpc = 1.155; % [kJ/kg ∫K]

losses = 0.90; % Assumed Shaft Losses

eta\_t = 0.95;

gamma\_h = 1.33;

T05 = T04.\*(1 - ((((Cpc./Cph).\*T02)./(losses.\*(1+fuel2air).\*T04))).\*((T03./T02) - 1));

P05 = P04.\*(1 - ((1./eta\_t).\*(1-(T05./T04)))).^(gamma\_h./(gamma\_h-1));

end

#### Codebase - Engine Model – Afterburner

%% Afterburner

%

% INSERT DOC

%% CODE

function [P06, T06, mdot\_e2, mdot\_f2, fuel2air] = afterburner(P05, T05, AB, mdot\_e1)

% Docstring

Cph2 = 1.155; % [kJ/kg ∫K]

Cph1 = 1.268; % [kJ/kg ∫K]

JP7LHV = 43682; % [kJ/kg]

T06 = ones(size(T05));

T06(AB == 0) = T05(AB == 0);

T06(AB == 1) = (1760+273.15); % [∫K] Assumption: Constant @ Nominal

eta\_b = 0.99; % Assumption: From Course Textbook

fuel2air = (Cph2.\*T06 - Cph1.\*T05)./(eta\_b.\*JP7LHV - Cph2.\*T06);

P06 = P05;

P06(AB == 1) = P05(AB == 1).\*(1-(1-eta\_b));

tmp1 = zeros(size(T05));

tmp1(AB == 1) = mdot\_e1.SI(AB == 1).\*(1+fuel2air(AB == 1));

tmp2 = zeros(size(T05));

tmp2(AB == 1) = mdot\_e1.SI(AB == 1).\*(1+fuel2air(AB == 1));

mdot\_e2 = struct('ENGLISH', tmp1,...

'SI', tmp2);

tmp1 = zeros(size(T05));

tmp1(AB == 1) = mdot\_e1.ENGLISH(AB == 1).\*(fuel2air(AB == 1));

tmp2 = zeros(size(T05));

tmp2(AB == 1) = mdot\_e1.SI(AB == 1).\*(fuel2air(AB == 1));

mdot\_f2 = struct('ENGLISH', tmp1,...

'SI', tmp2);

end

#### Codebase - Engine Model – Nozzle

%% Nozzle

%

% INSERT DOC

%% CODE

function [P8, T8, V8] = nozzle(P06, T06, AB, T02, P\_a)

% Docstring

eta\_n = 0.98;

R = 287;

T8 = nominalEGT(T02 - 273.15)+273.15;

Pc = zeros(size(P06));

P8 = zeros(size(P06));

V8 = zeros(size(P06));

gamma\_h = 1.33;

Cph = 1.155;

% Inoperant AB

Pc(AB == 0) = P06(AB == 0).\*((1 - (1/eta\_n).\*(gamma\_h-1)./(gamma\_h+1)).^(gamma\_h./(gamma\_h-1)));

P8(AB == 0 & Pc >= P\_a) = Pc(AB == 0 & Pc >= P\_a);

P8(AB == 0 & Pc < P\_a) = P\_a(AB == 0 & Pc < P\_a);

V8(AB == 0 & Pc >= P\_a) = sqrt(gamma\_h .\* R .\* T8(AB == 0 & Pc >= P\_a));

V8(AB == 0 & Pc < P\_a) = sqrt(((2.\*gamma\_h.\*eta\_n.\*R.\*...

T06(AB == 0 & Pc < P\_a))./(gamma\_h-1)).\*...

(1-((P\_a(AB == 0 & Pc < P\_a)./...

P06(AB == 0 & Pc < P\_a)).^((gamma\_h-1)./gamma\_h))));

% Operative AB

Pc(AB == 1) = P06(AB == 1).\*((1 - (1/eta\_n).\*(gamma\_h-1)./(gamma\_h+1)).^(gamma\_h./(gamma\_h-1)));

P8(AB == 1 & Pc >= P\_a) = Pc(AB == 1 & Pc >= P\_a);

P8(AB == 1 & Pc < P\_a) = P\_a(AB == 0 & Pc < P\_a);

V8(AB == 1 & Pc >= P\_a) = sqrt(gamma\_h .\* R .\* T8(AB == 1 & Pc >= P\_a));

V8(AB == 1 & Pc < P\_a) = sqrt(2.\*Cph.\*eta\_n.\*T06(AB == 1 & Pc < P\_a).\*...

(1-((P\_a(AB == 1 & Pc < P\_a)./P06(AB == 1 & Pc < P\_a)).^((gamma\_h-1)./gamma\_h))));

end

#### Codebase – Performance Parameters – Flight Conditions

%% Flight Conditions

%

% INSERT DOC

%% CODE

function [conditions] = flight\_conditions()

% Altitude: [ft]

% Mach as per Atmos

% Validation Condition 0

Condition0 = struct('altitude', 0, ...

