AEROTHERMODYNAMIC CYCLE DESIGN AND OPTIMIZATOIN METHOD FOR AICRAFT ENGINES

A Thesis
Presented to
The Academic Faculty

by

Sean T Ford

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in the
School of Aerospace Engineering

Georgia Institute of Technology
December 2014

AEROTHERMODYNAMIC DESIGN AND OPTIMIZATION METHOD FOR AIRCRAFT ENGINES

Approved by:

Dr. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Brian Kestner
GE Aviation

Dr. Jeff Schutte
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 31 July 2014

ACKNOWLEDGEMENTS

There are many people I would like to acknowledge, who without their support I would not have come so far. First is my advisor Professor Dimitri Mavris who provided me with the opportunity to pursue this thesis and has guided me on many aspects of not just this thesis but my education. His tireless leadership and mentorship has been a great source of encouragement. He has taught me not only the skills and knowledge to be a better engineer, but a better overall person.

I would also like to acknowledge the community at ASDL including Dr. Jeff Schutte. They have been an amazing source of help and guidance throughout this process. And last but most definitely not least I would like to acknowledge Dr. Brian Kestner. He has been a constant source of help through this thesis and my education at Georgia Tech. He has been instrumental in helping me push further than I thought possible. Without his help and occasional kicks in the right direction I would not have made it so far.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

a_0	Speed of Sound
Advent	Adaptive Versatile Engine Technology
A_e	Exit Area
A_i	Inlet Area
AIAA	American Institute of Aeronautics and Astronautics
AETD	Adaptive Engine Technology Development
AFRL	Air Force Research Lab
BPR	Bypass Ratio
CDS	Cycle Design Space
cs	Control Surface
C_v	Velocity Coefficient
D	Matrix of partial derivatives of objective function
∂	Partial Derivative
DPMM	Design Point Mapping Matrix
η_p	Propulsive Efficiency
f	Fuel Fraction
F	Force
F_n	Net Thrust
FAR	Fuel to Air Ratio
FPR	Fan Pressure Ratio
HP	High Pressure
HPC	High Pressure Compressor
HPCPR	High Pressure Compressor Pressure Ratio
HPT	High Pressure Turbine
IGV	Inlet Guide Vane
ISA	International Standard Atmosphere
IHPTET	Integrated High Performance Turbine Engine Technology

LMG	Linear Model Generator
LP	Low Pressure
LPC	Low Pressure Compressor
LPCPR	Low Pressure Compressor Pressure Ratio
LPT	Low Pressure Turbine
$\dot{m}_0$	Free Stream Mass Flow
M_0	Free Stream Mach Number
$\dot{m}_a$	Entrance Mass Flow
$\dot{m}_C$	Mass Flow Through Engine Core
$\dot{m}_e$	Exit Mass Flow
$\dot{m}_f$	Fuel Mass Flow
$\dot{m}_F$	Mass Flow Through Fan
MDP	Multi-design Point
MFTF	Mixed Flow Turbofan
MOBY	Modulating Bypass Ratio
NASA	National Aeronautics and Space Administration
NPSS	Numerical Propulsion System Simulation
OMDP	Optimized MDP
OPR	Overall Pressure Ratio
P_a	Atmospheric Pressure
P_e	Exit Pressure
R&TD	Requirements and Technology Definition
ρ	Air Density
SAE	Society of Automotive Engineers
SBO	Solver Based Optimization
SDP	Single Design Point
SLS	Sea Level Static
T4	Burner Exit Temperature
T4max	Maximum Burner Exit Temperature
TKO	Takeoff
TOC	Top of Climb

TSFC	Thrust Specific Fuel Consumption
u	Free Stream Velocity
u_e	Exit Velocity
u_x	Velocity in x Direction
VAATE	Versatile Affordable Advanced Turbine Engines
VABI	Variable Area Bypass Injector
VAPCOM	Variable Pumping Compressor
VCE	Variable Cycle Engine

SUMMARY

Gas turbine engines for aircraft traditionally establish the cycle through the process of on-design cycle analysis which involves the calculation of the cycle at a single design point. The performance of the engine is determined in off design analysis for fixed design choices at all flight conditions. Off design analysis can only begin once the design point and size of the engine have been chosen. It determines the performance of an engine with fixed design choices at all flight conditions and based on the off design performance over the entire aircraft mission, an engine can be selected for a particular mission. However, the selection of one design point to set the thermodynamic cycle can be difficult.

Variable geometry components have now been in use for decades. These components can alter their geometry and thus the flow through an engine. Typically these components are used to improve engine performance and more recently with variable cycle engines. VCE concepts are actively being explored as a means to meet competing performance demands of high thrust and low fuel consumption placed on aircraft. VCEs provide the possibility for achieving better performance in subsonic or supersonic flight and improved airflow matching through the use of VG features and additional flowpaths. It if from VCEs and variable geometry that it becomes possible to further maximize performance above what is determined from the fixed cycle design choices in on design. Thus the need arises for multi and single variable optimization techniques for aircraft engines.

There are a large variety of optimization algorithms available for a wide range of problems. Few are designed directly for aerothermodynamic cycle design. In most cases in available literature, an algorithm is chosen and then wrapped around the cycle solver. This thesis develops a solver based optimization (SBO) method which is able to control cycle design variables at all operating conditions to meet the performance requirements while controlling any additional variables which may be used to optimize the cycle. This is done while maintaining all operating limits and engine constraints to find a balanced and optimum cycle. SBO does this by utilizing features which are inherent in many typical cycle solvers, and in particular the common modified

Newton-Raphson type cycle solver

The method efficiently finds cycle designs for a wide range of engine architectures with extra degrees of freedom not needed to balance the cycle. Further, SBO can be directly applied to a multi-design point methodology to overcome the problem of selecting a single design point to size an engine. Selecting one design point in this manner generally results in an oversized engine with reduced performance at off design conditions. SBO effectively combines on design, off design, and optimization into a single simultaneous implementation.

Two research questions are identified in this thesis. SBO is demonstrated in this thesis on a separate flow turbofan model to explore these questions and test five hypotheses. Ten experiments are performed to highlight different aspects of the method. This includes a proof of concept on several analytical test functions to demonstrate the efficiency of the method and its ability to find a known optimum. It is then demonstrated using the SFTF model which provides insight into the implementation of SBO on a real world problem which is highly constraining, in the process pointing out several limitations on the method and how it is combined with an MDP method. SBO is successfully able to find a balanced and optimum cycle design for an SBO model for both the on design and off design spaces.

CHAPTER 1

INTRODUCTION AND MOTIVATION

This thesis addresses the need for an optimization method which can simultaneously optimize and balance an aerothermodynamic cycle. Traditionally for gas turbine engines, the cycle is established through the process of on-design cycle analysis which involves the calculation of the cycle at a single design point by selecting values of design variables such as fan pressure ratio (FPR), overall pressure ratio (OPR), bypass ratio (BPR), and combustor exit temperature (T4). Off design analysis determines the performance of an engine with fixed design choices at all flight conditions. Off design analysis can only be performed once the design point and size of the engine have been chosen. Based on the off design performance over the entire aircraft mission, an engine can be selected for a particular mission. However, the selection of a single design point to set the thermodynamic cycle can be difficult.

Over the past several decades engine designs have evolved from piston driven propeller aircraft to turbojet and turbofan designs. With them came the advent of variable geometry (VG) features which can alter their geometry and thus the flow through an engine. Typically these VG components are used to improve engine performance. Now, variable cycle engine (VCE) concepts are actively being explored as a potential approach for advanced military or commercial propulsion.[43] VCEs provide the possibility for achieving better performance in subsonic or supersonic flight and improved airflow matching through the use of VG features and additional flowpaths.

The need for an optimization methodology derives from the class of engines known as VCEs. "Steady-state operating schedules must be established to set the individual variable geometries, to constrain engine operation within acceptable limits, and to achieve the maximum performance capability which is potentially available."[43] Multi-variable optimization techniques are required for optimizing these designs, as well as multi and single variable optimization for the current generation of engines. However, it is important to understand the limitations of optimization techniques and that achieving the 'best' design can vary based on how best is defined.

"Expectations of achieving an absolute best design will invariably lead to maximum disappointment."[39]

Thus the optimization method used should be able to control cycle design variables at all operating conditions to meet the performance requirements while controlling any additional variables which may be used to optimize the cycle. This must be done while maintaining all operating limits and engine constraints to find a balanced and optimum cycle. The method should combine the cycle balance and optimization to efficiently find cycle designs for a wide range of engine architectures with extra degrees of freedom not needed to balance the cycle.

This thesis is divided into six chapters. This first chapter provides an introduction and a motivation for this optimization methodology. The second chapter provides background information on current paradigms in engine design and optimization. The third chapter discusses the methodology and the theory for the optimization technique developed, and provides a step by step procedure. The fourth chapter describes the implementation and setup of the method. The fifth chapter discusses the experiment setup and analysis of the experimental results. Finally, the last chapter includes some concluding remarks, summarization of the contributions to the engineering field, and suggestions for future work.

1.1 The Need for a New Engine Architecture

Most military fighter aircraft engines today utilize a low bypass ratio mixed flow turbofan (MFTF), such as that shown in Figure 1, in an attempt to reach an optimum balance between competing performance requirements. However, a new engine architecture is being pursued that can better meet requirements in the form of a variable cycle engine. As the name implies, it can operate with two or more thermodynamic cycles, theoretically allowing it to achieve high efficiency during cruise and high specific thrust when required at times such as acceleration to supersonic flight.

