CONCEPTUAL MODELLING OF CIVIL AND MILITARY AERO-ENGINES

Master of Science Thesis in Applied Mechanics

SUBRAMANYAM NATARAJAN

Department of Applied Mechanics,
Division of Fluid Dynamics,
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG, SWEDEN, 2013
Master's thesis 2013:14

MASTER'S THESIS 2013:14

CONCEPTUAL MODELLING OF CIVIL AND MILITARY AERO-ENGINES

Master of Science Thesis in Applied Mechanics

SUBRAMANYAM NATARAJAN

Department of Applied Mechanics

Division of Fluid Dynamics,

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden, 2013

Numerical modelling of a turbofan engine components and
optimization of its running parameters.

Master of Science Thesis in Applied Mechanics
SUBRAMANYAM NATARAJAN

Master's Thesis 2013:14
ISSN : 1652-8557
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
Fan of IAE V2500-A1 engine fitted to a Thomas Cook Airlines Airbus A320
Image © 2008, SempreVolando, http://en.wikipedia.org/wiki/User:SempreVolando
Used under the Creative Commons Attribution 3.0 Unported license.

Chalmers Reposervice Göteborg,
Sweden 2013

Dedicated to my parents

Contents

Acknowledgements

I would like to thank the Department of Applied Mechanics, Division of Fluid Dynamics, Chalmers University of Technology, for providing this opportunity to pursue my master thesis work. I also thank Konstantinos Kyprianidis for letting me use the simulation software EVA to help during development of the optimizer code and his continuous support through mentoring.

Special thanks to Vinay Shastry of Copyleft Solutions for his support with advanced C programming.

Finally I am ever grateful to my supervisor Tomas Grönstedt for his immense support and guidance since the very beginning of this thesis work.

Abstract

Statistical research has been performed on available turbofan engine configurations up to 1995 and documented by Grieb. A mathematical model is developed as a part of this thesis work to utilize the research data into predicting the parameters for 2015 and 2020 engines. This thesis work mainly focuses on modelling the polytropic efficiencies of the various components that constitute a 3-spool turbofan engine: fan, intermediate pressure compressor, high pressure compressor, high pressure turbine, intermediate pressure turbine and the low pressure turbine.

An optimizer code is developed in the C programming language as a second part of this project to feed the modelled values as input to the gas turbine analysis software EVA. It then varies the design parameters like inlet mass flow, velocity ratio, pressure ratios and turbine inlet temperature continuously between two arbitrary fixed points. The specific fuel consumption at mid-cruise conditions is read from the output of EVA. Installed SFC is then calculated by taking nacelle inputs from Weico (a program to calculate engine dimensions) to get the minimal value. Design parameters at this value of installed SFC are returned into the input of EVA. Successive iterations are performed to get a global minimum value of installed SFC.

Nomenclature

$\bar{\psi}$	Average stage loading parameter
η_{pol}	Polytropic Efficiency
γ	Ratio of specific heats
μ	Dynamic viscosity of air
π	Pressure Ratio
ρ	Density of air
C	Cruise velocity
C_d	Drag coefficient
d_{nac}	Nacelle diameter
EIS	Entry Into Service
HPC	High Pressure Compressor
HPT	High Pressure Turbine
IPC	Intermediate Pressure Compressor
IPT	Intermediate Pressure Turbine
l	Nacelle length
LPC	Low Pressure Compressor
LPT	Low Pressure Turbine
M	Mass flow
M_0	Mach number
m_{eng}	Mass of engine
P	Pressure
R	Gas constant
Re	Reynolds Number
RNI	Reynolds Number Index
RPM	Revolutions Per Minute
S	Exposed surface area
SFC	Specific Fuel Consumption
T	Temperature
TOC	Top Of Climb
U	Blade speed at mid-area

1 Introduction

Gas turbine engines are widely used for their high power to weight ratio and specific output. Power can be harnessed out of a gas turbine engine either as thrust or as torque at a shaft. Applications are seen as power sources for aircraft, helicopters, marine engines and power generating stations. Considering its application as an aero-engine, the gas turbine engine can be classified into four types: Turbojet, Turbofan, Turboprop and the Ramjet. Turbojet engines are relatively low mass flow, high specific power engines finding applications in military fighters and other supersonic aircraft. Turbofan engines are high mass flow high power engines and are employed in civilian aircraft of medium to high capacity. These engines are designed to cruise just below the sonic limit. Turboprop engines host a propeller. These are employed in micro-light to light civilian aircraft. The ramjet is an engine which involves no moving parts. This engine is used in tandem with usually a turbojet engine, and is used to accelerate the aircraft beyond supersonic speeds.

The turbofan engine is again classified, based on its bypass ratio into high bypass and low bypass ratio turbofans. The civil aircraft use the high bypass ratio turbofan engine as its power source due to its high mass flows.

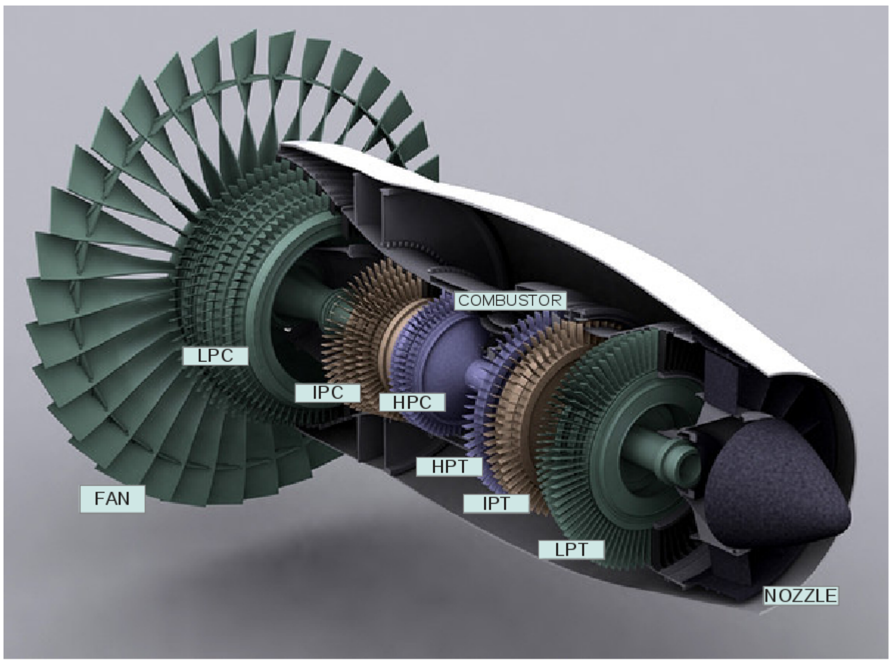


Figure 1.1: Parts of a turbofan engine.

Depending on the size and capacity of an aircraft, a suitable engine is designed to cater to peak requirements of power at take-off and top of climb and good fuel economy at mid cruise. The life cycle of an engine starts with conceptual modelling and heads to designing, manufacturing, testing and serial production.

The input to the conceptual design phase is the layout of the engine and its physical limitations like maximum pressure ratio and maximum temperature of any component. Using various models, optimal operating conditions are derived and these conditions serve as input to the design phase. In the design phase, solid modelling and analysis is done on the geometry of the components considering manufacturing feasibility. Expensive technology, materials and processes are used to manufacture the components and design verification and production validation tests. Tests for functionality are performed in the testing phase before an engine is commissioned. The below graphic illustrates the cost incurred during the different phases of product design.

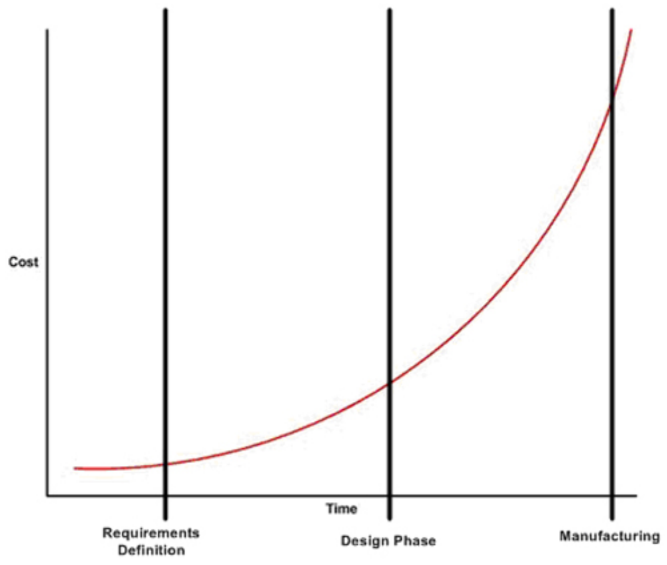


Figure 1.2: Costs incurred through the design process. [6]

It is clear from this illustration that any small design changes performed earlier in the design process incurs a smaller cost compared to changes downstream. Conceptual modelling is almost the first stage of the design phase and thus of the highest importance. This thesis work focuses on conceptual modelling alone and provides models built on statistical data to determine the polytropic efficiency of various components of a turbofan engine. The realistic values are then plugged into a performance analyser code along with starting values of operating conditions like pressure ratio, maximum temperature and nacelle size. The analyzer code used in this case is EVA, written in Fortran 2003 [5]. The specific fuel consumption is calculated and optimized varying other parameters to get a minimum value using an optimizer algorithm.

2 Modelling

Before starting the task of modelling, polytropic efficiencies for various engine components are obtained from previous experience. However, it ignores aerodynamic developments and also fails to include corrections due to them. Grieb has performed an extensive study on the various correction factors that could influence the initial assumption for these efficiencies and this study helps derive a strong starting point. His work is based on statistical data collected up to the year 1995 for various engine configurations.

We will be considering the data for civil turbofan engines and building mathematical models for each of the engine components, to extrapolate this data for the future. The output of these models will be the polytropic efficiency of each component. The various correction factors that are taken into consideration are indicated by:

$$\eta_{pol} = \eta_{pol}^{***} + \Delta\eta_{pol,EIS} - \Delta\eta_{pol,Re} - \Delta\eta_{pol,M} \tag{2.1}$$

η_{pol}^{***}= normalized efficiency
$\Delta\eta_{pol,EIS}$= Entry into service correction
$\Delta\eta_{pol,Re}$= Reynolds number correction
$\Delta\eta_{pol,M}$= Mass flow correction

This code is calibrated for 1995 Entry into service engines. Further improvement through aerodynamic progresses is captured through the $\Delta\eta_{pol,EIS}$ term. Changes in efficiencies due to the effect of Reynolds Number Index is modelled through $\Delta\eta_{pol,Re}$. Effects of change in efficiencies due to size variation not associated with Reynolds Number effects is captured by the $\Delta\eta_{pol,M}$. Thus in the above equation, for a 1995 engine with RNI=1.0 and M=70 kg/s,

$$\eta_{pol} = \eta_{pol}^{***} \tag{2.2}$$

The data from the print is read using the open source software 'Digitizer' and converted into digital data for creating the models.

2.1 Normalized efficiency

The term η_{pol}^{***} in the above equation is dependent on the aerodynamic loading of the turbomachinery component. For fans, the model relates pressure ratio to normalized efficiency whereas for the other turbomachinery average stage loading is used. This stage loading parameter is given by:

$$\bar{\psi} = \frac{2\Delta H}{\Sigma_{all\,rotors} U^2_{at\,mid\,area}} \tag{2.3}$$

where ΔH is the specific work for the respective turbomachinery component. The blade speeds should be taken at mid area as it is the average value.

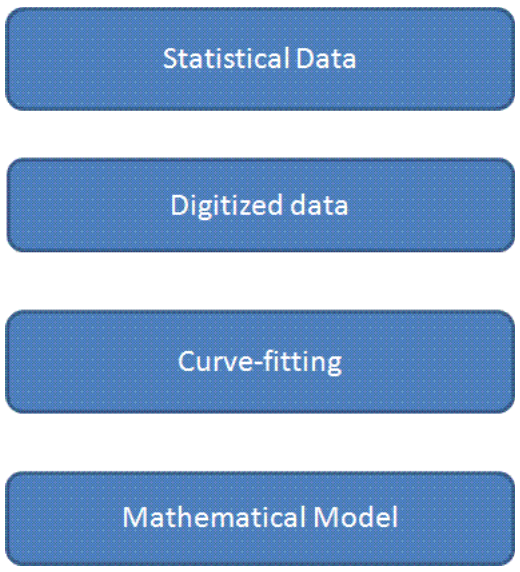


Figure 2.1: Modelling flow chart

2.2 Entry Into Service Correction

Aerodynamic, raw material and manufacturing developments are captured by the Entry Into Service (EIS) term. Based on this model, EIS correction for a 1995 engine is zero.

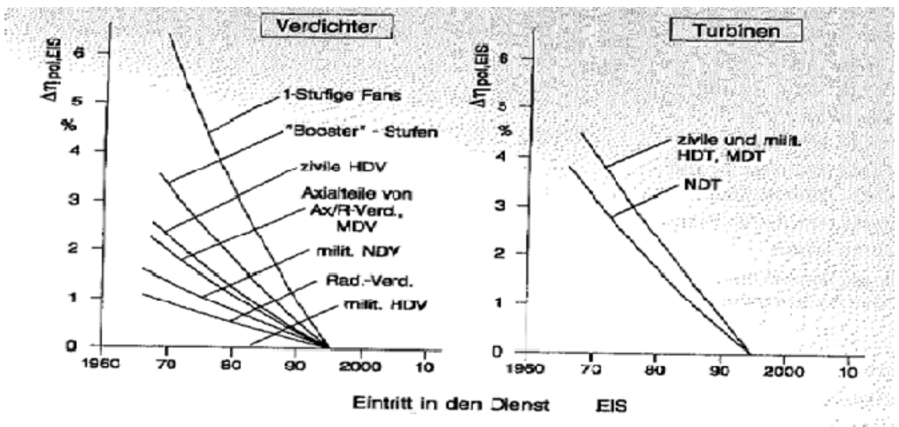


Bild 5.2.1.4: Normierung der polytropen Wirkungsgrad bei M_{korr} 70 kg/s auf EIS = 1995

Figure 2.2: EIS v/s $\Delta\eta_{pol,EIS}$ [1]

2.3 Mass Flow Correction

The effects of size on boosters and intermediate pressure compressors are captured by the Mass Flow correction term $\Delta\eta_{pol,M}$. It is captured for various engine configurations in the statistical study done by Grieb.