'Mach', 0.0,...

'Afterburner', 1);

% Takeoff Appendix 2-3

Condition1 = struct('altitude', 0, ...

'Mach', 0.3542,...

'Afterburner', 1);

% Refueling A3-3 & Buddy Mission A4-2

Condition2 = struct('altitude', 25000, ...

'Mach', 0.75,...

'Afterburner', 0);

% Climbing A3-3

Condition3 = struct('altitude', 30000, ...

'Mach', 1.25,...

'Afterburner', 1);

% Concorde Flight Conditions

Condition4 = struct('altitude', 60000, ...

'Mach', 2.00,...

'Afterburner', 1);

% YF12A Record Flight 03/18/65

Condition5 = struct('altitude', 65000, ...

'Mach', 2.2,...

'Afterburner', 1);

% A12 Max. Altitude at M=2.2

Condition6 = struct('altitude', 75500, ...

'Mach', 2.2,...

'Afterburner', 1);

% Takeoff at high altitude airstrip (Lake County Airport)

Condition7 = struct('altitude', 9928, ...

'Mach', 0.3545,...

'Afterburner', 1);

% Lowest operating altitude at M=1.0

Condition8 = struct('altitude', 15000, ...

'Mach', 1.00,...

'Afterburner', 1);

% MA139-XAA

Condition9 = struct('altitude', 40000, ...

'Mach', 1.9,...

'Afterburner', 1);

% French Griffon II

Condition10 = struct('altitude', 61000, ...

'Mach', 2.1,...

'Afterburner', 1);

% Constant climb

Condition11 = struct('altitude', 33000, ...

'Mach', 0.9,...

'Afterburner', 1);

% Supersonic transport flight

Condition12 = struct('altitude', 70000,...

'Mach', 2.5,...

'Afterburner', 1);

conditions = struct('Condition0', Condition0,...

'Condition1', Condition1,...

'Condition2', Condition2,...

'Condition3', Condition3,...

'Condition4', Condition4,...

'Condition5', Condition5,...

'Condition6', Condition6,...

'Condition7', Condition7,...

'Condition8', Condition8,...

'Condition9', Condition9,...

'Condition10', Condition10,...

'Condition11', Condition11,...

'Condition12', Condition12);

end

#### Codebase – Performance Parameters – Condition Vectorizer

%% Internal Condition Vectorization

%

%

%% Code

function [altitude, Mach, Afterburner] = condition\_vectorizer()

% Docstring

FlightConditions = flight\_conditions();

altitude = [FlightConditions.Condition0.altitude,...

FlightConditions.Condition1.altitude,...

FlightConditions.Condition2.altitude,...

FlightConditions.Condition3.altitude,...

FlightConditions.Condition4.altitude,...

FlightConditions.Condition5.altitude,...

FlightConditions.Condition6.altitude,...

FlightConditions.Condition7.altitude,...

FlightConditions.Condition8.altitude,...

FlightConditions.Condition9.altitude,...

FlightConditions.Condition10.altitude,...

FlightConditions.Condition11.altitude,...

FlightConditions.Condition12.altitude];

Mach = [FlightConditions.Condition0.Mach,...

FlightConditions.Condition1.Mach,...

FlightConditions.Condition2.Mach,...

FlightConditions.Condition3.Mach,...

FlightConditions.Condition4.Mach,...

FlightConditions.Condition5.Mach,...

FlightConditions.Condition6.Mach,...

FlightConditions.Condition7.Mach,...

FlightConditions.Condition8.Mach,...

FlightConditions.Condition9.Mach,...

FlightConditions.Condition10.Mach,...

FlightConditions.Condition11.Mach,...

FlightConditions.Condition12.Mach];

Afterburner = [FlightConditions.Condition0.Afterburner,...

FlightConditions.Condition1.Afterburner,...

FlightConditions.Condition2.Afterburner,...

FlightConditions.Condition3.Afterburner,...

FlightConditions.Condition4.Afterburner,...

FlightConditions.Condition5.Afterburner,...

FlightConditions.Condition6.Afterburner,...

FlightConditions.Condition7.Afterburner,...

FlightConditions.Condition8.Afterburner,...

FlightConditions.Condition9.Afterburner,...

FlightConditions.Condition10.Afterburner,...

FlightConditions.Condition11.Afterburner,...