In designing a military aircraft engine for example, there are several performance parameters that designers attempt to optimize. Often these parameters are at odds with each other and arise from the fact that military fighters require engines that can operate efficiently at both supersonic and subsonic speeds. Two of these parameters are specific thrust and fuel consumption. Generally the predominant goal for a fighter aircraft is to achieve a high aircraft thrust to weight ratio attained through high engine specific thrust.[4] High specific thrust is desirable for a number

of reasons including combat maneuvering, supersonic flight, short takeoff, and intercept to name a few.[1] Normally specific thrust is maximized by using a traditional turbojet or very low bypass ratio turbofan. The turbojet offers the best option for high specific thrust at subsonic to low supersonic speeds.

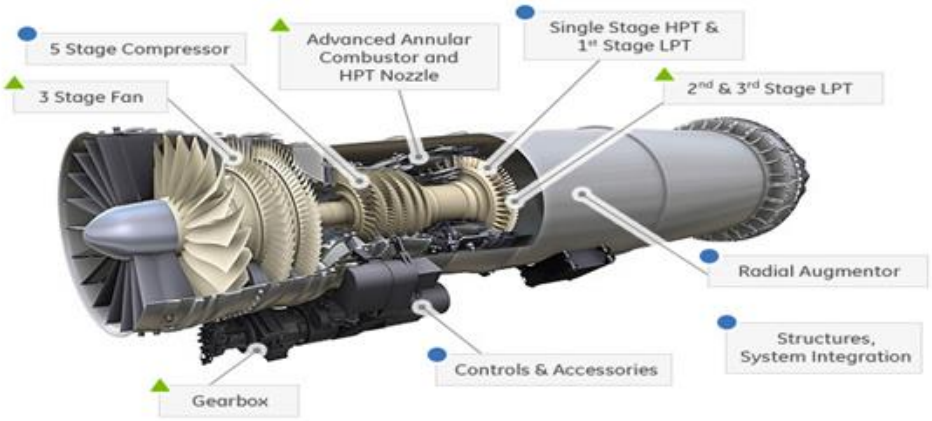

Figure 1: 3-D Illustration of MFTF[25]

Requirements for long duration cruise (i.e. low fuel consumption), loiter, noise reduction, and operating costs often are competing demands on the engine which favor a different engine architecture, namely, the high bypass ratio turbofan. Most large commercial aircraft utilize a form of high bypass ratio turbofan. This allows relatively high specific thrust, some improvement of fuel consumption over the turbojet, and does not prohibitively increase the size and weight of the aircraft. Engine diameter has a considerable effect on the airframe size and weight.[4]

Meeting the competing performance demands requires a new engine architecture that is a departure from the traditional turbofan and turbojet designs of the past 60 years. In that time there has been a considerable amount of research into solving the problems of competing performance demands. However, it has been several decades since the Unites States has sought to develop an all-new combat aircraft engine.[5]

Early jet engines were single stream turbojets. These provided a high level of specific thrust by moving a relatively small amount of air very fast. Early turbojets offered poor fuel efficiency

however. Later, double stream turbofans were introduced which exhaust air at a relatively slow velocity. These give a much lower fuel consumption. The "Holy Grail" of engines would effectively combine the best characteristics of both engine types to provide high specific thrust at low fuel consumption over the engine's operating envelope. After decades of research, the type of engine envisioned to meet the need is a double bypass variable cycle engine like that shown in Figure 2. The US Air Force is currently actively pursuing variable bypass, adaptive engine technology in its Advent and AETD programs. AFRL calculates this technology will improve engine efficiency by 25% and increase aircraft combat radius by 25-30%.[5] With improvements like this, it is easy to see why this technology is being pursued.

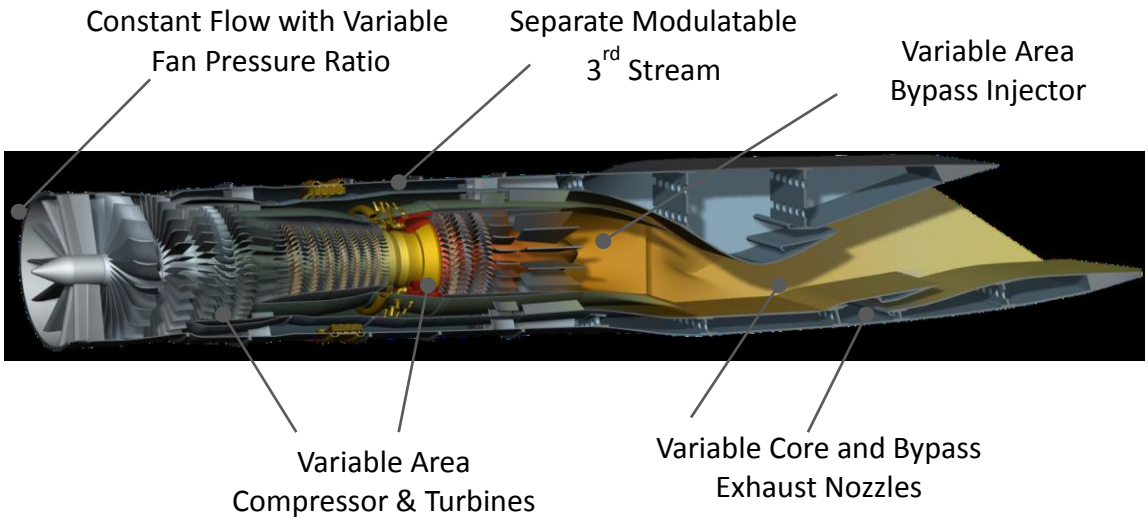

Figure 2: Notional Illustration of Double Bypass VCE[1]

1.2 The Need for an Optimization Method

With the increase in the number of available variable features the question remains as to how such variable feature geometries are chosen. Simmons notes that for a VCE of the type shown in Figure 2, "just finding locally optimal solutions would require far too great a time investment; a truly comprehensive search of the design space would require a more automated process."[1] This statement succinctly sums up a problem that has faced VCE designers and those wishing to show the extent to which a VCE can offer performance improvements. It also provides motivation for

a more automated optimization method.

There are a large variety of optimization algorithms available for a wide range of problems. However, few have been designed to be directly used for aerothermodynamic cycle design. In most cases in available literature, an algorithm is chosen and then wrapped around the cycle solver. This method has one main advantage – it allows almost any type of optimization algorithm to be chosen to best fit the problem at hand. This is generally accomplished by some type of nested loop structure where the optimizer varies the available values in one loop, and then the cycle solver rebalances the cycle in an inner loop. Both the solver and optimizer iterate until a balanced and optimal cycle is found. These loops can be built directly into the thermodynamic analysis package, or the cycle solver and optimizer package can be left separate and combined through use of a third piece of software such as ModelCenter which combines the two.

There are several drawbacks of the approach just mentioned. First is the need to in some way link the optimization package and thermodynamic analysis. This could represent a significant amount of time and energy invested in making the two packages compatible. Second, having to rebalance the cycle separately from the optimization is typically extremely inefficient.

Ideally, it would be advantageous to balance the cycle while simultaneously optimizing it. Brown (Reference 43) notes that many typical cycle solvers already incorporate much of the required information to do this. Firstly, they incorporate the unknown parameters needed to balance the cycle. Additionally they are capable of handling the many constraints placed on an engine system. And most importantly, they already are capable of computing derivative information to determine the search direction and step size. The only remaining piece of information required to create a simultaneous optimization and cycle balance method is knowing when the optimum has been reached.

CHAPTER 2

BACKGROUND INFORMATION

This chapter provides the background information necessary to understand the challenges associated with engine optimization and the historical backdrop which led to the motivation for a new methodology. The first section provides a brief history of VCE research since its beginning in the 1960s. The second section discusses the different VG components available in current propulsion systems. The third section discusses optimization methodologies in general and their specific application to aircraft engines. Throughout this chapter, observations are highlighted which lead to the research questions this thesis will answer.

Before proceeding, it will be helpful to provide a common and general nomenclature for aerospace propulsions systems. The Society of Automotive Engineers (SAE) has provided such a nomenclature in publishing ARP 755B.[48] ARP 755B creates a common nomenclature for designating stations for various engine architectures with each station locating thermodynamic properties of the flow within the engine. This station numbering system will be used throughout the remainder of this thesis.