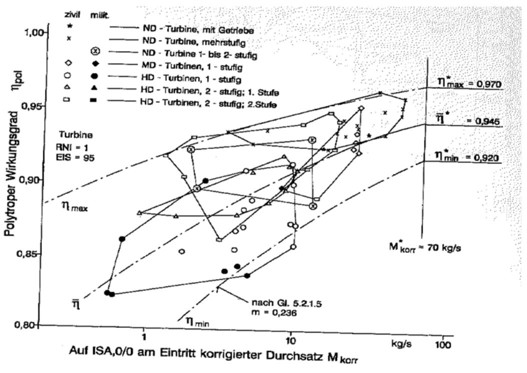


Figure 2.3: Mass flow v/s $\Delta\eta_{pol}$ [1]

The M_{corr} term in the models for the various components is defined as:

$$M_{corr} = \frac{M\sqrt{\frac{T_{0,entrance}}{288.15}}}{\sqrt{\frac{P_{0,entrance}}{101325}}} \tag{2.4}$$

$M^*_{corr} = 70\ kg/s$. The above correction must hence be applied to components with $M_{corr} < 70\ kg/s$. For larger components, $M_{corr} = 0$ is used. The need for having a size correction other than the Reynolds number term indicated below, is because the following factors are considered in the size correction term:

- Blade manufacturing tolerances and profile surface quality in relation to size
- Turbomachinery blade aspect ratios.
- Relative size of blade tip gaps.
- Aerodynamic quality of end walls.

2.4 Reynolds Number Index Correction

Reynolds Number Index is the ratio of actual Reynolds number to a reference Reynolds number at a constant Mach number [?].

$$RNI = \frac{Re}{Re_{ref}} = \frac{\frac{\rho LU}{\mu}}{\frac{\rho_{ref}L_{ref}U_{ref}}{\mu_{ref}}} \tag{2.5}$$

Since there are no length changes between actual and reference conditions, $L = L_{ref}$. Density is given by:

$$\rho = \frac{P_s}{R * T_s} \tag{2.6}$$

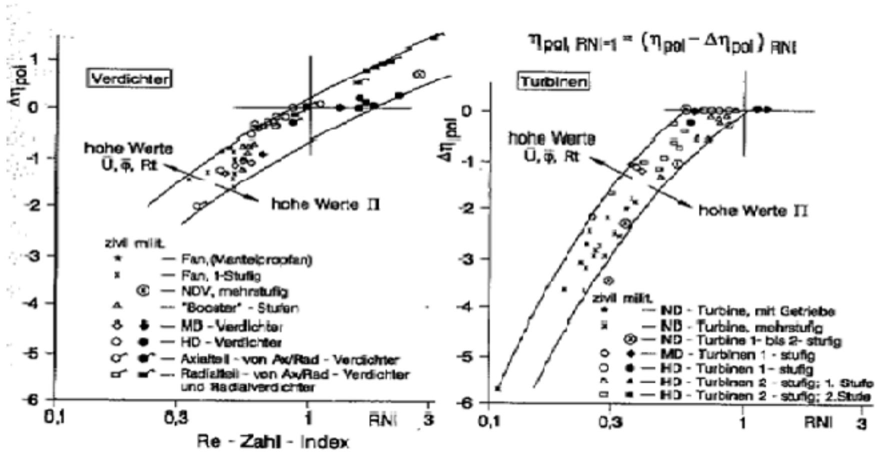


Figure 2.4: RNI v/s $\Delta\eta_{pol}$ [1]

P_s is the static pressure.
T_s is the static temperature.
R is the gas constant.

Substituting,

$$RNI = \frac{P_s}{R * T_s} * \frac{R_{ref} * T_{s,ref}}{P_{s,ref}} * \frac{V}{V_{ref}} * \frac{\mu_{ref}}{\mu} \tag{2.7}$$

$$RNI = \frac{P_s}{P_{s,ref}} * \frac{V}{\sqrt{\gamma * R * T_s}} * \frac{\sqrt{\gamma}}{\sqrt{R * T_s}} * \frac{\sqrt{\gamma_{ref} * R_{ref} * T_{s,ref}}}{V_{ref}} * \frac{\sqrt{R_{ref} * T_{s,ref}}}{\sqrt{\gamma_{ref}}} * \frac{\mu_{ref}}{\mu} \tag{2.8}$$

RNI compares conditions at the same Mach number:

$$M = \frac{V}{\sqrt{\gamma * R * T_s}} = \frac{V_{ref}}{\sqrt{\gamma_{ref} * R_{ref} * T_{s,ref}}} \tag{2.9}$$

$$\implies RNI = \frac{P_s}{P_{s,ref}} * \sqrt{\frac{T_{s,ref}}{T_s}} * \sqrt{\frac{R_{ref}}{R} * \frac{\gamma}{\gamma_{ref}}} * \frac{\mu_{ref}}{\mu} \tag{2.10}$$

Introducing total pressure P_t and total temperature T_t,

$$RNI = \frac{P_s/P_t}{P_{s,ref}/P_{t,ref}} * \frac{P_t}{P_{t,ref}} * \sqrt{\frac{T_{s,ref}/T_{t,ref}}{T_s/T_t} * \frac{T_{t,ref}}{T_t}} * \sqrt{\frac{R_{ref}}{R} * \frac{\gamma}{\gamma_{ref}}} * \frac{\mu_{ref}}{\mu} \tag{2.11}$$

Since the Mach number is the same, the ratio of static and total pressures and temperatures are the same for actual and reference conditions. Thus the ratio of actual and reference Reynolds numbers becomes:

$$RNI = \frac{P_t}{P_{t,ref}} * \sqrt{\frac{R_{ref} * T_{t,ref}}{R * T_t}} * \frac{\mu_{ref}}{\mu} \tag{2.12}$$

Reference conditions are taken as $P_{t,ref} = 101.325kPa$, $T_{t,ref}288.15K$ and $R_{ref} = 288J/(kg * K)$. Thus at ISA sea level conditions, $RNI = 1$.

2.5 The Sigmoid

The statistical study conducted by Grieb is time based. Like many natural processes, they start with small beginnings that accelerates and approaches a steady state over time. To model this, the Sigmoid curve is best suited. The general form is represented by:

$$P(x) = \frac{A}{B + e^{Cx+D}} \tag{2.13}$$

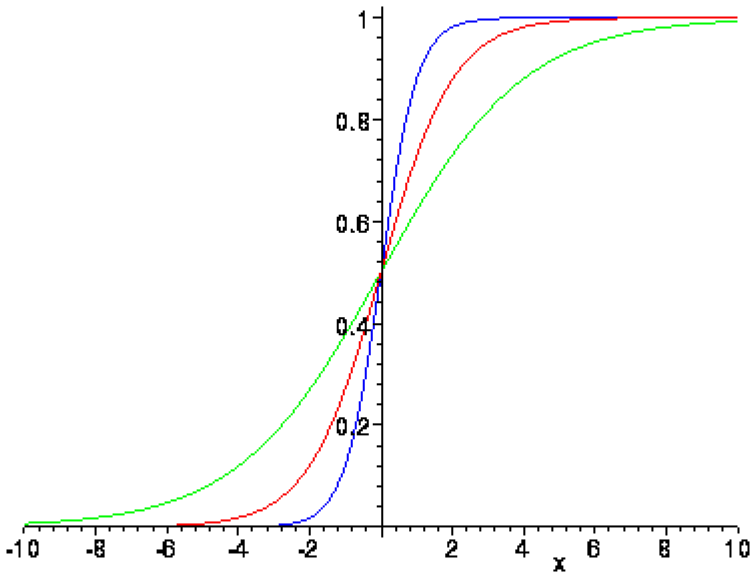


Figure 2.5: The sigmoid

2.6 Polytropic Efficiency - Fan

The fan is the first component in the turbofan engine along the air stream. It accelerates air and a fraction of this air flows into the engine downstream to burn fuel to run compressors and provide hot stream thrust. The rest of it flows around the engine core and converges to provide cold stream thrust.

Engines are classified as high-bypass ratio and low-bypass ratio turbofans depending on the amount of air that travels into the engine and around it. This ratio is called the bypass ratio. Generally, military aircraft engines are low-bypass ratio and civil aircraft use high bypass ratio ones. The fan derives mechanical energy from the low pressure turbine of a two or three spool engine configuration. The Revolutions Per Minute (RPM) of a fan is nearly one order lower in magnitude compared to the other compressor stages. This is mainly to reduce shock waves at the blade tips due to its large diameter.

The fan configuration considered here is a single stage fan with a mass flow rate > 70 kg/s. No size variation correction is applied due to this. The relation between aerodynamic loading and the normalized efficiency η^{***} is based on the

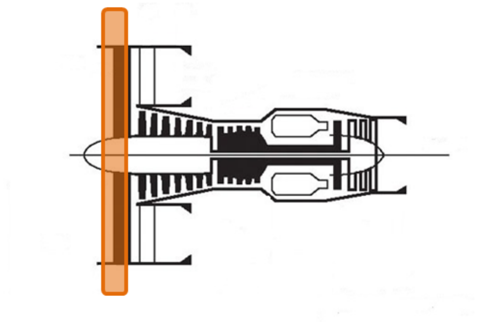


Figure 2.6: Fan rotor

fan pressure ratio at mid cruise (π). The following model was arrived at for the normalized efficiency:

$$\eta^{***} = \frac{0.3721}{0.4004 + e^{12.08\Pi - 25.94}} \tag{2.14}$$

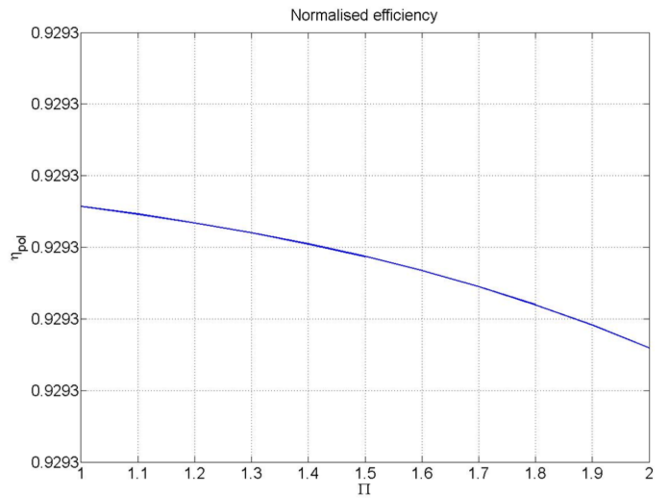


Figure 2.7: Pressure ratio (π) v/s η_{pol}

The choice of pressure ratio for a fan depends on various parameters. Based on prior experience, the starting point for our iteration process would be to assume a fan pressure ratio $\pi = 1.6$.
The EIS correction for a one stage fan is modelled by:

$$\Delta\eta_{pol,EIS} = \frac{-0.8400}{0.8658 + e^{-0.2428*EIS+45.52}} + 0.9130 \tag{2.15}$$

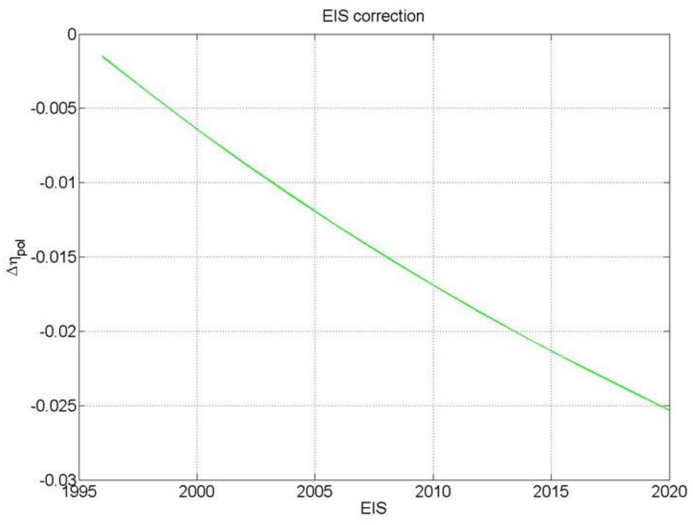


Figure 2.8: EIS v/s $\Delta\eta_{pol}$

The Reynolds Number Index variation is computed from :

$$\Delta\eta_{pol,Re} = 0.095 * \left(1 - \frac{1}{RNI^{0.14}}\right) \tag{2.16}$$

2.7 Polytropic Efficiency - Low speed boosters and Intermediate pressure compressors

Intermediate pressure compressors form the first set of components inside the engine core. As the name suggests, they develop compression values in the order of a fan up to that of a high pressure compressor. IPCs are found in three spool configurations. They derive their energy from the Intermediate pressure turbines.