FlightConditions.Condition12.Afterburner];

end

#### Codebase – Performance Parameters – Impulse

%% Impulse

%

% INSERT DOC

%% CODE

function [output] = impulse(air\_massflow,air2fuel,thrust)

% Docstring

output = air\_massflow.\*air2fuel./thrust

end

#### Codebase – Performance Parameters – Propulsive Efficiency

%% Propulsive Efficiency

%

%

%% CODE

function [output] = propulsive\_efficiency(air\_massflow,air2fuel,exhaust\_velocity,flight\_velocity,thrust)

% Docstring

efprop\_num = flight\_velocity.\*thrust;

efprop\_den = efprop\_num + .5.\*air\_massflow.\*(1+air2fuel).\*(exhaust\_velocity-flight\_velocity).^2;

output= efprop\_num./efprop\_den;

end

#### Codebase - Performance Parameters – Thermal Efficiency

%% Thermal Efficiency

%

% INSERT DOC

%% CODE

function [output] = thermal\_efficiency(air\_massflow,air2fuel,...

exhaust\_velocity,flight\_velocity,...

LHV)

% Docstring

efther\_num = (thrust.\*flight\_velocity)+0.5\*air\_massflow.\*...

(1+air2fuel).\*(exhaust\_velocity-flight\_velocity).^2;

efther\_den = air\_massflow.\*air2fuel.\*LHV;

output = efther\_num./efther\_den;

end

#### Codebase - Performance Parameters – Overall Efficiency

%% overall\_effiency

%

% INSERT DOC

%% CODE

function [output] = overall\_effiency(propulsive\_efficiency,...

thermal\_efficiency)

% Docstring

output = propulsive\_efficiency.\*thermal\_efficiency;

end

#### Codebase - Performance Parameters – Overall Efficiency

%% Range

%

% INSERT DOC

%% CODE

function [output] = range()

% Docstring

end

#### Codebase – Performance Parameters – Thrust

%% Thrust

%

% Thrust force is considered an important performance parameter for any

% turbojet engine.

%

%% CODE

function [thrust] = thrust(air\_massflow, air2fuel, exhaust\_velocity,...

flight\_velocity, atmospheric\_pressure, exhaust\_pressure, exhaust\_area)

% thrust computes the thrust force for multiple flight conditions. Inputs

% should be floating point vectors with one entry per flight condition.

pthrust = (exhaust\_pressure - atmospheric\_pressure).\*exhaust\_area;

vthrust = air\_massflow.\*((1+air2fuel).\*exhaust\_velocity + flight\_velocity);

thrust = pthrust + vthrust;

end

#### Codebase – Performance Parameters - TSFC

%% TSFC

%

% INSERT DOC

%% CODE

function [output] = tsfc(air\_massflow,air2fuel,thrust)

% Docstring

output = air\_massflow.\*air2fuel./thrust;

end

#### Codebase – Utils – Atmos – License

Copyright (c) 2010, Sky Sartorius

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are

met:

\* Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in

the documentation and/or other materials provided with the distribution

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

POSSIBILITY OF SUCH DAMAGE.

#### Codebase – Utils – Atmos – Atmos

function varargout = atmos(h,varargin)

% ATMOS Find gas properties in the 1976 Standard Atmosphere.

% [rho,a,T,P,nu,z] = ATMOS(h,varargin)

%

% ATMOS by itself gives atmospheric properties at sea level on a standard day.

%

% ATMOS(h) returns the properties of the 1976 Standard Atmosphere at

% geopotential altitude h, where h is a scalar, vector, matrix, or ND array.

%

% The input h can be followed by parameter/value pairs for further control of

% ATMOS. Possible parameters are:

% tOffset - Returns properties when the temperature is tOffset degrees

% above or below standand conditions. h and tOffset must be

% the same size or else one must be a scalar. Default is no

% offset. Note that this is an offset, so when converting

% between Celsius and Fahrenheit, use only the scaling factor

% (dC/dF = dK/dR = 5/9).

% tAbsolute - Similar to tOffest, but an absolute air temperature is

% provided (∞K or ∞R) instead of an offset from the standard

% temperature. Supersedes tOffset if both are provided.

% altType - Specify type of input altitude, either 'geopotential' (h)

% or 'geometric' (z). Default altType = 'geopotential'.

% structOutput - When set, ATMOS produces a single struct output with fields

% rho, a, T, P, nu, and either z or h (whichever complements

% input altType). Default structOutput = false.

% units - String for units of inputs and outpus, either 'SI'

% (default) or 'US'. This is ignored if the provided input h

% is a DimVar, in which case all outputs are also DimVars and

% expected tOffset is either a DimVar or in ∞C/∞K.

% Description: SI: US:

% Input: -------------- ----- -----

% h | z Altitude or height m ft

% tOffset Temp. offset ∞C/∞K ∞F/∞R

% Output: -------------- ----- -----

% rho Density kg/m^3 slug/ft^3

% a Speed of sound m/s ft/s

% T Temperature ∞K ∞R

% P Pressure Pa lbf/ft^2

% nu Kinem. viscosity m^2/s ft^2/s

% z | h Height or altitude m ft

%

% ATMOS returns properties the same size as h and/or tOffset (P does not vary

% with temperature offset and is always the size of h).