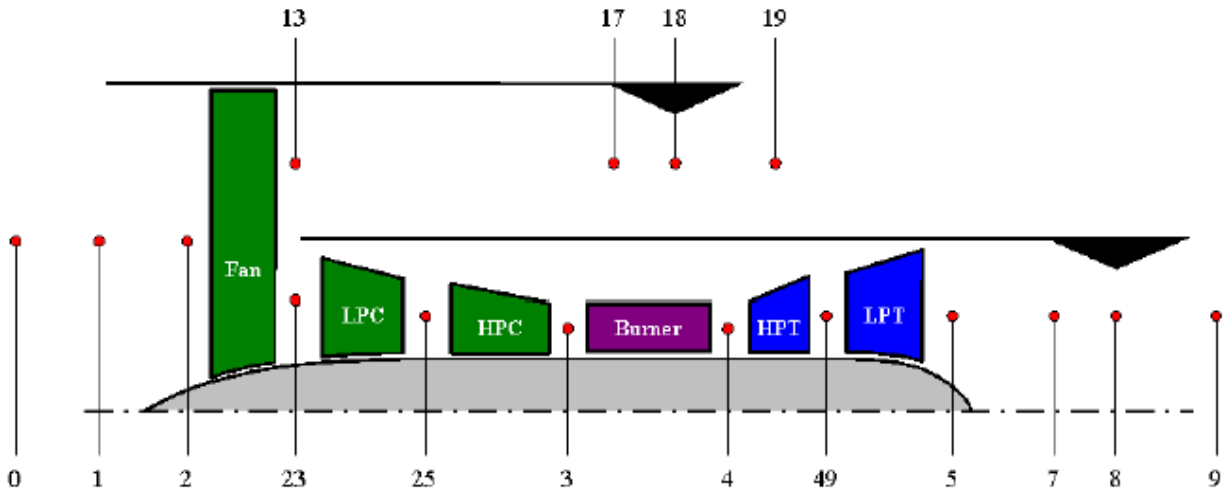

Figure 3: Example of SFTF Station Numbering[27]

2.1 Variable Cycle Engines and Variable Geometry

This section will briefly describe the history of the VCE. It is this architecture which forms part of the motivation for this thesis. Research in this area has been ongoing for many years. VCEs, as do many other engines, incorporate variable geometry components. The second part of this section will summarize the most common variable geometry components used.

2.1.1 VCE History

"A 'variable cycle' engine generally refers to a family of hybrid gas turbine engines which exhibit the high specific thrust characteristic of a low BPR turbofan or turbojet at high power settings, and yet also exhibit relatively low specific thrust, noise, and fuel consumption levels typically characteristic of moderate BPR turbofan engines at part power settings."[10] To this end, there have been numerous engine designs proposed over the years. One of the first attempts was the Variable Pumping Compressor shown in Figure 4.

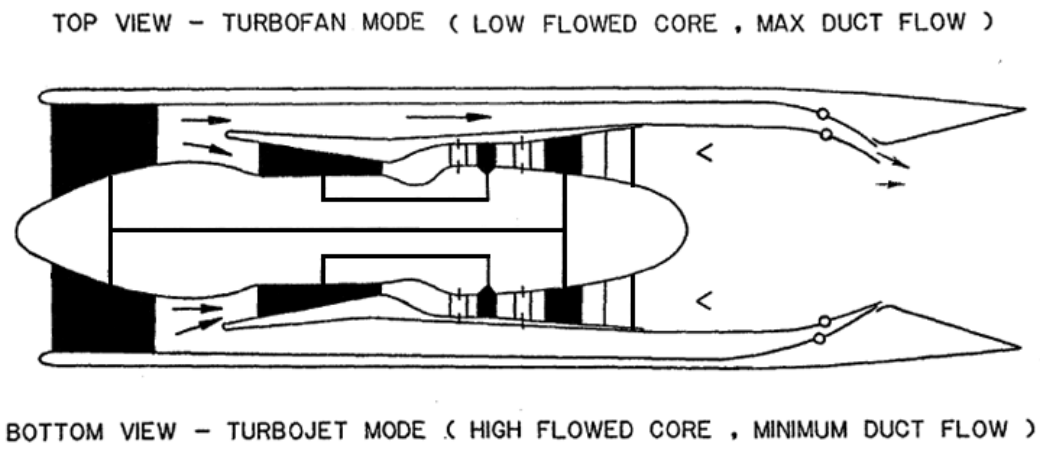

Figure 4: VAPCOM[28]

The General Electric Company has been especially active in VCE research and in 1973 they invented the MOBY (Modulating Bypass Ratio) engine in response to the Air Force's request for engine concepts that address the problems of throttle dependent losses. The system was successful in minimizing spillage drag at part power and offered significant fuel savings, but overall system complexity prohibited further development.[28]

These few examples are just the tip of the iceberg for variable cycle research; shown here to give an idea of some previous concepts and the length of time this line of research has been pursued. There is a large variety of sources giving more detail on the advances in variable cycle engine technology, several excellent ones being References 15, 19 , and 28.

2.1.2 Variable Geometry

"Variable geometry is used extensively in advanced aircraft engines."[35] Common forms of variable geometry include variable inlet guide vanes (IGVs) and variable area nozzles. A variable cycle, while it almost certainly will incorporate variable geometry features, is different in that it can operate with two or more thermodynamic cycles – hence the name variable cycle engine. There are a large number of proposed VCE designs, however this cycle variability is accomplished through use of these variable components often in conjunction with the additional 3rd stream. This leads to the first observation:

> Observation 1: Variable geometry features are also used extensively on current engine designs and may be used to provide a small measure of cycle variability or for operability requirements. Operating schedules are required to set the individual variable geometries and to achieve the maximum performance capability which is potentially available.

In other words, there are additional degrees of freedom available to the cycle that are not required to satisfy conservation of mass, momentum, and energy. The added degrees of freedom in variable geometry designs makes this inherently an optimization problem.

2.1.2.1 Variable Area Bypass Injector

A Variable Area Bypass Injector (VABI) is a device used to vary the relative proportions of flow in a mixer. Effectively, it acts like a variable area mixer. The key function of the rear VABI is to enable control of the fan operating line independently of the core gas generator system.[19] It does this by matching static pressures of the streams entering the mixer by varying the Mach number in the bypass stream to attain the static pressure balance for mixing the flows.[30] VABI settings may range from 0.5 to 1.5. In any condition other than the nominal position, a pressure drop

must occur across the VABI in order to match static pressures resulting in increased mixing losses associated with the velocity mismatch of the two streams.[19]

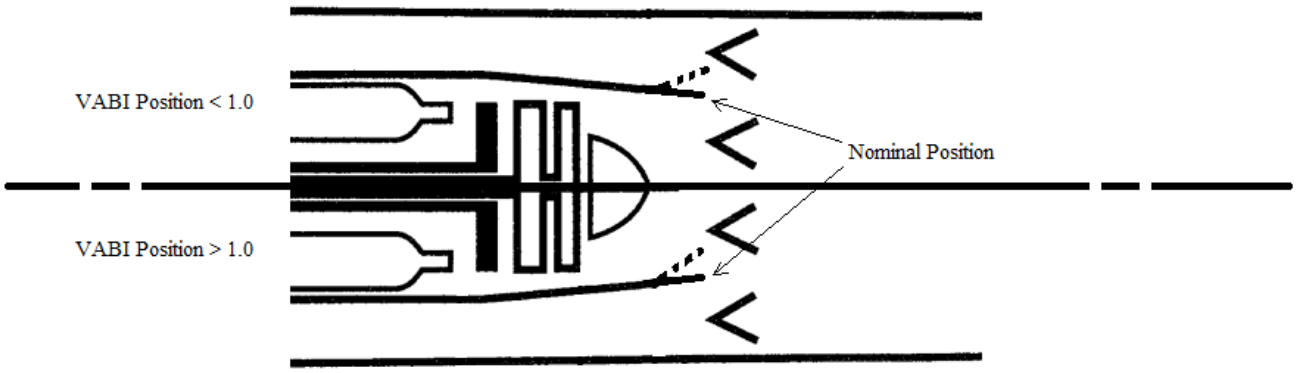

Figure 5: VABI Position Diagram

2.1.2.2 Variable Area Nozzle

The basic effect of the nozzle is most easily understood when considering a simple arrangement of a compressor on a test rig. At constant speed a nozzle downstream of the compressor is closed. This increases the pressure ratio in the compressor because the compressor must now increase the density of the gas to push it through the smaller flow area. In such a way the working line of a compressor can be controlled.[31] For an engine with an afterburner a variable area nozzle becomes a necessity. Due to the increase in temperature in the afterburner, the pressure increases to the point where compressor stall would occur if the nozzle area where not adjusted.[31]

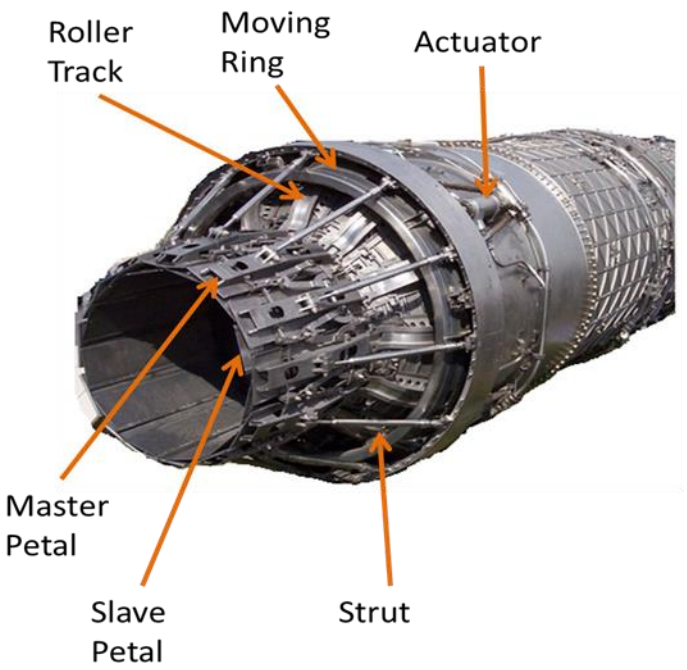

Figure 6: EJ200 Variable Exhaust Nozzle[37]

2.1.2.3 Variable Inlet Guide Vanes

The aim of using VIGVs in a VCE is to enable transition from one cycle to another.[35] They may be used in an architecture such as an MFTF to enable a small amount of flow holding.[25] Variable inlet guide vanes operate by varying the inlet axial velocity. As can be seen in the vector diagrams in Figure 7, closing the vanes produces a smaller value of axial velocity and thus a smaller mass flow. Similarly, opening the vanes away from the nominal position produces a larger axial velocity and thus higher mass flow. This follows from Eq. (1) where *c* is the axial velocity component.

$$\dot{m} = \rho c A \qquad (1)\,.$$

By varying the axial velocity through changing the IGV angle, one can effectively vary the mass flow through the component. In addition to enabling cycle transition, VIGVs offer better performance at off design, surge margin control, and increasing the overall compressor range of operation.