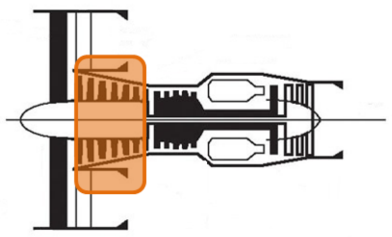


Figure 2.9: Intermediate Pressure Compressor

The relation between aerodynamic loading and the normalized efficiency η^{***} is based on the average stage loading parameter $\bar{\psi}$ at mid cruise. Normalized efficiency is modelled by:

$$\eta_{pol}^{***} = \frac{0.3227}{0.3516 + e^{1.448\bar{\psi} - 4.252}} \tag{2.17}$$

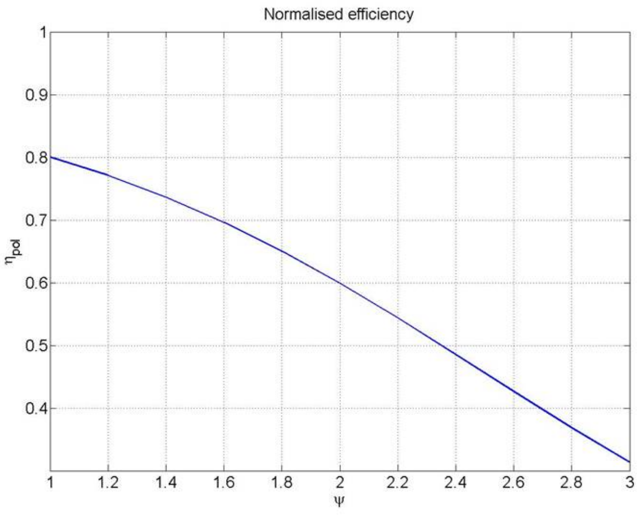


Figure 2.10: ψ v/s η_{pol}

A typical value assumed to start the conceptual design process is $\bar{\psi} = 0.8$ for low speed boosters and $\bar{\psi} = 0.6$ for high speed boosters.

The EIS correction for low speed boosters and intermediate pressure compressors is modelled by:

$$\Delta\eta_{pol,EIS} = \frac{-0.1600}{0.1700 + e^{-0.02297 * EIS + 43.08}} + 0.9073 \tag{2.18}$$

The Reynolds number index variation is modelled by:

$$\Delta\eta_{pol,Re} = 0.095 * \big(1 - \frac{1}{RNI^{0.12}}\big) \tag{2.19}$$

The effect of size on boosters and intermediate pressure compressors are captured by the expression:

$$\Delta\eta_{pol,M} = 0.095 * \big(1 - \frac{1}{\big(\frac{M_{corr}}{M_{corr}^*}\big)^{0.063}}\big) \tag{2.20}$$

where M_{corr} is the normalized mid cruise mass flow defined as $M_{corr} = M\frac{\sqrt{\frac{T_{0,entrance}}{288.15}}}{\sqrt{\frac{P_{0,entrance}}{101325}}}$. $M_{corr}^* = 70\ kg/s$. For flow greater than 70 kg/s, $\Delta\eta_{pol,M} = 0$.

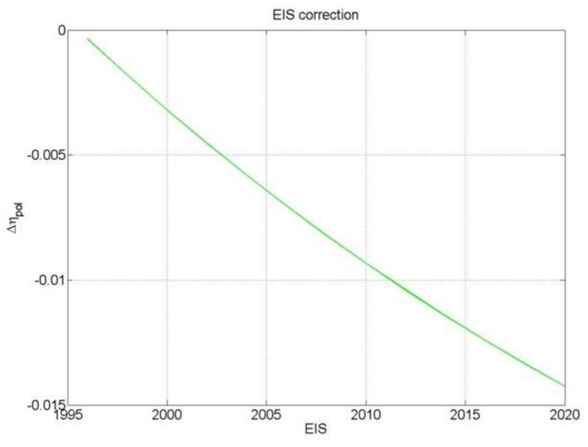


Figure 2.11: EIS v/s $\Delta\eta_{pol}$

2.8 Polytropic Efficiency - High pressure compressors

The high pressure compressor constitutes the third compression step along the air stream in a 3-spool turbofan engine. This is the final stage of compression and output is fed to the combustor where fuel is injected for the combustion process. Pressure ratios of the high pressure compressor ranges from 10-15. The HPC is driven by the High Pressure Turbine.

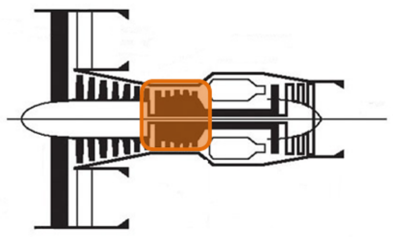


Figure 2.12: High Pressure Compressor

For high pressure compressors the relation between aerodynamic loading and the normalized efficiency η^{***} is based on the average stage loading parameter $\bar{\psi}$ at mid cruise. Normalized efficiency is modelled by:

$$\eta_{pol}^{***} = \frac{0.2492}{0.2643 + e^{1.353\bar{\psi} - 4.183}} \tag{2.21}$$

Based on experience, a typical value to start the conceptual design process is $\bar{\psi} = 0.8$.

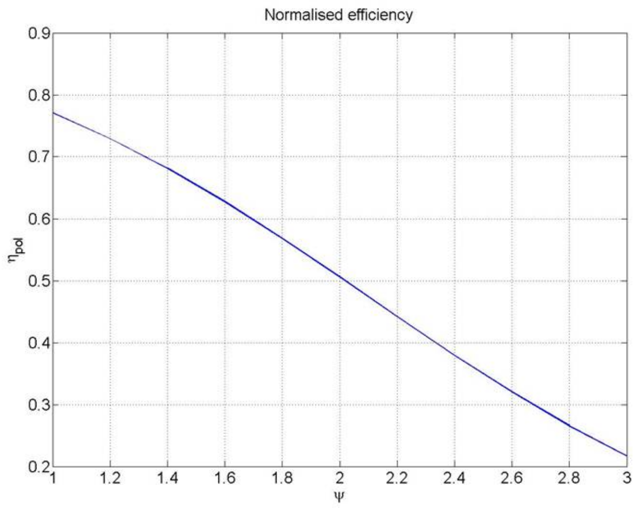


Figure 2.13: ψ v/s η_{pol}

The EIS correction parameter is modelled by:

$$\Delta\eta_{pol,EIS} = \frac{1.800}{1.860 + e^{-0.01463EIS+27.00}} - 0.9131 \tag{2.22}$$

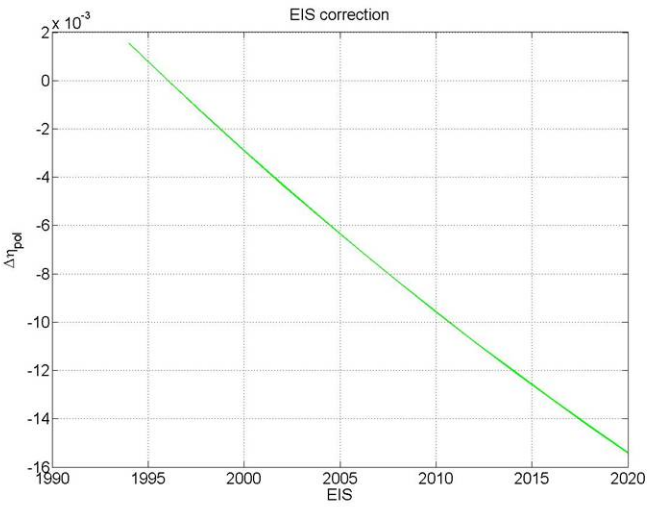


Figure 2.14: EIS v/s $\Delta\eta_{pol}$

The Reynolds Number Index variation is modelled by:

$$\Delta\eta_{pol,Re} = 0.095\left(1 - \frac{1}{RNI^{0.10}}\right) \tag{2.23}$$

The effect of size on high pressure compressors is captured by:

$$\Delta\eta_{pol,M} = 0.095\left(1 - \frac{1}{\left(\frac{M_{corr}}{M^*_{corr}}\right)^{0.063}}\right) \tag{2.24}$$

2.9 Polytropic Efficiency - High and intermediate pressure turbines

The high pressure turbine is the first stage of turbines. It's function is to only drive the high pressure compressor in a three spool configuration.

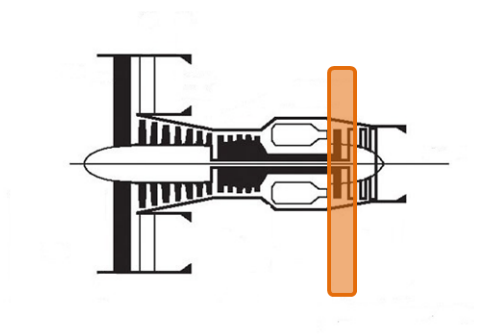


Figure 2.15: High Pressure Turbine

For high pressure turbines the relation between aerodynamic loading and the normalized efficiency η^{***} is based on the average stage loading parameter $\bar{\psi}$ at mid cruise. Normalized efficiency is modelled by:

$$\eta^{***}_{pol} = \frac{1.785}{1.859 + e^{1.303\bar{\psi}-6.678}} \tag{2.25}$$

Based on experience, a typical value to start the conceptual design process is $\bar{\psi} = 3.5$.

The EIS parameter is modelled by:

$$\Delta\eta_{pol,EIS} = \frac{-0.05001}{0.05265 + e^{-0.06720EIS+126.6}} + 0.9400 \tag{2.26}$$

The RNI correction is modelled by:

$$\Delta\eta_{pol,Re} = 0.055 * \left(1 - \frac{1}{RNI^{0.18}}\right) \tag{2.27}$$

The effect of size on high and intermediate pressure turbines is modelled by:

$$\Delta\eta_{pol,M} = 0.055 * \left(1 - \frac{1}{\left(\frac{M_{corr}}{M^*_{corr}}\right)^{0.236}}\right) \tag{2.28}$$

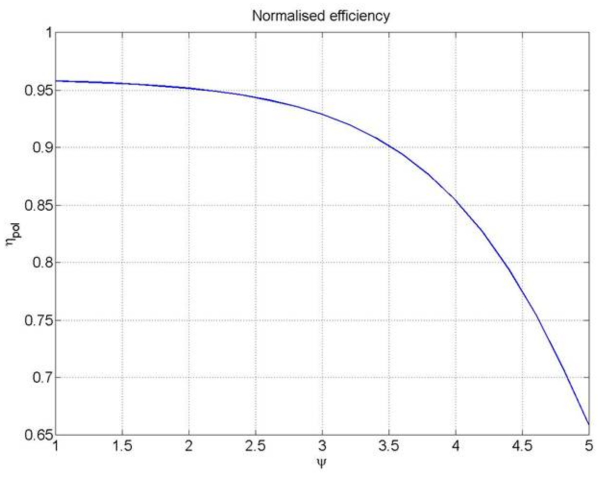


Figure 2.16: ψ v/s η_{pol}

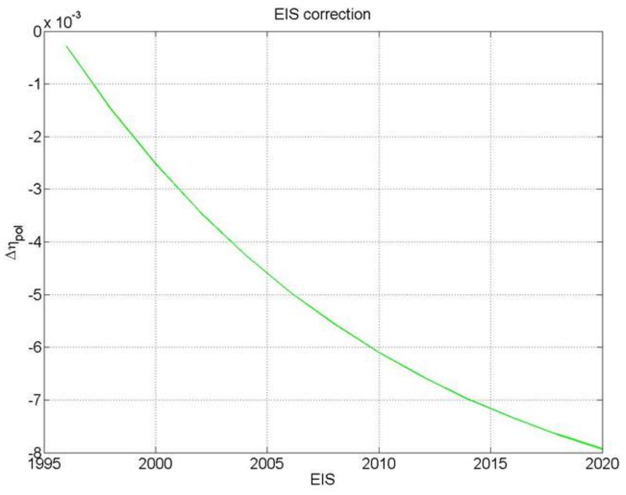


Figure 2.17: EIS v/s $\Delta\eta_{pol}$

Efficiency corrections based on cooling flows is modelled as:

$$\Delta\eta_{cooling} = 0.00467\phi + 0.006936\phi^2 \tag{2.29}$$

The parameter ϕ is determined as:

$$\phi = \frac{\dot{m}_{rotor+stator}}{\dot{m}_{compressorExit}} \tag{2.30}$$

2.10 Polytropic Efficiency - Low pressure turbines

This forms the last step of the turbine expansion in a 3-spool configuration (in a civil aircraft engine) and outputs to the nozzle. Its functional requirement is to drive the fan.

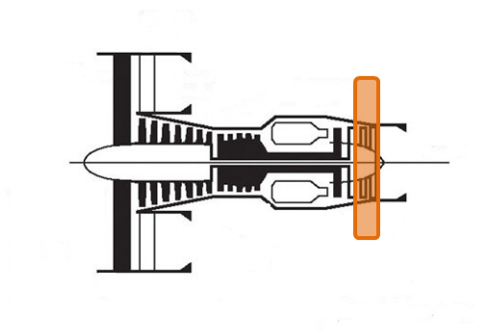


Figure 2.18: Low Pressure Turbine

For low pressure turbines the relation between aerodynamic loading and the normalized efficiency η^{***} is based on the average stage loading parameter $\bar{\psi}$ at mid cruise. Normalized efficiency is modelled by:

$$\eta_{pol}^{***} = \frac{1.785}{1.859 + e^{1.403\bar{\psi} - 8.103}} \tag{2.31}$$

Based on experience, a typical value to start the conceptual design process is $\bar{\psi} = 4.5$.