%

% Example 1: Find atmospheric properties at every 100 m of geometric height

% for an off-standard atmosphere with temperature offset varying +/- 25∞C

% sinusoidally with a period of 4 km.

% z = 0:100:86000;

% [rho,a,T,P,nu,h] = atmos(z,'tOffset',25\*sin(pi\*z/2000),...

% 'altType','geometric');

% semilogx(rho/atmos,h/1000)

% title('Density variation with sinusoidal off-standard atmosphere')

% xlabel('\sigma'); ylabel('Geopotential altitude (km)')

%

% Example 2: Create tables of atmospheric properties up to 30,000 ft for a

% cold (-20∞C), standard, and hot (+20∞C) day with columns

% [h(ft) z(ft) rho(slug/ft≥) sigma a(ft/s) T(R) P(psf) µ(slug/ft-s) nu(ft≤/s)]

% leveraging n-dimensional array capability.

% [~,h,dT] = meshgrid(0,-5000:1000:30000,[-20 0 20]);

% [rho,a,T,P,nu,z] = atmos(h,'tOffset',dT\*9/5,'units','US');

% t = [h z rho rho/atmos(0,'units','US') a T P nu.\*rho nu];

% format short e

% varNames = {'h' 'z' 'rho' 'sigma' 'a' 'T' 'P' 'mu' 'nu'};

% ColdTable = array2table(t(:,:,1),'VariableNames',varNames)

% StandardTable = array2table(t(:,:,2),'VariableNames',varNames)

% HotTable = array2table(t(:,:,3),'VariableNames',varNames)

%

% Example 3: Use the unit consistency enforced by the DimVar class to find the

% SI dynamic pressure, Mach number, Reynolds number, and stagnation

% temperature of an aircraft flying at flight level FL500 (50000 ft) with

% speed 500 knots and characteristic length of 80 inches.

% V = 500\*u.kts; c = 80\*u.in;

% o = atmos(50\*u.kft,'structOutput',true);

% Dyn\_Press = 1/2\*o.rho\*V^2;

% M = V/o.a;

% Re = V\*c/o.nu;

% T0 = o.T\*(1+(1.4-1)/2\*M^2);

%

% This model is not recommended for use at altitudes above 86 km geometric

% height (84852 m / 278386 ft geopotential) but will attempt to extrapolate

% above 86 km (with a lapse rate of 0∞/km) and below 0.

%

% See also ATMOSISA, ATMOSNONSTD, TROPOS,

% DENSITYALT - http://www.mathworks.com/matlabcentral/fileexchange/39325,

% UNITS - http://www.mathworks.com/matlabcentral/fileexchange/38977.

%

% [rho,a,T,P,nu,z] = ATMOS(h,varargin)

% Copyright 2015 Sky Sartorius

% www.mathworks.com/matlabcentral/fileexchange/authors/101715

%

% References: ESDU 77022; www.pdas.com/atmos.html

defaultUnits = 'SI'; % Alternate: 'US'

defaultStructOutput = false;

%% Parse inputs:

if nargin == 0

h = 0;

end

if nargin <= 1 && ~nnz(h)

% Quick return of sea level conditions.

rho = 1.2250000;

a = 340.293988026089;

temp = 288.15;

press = 101325;

kvisc = 1.46071857273722e-05;

ZorH = 0;

if isa(h,'DimVar')

rho = rho\*u.kg/(u.m^3);

if nargout == 1

varargout = {rho};

return

end

a = a\*u.m/u.s;

temp = temp\*u.K;

press = press\*u.Pa;

kvisc = kvisc\*u.m^2/u.s;

ZorH = ZorH\*u.m;

end

varargout = {rho,a,temp,press,kvisc,ZorH};

return

end

validateattributes(h,{'DimVar' 'numeric'},{'finite' 'real'});

p = inputParser;

addParameter(p,'tOffset',0,@(x)validateattributes(x,{'DimVar','numeric'},...

{'finite' 'real'}));

addParameter(p,'tAbsolute',[],@(x)validateattributes(x,{'DimVar','numeric'},...

{'finite' 'real' 'positive'}));

addParameter(p,'units',defaultUnits);

addParameter(p,'altType','geopotential');

addParameter(p,'structOutput',defaultStructOutput,...

@(x)validateattributes(x,{'numeric','logical'},{'scalar'}));

parse(p,varargin{:});

tOffset = p.Results.tOffset;

tAbsolute = p.Results.tAbsolute;

convertUnits = strcmpi('US',validatestring(p.Results.units,...

{'US' 'SI'},'atmos','units'));

% Flag if I need to convert to/from SI.

geomFlag = strcmpi('geometric',validatestring(p.Results.altType,...