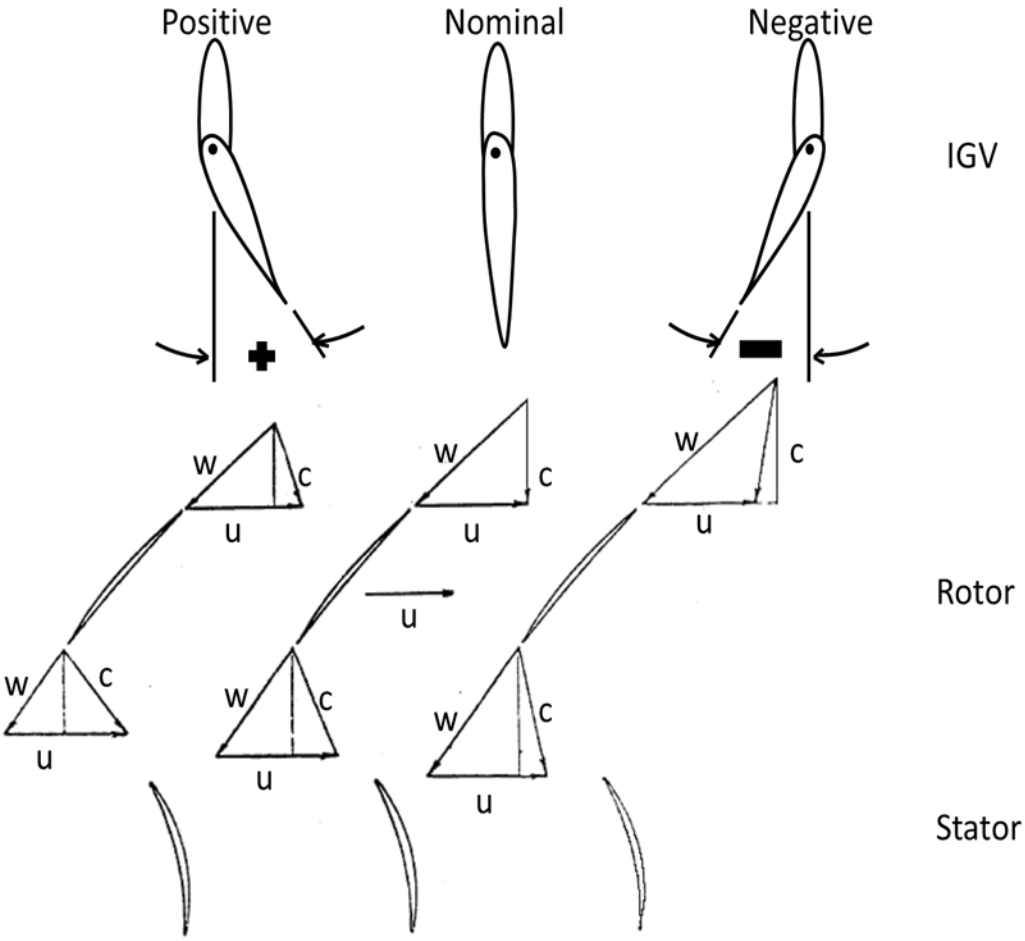

Figure 7: Vector Diagram for VIGV at Constant Rotor Incidence and Deviation Angle

2.1.2.4 Variable Nozzle Area Turbine

Variable area turbines primarily attempt to "change the speed-speed relationship of the high and low rotors in order to enable optimal engine efficiency over the entire flight envelope", thus allowing OPR to be controlled as turbine inlet temperature varies to match the fan power demand.[19] They also allow independent control of high and low rotor speeds and provide increased cycle matching capability. This variable geometry occurs at the inlet of the turbine and can be attained two ways. The first method involves re-staggering the stator blades and acts in a similar fashion to VIGVs. The second method is mechanically simpler and involves introducing an obstruction into the flow or by introducing secondary airflow.[35] Use of variable geometry in

an HP turbine is seen as a large technology risk but use in an LP turbine can lead to modest alteration of bypass ratio, TSFC, and specific thrust at subsonic speeds. The specific improvement depends largely on the cycle engine design.[35]

2.2 Current Paradigms in Aero Propulsion Design and Cycle Analysis

Cycle analysis studies the thermodynamic changes of the working fluid as it flows through the engine.[26] It can be broken into two types of analysis: parametric cycle analysis (on design analysis) and performance analysis (off design analysis). The main goal of on design analysis is to relate performance parameters to design choices. Off design analysis determines the performance of an engine with fixed design choices at all flight conditions.[26] These flight conditions are points within the operational envelope defined by some combination of environmental conditions, flight Mach number, and throttle setting. The points are selected at key segments in a vehicle mission profile, thus the design of an aircraft engine is a compromise amongst the various operational requirements of the mission it is expected to fulfill.[27]

A single design point (SDP) approach is typically instituted for the sake of simplicity. This approach is appropriate for some gas turbine applications and for providing a basic understanding of the thermodynamic performance trends for various engine architectures. SDP is not aimed at matching an engine to meet a set of performance requirements. Thus many of the texts available on cycle analysis do not take into account requirements or technology until after the analysis. "In many of the texts, mention is made of the need for a cycle design to meet requirements at multiple operating conditions however for simplicity, a single design point approach is instituted and a process to meet all of the requirements at different operating conditions barely discussed."[27] The selection of a single aerothermodynamic cycle design point is a difficult yet important part of cycle analysis. But, an engine designer must recognize the differing requirements for takeoff, climb, cruise, maneuvering, etc.

The actual calculation of the cycle is an iterative process requiring an initial estimate and typically utilizes a steady-state mathematical simulation of an engine to provide internal and external flow characteristics as a function of operating conditions and power setting.[43] The basic elements of the cycle calculation for a conventional commercial turbofan is illustrated in Figure 8. The calculation iterates from the initial estimate of those parameters which are undetermined and are necessary to complete the calculation in order to solver for the internal flow and energy

balance. Error functions are used to represent the degree of unbalance in each iteration. Each iteration successively perturbs each parameter and the resulting unbalanced cycled used to calculate a matrix of partial derivatives of each error with respect to each variable. This matrix is then used to calculate the changes in the variables which will drive the unbalanced cycle towards a balanced solution.[43]

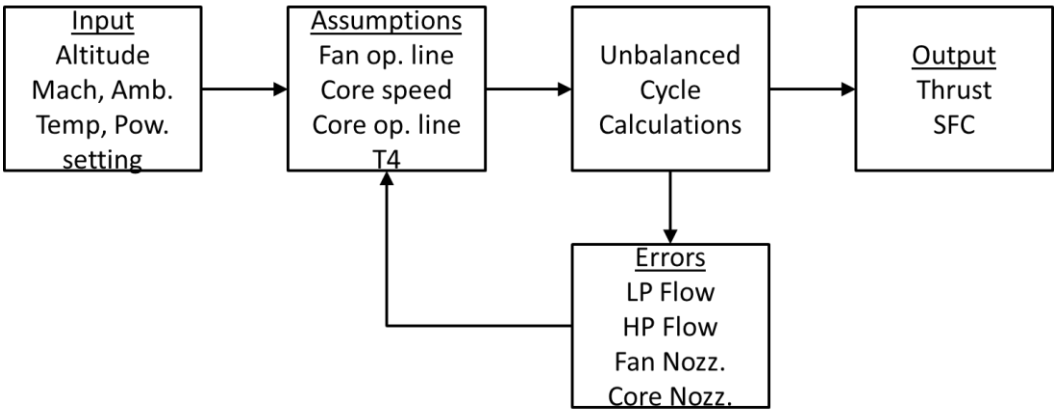

Figure 8: Turbofan Cycle Calculation[43]

There are a number of studies showing that methods using a single design point are deficient in their ability to choose an optimum gas turbine engine cycle for a given set of requirements. This is due to the assumptions about the cycle made a priori resulting in an engine that is oversized and suffers a performance penalty in order to ensure feasibility at off design operating conditions. The deficiencies are only compounded as the complexity of the cycle increases, leading to designs that are underperforming at one or more points in the flight envelope.