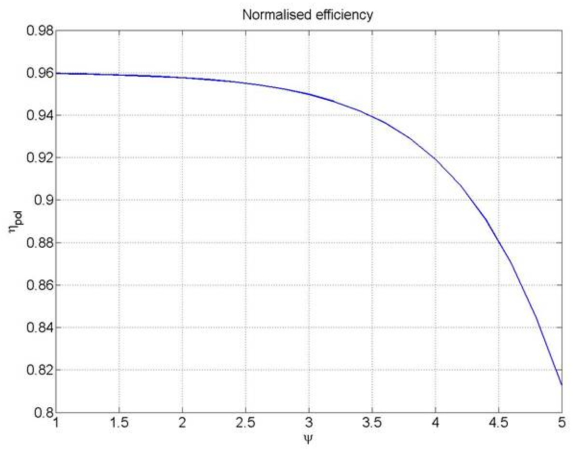


Figure 2.19: ψ v/s η_{pol}

The EIS parameter is modelled by:

$$\Delta\eta_{pol,EIS} = \frac{-1.2745}{1.2952 + e^{-0.026598EIS + 49.87} + 0.9538} \tag{2.32}$$

The RNI correction is modelled using:

$$\Delta\eta_{pol,Re} = 0.055 * \left(1 - \frac{1}{RNI^{0.18}}\right) \tag{2.33}$$

The effect on size for low pressure turbines is modelled using:

$$\Delta\eta_{pol,M} = 0.055 * \left(1 - \frac{1}{\left(\frac{M_{corr}}{M^*_{corr}}\right)^{0.236}}\right) \tag{2.34}$$

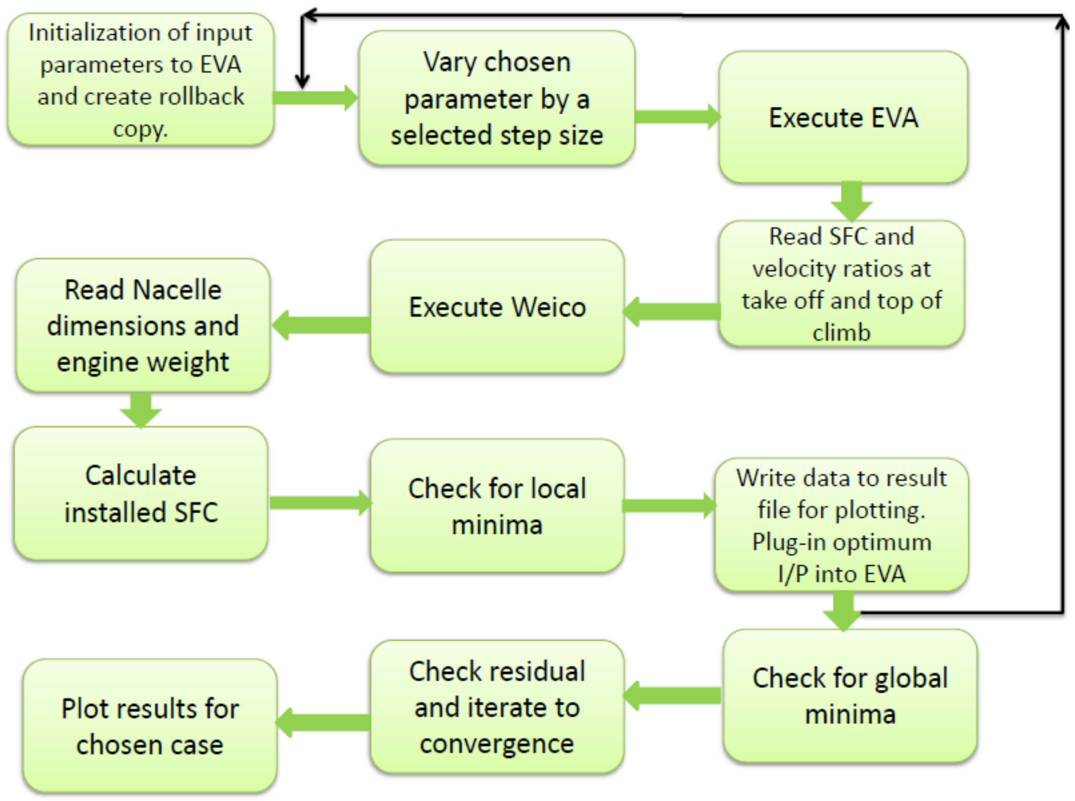
$\Delta\eta_{pol,M} = 0$ for $M_{corr} > 70\,kg/s$.

3 Optimization of running conditions

Power output of an engine increases with pressure ratios and turbine inlet temperatures. This is limited by factors like material limitations and loading parameters. Specific fuel consumption (SFC) is inversely proportional to pressure ratios. Thus it becomes essential to optimize the operating conditions so that we obtain a minimal SFC. Since the design considerations are taken at three points: take-off, top of climb and mid cruise, we limit turbine temperatures for peak power demand points and limit SFC for mid cruise which lasts for the longest duration in a flight.

3.1 Optimizer code

Optimization is achieved using 'Optimizer.exe' which is an optimizer code built in the C language as a part of this thesis work. It uses the brute-force algorithm to give a full picture of performance at equally spaced intervals and is compiled for Intel 64-bit machines on Microsoft Windows 7. The graphic below demonstrates the steps performed by the code.



Initialization of input parameters to EVA and create rollback copy.
Vary chosen parameter by a selected step size
Execute EVA
Read SFC and velocity ratios at take off and top of climb
Execute Weico
Read Nacelle dimensions and engine weight
Calculate installed SFC
Check for local minima
Write data to result file for plotting. Plug-in optimum I/P into EVA
Check for global minima
Check residual and iterate to convergence
Plot results for chosen case

Figure 3.1: Optimizer code algorithm

EVA [5] is the gas turbine performance analyzer code compatible with the Microsoft Windows operating system. Input is provided in the form of operating conditions, like pressure ratios, turbine operating temperatures, correction factors, polytropic efficiencies and efficiencies of mechanical components. These are fed through the Engine_dp.dat file for the design point and Engine_od.dat for off design conditions respectively, located in the installation directory.

The optimization is done for two different engine models: 2015 and 2020. The first task would be to set default values for all independent variables and create a roll back copy of the input file as a back up in case of an error downstream. The Engine_dp.dat file is as shown in Appendix A. The polytropic efficiencies for components in the file are calculated from the models for respective engine models, in the previous section. However, the scope of this project is limited to optimizing the fan, HPC pressure ratio and HPT inlet temperature, which contribute most strongly to the engine performance. Similar procedures can be extended to other components to get a better performance.

Based on statistical data of existing engines, the mass flow rate captured by the variable $windes$ in the input file is varied between $450 - 550\, kg/s$ at a step size of $20\, kg/s$ and velocity ratio shown by the variable $VcoldQhotDes$. It is varied between 0.68 and 0.99 at a step size of 0.02. The pressure ratio of the High Pressure Compressor denoted by the term $PRdes$ is varied between 4 and 5.5 in steps of 0.1. The High Pressure Turbine inlet temperature denoted by $Tmdes$ is varied at a step size of 2 from a minimum of $1170\, K$ to an upper limit defined by:

$$Tmdes = 1370 - (2020 - EIS) * 10\, K \tag{3.1}$$

This empirical formula correlates the advancements in material and manufacturing technology as a function of the Entry Into Service term, resulting in an increased metal temperature. Again modelled based on statistical data, it approximates to nearly $10\, K$ rise in the upper limit of inlet temperature every year. Thus on this basis, a 2020 EIS engine would have a turbine inlet temperature of $1370\, K$.

Mass flow rate and velocity ratio are optimized in the first block of the code. For every combination of Mass flow rate and velocity ratio, the executable file of EVA is called using the *system* command. The output of the simulation is written to the $PerformanceResults.txt$ file in the installation directory. Specific Fuel Consumption and Velocity ratios are read at take-off and top-of-climb conditions. This output file is read using $Weico$.

$Weico$ is a software to calculate engine dimensions based on parameters like SFC, net thrust and pressure ratios. The input to this code is the output file of EVA. The executable file is $Weico.exe$, called using the system command. Output of the code is present in the $weightAircraft.txt$ and $weightOutput.txt$ files. As part of this project we would be interested in nacelle dimensions and engine weight.

Installed SFC literally means the actual SFC of the engine after installing on aircraft. It is the specific fuel consumption after considering all adverse effects after installing the engine like compressed air bleeding losses for cabin pressurization, nacelle drag etc.

Reynolds Number for flow through the Nacelle is given by:

$$Re_{nac} = \frac{C_0 l_{nac} \rho_0}{\mu_0} \tag{3.2}$$

C_0 is the cruise velocity in m/s.
l_{nac} is the nacelle length in m.
ρ_0 is air density at cruise altitude.
μ_0 is the dynamic viscosity of air at cruise altitude.

Nacelle drag is calculated using:

$$D_{nac} = \frac{\rho_0 C_0^2 S_{ref} C_d}{2} \tag{3.3}$$

S_{ref} is area exposed assumed to be 363 m^2.
C_d is the drag coefficient given by:

$$C_d = \frac{C_f * FF * IF * A_{wet}}{S_{ref}} \tag{3.4}$$

$$C_f = \frac{0.455}{(log_{10} Re_{nac})^{2.58} * (1 + 0.144 * M_0^2)^{0.65}} \tag{3.5}$$

where, M_0 is the Mach number
$FF = 1.25$
$IF = 1.0$

$$A_{wet} = \pi * l_{nac} * d_{nac} \tag{3.6}$$

d_{nac}being the nacelle diameter read from $weightAircraft.txt$.

Momentum drag is calculated as:

$$D_{m,eng} = \frac{m_{eng} * g}{L/D} kN \tag{3.7}$$

m_{eng} is the engine weight in kg and $g = 9.81\ m/s^2$.
L/D is the length to diameter ratio.

Installed SFC is now calculated using the relation:

$$SFC_{installed} = \frac{SFC * F_N}{F_N - (D_{nac} + D_{m,eng})} g/kNs \tag{3.8}$$

Installed SFC is calculated in every iteration varying one parameter at the chosen step size at a time. Local minima of installed SFC is identified per parameter and recorded in the respective result files.

4 Results

The code simulates two engine configurations based on Entry Into Service: 2015 and 2020. Results of the various iterations are stored under the following files:

- 'WindesValue'.dat: Contains Velocity ratio at Top Of Climb, Mid Cruise and Take-Off against SFC and installed SFC.
- HPC_Prdes.dat: Contains HPC Pressure ratio against SFC and installed SFC.
- T41M.dat: Contains High pressure turbine inlet temperature against SFC and installed SFC.

The improvement achieved using these models is reflected in the form of increase in SFC after optimising the operating conditions. Thus we shall compare the results achieved for the 2015 EIS engine configuration, by using the conventional method to that achieved using the above models. We shall also make a comparison between the operating conditions and engine dimensions for the 2015 and 2020 EIS engine configurations to see the improvement using these models. *plotter.m* as given in Appendix B, is a Matlab code to plot these various data as graphs.

Figures 4.1, 4.2 and 4.3 represent variation of SFC and installed SFC by varying velocity ratios for six different mass flow rates achieved using conventional and calculated polytropic efficiencies for 2015 and 2020 engine configurations. Comparing SFC and installed SFC values for subsequent velocity ratios, values obtained using the calculated method are lower than those obtained using the conventional method. We get better polytropic efficiencies for individual components using the models compared to the conventional assumptions. Since polytropic efficiency is inversely proportional to SFC, it is reflected as lowered SFC.

Figure 4.4 represents variation of SFC and installed SFC with variation in pressure ratio of the HPC for a 2015 engine configuration, using conventional assumption of polytropic efficiencies. Figure 4.5 represent variation of SFC and installed SFC with variation in pressure ratio of the HPC for 2015 and 2020 engine configurations, using predicted polytropic efficiencies. SFC values for the pressure ratios indicate a better value in the calculated method compared to the conventional method. Optimal operating pressure ratios have been found from running the optimizer code. Designing the component for optimal pressure ratio considering the system is most important to get the best SFC.

Figures 4.6 and 4.7 represent variation of SFC and installed SFC with variation in turbine inlet temperature of HPT, for 2015 and 2020 engine configurations, using conventional and calculated polytropic efficiencies. It is seen that we are able to obtain higher SFC and installed SFC values at a given temperature with the use of the models and it is seen to better with time. This is attributed to the various developments in materials technology resulting in higher operating temperatures given by the empirical equation 3.1. Since higher operating temperatures improves work output, smaller dimensions are possible to derive the same output. This reduces overall weight of the system.

Figures 4.6 and 4.7 plot the engine weight for 2015 EIS engines based on conventional and calculated methods and 2020 EIS engines with corresponding SFC values.

Figures 4.9 and 4.10 are a plot of engine dimensions obtained through *weico* [4] and it can be seen that the engine dimensions and thus weight get smaller with the use of these models and get better with time.

It can thus be inferred that it is possible to start the conceptual modelling phase with better assumptions of polytropic efficiency of the components with the aid of Grieb's work than to stick with conventional method of assumptions based on experience. This can help the designer to derive the best out of what is physically possible.

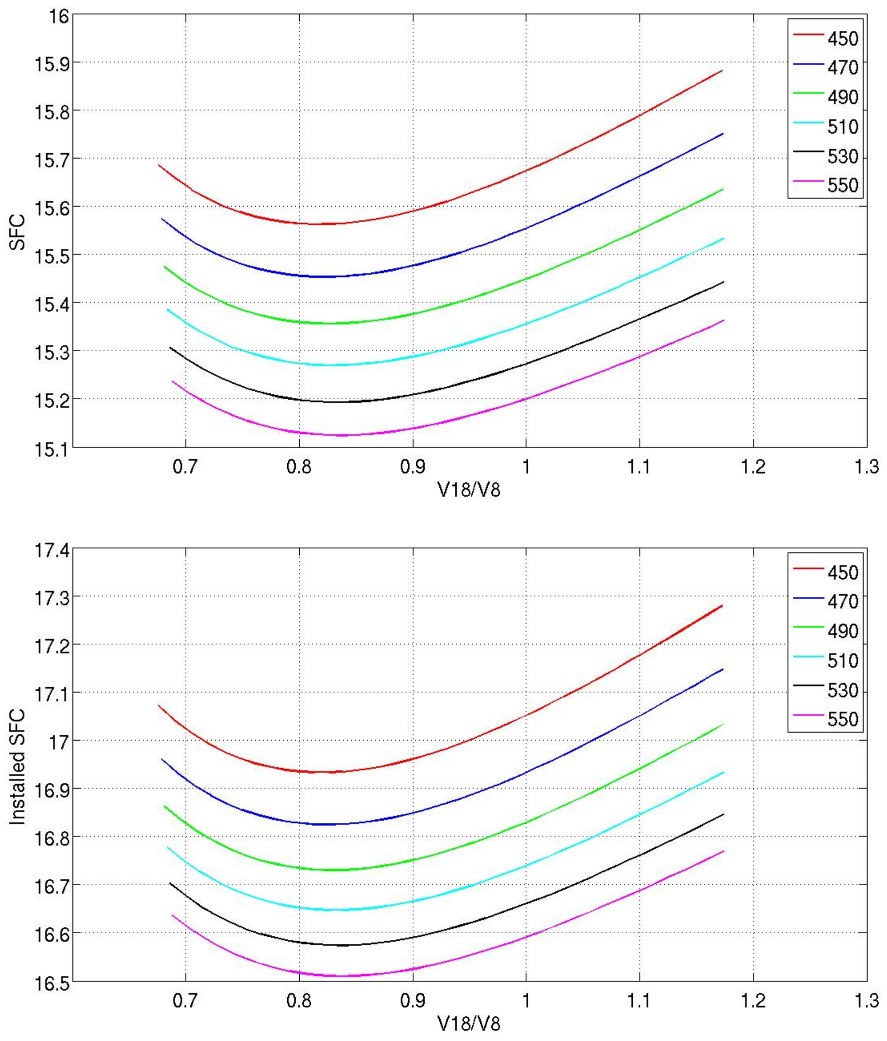
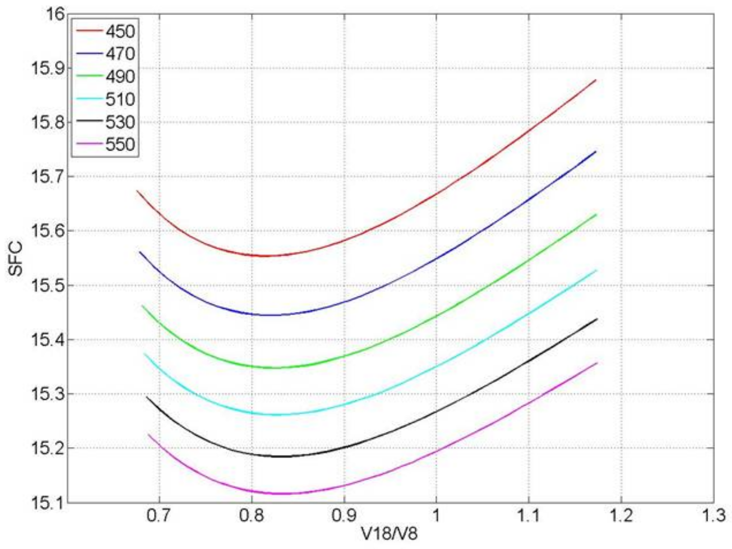