{'geopotential' 'geometric'},'atmos','altType'));

% Flag specifying z provided as input.

structOutput = p.Results.structOutput;

%% Deal with different input types:

dimVarOut = false;

if isa(h,'DimVar')

h = h/u.m;

dimVarOut = true;

convertUnits = false; % Trumps specified units.

end

if isa(tOffset,'DimVar')

tOffset = tOffset/u.K;

% It is allowed to mix DimVar h\_in and double tOffset (or reverse).

end

if isa(tAbsolute,'DimVar')

tAbsolute = tAbsolute/u.K;

end

if convertUnits

h = h \* 0.3048;

tOffset = tOffset \* 5/9;

tAbsolute = tAbsolute \* 5/9;

end

%% Constants, etc.:

% Lapse rate Base Temp Base Geop. Alt Base Pressure

% Ki (∞C/m) Ti (∞K) Hi (m) P (Pa)

D =[-0.0065 288.15 0 101325 % Troposphere

0 216.65 11000 22632.0400950078 % Tropopause

0.001 216.65 20000 5474.87742428105 % Stratosphere1

0.0028 228.65 32000 868.015776620216 % Stratosphere2

0 270.65 47000 110.90577336731 % Stratopause

-0.0028 270.65 51000 66.9385281211797 % Mesosphere1

-0.002 214.65 71000 3.9563921603966 % Mesosphere2

0 186.94590831019 84852.0458449057 .373377173762337];% Mesopause

% Constants:

R = 287.05287; %N-m/kg-K; value from ESDU 77022

% R = 287.0531; %N-m/kg-K; value used by MATLAB aerospace toolbox ATMOSISA

gamma = 1.4;

g0 = 9.80665; %m/sec^2

RE = 6356766; %Radius of the Earth, m

Bs = 1.458e-6; %N-s/m2 K1/2

S = 110.4; %K

K = D(:,1); %∞K/m

T = D(:,2); %∞K

H = D(:,3); %m

P = D(:,4); %Pa

%% Convert from geometric altitude to geopotental altitude, if necessary.

if geomFlag

hGeop = (RE\*h) ./ (RE + h);

else

hGeop = h;

end

%% Calculate temperature and pressure:

% Pre-allocate.

temp = zeros(size(h));

press = temp;

nSpheres = size(D,1);

for i = 1:nSpheres

% Put inputs into the right altitude bins:

if i == 1 % Extrapolate below first defined atmosphere.

n = hGeop <= H(2);

elseif i == nSpheres % Capture all above top of defined atmosphere.

n = hGeop > H(nSpheres);

else

n = hGeop <= H(i+1) & hGeop > H(i);

end

if K(i) == 0 % No temperature lapse.

temp(n) = T(i);

press(n) = P(i) \* exp(-g0\*(hGeop(n)-H(i))/(T(i)\*R));

else

TonTi = 1 + K(i)\*(hGeop(n) - H(i))/T(i);

temp(n) = TonTi\*T(i);

press(n) = P(i) \* TonTi.^(-g0/(K(i)\*R)); % Undefined for K = 0.

end

end

%% Switch between using standard temp and provided absolute temp.

if isempty(tAbsolute)

% No absolute temperature provided - use tOffset.

temp = temp + tOffset;

else

temp = tAbsolute;

end

%% Populate the rest of the parameters:

rho = press./temp/R;

a = sqrt(gamma \* R \* temp);

kvisc = (Bs \* temp.^1.5 ./ (temp + S)) ./ rho; %m2/s

if geomFlag % Geometric in, ZorH is geopotential altitude (H)

ZorH = hGeop;

else % Geop in, find Z

ZorH = RE\*hGeop./(RE-hGeop);

end

%% Process outputs:

if dimVarOut

rho = rho\*u.kg/(u.m^3);

a = a\*u.m/u.s;

temp = temp\*u.K;

press = press\*u.Pa;

kvisc = kvisc\*u.m^2/u.s;

ZorH = ZorH\*u.m;

elseif convertUnits

rho = rho / 515.3788;

a = a / 0.3048;

temp = temp \* 1.8;

press = press / 47.88026;

kvisc = kvisc / 0.09290304;

ZorH = ZorH / 0.3048;

end

varargout = {rho,a,temp,press,kvisc,ZorH};

if structOutput

if geomFlag

ZorHname = 'h';

else

ZorHname = 'z';

end

names = {'rho' 'a' 'T' 'P' 'nu' ZorHname};

varargout = {cell2struct(varargout,names,2)};

end

end

#### Codebase – Utils – Atmos – Tropos

function [rho,a,temp,press,kvisc]=tropos(h\_in,tOffset)

% TROPOS Stripped-down version of atmos, applicable only to the troposphere

% (covers the vast majority of atmospheric flight), for when computation speed

% is a priority.