As already stated, an SDP methodology has several limitations which reduce its usefulness as a tool for designing and matching an engine to meet a set of performance requirements.[27] Multi design point methodology aims to provide the needed tools. The MDP method is not intended to improve the accuracy of performance predictions or work as an optimizer. The intent of MDP methodology is to help the designer ensure the feasibility of all the cycle designs for a particular application. It does so by adjusting the design to simultaneously meet the performance requirements and constraints at different operating conditions. Due to necessary a priori

assumptions when implementing SDP, the engine is oversized and thus suffers a performance penalty in order to ensure feasibility at off design operating conditions.[2,35] Assumptions are made about the design variables, how they interact, and how they vary at off-design as a function of many possible variables. This leads to the third observation, which should be considered for any engine optimization problem:

Observation 2: "Previous studies have shown the deficiencies of Single Design Point methods to determine an optimum gas turbine engine cycle for a given set of requirements." [2] An MDP method always meets the performance requirements throughout the operating envelope ensuring a properly matched engine and aircraft.

It has also been shown that the benefits of MDP over SDP are further enhanced as the complexity of the cycle or number of requirements increase.[2,27] However, it should be emphasized again that the MDP method is not intended to improve the accuracy of performance predictions or work as an optimizer. It is intended to ensure the feasibility of all the cycle designs for a particular application. It does so by changing the topography of the cycle design space, in comparison to SDP which only can change the feasible boundaries of the cycle design space, thus reducing the available designs for a given set of requirements.[7]

The selection of the points to include in the cycle analysis is strongly dependent upon the vehicle mission. However, some general knowledge about possible design points of interest should help identify the point or points necessary to include in analysis.

The Top of Climb (TOC) point is derived from the requirements and constraints and as the name suggests, it is the highest point reached at the end of the climb phase of a mission. It is used to specify the highest referred fan speed and flow for a thrust requirement at a given flight condition (Mach and altitude). This determines the mass flow schedule. TOC can be further subdivided into a subsonic point and a supersonic point, usually at a higher altitude.

The Sea Level Static (SLS) point may be considered because it is at this point where the engine is manufactured and tested, thus it serves as a common point for comparison of various engine designs. Also, it is generally the point used for emissions testing. SLS can be further subdivided into an installed and uninstalled point.

The takeoff (TKO) point is also derived from requirements and constraints. It is used to

specify the $T_{4,max}$ for a thrust requirement at a given flight condition. It determines the T_4 schedule. TKO should be considered at ISA standard day conditions and the more constraining high hot day condition.

The Cruise point is probably where the engine will operate most of the time and is where the engine will be at a part power setting. It should be ensured that the operation at cruise is as efficient as feasible. Again, this point could be further subdivided into a subsonic and supersonic point.

The Transonic breakthrough point is where the drag is at its highest. The thrust must be high enough here to enable the aircraft to punch through this high drag region into supersonic flight – if supersonic flight is required by the mission.

Finally, there should be a reference point for the turbomachinery components on each shaft. The LP point (LPCDP) is used to incorporate the technology rules for the fan and LPC. It is also used to find the cruise operating line for the fan and LPC. The HP point (HPCDP) is used to ensure the engine core remains fixed for performance and geometry for all engine designs. Additionally, the HP and LP points are actually interconnected making one a function of OPR. For example, by setting FPR the HPC corrected speed is adjusted as a function of this FPR to meet the required OPR.

2.3 Optimization Techniques for an Engine Model

The basic elements of any optimization technique generally start with the selection of a starting point. From there, a direction must be found from the starting point to a point of improved system performance. Then a step size must be determined in the chosen direction. Finally, the new point must be evaluated to determine if it is accepted as the optimum. If not, the process repeats. This process repeats until either a predefined acceptance criteria is met or a maximum number of iterations is reached.

The simplest approach to minimizing a function is to randomly select a large number of variable values and evaluate the result for each. This type of method only requires function values in searching for a minimum and is referred to as a zero-order method. These types of methods are generally not considered efficient or reliable.[39] However they may be better able to handle cases with local minima and functions without nice derivatives. One very popular zero-order method is the genetic algorithm. In concept it is quite simple and is basically a refined

random search method. The main attraction of genetic algorithms is their improved probability of finding a global optimum and the fact that they operate directly on the function of interest and do not require computing a derivative. This is attained at the cost of a relatively large number of function evaluations.

Usually a more efficient approach to an optimization problem is to use gradient information. Use of such information limits the search to a specific direction but may require many calculations of the gradient if the function is highly non-linear. Methods that use the gradient are called first-order methods. First-order methods can generally be expected to perform better than zero-order methods but often perform poorly for functions which have discontinuous first derivatives. Methods using second derivative information are then called second-order methods. However second derivative information is seldom available analytically and even if it is, it is computationally costly to compute numerically – reducing or eliminating any efficiency gains.[39]

Choice of a suitable search algorithm for engine optimization is important and to date, there has already been some research into optimizing an aerothermodynamic cycle. One of the simplest methods is a simple brute force grid search. The search can be used to explore an n-dimensional optimization space where each optimization parameter is varied between its upper and lower bounds by some step size. The objective function is then calculated at each grid point and compared against other points in the grid. This method was demonstrated successfully for an MFTF, and while it is reliable, it is extremely inefficient and becomes increasingly time consuming as the number of optimization parameters and design points grows.[19]

Simmons utilized a genetic algorithm and a complicated nested structure involving ModelCenter for finding an optimal and balanced VCE cycle, shown in Figure 9.[1] A genetic algorithm is a random search technique and requires several nested loops to search the on and off design space as well as modification of the thermodynamics package to repair the internal solver when the algorithm chooses bad points.

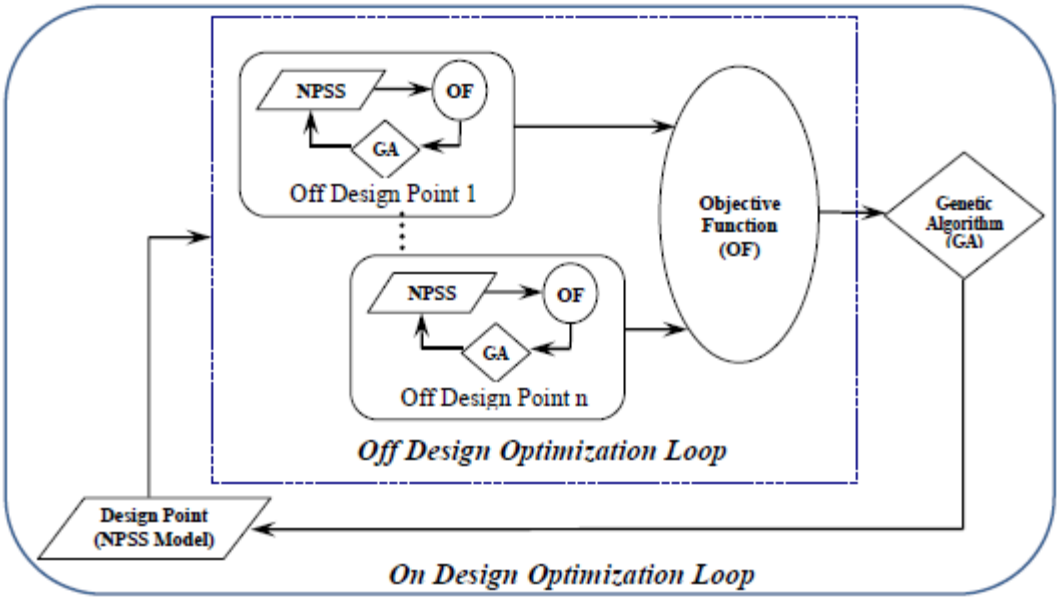

Figure 9: Nested Optimization Loops for Genetic Algorithm[1]

Another approach is that of the nested optimizer/solver approach, which was successfully utilized for a core driven fan stage VCE.[19] This approach allows the independent variables to be used to balance the cycle while the optimization variables are left as fixed inputs. The chosen optimizer is wrapped around the solver and varies the optimized parameters to attain optimal engine performance. Constraints can be handled either in the inner structure or the outer optimization structure. In Reference 19 the Fletcher-Reeves method was coded directly into the thermodynamic analysis package. The algorithm is a conjugate-gradient method for optimizing an n-dimensional function. The nested optimizer/solver approach is conceptually simple but relatively inefficient because it requires the solver to balance the cycle for each iteration of the solver requiring many model passes.[19]

The nested solver approach is similar to the previous approach except that it directly uses the solver to find the optimal settings. This is done by calculating gradients of the objective function and then using the solver to drive the gradients to zero as well as driving the cycle balance error terms to zero. To do this, a high level solver is used to optimize the control inputs by driving the derivative of the objective function with respect to the control inputs to zero.[19] This requires a nested solver to re-balance the cycle at each derivative perturbation. Obviously, this method requires some means of acquiring the derivative of the objective function. Fortunately, most

cycle solvers use a Newton-Raphson or Broyden algorithm which typically require the calculation of gradients in the process of matching the cycle. This is similar to the information needed by the nested solver approach, which has led to the development of methods designed to directly use Newton type solvers and their derivative information to optimize a cycle.[19]

In fact, Brown developed two multi-variable cycle optimization techniques specifically for optimizing part power performance capabilities of VCEs.[43] One is the internal gradient approach which uses an optimizer to satisfy both the cycle balance and optimization. This is done by representing the cycle balance equality constraints as pairs of inequality constraints. Then the errors are constrained to be less than or equal to zero while simultaneously being constrained to be greater than or equal to zero.[19] The internal cycle balance requirements are combined with the external optimization constraints, all of which are expressed as pairs of inequalities to constitute the total system independent variables for the internal gradient method.[43]

The gradient integration approach involves using a numerical integration technique to push toward reaching an objective gradient value of zero. Its implementation is a cross between a Runga-Kutta numerical integration routine and a steepest descent optimization routine.[19] The method makes use of the derivative information produced as a byproduct of the cycle balance process to update the search direction after every cycle balance point enabling it to find an optimum in the minimum number of model passes.