Figure 4.1: SFC and installed SFC for 2015 engines using the conventional method



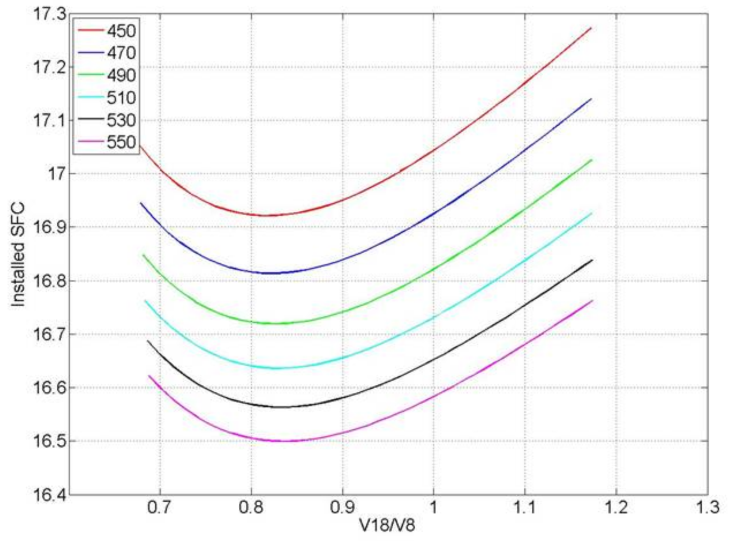
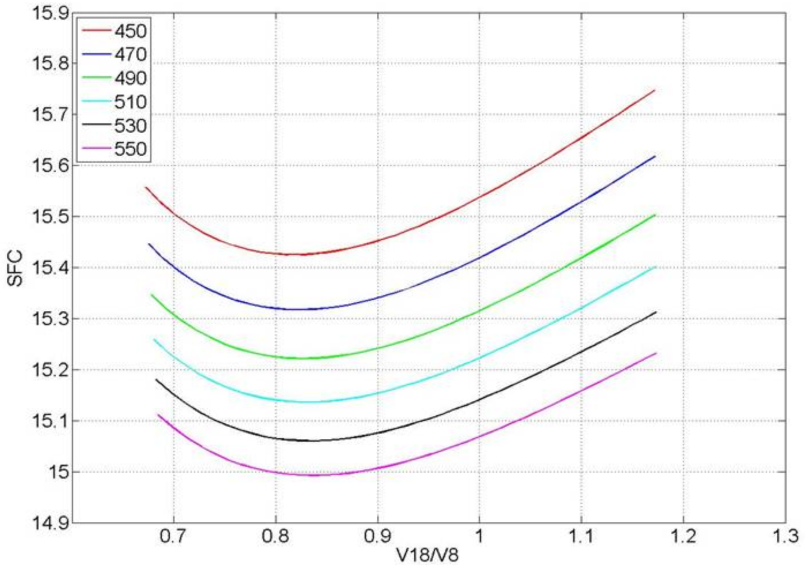


Figure 4.2: SFC and installed SFC for 2015 engines predicted using models



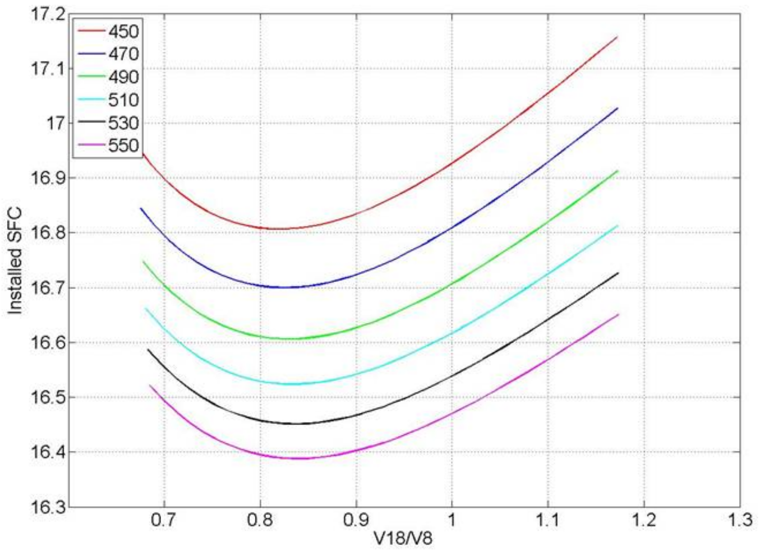


Figure 4.3: SFC and installed SFC for 2020 engines predicted using models

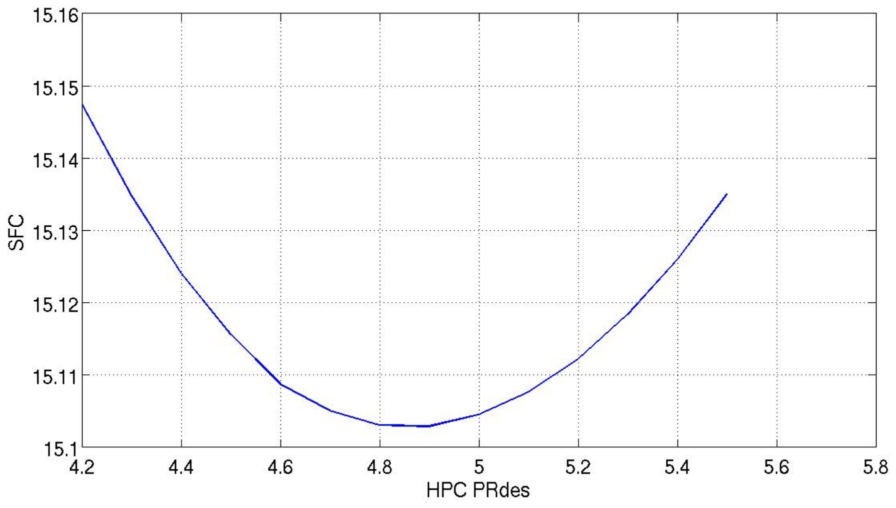


Figure 4.4: SFC vs. HPC pressure ratio for 2015 engines based on conventional assumption.

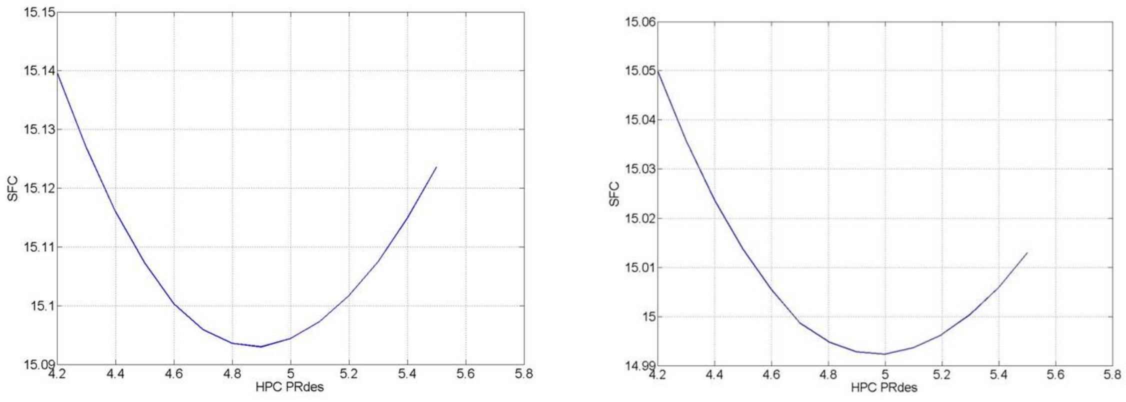


Figure 4.5: SFC vs. HPC pressure ratio for 2015 and 2020 engines based on prediction using models.

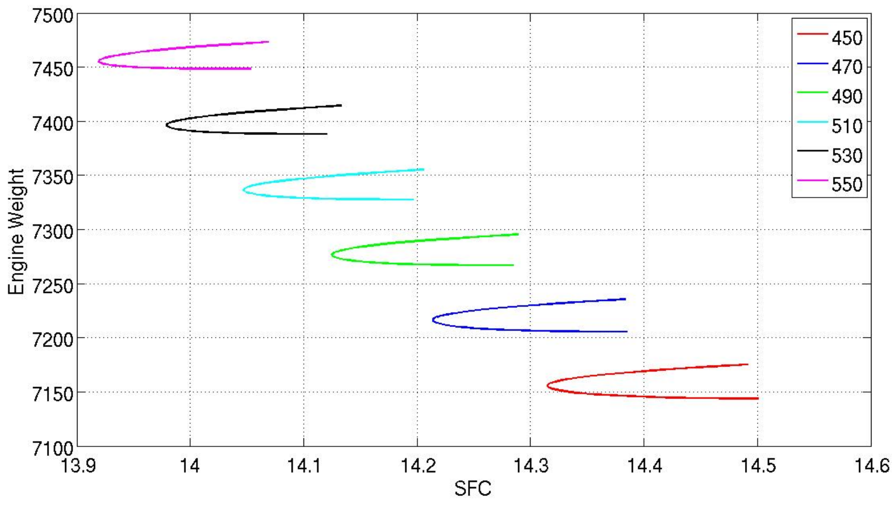


Figure 4.6: SFC vs Engine weight for 2015 EIS obtained by calculated method.

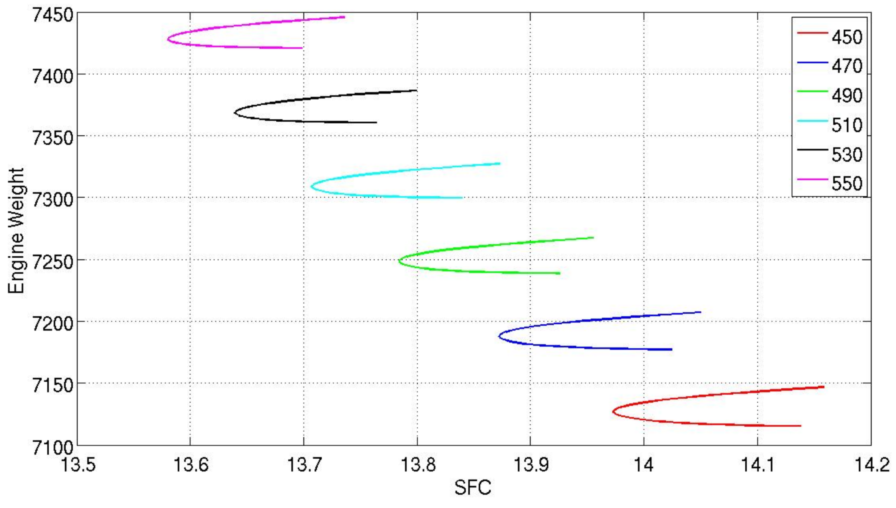


Figure 4.7: SFC vs Engine weight for 2020 EIS obtained by calculated method.