%

% [rho,a,T,P,nu] = TROPOS(h)

% [rho,a,T,P,nu] = TROPOS(h,dT)

%

% See also ATMOS.

if nargin < 2

tOffset = 0;

end

if nargin < 1

h\_in = 0;

end

dimVarOut = false;

if isa(h\_in,'DimVar')

h\_in = h\_in/u.m;

dimVarOut = true;

end

if isa(tOffset,'DimVar')

tOffset = tOffset/u.K;

% It is allowed to mix DimVar h\_in and double tOffset (or reverse).

end

% h\_in(h\_in>11000 | h\_in<0) = NaN;

TonTi=1-2.255769564462953e-005\*h\_in;

press=101325\*TonTi.^(5.255879812716677);

temp = TonTi\*288.15 + tOffset;

rho = press./temp/287.05287;

a = sqrt(401.874018 \* temp);

kvisc = (1.458e-6 \* temp.^1.5 ./ (temp + 110.4)) ./ rho;

if dimVarOut

rho = rho\*u.kg/(u.m^3);

a = a\*u.m/u.s;

temp = temp\*u.K;

press = press\*u.Pa;

kvisc = kvisc\*u.m^2/u.s;

end

#### Codebase – Utils – Engine Parameters – Engine

%% Engine Specs

%

% Basic engine parameters are provided in a structure.

%

% engine\_specs:

%

% \* WET\_THRUST

% \* DRY\_THRUST

% \* SFC

% \* AFTERBURNER\_FUEL\_perHour

% \* NO\_AFTERBURNER\_FUEL\_perHour

% \* AFTERBURNER\_FUEL\_perSecond

% \* NO\_AFTERBURNER\_FUEL\_perSecond

%% CODE

function [engine\_specs] = engine()

% Published Specs

WET\_THRUST = 34000; % lbf

DRY\_THRUST = 25000; % lbf

WET\_SFC = 1.9; % lb/lbf\*hr

DRY\_SFC = 0.8; % lb/lbf\*hr

CORE\_AIRFLOW = 450; % lb

COMPRESSION\_RATIO = 8.8;

WET\_FUEL\_hr = WET\_SFC \* WET\_THRUST; % lb/hr

DRY\_FUEL\_hr = DRY\_SFC \* DRY\_THRUST; % lb/hr

WET\_FUEL\_sec = WET\_FUEL\_hr/3600; % lb/sec

DRY\_FUEL\_sec = DRY\_FUEL\_hr/3600; %lb/sec

engine\_specs = struct('WET\_THRUST', WET\_THRUST,...

'DRY\_THRUST', DRY\_THRUST,...

'WET\_FUEL\_perHour', WET\_FUEL\_hr,...

'DRY\_FUEL\_perHour', DRY\_FUEL\_hr,...

'WET\_FUEL\_perSecond', WET\_FUEL\_sec,...

'DRY\_FUEL\_perSecond', DRY\_FUEL\_sec,...

'DRY\_SFC', DRY\_SFC,...

'WET\_SFC', WET\_SFC,...

'CORE\_AIRFLOW', CORE\_AIRFLOW,...

'COMPRESSION\_RATIO', COMPRESSION\_RATIO);

end

#### Codebase – Utils – Engine Parameters – Recovery

%% Recover vs Mach

%% Code

function [ratio] = recovery(Mach)

% Docstring

ratio = 0.004105.\*(Mach.^4)+...

-0.02213.\*(Mach.^3)+...

0.0006246.\*(Mach.^2)+...

0.03737.\*(Mach) + 0.951;

end

#### Codebase – Utils – Flight Manual Utilities – Altitude Converter

%% Altitude Converter

%

%

%% Code

function [length] = altitude\_converter(z, given\_units, required\_units)

% Docstring

given\_units = lower(given\_units);

required\_units = lower(required\_units);

if given\_units == 'ft'

if required\_units == 'm'

length = z .\* 0.3;

elseif required\_units == 'km'

length = (z .\* 0.3)./1000;

end

elseif given\_units == 'm'

if required\_units == 'ft'

length = z .\* 3.281;

elseif required\_units == 'km'

length = z./1000;

end

elseif given\_units == 'km'

z = z .\* 1000;

if required\_units == 'm'

length = z;

else

length = altitude\_converter(z, 'm', required\_units);

end

end

end

#### Codebase – Utils – Flight Manual Utilities – Knots

%% knots

%% Code

function speed = knots(x, units)

% Docstring

if units == 'ft/s'

speed = x .\* 1.69;

elseif units == 'mph'

speed = x .\* 1.15;

elseif units == 'm/s'

speed = x .\* 0.51;

elseif units == 'km/h'

speed = x .\* 1.85;

end

end

#### Codebase – Utils – Flight Manual Utilities – Nominal Exhaust Gas Temperature

%% Nominal Exhaust Gas Temperature

%

%

%% CODE

function egt = nominalEGT(CompressorInletTemperature)

assert(all(CompressorInletTemperature) > -50, 'Off Limits');

egt = zeros(size(CompressorInletTemperature));

egt(CompressorInletTemperature <= 10) = 570 + ((810 - 570)/(10-(-50)))...