Several of Brown's ideas are extremely appealing leading to the second observation:

Observation 3: The use of derivative information that is already produced as a byproduct of the cycle balancing can lead to a fast and efficient method. If the derivative information could be used along with the solver, theoretically one could drive the derivatives to zero while simultaneously balancing the cycle.

The remaining piece is then to provide the solver with the derivative of the optimization response with respect to the optimization variables. This would result in solver based optimization, or SBO. A Venn diagram of the commonalities and differences of the cycle balance and optimization processes is illustrative in showing why it is advantageous to develop a method which combines the two. Aircraft engine cycle design and optimization have several key features which the proposed methodology will address, many of which overlap. Both must in

some way determine the direction to adjust the available variables to reach a solution and must also be able to find an appropriate amount to vary each variable. Both are iterative processes which must be given some starting point, or initial guess. Finally, both must be able to handle any limits placed on the engine system. Thus the only remaining piece of information needed to include optimization in the cycle balance process is computation of the derivative information.

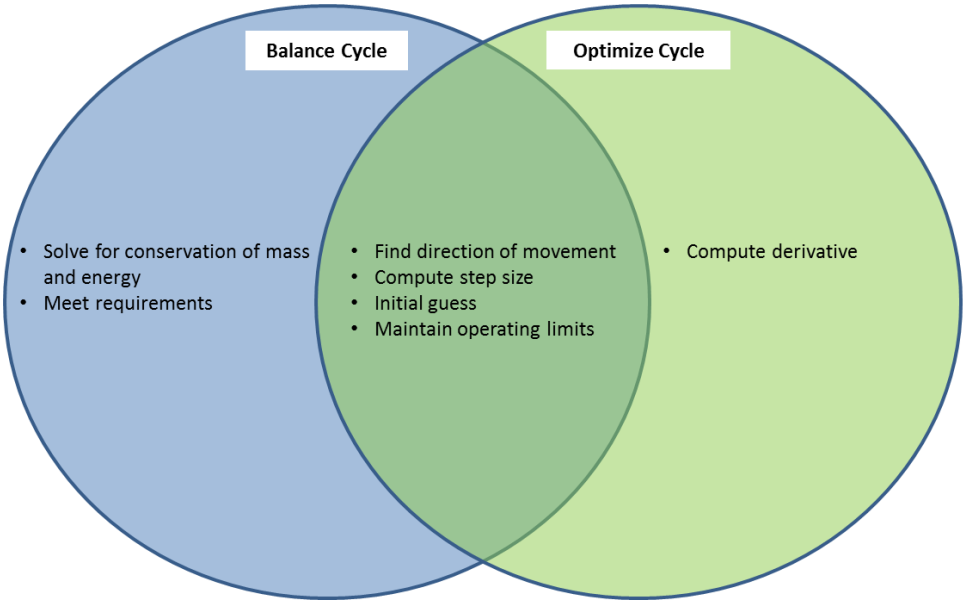

Figure 10: Cycle Design and Optimization Venn Diagram

2.4 Research Questions and Hypotheses

Variable geometry was once seen as the future of aircraft engines and has since become a major player in the engine improvements seen over the past few decades. Variable cycle engines are now envisioned as the future of aircraft engines, especially for military fighters. Analysis of the benefits these engines afford and accurate performance prediction is necessary if these engine architectures are to be fully exploited. However, the design and analysis paradigms that exist in aero propulsion system design are admittedly problematic. With more complex engine architectures being pursued, the problems are only compounded. Additionally, the added degrees of freedom offered by variable geometry above what is required to balance the cycle implies optimization is possible. This leads to the problem statement that this thesis addresses.

Problem Statement: The determination of the full benefit of variable geometry/variable cycle technology is hampered by the added complexity inherent in variable geometry/variable cycle designs, and the current design and analysis paradigms in the research community.

During the research of variable cycle engine architectures, or any engine architecture that utilizes variable geometry components, an obvious question comes to mind. How are the geometries of the components set? In many advanced aircraft engines today, the use of variable geometry is not intended to change the thermodynamic cycle of the engine but simply to improve the engine performance.[35] Setting VG components is made more complicated for a VCE by the fact that the VG components are now also used to enable cycle variability. Thus, the question just mentioned must first be answered before any meaningful analysis can be performed on variable geometry engine architectures, and leads to the first research question.

Research Question 1: How can optimal settings be found for engines incorporating variable geometry in a way that is efficient and robust enough to handle many configurations quickly?

This question focuses on finding optimal and balanced cycle designs for architectures using variable geometry. This leads to the first two hypotheses:

Hypothesis 1: Combining a Newton-Raphson type cycle solver and local linear model for derivative information should result in a gradient based optimization method that can be used to find an optimum of an engine model with user selectable input parameters.

Hypothesis 2: The method developed will be able to optimize an engine cycle while at the same time balancing the cycle and maintaining all operating limits.

These hypotheses address the possibility for large efficiency gains by combination of the optimization with the cycle solver. With the assumption of differentiability, a local linear model may be created which can create derivative information used to drive an objective function to an optimum. Specifically, the derivatives produced will be driven to zero – relying on an elementary

principle of calculus which says that a minimum or maximum of a function will be found when the derivative is exactly zero. The solver can be set up in such a way that the derivative information at each pass through the model can be included to drive the derivatives to zero while simultaneously balancing the cycle.

The ability to find an optimum for an engine cycle in this way is extremely powerful. However, there still exist some problems. Simmons states "the amount of flow variation from the core to the second and third streams is a strong function of the design point selected. Therefore to be effective, engine optimization must simultaneously investigate both the on design search space and the associated off design search space."[1] Aleid states that there are several design points that may size a VCE and asks which one of the points is the most important for designing a VCE.[35] This leads to the second research question:

Research Question 2: Can an optimization method be developed which generates optimized aerothermodynamic cycles and includes all desired cycle performance requirements and constraints at multiple design points?

This question focuses on the need to find an optimum for an engine cycle while still being able to meet all performance requirements and constraints. Additionally, multiple design points may size an engine.

Hypothesis 3: Use of SBO with MDP would combine on design, off design, and optimization into a single, general, and simultaneous implementation which will produce a cycle design that is optimum while still being feasible at all design points, meeting all performance requirements, and not exceeding any constraints.

There is an added appeal when using an approach that only uses the solver along with MDP, which itself ties the on and off design phases into a single simultaneous implementation within the solver. MDP is an elegant solution to ensuring that an engine meets all requirements at all design points. Use of SBO only requires appending the derivative information to the solver and corresponding independent/dependent combinations to drive the derivatives to zero. Thus the solver does not "see" these added solver variables as anything other than something needed to

balance the cycle when in fact they are used to optimize the cycle.

As already mentioned, the objective function is a complex function of many design variables. MDP introduces the possibility to add variables specific to each design point included. Combining SBO with MDP also allows the designer pick at which design point to optimize the objective. This obviously has a great deal to do with the purposes and requirements of the designer, however an optimized MDP (OMDP) method will address many of the problems experienced with other optimization techniques and leads to the next hypothesis:

Hypothesis 4: OMDP will always produce feasible designs and the available derivative information may be utilized for single or multi-variable optimization on one or more responses at one or more design points.

Hypothesis 4 addresses the fact that what is considered as the optimum can vary based on the intentions of the designer. In any case, it must be demonstrated that OMDP can produce fully feasible designs no matter the choice of optimization point.

Generally it has been found that the MDP method works best when given an initial starting point in the feasible region, i.e. the initial iterate is itself a solution to one of the candidate engines within the cycle design space. Because of this, the optimization will also start in the feasible region of the design space. This does not however mean that this starting point will allow the optimization to find the truly optimal point. A poor choice of starting point may in reality cause the optimizer to fail altogether.

Hypothesis 5: The ability of OMDP to converge to the optimal, balanced cycle will be insensitive to starting point.

The design space represents an n-dimensional space that may consist of local minima and infeasible regions which can cause the optimizer to get stuck or fail altogether. Considering that MDP will always start at a solution to one of the candidate engine cycles, and assuming that the space is relatively smooth, it is reasonable to also assume that OMDP will be relatively insensitive to the choice of starting point for the VG components as long as the cycle itself is feasible at the start.

CHAPTER 3

RESEARCH FORMULATION

The objective of the optimization method here developed, SBO, is to simultaneously balance and optimize the aerothermodynamic cycle. The benefits an MDP method provides has also been discussed. Therefore SBO should be left general enough to be used for a wide variety of applications, including its incorporation into MDP. This represents a simplification and improvement of optimization for aircraft engines in the on design and off design search spaces. This chapter lays out the mathematical and practical formulation for the optimization technique developed. It also provides a brief overview of MDP. It should be noted that his methodology focuses on SBO does not go into detail on creation of the cycle design relations. It is assumed that the cycle analyst is already familiar with the cycle design process and has all the information required to perform the cycle analysis in the absence of optimization.