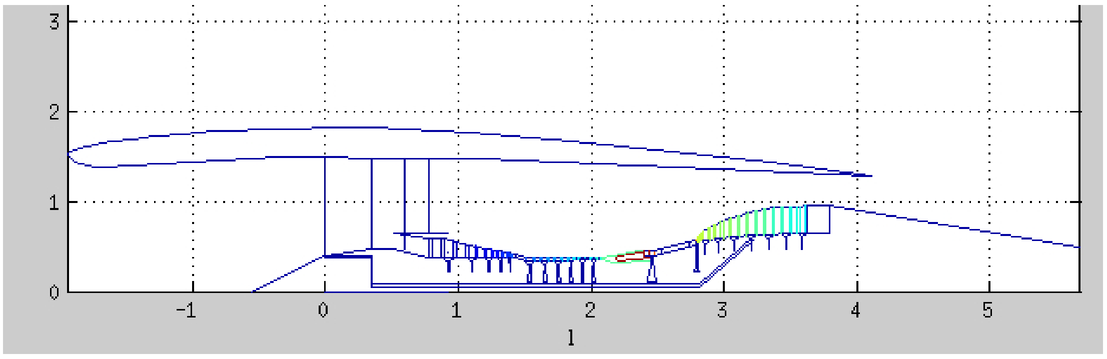


Figure 4.8: Engine dimensions for 2015 EIS using conventional method

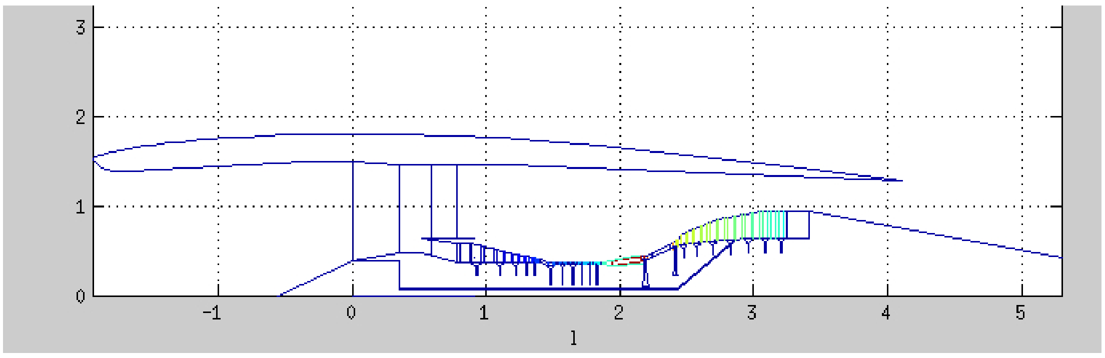


Figure 4.9: Engine dimensions for 2015 EIS using predicted method

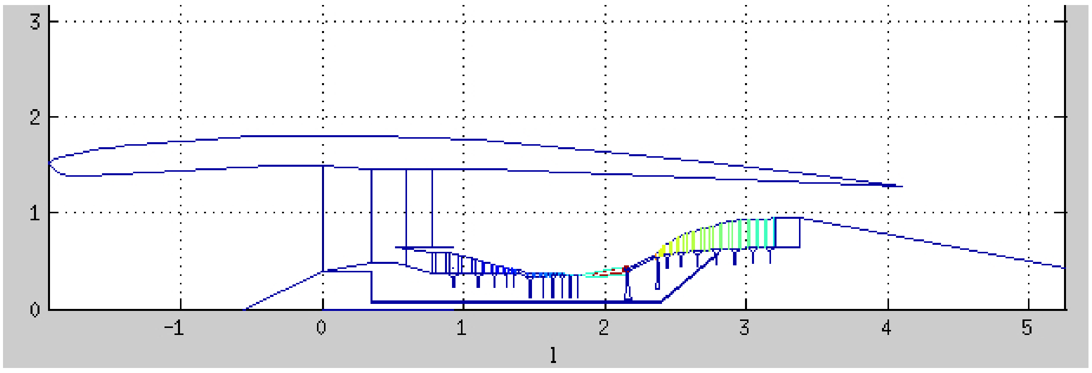


Figure 4.10: Engine dimensions for 2020 EIS using predicted method

5 Future Work

It has been established from this thesis work that the models behave in the expected manner to improve the SFC. Four key parameters have been varied and optimized to see the behaviour. More parameters can be optimized with the developed models to get a better accuracy of the conceptual model of an engine.

This exercise can be performed on a existing engine say a 2010 EIS engine to calibrate the models and fine tune the factors if needed. This can take care of linear bias in the model behaviour if found.

This exercise uses brute-force algorithm for optimization. Optimization algorithm like the *Simplex* method can be implemented for faster and more accurate simulations.

6 References

[1] Grieb, H., 2004, “Projektierung von Turboflugtriebwerken”, Birkhäuser verlag

[2] Saravanamuttoo, HIH et al., 2009, “Gas Turbine Theory”, Pearson Education

[3] Gronstedt, T., 2011, “Conceptual aero engine design modeling- Efficiency modeling”

[4] Gronstedt, T., 2011, “Turbofan sizing and aerodynamics”

[5] Kyprianidis, K., 2011, “EVA Short manual”

[6] Pawlowski, F., 2009, “Smooth Transition”, www.pddnet.com/articles/2009/07/smooth-transition

[7] SempreVolando, 2008, “File:V2500.jpg”, http://en.wikipedia.org/wiki/File:V2500.jpg