\*(CompressorInletTemperature(CompressorInletTemperature <= 10)+50);

egt(CompressorInletTemperature > 10 & CompressorInletTemperature < 50)=...

820 + ((780-810)/(50-10))\*...

(CompressorInletTemperature(CompressorInletTemperature > 10 ...

& CompressorInletTemperature < 50) - 10);

egt(CompressorInletTemperature >= 50) = 795;

end

#### Codebase – Main

%% Main Routine

%

% This file contains the main routine for the J58 Thermal Model.

%

% All other files are required to be in the same directory.

%% Routine Body

%

% Pressures [Pa]

% Temperatures [∫K]

% mdot\_a.ENGLISH [lb/s]

% mdot\_a.SI [kg/s]

% Generating Condition Vectors

[altitude, Mach, AB] = condition\_vectorizer();

% Inlet Model

[P\_a, P0A, P02, T\_a, T02, mdot\_a\_nonbleed, V\_inf] = inlet(altitude, Mach, 0);

% Shock Trap Bleed Model

[P02, T02, mdot\_a] = shock\_trap(P02, T02, mdot\_a\_nonbleed, Mach);

% Compressor Model

[P03, T03, mdot\_a, eta\_c] = compressor(P02, T02, mdot\_a);

% Burner Model

[P04, T04, fuel2air, mdot\_a, mdot\_e1, mdot\_f, LHV] = burner(P03, T03, mdot\_a);

% Turbine Model

[P05, T05] = turbine(T02, T03, P04, T04, fuel2air);

% Afterburner Model

[P06, T06, mdot\_e2, mdot\_f2, ab\_fuel2air] = afterburner(P05, T05, AB, mdot\_e1);

total\_fuel2air = struct('ENGLISH', (mdot\_f2.ENGLISH+mdot\_f.ENGLISH)./mdot\_a.ENGLISH,...

'SI', (mdot\_f2.SI+mdot\_f.SI)./mdot\_a.SI);

%total

% Nozzle Model

[P8, T8, V8] = nozzle(P06, T06, AB, T02, P\_a);

fprintf('V\_inf\tV\_ext\tMach\tProportion\n')

for ii = [1:12]

fprintf('%3.2f\t%3.2f\t%3.2f\t%3.2f\n',V\_inf(ii), V8(ii), Mach(ii), V8(ii)/V\_inf(ii))

end

%% Postprocessing

%

% Nozzle Area

rho\_nozzle = P8./(287.\*T8); % kg/m^3

total\_mdot = struct('ENGLISH', (mdot\_a\_nonbleed.ENGLISH + mdot\_f.ENGLISH...

+ mdot\_f2.ENGLISH),...

'SI', (mdot\_a\_nonbleed.SI + mdot\_f.SI + mdot\_f2.SI));

nozzle\_area = (mdot\_a\_nonbleed.SI + mdot\_f.SI + mdot\_f2.SI)./...

(rho\_nozzle + V8); % m^2

% Thrust

[thrust\_SI] = thrust(mdot\_a.SI, total\_fuel2air.SI, ...

V8,V\_inf, P\_a, P8, nozzle\_area); % Newtons

thrust\_ENGLISH = 0.224808943.\* thrust\_SI; % lbf

% Propulsive Efficiency

prop\_efficiency = (V\_inf.\*thrust\_SI)./((V\_inf.\*thrust\_SI)...

+ 0.5.\*total\_mdot.SI.\*((V8 - V\_inf).^2));

% Thermal Efficiency

thermal\_efficiency = ((thrust\_SI.\*V8) + ...

(0.5.\*mdot\_a.SI.\*(1+total\_fuel2air.SI).\*((V8-V\_inf).^2)))./...

((LHV.\*1000).\*(mdot\_f2.SI+mdot\_f.SI));

% Overall Efficiency

overall\_efficiency = prop\_efficiency .\* thermal\_efficiency;

%% Viz

%

figure('Name','PressureVStation')

plot([P\_a; P0A; P02; P03; P04; P05; P06; P8])

xlabel('Station')

ylabel('P [Pa]')

legend('Validation','Takeoff', 'Refueling\_Buddy', 'Climbing', 'Concorde',...

'YF12A', 'A12Max', 'Takeoff\_High', 'LowestM1',...

'MA139XAA', 'FrenchGriffon2', 'ConstantClimb', 'Out\_Of\_Model')

figure('Name','TemperatureVStation')

plot([T\_a; T\_a; T02; T03; T04; T05; T06; T8])

xlabel('Station')

ylabel('T [∫K]')

legend('Validation','Takeoff', 'Refueling\_Buddy', 'Climbing', 'Concorde',...

'YF12A', 'A12Max', 'Takeoff\_High', 'LowestM1',...