3.1 Solver Based Optimization Overview

A broad overview of the methodology for this thesis is found in Figure 11. Typically the data generated from on-design cycle analysis such as geometry areas and design pressure ratios are inputs for off-design analysis. In off-design, the performance of each component is determined from engine component performance maps scaled around a design point. The location on the maps are unknown for a given flight condition so an iterative process must take place to determine engine cycle performance. This ”matching” is done by means of a thermodynamic model that must satisfy continuity and conservation of energy to determine the pressure ratio, airflow, rotor speed and efficiency.[27]

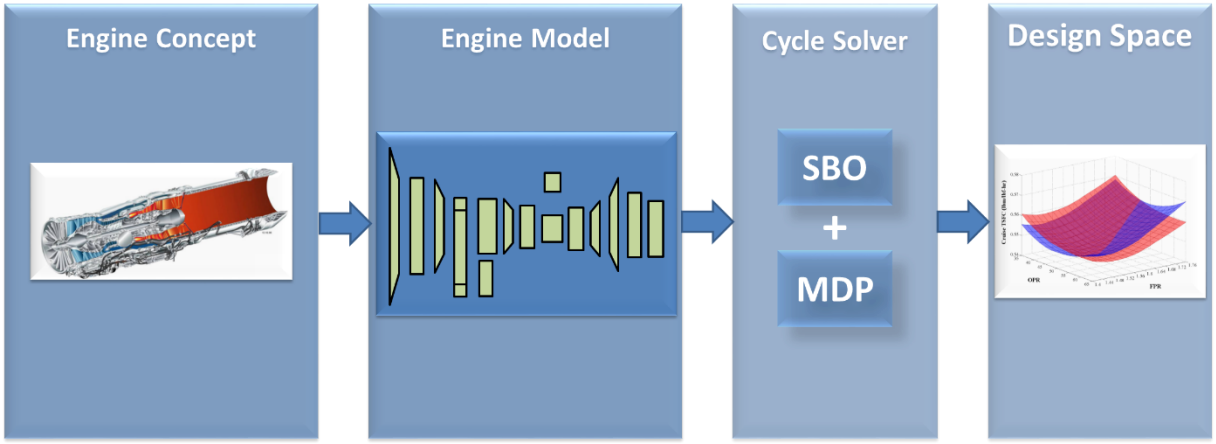

Figure 11: Methodology Overview

The method chosen for finding the solutions is a form of Newton's method. MDP allows many design points to be included in the analysis to ensure that all performance requirements are met at all design points without violating any constraints. This is a more elegant approach than separate on and off design analyses. However, as previously stated, MDP methods are not intended to improve the accuracy of the performance prediction. The accuracy is dependent on the quality of the cycle model and performance maps. Finding improved performance estimates, ie optimization, is the role of SBO. Thus combination of SBO with MDP will result in an optimal solution that is fully feasible, and meets all requirements and constraints at all design points of interest.

It will be useful to first define the parameters incorporated in the cycle design and optimization process. The first category is the cycle design variables. These are varied to create the Cycle Design Space (CDS). Cycle design variables define the performance and include, BPR, FPR, and OPR. By definition design variables are independent cycle parameters which the cycle analyst has complete authority to set.[27] The next category of parameters is the operating conditions that define the different design points. Each unique combination of operating conditions will define a new design point. The next two categories are the cycle independent and dependent functions. The independent parameters are those used to reach desired performance targets specified by the dependent functions.[2] The fifth category of parameters are the constraints. These are limits placed on the cycle which the engine may not exceed. Generally, a constraint is composed of both technology and performance limits. Each constraint is related to

one or more of the dependent parameters. The final category of parameters are the optimization variables. These are variables selected by the user to minimize or maximize the objective function. They represent additional degrees of freedom in the system above which are required to satisfy conservation of mass and energy.

3.1.1 SBO Process

Incorporation of SBO into the solver is in general no different than setting up the solver for balancing the cycle, which is one of the main advantages of the method. The main point of difference is the linearization of the nonlinear model to obtain the derivative information. The flow of information for SBO, shown in Figure 12, can be broken into 3 parts. The first part identifies the objective function to be optimized and the parameters available to optimize the function. The next part prepares the engine cycle for analysis and optimization, as well as set up of the linearization object. The final part performs the cycle analysis and optimization for the chosen combination of design variables.

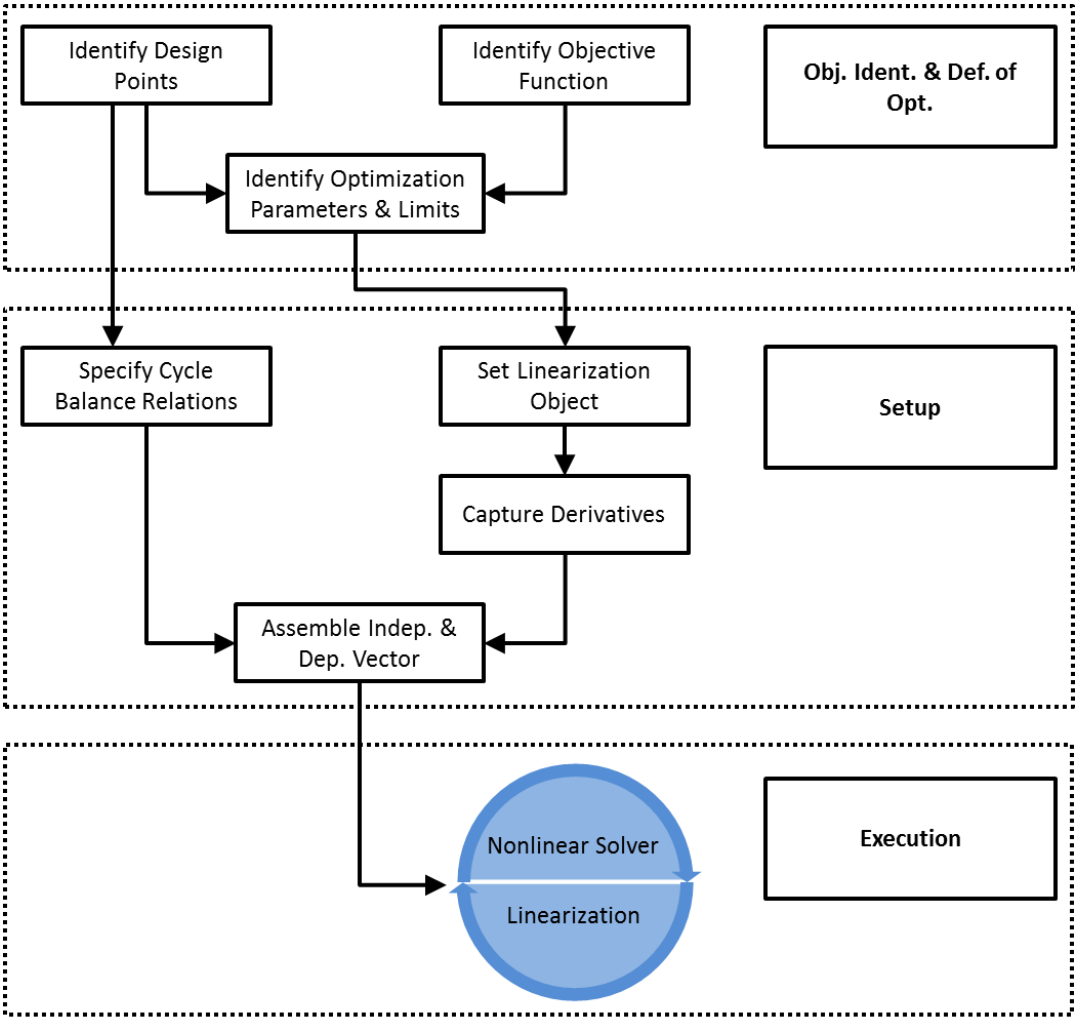

Figure 12: Process Flowchart for Solver Based Optimization Method

3.1.1.1 Objective Identification and Definition of Optimum

The first part of SBO first involves identifying the objective functions to be optimized. The objective function represents the system performance parameter to be maximized or minimized. Obviously the function identified must be attainable from the cycle evaluation process. Examples of typical performance objectives for aircraft engines include SFC and specific thrust, and are usually available from any cycle analysis. This process should also include identification of the parameters which can be used to optimize this function and any limits which must be placed on these parameters. These limits may include physical limits such as a maximum or minimum physical area for a VG exhaust nozzle, or limits placed on the cycle such as a maximum fan diameter as a function of BPR. Once the objective function is identified, it must be determined

whether this function is to be minimized or maximized. Finally, the design point(s) must be selected. For SDP, both the objective function and optimization variables must be at the same point. For MDP, the objectives and variables may be at any point.

3.1.1.2 Setup

This step assumes that the cycle design relations and flight conditions have already been specified. The cycle design relations are created by selecting cycle variables for use as independent parameters to be controlled by the solver to satisfy the target requirements. These include design variables needed to balance the cycle, performance requirements, and constraints. With this assumption, the linearization object must be set up next. The input and output variables are added to the object, which are the previously identified optimization variables and objective functions, respectively. Linearization then takes place at the current state of the model, producing the needed derivative information.