A optimizer.c

```
/*===========================================================================
    Optimizer code for EVA
    Author:  Subramanyam Natarajan
    Dept.  of Fluid Dynamics, Chalmers University of Technology, Gothenburg
    ===========================================================================*/
    //Integrates Weico and PSI AND minimises based on SFC_installed and also writes all output
to one file
    // Separates velrat and windes calculations
    #include<stdio.h>
    #include<stdlib.h>
    #include<math.h>
    #include<string.h>
    // If an error occurs in EVA, the Engine_DP.dat is restored to initial values
    void rollback() {
    int linCount;
     char bufferLine[1000];
     FILE *in;
     FILE *out;
    printf("\nEngine_DP.dat Roll-back in progress...\n");
     in = fopen("Engine_DP_Rollback.dat","r");
     out = fopen("Engine_DP.dat","w");
     linCount=1;
    while (linCount!=1647)
     {
     fgets(bufferLine,200,in);
     fputs(bufferLine,out);
     linCount++;
     }
     fclose(in); fclose(out);
    }
    /* // ******************************** EFFICIENCY MODELS **********************************
    void fanEta(float PR, int EIS)
    {
     FILE *in;
    FILE *out;
     float e = 2.7182818;
     float etaNorm, deltaEtaEIS, etaPol;
    char bufferLine[1000];
     int linCount;
     etaNorm = 0.3721 / (0.4004+ pow(e,((12.08*PR) - 25.94)));
    deltaEtaEIS = (0.84 / (0.8658+ pow(e,((-0.0242428*EIS) + 45.52))) ) - 0.9130;
    etaPol = etaNorm + deltaEtaEIS;
    // update in Engine_DP.dat
     in = fopen("Engine_DP.dat","r");
     if (in==NULL
     {
     printf("\n6.  File Engine_DP.dat not found.  Please check!\n");
     fclose(in);
     exit(1);
    }
     out = fopen ("bufferFile","w");
     linCount=1;
    // Reset linCount for rewrite
    // Start writing the buffer file
    while (linCount < 317)
     {
     fgets(bufferLine,200,in);
     fputs(bufferLine,out);
     linCount ++;
     }
    // Update ETApol
     fputs("ETApol = ",out);
     fprintf(out,"%f [-] !  Polytropic Efficiency <<<< EDITED AUTOMATICALLY\n",etaPol);
     fgets(bufferLine,200,in);
    // Moves the seek by one line in 'in'
     linCount++;
     while (linCount < 1647)
    {
    fgets(bufferLine,200,in);
     fputs(bufferLine,out);
     linCount ++;
     }
```

```
 fclose(in);
 fclose(out);
 printf("\n Fan etaPol = %f\n",etaPol);
 }
void hpcEta(EIS)b
{
 FILE *in; FILE *out;
 float e = 2.7182818;
 float etaNorm, deltaEtaEIS, deltaEtaRNI=0, etaPol, RNI=0.37397359, psi=0.8;
 char bufferLine[1000];
 int linCount;
 // etaNorm = 0.2492 / (0.2643+ pow(e,((1.353*psi) - 4.183)));
//printf("\nEtaNormHPC = %f\n",etaNorm);
 //deltaEtaEIS = (0.05001 / (0.05265+ pow(e,((-0.0672*EIS) + 126.6))) ) - 0.94;
 //deltaEtaRNI = 0.055 * (1-(1/pow(RNI,0.18)));
 //printf("\ndeltaEtaRNI = %f\n",deltaEtaRNI);
 //etaPol = 0.9;
 etaPol = etaNorm + deltaEtaEIS - deltaEtaRNI;
 // update in Engine_DP.dat
 in = fopen("Engine_DP.dat","r");
 if (in==NULL) {
printf("\n6.  File Engine_DP.dat not found.  Please check!\n");
 fclose(in);
exit(1);
}
out = fopen ("bufferFile","w");
 linCount=1;
// Reset linCount for rewrite
 // Start writing the buffer file
 while (linCount < 490)
 {
 fgets(bufferLine,200,in);
 fputs(bufferLine,out);
 linCount ++;
 }
 // Update ETApol
 fputs("ETApol = ",out);
 fprintf(out,"%f [-] !  Polytropic Efficiency <<<< EDITED AUTOMATICALLY\n",etaPol);
 fgets(bufferLine,200,in);
// Moves the seek by one line in 'in'
 linCount++;
 while (linCount < 1647)
 {
 fgets(bufferLine,200,in);
 fputs(bufferLine,out);
 linCount ++;
 }
 fclose(in);
 fclose(out);
 printf("\nHPC etaPol = %f\n",etaPol);
 }
// ************************************************************************************ */
// Executes EVA and checks PerformanceLog.out for error-free execution
void execEVA()
{
 char bufferLine[1000];
 int x=1, linCount, logErrorCheck;
 FILE *in;
 // BLOCK TO EXECUTE performance.exe HERE and check for errorfree execution
 system("Performance.exe");
 in = fopen("PerformanceLog.out","r");
 if (in==NULL)
{
printf("\nPerformanceLog.out not found!  Exiting.\n");
fclose(in);
rollback();
exit(1);
}
 logErrorCheck=1;
 bufferLine[200]='a';
 linCount=1;
 x=1;
 while (x!=0 && linCount!=500)
 {
```

```
 // printf("now here %d\n",strncmp(bufferLine," ERROR: Code (0)",15));
 x=strncmp(bufferLine," ERROR: Code (0)",16);
 fgets(bufferLine,500,in);
 if(x==0) logErrorCheck=0;
 linCount++;
// if (strncmp(bufferLine," ERROR: Code (0)",16)==0)
{
logErrorCheck=0;
printf("\nJust read log\n");
}
 }
 if(logErrorCheck==1)
{
printf("\nPerformance.exe failed.  Check PerformanceLog\n");
fclose(in);
rollback();
exit(1);
}
 fclose(in);
// close PerformanceLog.out
 printf("\nExecEva");
}
void execWeico()
{
 system("weico.exe");
 printf("\nExecWeico");
}
// Grabs and returns SFC data from PerformanceResults.txt
double readSFC()
{
 char bufferLine[1000];
 char MC_SFCstr[50];
int linCount, charCount;
 double MC_SFC;
 FILE *in;
 // Open PerformanceResults.txt
 in = fopen("PerformanceResults.txt","r");
 if (in==NULL)
{
printf("\nFile PerformanceResults.txt not found.  Please check!\n");
fclose(in);
exit(1);
}
 linCount =1; // Reset
 // Check for presence of 'cruise' section in the PerformanceResults.txt
 while (linCount<372)
 {
 fgets(bufferLine,1000,in);
 linCount++;
 }
 if(strncmp(bufferLine,"Point Name:  Cruise",18)!=0)
 {
 printf("\nCruise data absent in PerformanceResults.txt.  Exiting optimizer!\n");
 fclose(in);
 exit(1);
 }
 // Grab SFC value
while (linCount<554)
 {
fgets(bufferLine,1000,in);
 linCount++;
}
 // Get to position
 while (fgetc(in)!='0');
 // Scans the SFC value character wise and loads into MC_SFCstr array
 charCount=0;
 while ((MC_SFCstr[charCount]=fgetc(in))!=' ')
 charCount++;
 MC_SFCstr[charCount]='\0';
 //printf("\n%s",MC_SFCstr);
 // Convert string into a floating point value
 MC_SFC = atof(MC_SFCstr);
 //printf("\n%5.10G",MC_SFC);
 fclose(in);
```

```
    // Close PerformanceResults.txt
     return MC_SFC;
    }
    // Write the results of final iteration to respective files
    void outputFile(char filename[20], float value1, float value2, float value3, float value4,
double SFC, double INST_SFC, int firstWriteFlag)
    {
     FILE *out;
     //char location[20]="optimizerResults\\";
    //strcat(location,filename);
     if (firstWriteFlag==1)
     out = fopen(filename,"w");
     else
     out = fopen(filename,"a");
    //printf("%f\t%G\t%G\n",value1,value2,SFC);
     fprintf(out,"%f\t\t%f\t\t%f\t\t%f\t\t%G\t\t%G\n",value1,value2,value3,value4,SFC,INST_SFC);
    printf("\n%s\n", filename);
     fclose(out);
    }
    // PSI // Grab engine weight, Nacelle Length and Nacelle Dia from weightOutput.txt
    double SFCinst(double SFC, int flag)
    {
     char bufferLine[1000];
     char engineWeightStr[50], nacelleDiaStr[50], nacelleLenStr[50];
     int linCount, charCount;
     float S_ref, FF, IF, C_0, rho_0, mu_0, pi, LtoD, Re_nac, C_f, Mach_0, A_wet, C_d, F_N, D_mEng,
g;
     double engineWeight, nacelleDia, nacelleLen, SFC_installed, nacelleDrag;
     FILE *in;
     // Open weightAircraft.txt
     in = fopen("weightAircraft.txt","r");
     if (in==NULL)
    {
    printf("\nFile weightAircraft.txt not found.  Please check!\n");
    fclose(in);
    exit(1);
    }
     linCount=1;
     while (linCount<4)
     {
     fgets(bufferLine,1000,in);
     linCount++;
    }

    // Grab Weight
     // Get to position
     while (fgetc(in)!='O');
     // Scans the engineWeight value character wise and loads into engineWeightStr array
     charCount=0;
     while ((engineWeightStr[charCount]=fgetc(in))!='\n')
     charCount++;
     engineWeightStr[charCount]='\0';
     engineWeight = atof(engineWeightStr);
     linCount++;
     printf("Engine Weight :  %f", engineWeight);
     // skip next line fgets(bufferLine,1000,in);
     linCount++;
     //Grab Eng Nacelle Dia
     while (fgetc(in)!='O');
     // Scans the nacelleDia value character wise and loads into nacelleDiaStr array
     charCount=0;
     while ((nacelleDiaStr[charCount]=fgetc(in))!='\n')
    charCount++;
     nacelleDiaStr[charCount]='\0';
     nacelleDia = atof(nacelleDiaStr);
     linCount++;
     printf("\nnacelleDia :  %f",nacelleDia);
     //Grab Eng Nacelle Len
     while (fgetc(in)!='O');
     // Scans the nacelleDia value character wise and loads into nacelleDiaStr array
     charCount=0;
     while ((nacelleLenStr[charCount]=fgetc(in))!='\n')
     charCount++;
     nacelleLenStr[charCount]='\0';
```

```
 nacelleLen = atof(nacelleLenStr);
 linCount++;
 // Assumptions:
 S_ref = 363; //m2
 FF = 1.25;
 IF = 1.0;
 C_0 = 243.2;
// cruise velocity
 rho_0 = 0.38;
 mu_0 = 0.000014;
 pi = 3.141592654;
 LtoD = 18;
 F_N=51000; // Thrust in N
g = 9.81;
 printf("\nnacelleLength :  %f",nacelleLen);
 Re_nac = (C_0 * nacelleLen * rho_0)/ mu_0;
 printf("\nRe_nac :  %f",Re_nac);
 C_f = 0.455/ ( pow(log(Re_nac)/log(10),2.58) * pow((1 + 0.144*Mach_0*Mach_0),0.65));
 printf("\nC_f :  %f",C_f);
 A_wet = pi * nacelleLen * nacelleDia; // surface area
 printf("\nA_wet :  %f",A_wet);
 C_d = (C_f * FF * IF * A_wet) / S_ref; // Drag co-efficient
 printf("\nC_d :  %f",C_d);
 nacelleDrag = rho_0 * C_0*C_0 * S_ref * C_d/2;
 printf("\nnacelleDrag :  %f", nacelleDrag);
 D_mEng = (engineWeight * g) / (LtoD);
 printf("\nD_mEng :  %f",D_mEng);
 SFC_installed = SFC*F_N / (F_N - (nacelleDia + D_mEng));
 // printf("\nSFC_installed :  %G\n",SFC_installed);
 // outputFile("DNac_DmEng",nacelleDrag,D_mEng,0,SFC,SFC_installed,flag); //printf("\n%5.10G",totalEngineWeight);
 fclose(in); // Close weightAircraft.txt
 return SFC_installed;
}
/* readTestVariables(int flag)
{
int linCount, charCount;
char BPR_MCStr[50], OPR_MCStr[50], bufferLine[1000];
double BPR_MC, OPR_MC;
 FILE *in;
in = fopen("PerformanceResults.txt","r");
 if (in==NULL)
{
printf("\nFile weightAircraft.txt not found.  Please check!\n");
fclose(in);
exit(1);
}

linCount=1;
 while (linCount<404)
 {
 fgets(bufferLine,1000,in);
 linCount++;
}
// Grab BPR_MC from PerformanceResults.txt
 // Get to position
 while (fgetc(in)!='0');
 // Scans the BPR_MC value character wise and loads into BPR_MCStr array
 charCount=0;
 while ((BPR_MCStr[charCount]=fgetc(in))!='\n')
 charCount++;
 BPR_MCStr[charCount]='\0';
 BPR_MC = atof(BPR_MCStr);
 linCount++;
 printf("BPR_MC : %f", BPR_MC);
 while (linCount<552)
 {
 fgets(bufferLine,1000,in);
 linCount++;
}
// Grab OPR_MC from PerformanceResults.txt
 // Get to position
 while (fgetc(in)!='0');
 // Scans the OPR_MC value character wise and loads into OPR_MCStr array
 charCount=0;
```

```
     while ((OPR_MCStr[charCount]=fgetc(in))!='\n')
     charCount++;
     OPR_MCStr[charCount]='\0';
     OPR_MC = atof(OPR_MCStr);
     linCount++;
     printf("OPR_MC : %f", OPR_MC);
     fclose(in);
     outputFile("BPR_OPR",BPR_MC,OPR_MC,0,0,0,flag);
     } */
    int main() {
     // Global
     int EIS=2020, x, firstWriteFlag=1;
     int linCount=1, charCount, iteCount=1, oneWriteFlag=0, logErrorCheck=1;
     double MC_SFC=1000, minSFC=1000, minInstalledSFC=1000, installedSFC=1000, minInstalledSFCPrev=0,
minSFCPrev=0, SFCInit=1;
     double res=1;
     char PRdesString[3], bufferLine[1000], MC_SFCstr[50];
     char testString[100];
     // FAN
     float fan_PR=1.6;
     // W2 & V18/V8
     float Windes=450, Velrat_TOC=0.58, Velrat_MC, Velrat_TO;
    // W2 & Velocity Ratio V18/V8
     float optWindes=450, optVelrat_TOC=0.58, optVelrat_MC, optVelrat_TO, BPR_MC, OPR_MC;
    //optimum values
     char WindesString[3], WindesFilenameChar[8], VelratStr[50], engineWeightStr[50], BPR_MCStr[50],
OPR_MCStr[50];
     double engineWeight;
     // HPC float HPC_PRdes=4;
     // HPC Pressure ratio float optHPC_PRdes=4; // optimum
     // T41M_TOC
     float upperTemp=1370-(2020-EIS)*10;
     float lowerTemp=upperTemp-200;
     float T41M_TOC;
     float optT41M_TOC=lowerTemp;
     FILE *in1;
     FILE *in;
     FILE *out;
     // Back up Engine_DP.dat to rollback
     linCount=1;
     in = fopen("Engine_DP.dat","r");
     if (in==NULL)
    {
    printf("\n1.  File Engine_DP.dat not found.  Please check!\n");
    fclose(in);
    exit(1);
    }
     out = fopen("Engine_DP_Rollback.dat","w");
     while (linCount!=1647)
     {
     fgets(bufferLine,500,in);
     fputs(bufferLine,out);
     linCount++;
     }
     fclose(in);
     fclose(out);
     // Reset Engine_DP.dat to initial values
     linCount=1;
     in = fopen("Engine_DP_init.dat","r");
     if (in==NULL)
    {
    printf("\n2.  File Engine_DP_init.dat not found.  Please check!\n");
    fclose(in);
    exit(1);
    }
     out = fopen("Engine_DP.dat","w");
     while (linCount!=1647)
     {
     fgets(bufferLine,500,in);
     fputs(bufferLine,out);
     linCount++;
     }
     fclose(in);
    fclose(out);
```

```
 printf("\nFor EIS : %d\n",EIS);
 // ==========================================================================
 /*
 // UPDATE FAN EFFICIENCY
 fanEta(fan_PR,EIS);
 //UPDATE HPC EFFICIENCY
 hpcEta(EIS);
 */
 while (res>0.001) {
 // Initialize necessary variables
 minSFC=1000;
 HPC_PRdes=4.2;
 Windes=450;
 upperTemp = 1370-(2020-EIS)*10;
 lowerTemp = upperTemp-200;
 T41M_TOC=lowerTemp;

 // ===========================================================================
 // OPTMIZATION OF W2 AND V18/V8
 while (Windes <= 550)
{
firstWriteFlag=1; // Creates a fresh file during every iteration
Velrat_TOC = 0.58;
//Initial V18/V8
while (Velrat_TOC < 0.99)
{
in = fopen("Engine_DP.dat","r");
if (in==NULL)
{
printf("\n3.  File Engine_DP.dat not found.  Please check!\n");
fclose(in);
exit(1);
}
out = fopen ("bufferFile","w");
linCount=1; // Reset linCount for rewrite
// Start writing the buffer file
while (linCount < 276)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount ++;
}
// Update Windes
fputs("Windes = ",out);
 fprintf(out,"%f [kg/s] !  Inlet Mass Flow in [kg/s] <<<< EDITED AUTOMATICALLY\n",Windes);
fgets(bufferLine,200,in); // Moves the seek by one line in 'in' linCount++;
 while (linCount < 832)
{
fgets(bufferLine,200,in);
 fputs(bufferLine,out);
linCount ++;
}
// printf ("\nI'm here\n");
// Change Velrat_TOC
fputs("VcoldQhotDes = ",out);
fprintf(out,"%f [-] !  Velocity Ratio <<<< EDITED AUTOMATICALLY\n",Velrat_TOC);
fgets(bufferLine,200,in);
// Moves the seek by one line in 'in'
linCount++;
 while (linCount < 1647)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount ++;
}
fclose(in);
fclose(out);
// Rewrite Engine_DP.dat with bufferFile
 linCount=1;
in = fopen("bufferFile","r");
out = fopen("Engine_DP.dat","w");
while (linCount!=1647)
{
fgets(bufferLine,200,in);
```

```
    fputs(bufferLine,out);
    linCount++;
    }
    fclose(in);
    fclose(out);
    // run EVA
    execEVA();
    execWeico();
    // grab SFC value from PerformanceResults.txt
    MC_SFC = readSFC();
    installedSFC = SFCinst(MC_SFC, firstWriteFlag);
    // grab V18V8 values for mid-cruise and take-off conditions too
    // Open PerformanceResults.txt
    in = fopen("PerformanceResults.txt","r");
     if (in==NULL) {printf("\nFile PerformanceResults.txt not found.  Please check!\n");
    fclose(in);
    exit(1);
    }
    linCount =1; // Reset

    // Grab VcoldQhot at Cruise
    while (linCount<553)
    {
    fgets(bufferLine,1000,in);
    linCount++;
    }
    // Get to position
    while (fgetc(in)!='O'); // Scans the VcoldQhot value character wise and loads into VelratStr
array
    charCount=0;
    while ((VelratStr[charCount]=fgetc(in))!=' ')
    charCount++;
    VelratStr[charCount]='\0';
    //printf("\n%s",VelratStr); // Convert string into a floating point value
    Velrat_MC = atof(VelratStr); // Grab VcoldQhot at Cruise
    while (linCount<746)
    {
    fgets(bufferLine,1000,in);
    linCount++;
    }
    // Get to position
    while (fgetc(in)!='O'); // Scans the VcoldQhot value character wise and loads into VelratStr
array
    charCount=0;
    while ((VelratStr[charCount]=fgetc(in))!=' ')
    charCount++;
    VelratStr[charCount]='\0';
    //printf("\n%s",VelratStr);
    // Convert string into a floating point value
    Velrat_TO = atof(VelratStr);

    //printf("\n%5.10G",VelratStr);
    fclose(in); // Close PerformanceResults.txt
    // Open weightAircraft.txt
    in = fopen("weightAircraft.txt","r");
     if (in==NULL) {
    printf("\nFile weightAircraft.txt not found.  Please check!\n"); fclose(in); exit(1);
    }
    linCount=1;
    while (linCount<4)
    {
    fgets(bufferLine,1000,in);
    linCount++;
    }
    // Grab Weight
    // Get to position
    while (fgetc(in)!='O');
    // Scans the engineWeight value character wise and loads into engineWeightStr array
    charCount=0;
    while ((engineWeightStr[charCount]=fgetc(in))!='\n')
    charCount++;
    engineWeightStr[charCount]='\0';
    engineWeight = atof(engineWeightStr);
    linCount++;
```

```
fclose(in);

in = fopen("PerformanceResults.txt","r");
 if (in==NULL)
{
printf("\nFile weightAircraft.txt not found.  Please check!\n");
fclose(in);
exit(1);
}
 linCount=1;
 while (linCount<404)
{
 fgets(bufferLine,1000,in);
 linCount++;
}
// Grab BPR_MC from PerformanceResults.txt
 // Get to position
 while (fgetc(in)!='0');
 // Scans the BPR_MC value character wise and loads into BPR_MCStr array
 charCount=0;
 while ((BPR_MCStr[charCount]=fgetc(in))!='\n')
 charCount++;
 BPR_MCStr[charCount]='\0';
 BPR_MC = atof(BPR_MCStr);
 linCount++;
 printf("BPR_MC : %f", BPR_MC);
 while (linCount<552)
 {
 fgets(bufferLine,1000,in);
 linCount++;
}
// Grab OPR_MC from PerformanceResults.txt
 // Get to position
 while (fgetc(in)!='0');
 // Scans the OPR_MC value character wise and loads into OPR_MCStr array
 charCount=0;
 while ((OPR_MCStr[charCount]=fgetc(in))!='\n')
 charCount++;
 OPR_MCStr[charCount]='\0';
OPR_MC = atof(OPR_MCStr);
 linCount++;
 printf("OPR_MC : %f", OPR_MC);
 fclose(in);

/*
// Check if the latest SFC is the minimum.  If so, update optWindes and optVelrat_TOC
if(MC_SFC<minSFC)
{
minSFC=MC_SFC;
optWindes=Windes;
optVelrat_TOC=Velrat_TOC;
optVelrat_MC=Velrat_MC;
optVelrat_TO=Velrat_TO;
//printf("\nMinimum noted:  %G\n",minSFC);
} //printf("\n%G\n",minSFC);
*/

// optimising based on minimal installed SFC
if(installedSFC < minInstalledSFC)
{
minInstalledSFC=installedSFC;
optWindes=Windes;
optVelrat_TOC=Velrat_TOC;
optVelrat_MC=Velrat_MC;
optVelrat_TO=Velrat_TO;
//printf("\nMinimum noted:  %G\n",minSFC);
}
printf("\n Itn:%d W2:%f V18/V8:%f HPC-PR:%f T41M_TOC:%f\ninstalledSFC: %5.8G",iteCount,Windes,Velrat_TOC,optHPC_PRdes,optT4
printf(WindesFilenameChar,"%f",Windes);
outputFile(WindesFilenameChar,Velrat_MC,BPR_MC,OPR_MC,engineWeight,MC_SFC,installedSFC,firstWriteFlag);
firstWriteFlag=0;
Velrat_TOC=Velrat_TOC+0.01;
}
// Ends Velrat_TOC
```

```
    // printf("\nprinting windes :  %f\n",Windes);
    Windes=Windes+20;
    } // Ends Windes
     // printf("\n%f\n",optWindes);
     // printf("\n%f\n",optVelrat_TOC);
     // Update the Engine_DP.dat file with optimum Windes and VcoldQhotDes values
     in = fopen("Engine_DP.dat","r");
     if (in==NULL)
    {
    printf("\nAttempting to write optimum Windes and VcoldQhotDes values.  File Engine_DP.dat
not found.\n");
    fclose(in);
    exit(1);
    }
     out = fopen ("bufferFile","w");
     linCount=1; // Reset linCount for rewrite
     // Start writing the buffer file
     while (linCount < 276)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     // Update Windes
     fputs("Windes = ",out);
     fprintf(out,"%f [kg/s] !  Inlet Mass Flow in [kg/s] <<<< EDITED AUTOMATICALLY\n",optWindes);
     fgets(bufferLine,200,in);
    // Moves the seek by one line in 'in'
     linCount++;
     while (linCount < 832)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     // Update Velrat_TOC
     fputs("VcoldQhotDes = ",out);
     fprintf(out,"%f [-] !  Velocity Ratio <<<< EDITED AUTOMATICALLY\n",optVelrat_TOC);
     fgets(bufferLine,200,in);
    // Moves the seek by one line in 'in'
     linCount++;
     while (linCount < 1647)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     fclose(in);
     fclose(out);
     // Rewrite Engine_DP.dat with bufferFile
     linCount=1;
     in = fopen("bufferFile","r");
     out = fopen("Engine_DP.dat","w");
     while (linCount!=1647)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount++;
    }
     fclose(in);
     fclose(out);
     printf("\n Just got here!!!!!  " );
     // printf("\nEnd of stage1 \n");
    // ========================================================================== // OPTIMIZATION
OF HPC PRESSURE RATIO
     // Initialize necessary variables
     firstWriteFlag=1;
     while (HPC_PRdes <= 5.5)
    {
    in = fopen("Engine_DP.dat","r");
    if (in==NULL)
    {
    printf("\n4.  File Engine_DP.dat not found.  Please check!\n");
    fclose(in);
exit(1);
}
out = fopen ("bufferFile","w");
linCount=1; // Reset linCount for rewrite
// Start writing the buffer file
while (linCount < 489)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount ++;
}
// Update HPC PRdes
fputs("PRdes = ",out);
fprintf(out,"%f [-] !  Pressure Ratio <<<< EDITED AUTOMATICALLY\n",HPC_PRdes);
fgets(bufferLine,200,in);
// Moves the seek by one line in 'in'
linCount++;
while (linCount < 1647)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount ++;
}
fclose(in);
fclose(out);
// Rewrite Engine_DP.dat with bufferFile
linCount=1;
in = fopen("bufferFile","r");
out = fopen("Engine_DP.dat","w");
while (linCount!=1647)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount++;
}
fclose(in);
fclose(out);
// run EVA
 execEVA();
execWeico();
// grab SFC value from PerformanceResults.txt
MC_SFC = readSFC();
installedSFC = SFCinst(MC_SFC,firstWriteFlag);
// Check if the latest SFC is the minimum.  If so, update optHPC_PRdes
if(installedSFC < minInstalledSFC)
{
minInstalledSFC=installedSFC;
optHPC_PRdes=HPC_PRdes;
// printf("\nMinimum noted:  %G\n",minSFC);
}


// Read Engine weight
// Open weightAircraft.txt
in = fopen("weightAircraft.txt","r");
 if (in==NULL)
{
printf("\nFile weightAircraft.txt not found.  Please check!\n");
fclose(in);
exit(1);
}
linCount=1;
while (linCount<4)
{
fgets(bufferLine,1000,in);
linCount++;
}
// Grab Weight
// Get to position
while (fgetc(in)!='O');
// Scans the engineWeight value character wise and loads into engineWeightStr array
charCount=0;
while ((engineWeightStr[charCount]=fgetc(in))!='\n')
charCount++;
```

```
    engineWeightStr[charCount]='\0';
    engineWeight = atof(engineWeightStr);
    linCount++;
    fclose(in);
    printf("\n Itn:%d W2:%f V18/V8:%f HPC-PR:%f T41M_TOC:%f\nMC-SFC: %5.8G",iteCount,optWindes,optVelrat_TOC,HPC_PRdes,optT41M_
    outputFile("HPC_PRdes",HPC_PRdes,0,0,engineWeight,MC_SFC,installedSFC,firstWriteFlag);
    firstWriteFlag=0;
    HPC_PRdes=HPC_PRdes+0.1;
    }
    // Ends HPC PRdes increments
     // Update the Engine_DP.dat file with Optimum HPC PRdes
     in = fopen("Engine_DP.dat","r");
     if (in==NULL)
    {
    printf("\nAttempting to write optimum HPC PRdes.  File Engine_DP.dat not found.\n");
    fclose(in);
    exit(1);
    }
     out = fopen ("bufferFile","w");
     linCount=1; // Reset linCount for rewrite
     // Start writing the buffer file
     while (linCount < 489)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     // Update to optimum HPC PRdes
     fputs("PRdes = ",out);
     fprintf(out,"%f [-] !  Inlet Mass Flow in [kg/s] <<<< EDITED AUTOMATICALLY\n",optHPC_PRdes);
     fgets(bufferLine,200,in);
    // Moves the seek by one line in 'in'
     linCount++;
     while (linCount < 1647)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     fclose(in);
     fclose(out);
    // Rewrite Engine_DP.dat with bufferFile
     linCount=1;
     in = fopen("bufferFile","r");
     out = fopen("Engine_DP.dat","w");
     while (linCount!=1647)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount++;
    }
     fclose(in);
     fclose(out);

     // =============================================================================== // OPTIMIZATION
OF TAKE-OFF T41M_TOC BASED ON EIS
     // initialize necessary variables
     firstWriteFlag=1;
    while (T41M_TOC <= upperTemp)
    {
    in1 = fopen("Engine_DP.dat","r");
    if (in1==NULL)
    {
    printf("\n5.  File Engine_DP.dat not found.  Please check!\n");
    fclose(in1);
    exit(1);
    }
    out = fopen ("bufferFile","w");
    linCount=1;
    // Reset linCount for rewrite
    // Start writing the buffer file
    while (linCount < 600)
    {
    fgets(bufferLine,200,in1);
    fputs(bufferLine,out);
    linCount ++;
    }
    // Update Tmdes
    fputs("Tmdes = ",out);
    fprintf(out,"%f [K] ! Turbine rotor metal temperature in [K] at design point <<<< EDITED AUTOMATICALLY\n",T41M_TOC);
    fgets(bufferLine,200,in1);
    // Moves the seek by one line in 'in'
    linCount++;
    while (linCount < 1647)
    {
    fgets(bufferLine,200,in1);
    fputs(bufferLine,out);
    linCount ++;
    }
    fclose(in1);
    fclose(out);
    // Rewrite Engine_DP.dat with bufferFile
    linCount=1;
    in = fopen("bufferFile","r");
    out = fopen("Engine_DP.dat","w");
    while (linCount!=1647)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount++;
    }
    fclose(in);
    fclose(out);
    // run EVA
    execEVA();
    execWeico();

    // grab SFC value from PerformanceResults.txt
    MC_SFC=readSFC();
    installedSFC = SFCinst(MC_SFC,firstWriteFlag);
    // Check if the latest SFC is the minimum.  If so, update PRdes
    if(installedSFC < minInstalledSFC)
    {
    minInstalledSFC=installedSFC;
    optT41M_TOC = T41M_TOC;
    // printf("\nMinimum noted:  %G\n",minSFC);
    }
    printf("\n Itn:%d W2:%f V18/V8:%f HPC-PR:%f T41M_TOC:%f\ninstalledSFC: %5.8G",iteCount,optWindes,optVelrat_TOC,optHPC_PRdes
    outputFile("T41M_TOC",T41M_TOC,0,0,0,MC_SFC,installedSFC,firstWriteFlag);
    firstWriteFlag=0;
    T41M_TOC = T41M_TOC + 5;
    }
    // Ends T41M_TOC increments
     // Update the Engine_DP.dat file with Optimum T41M_TOC
     in = fopen("Engine_DP.dat","r");
     if (in==NULL)
    {
    printf("\nAttempting to write optimum HPC PRdes.  File Engine_DP.dat not found.\n");
    fclose(in);
    exit(1);
    }
     out = fopen ("bufferFile","w");
     linCount=1;
     // Reset linCount for rewrite
     // Start writing the buffer file
     while (linCount < 600)
    {
    fgets(bufferLine,200,in);
    fputs(bufferLine,out);
    linCount ++;
    }
     // Update to optimum T41M_TOC
     fputs("Tmdes = ",out);
     fprintf(out,"%f [K] ! Turbine rotor metal temperature in [K] at design point <<<< EDITED
AUTOMATICALLY\n",optT41M_TOC);
     fgets(bufferLine,200,in);
   // Moves the seek by one line in 'in'
     linCount++;
```

``` 
 while (linCount < 1647)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount ++;
}
 fclose(in);
 fclose(out);
 // Rewrite Engine_DP.dat with bufferFile
 linCount=1;
 in = fopen("bufferFile","r");
 out = fopen("Engine_DP.dat","w");
 while (linCount!=1647)
{
fgets(bufferLine,200,in);
fputs(bufferLine,out);
linCount++;
}
 fclose(in);
 fclose(out);
 // ==============================================================================
// ITERATION CHECK
 if (oneWriteFlag==0)
{
SFCInit = minInstalledSFC;
oneWriteFlag=1;
}
 // printf("\nPrev Min SFC: %G",minSFCPrev);
// printf("\nInitial SFC: %G",SFCInit);
 res=fabsl((minInstalledSFC-minInstalledSFCPrev)/SFCInit);
printf("\n================================================================\n");
printf("Iteration:\t\t%d\n",iteCount);
 printf("Residual:\t\t%f\n",res);
printf("Minimum SFC:\t\t%G\n",minInstalledSFC);
 printf("Opt W2:\t\t%f\n",optWindes);
 printf("Opt V18/V8:\t\t%f\n",optVelrat_TOC);
 printf("Opt HPC-PR:\t\t%f\n",optHPC_PRdes);
 printf("Opt T41M_TOC:\t\t%f\n",optT41M_TOC);
 printf("\n================================================================\n");
 // ==============================================================================
// Write all results to one file
 //
 out = fopen("finalResults.txt","w");
iteCount++;
minInstalledSFCPrev=minInstalledSFC;
 }
// Ends iteration loop
return 0;
}
```