'MA139XAA', 'FrenchGriffon2', 'ConstantClimb', 'Out\_Of\_Model')

figure('Name','ThermalEfficiencyVV\_inf')

scatter(V\_inf', thermal\_efficiency')

xlabel('V\_inf [m/s]')

ylabel('\eta\_t')

figure('Name','PropulsiveEfficiencyVV\_inf')

scatter(V\_inf', prop\_efficiency')

xlabel('V\_inf [m/s]')

ylabel('\eta\_p')

figure('Name','OverallEfficiencyVV\_inf')

scatter(V\_inf', overall\_efficiency')

xlabel('V\_inf [m/s]')

ylabel('\eta\_o')

## Biosketch

Christian Lagares is currently an undergraduate student at the Department of Mechanical Engineering at the University of Puerto Rico at Mayaguez and an Artificial Intelligence/Machine Learning Researcher at SIL Technologies, LLC. His main research interests include Supervised Learning Strategies for Advanced Signal Analysis, Real Time Systems for Simultaneous DAQ/Processing in low power portable devices and AI-Enabled Materials.

# References

|  |  |
| --- | --- |
| [1] | P. Law, *SR-71 Propulsion System P&W J58 Engine (JT11D-20),* 2013. |
| [2] | *SR-71 Flight Manual,* Norton, CA: Norton, AFB, 1986. |
| [3] | J. T. Anderson, "How Supersonic Inlets Work: Details of the Geometry and Operation of the SR-71 Mixed Compression Inlet," Lockheed Martin Corporation, 2013. |
| [4] | C. L. Johnson, "Development of the Lockheed SR-71 Blackbird," *Lockheed Horizons,* 1982. |
| [5] | T. R. Conners, "Predicted Performance of a Thrust- Enhanced SR-71 Aircraft with an External Payload," *NASA Technical Memorandum 104330,* 1997. |
| [6] | Jet Engine Specification Database, "Military Turbojet/Turbofan Specifications," [Online]. Available: http://www.jet-engine.net/miltfspec.html. [Accessed 11 04 2018]. |
| [7] | Atomic Toaster, "A Look at the Pratt & Whitney J-58JT11D-20," 22 August 2012. [Online]. Available: http://atomictoasters.com/2012/08/a-look-at-the-pratt-whitney-j-58jt11d-20/. [Accessed 17 03 2018]. |
| [8] | R. H. Graham, SR-71 Revealed: The Untold Story, Osceola, WI: Zenith Imprint, 1996. |
| [9] | Coordinating Research Council, Inc., "Handbook of Aviation Fuel Properties (CRC Report No. 530)," Society of Automotive Engineers, Inc., Warrendale, PA, 1983. |
| [10] | P. W. Merlin, "Design and Development of the Blackbird: Challenges and Lessons Learned," in *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, Orlando, FL, 2009. |
| [11] | National Air and Space Museum, "Pratt & Whitney J58 (JT11D-20) Turbojet Engine," [Online]. Available: https://airandspace.si.edu/collection-objects/pratt-whitney-j58-jt11d-20-turbojet-engine. [Accessed 17 03 2018]. |
| [12] | S. Sartotius, "Standard Atmosphere," 3 07 2017. [Online]. Available: https://github.com/sky-s/standard-atmosphere. [Accessed 06 04 2018]. |
| [13] | Public Domain Aeronautical Software, "Properties Of The U.S. Standard Atmosphere 1976," 09 07 2017. [Online]. Available: http://www.pdas.com/atmos.html. |
| [14] | NOAA; NASA; USAF, "U.S. Standard Atmosphere, 1976," NOAA, Washington, D.C., 1976. |
| [15] | R. F. Boehme and et.al., "Proposal - A-11 - Appendix," Lockheed Aircraft Corporation California Division, 1959. |
| [16] | U. K. Saha, *Jet Propulsion: The Concorde Aircraft,* Guwahiti, India: Indian Institute of Technology Guwahiti. |
| [17] | L. Haynes, "Lockheed YF12A," [Online]. Available: http://www.sr71.us/yf12~1.htm. [Accessed 14 04 2018]. |
| [18] | "A12 Flight Manual with Technical Data Change," 1968. |
| [19] | T. R. Conners, "Predicted Performance of a Thrust-Enhanced SR-71 Aircraft With an External Payload," , 1995. [Online]. Available: https://ntrs.nasa.gov/search.jsp?r=19970019923. [Accessed 24 3 2018]. |
| [20] | P. Ricco, "The heart of the SR-71 "Blackbird": The Mighty J-58 engine". |

1. Undergraduate Researcher, Bubble Dynamics Laboratory & SIL Technologies. [↑](#footnote-ref-1)
2. Undergraduate Student, Department of Mechanical Engineering, University of Puerto Rico at Mayaguez. [↑](#footnote-ref-2)
3. Undergraduate Student, Department of Mechanical Engineering, University of Puerto Rico at Mayaguez. [↑](#footnote-ref-3)