B plotter.m

```
clear all
    close all

    load 450.000000
    load 470.000000
    load 490.000000
    load 510.000000
    load 530.000000
    load 550.000000
    load HPC-PRdes
    load T41M-TOC

    load DNac-DmEng

    figure(1);
    plot (X450(:,1),X450(:,5),'r','Linewidth',2);
    hold on;
    plot (X470(:,1),X470(:,5),'b','Linewidth',2);
    plot (X490(:,1),X490(:,5),'g','Linewidth',2);
    plot (X510(:,1),X510(:,5),'c','Linewidth',2);
    plot (X530(:,1),X530(:,5),'k','Linewidth',2);
    plot (X550(:,1),X550(:,5),'m','Linewidth',2);
    grid on;
    legend('450','470','490','510','530','550',1);
    xlabel ('V18/V8','FontSize',20);
    ylabel ('SFC','FontSize',20);
    handle=gca;
    set(handle,'fontsi',[20]);

    figure(2)
    plot (HPC-PRdes(:,1),HPC-PRdes(:,5),'Linewidth',2);
    grid on;
    xlabel ('HPC PRdes','FontSize',20);
    ylabel ('SFC','FontSize',20);
    handle=gca;
    set(handle,'fontsi',[20]);

    figure(3)
    plot(T41M-TOC(:,1),T41M-TOC(:,5),'Linewidth',2);
    grid on;
    xlabel ('T41M','FontSize',20);
    ylabel ('SFC','FontSize',20);
    handle=gca;
    set(handle,'fontsi',[20]);

    figure(4)
    plot (X450(:,1),X450(:,6),'r','Linewidth',2);
    hold on;
    plot (X470(:,1),X470(:,6),'b','Linewidth',2);
    plot (X490(:,1),X490(:,6),'g','Linewidth',2);
    plot (X510(:,1),X510(:,6),'c','Linewidth',2);
    plot (X530(:,1),X530(:,6),'k','Linewidth',2);
    plot (X550(:,1),X550(:,6),'m','Linewidth',2);
    grid on;
    legend('450','470','490','510','530','550',1);
    xlabel ('V18/V8','FontSize',20);
    ylabel ('Installed SFC','FontSize',20);
    handle=gca;
    set(handle,'fontsi',[20]);

    figure(5)
    plot (X450(:,6),X450(:,4),'r','Linewidth',2);
    hold on;
    plot (X470(:,6),X470(:,4),'b','Linewidth',2);
    plot (X490(:,6),X490(:,4),'g','Linewidth',2);
    plot (X510(:,6),X510(:,4),'c','Linewidth',2);
    plot (X530(:,6),X530(:,4),'k','Linewidth',2);
    plot (X550(:,6),X550(:,4),'m','Linewidth',2);
    grid on;
    legend('450','470','490','510','530','550',1);
    xlabel ('SFC','FontSize',20);
    ylabel ('Engine Weight','FontSize',20);
```

```
handle=gca;
set(handle,'fontsi',[20]);

figure(6)
plot (X450(:,1),X450(:,4),'r','Linewidth',2);
hold on;
plot (X470(:,1),X470(:,4),'b','Linewidth',2);
plot (X490(:,1),X490(:,4),'g','Linewidth',2);
plot (X510(:,1),X510(:,4),'c','Linewidth',2);
plot (X530(:,1),X530(:,4),'k','Linewidth',2);
plot (X550(:,1),X550(:,4),'m','Linewidth',2);
grid on;
legend('450','470','490','510','530','550',1);
xlabel ('V18/V8','FontSize',20);
ylabel ('Engine Weight','FontSize',20);
handle=gca;
set(handle,'fontsi',[20]);
```