This part of SBO involves adding the optimization variables identified in the previous step to the solver. The advantage of this method lies in the fact that the solver itself does not need to be modified in any way. To the solver, the additional variables look like any other variable. However these variables must use an absolute transformation since they will by definition be operating close to zero. Any constraints on the optimization variables are then attached to the dependents, thus instructing the solver how to proceed if a constraint is violated.

Finally, all cycle design relations and optimization relations have been specified. The relations appear identical to the solver – the cycle design relations use the independent parameters to satisfy target requirements and the optimization relations use independent parameters to drive the derivatives to zero. This is done such that each independent parameter is linked to achieve a target requirement or zero derivative so that each requirement and derivative is guaranteed to be a function of at least one independent parameter. However, the independent parameters must be unique and two dependent parameters cannot be linked to the same independent parameter. While many dependents may be functions of multiple independent parameters, only one unique independent parameter can be used by the solver to control the value of a dependent parameter.

3.1.1.3 Execution

This final part of SBO requires two key pieces. The first is the cycle design relations, and the second is the derivative information provided by the linearization object. The linearization object should be used at each new iteration of the solver to produce derivatives accurate at that point. Thus the execution represents a cyclic process where the solver for the nonlinear model and the linearization object run in series. If a specified maximum number of iterations has not been reached then the final iteration the solver will represent a converged point. This means that all cycle design relations have been satisfied and the linearization object at this point produces derivative values equal to zero, unless a constraint has been reached.

3.1.2 MDP

MDP can be broken into three phases. The first establishes the design problem to be addressed. The second phase organizes the information from the first phase and prepares the engine cycle for analysis. The final phase performs the analysis.

3.1.2.1 Requirements and Technology Definition Phase

This phase establishes the design problem to be addressed and sets the level of technology to be incorporated; i.e., it establishes the performance requirements and technology rules. The cycle performance requirements are specific requisites defining the expected performance characteristics of the cycle throughout the operational envelope. An example would be the requirement of 50,500 lbf (wet) thrust at SLS conditions.

Technology rules describe how the technology parameters change as a function of the cycle design variables. Component performance estimates are functions, tables, or maps which estimate component technology parameters for a given technology level as a function of design variables. Technology limits are constraints established by the technology level which cannot be exceeded.

3.1.2.2 Setup Phase

The Setup phase organizes the information from the R&TD phase and prepares the candidate engine for cycle analysis. Each design point of interest is composed of a unique set of operating conditions. Here also occurs the creation of the design rules and selection of design variables

available to create the CDS for the chosen engine architecture. It is the design rules that establish how the design points, performance requirements, technology rules, and design variables link together. A system of nonlinear equations is created that includes the cycle design relations, component matching relations, and constraint relations by use of an auto setup. This adds to the solver the variables needed to satisfy continuity and conservation of energy and by user selection of cycle variables for use as independent parameters. Some care must be taken when choosing these variables, but if chosen correctly the modified Newton-Raphson solver has all the inputs required.

This is the most complex phase of MDP. Extreme care must be taken to correctly create the design rules and solver setup. This is different than SDP because a constraint or dependent/independent linkage can exist across design points. Not setting this correctly will cause erroneous results while making it appear that MDP is working since the model itself will work fine.

Additionally, the MDP method postulates that an initial iterate can be used to find solutions to the candidate engine cycles provided that the initial iterate is itself a solution to one of the candidate engine cycles.[27] For a new MDP model, this initial guess is created using a standard SDP approach. Usually the design point chosen is the map scaling point from MDP. Then the cycle is designed using the SDP model and recreated at other design point operating conditions in off-design analysis as best as possible. From the on and off design points the initial guess for the values of the independent parameters may be obtained for MDP. If an MDP model already exists, and only requires some modification or addition of design points, the MDP model along with the original initial iterate may be used to obtain the new independent parameter values. [27]

3.1.2.3 Execution Phase

Finally, the Execution phase performs the cycle analysis to generate the CDS. This involves incorporation of design variables and initial values for each of the independent parameters in the solver. The execution of the solver finds a solution that sets the engine cycle; the engine is sized to meet all of the design criteria.

3.1.2.4 MDP Initial Iterate

The selection of an initial iterate is an important part of the method. A poor initial iterate will

more often than not fail to converge. If it does the number of iterations required to converge will be significant. MDP postulates that a single initial iterate can be used to efficiently and robustly find solutions provided that the iterate is itself a solution to one of the candidate engines within the cycle design space.[27] For a newly built MDP model, this initial iterate can be found by utilizing a single point design cycle model coupled with off design analysis. First one of the design points is selected for the SDP model. This is usually the map scaling point. Then the cycle is designed using the SDP model by specifying the design variables at that point pertaining to the point within the MDP cycle design space. The other design point operating conditions are recreated in off design analysis as closely as possible to their MDP setup. From the on design and off design analysis of the SDP cycle, the values of the independent parameters can be obtained and used as the initial iterate for MDP.

This method may not produce a perfect initial iterate due to the assumptions required for SDP. However this method will provide a starting point that is good enough. Once the MDP model is running, it can then be used to create better initial iterates if any modifications to the original setup are necessary.

3.2 Theoretical Framework

This chapter provides the background information necessary to understand the technical aspects of the optimization problem associated with designing and optimizing an aerothermodynamic cycle.

Table 1: SBO Theory Nomenclature

Symbol	Description
α	Design variable
β	Design point operating condition
f	Cycle dependent function
F	Vector of dependent functions
g	Constraint parameter
h	Function of one variable
i	Index for dependent functions
J	Jacobian matrix
L	Minimization function
m	Index for decision parameters
n	Index for state parameters
q	Index for optimization equations
u	User defined decision parameter
x	State parameter/ cycle independent parameter
ξ	Optimization equation
X	Vector of cycle independent parameters
y	Cycle dependent parameter
Z(X)	Vector of constraint functions
$\hat{y}$	Calculated dependent value

3.2.1 Thrust and Efficiency

The optimization methodology will be operating on a parameter commonly used to assess engine performance. This response parameter is thrust specific fuel consumption. To properly understand the use of this parameter, it is here defined. Hill and Peterson (Reference 29) provide excellent explanations, summarized below.

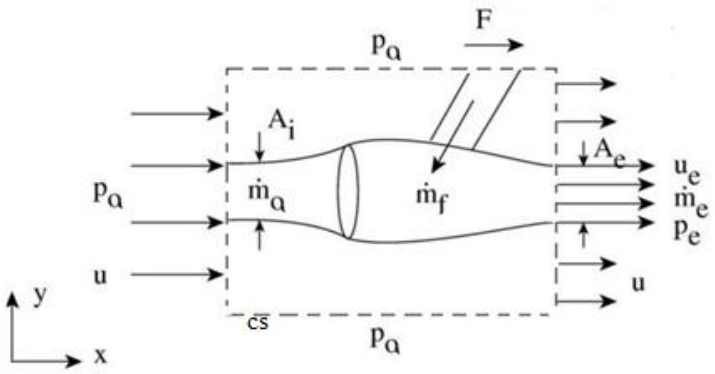

Figure 13: Generalized Thrust Producing Device[29]

Engine thrust is the vector summation of all forces on internal and external surfaces of the engine and nacelle. The thrust of a generalized thrust producer can be found using Newton's second law and a control volume, shown as the dotted line in Figure 13. Assuming steady flow and reversible external flow, the sum of the forces acting on the control surface in the *x* direction reduces to

$$\sum F_x = (P_a - P_e)A_e + F \tag{2}$$

The net efflux is

$$\int_{cs} u_x \rho(\boldsymbol{u} \cdot \boldsymbol{n}) dA = \dot{m}_e u_e - \dot{m}_a u \tag{3}$$

The momentum equation for the flow carried out by the control volume becomes

$$F = \dot{m}_e u_e - \dot{m}_a u + (P_a - P_e)A_e \tag{4}$$

The term $(P_a - P_e)A_e$ is only not zero if the exhaust jet is supersonic and the nozzle doesn't expand the exhaust jet to ambient pressure. Assuming $\dot{m}_e \cong \dot{m}_a = \dot{m}_0$, the specific thrust is then

$$F/\dot{m}_0 = u_e - u \tag{5}$$

The propulsive efficiency is a measure of the effectiveness with which the propulsion system is propelling the aircraft. It is the ratio of thrust power to the engine mechanical power required to generate said thrust. This ratio can be expressed as

$$\eta_p = \frac{Fu}{\dot{m}_0\left[(1+f)\left(\frac{u_e^2}{2} - \frac{u^2}{2}\right)\right]} \quad (6)\,.$$

Now assuming that *f* is much less than 1.0 in Eq. (6), and that the pressure term in the thrust equation is negligible, the propulsive efficiency can be written as

$$\eta_p = \frac{2u}{u_e + u} \quad (7)\,.$$

Looking at the equations for specific thrust Eq. (5) and propulsive efficiency Eq. (7), it is easy to see where one problem arises that the VCE is attempting to solve. The thrust equation shows that the exhaust velocity must be greater than the free stream velocity and the efficiency is maximized when $u_e = u$. However as u_e approaches u the specific thrust is practically zero and the engine required to produce a finite amount of thrust would be infinitely large. This obviously is not realistic and thus it is not practical to maximize propulsive efficiency. Therefore, other parameters are required to evaluate an engine's overall performance.

The simplified thrust equation suggests two obvious ways to attain thrust. The first is to attain a high exhaust velocity. This is the approach taken by the turbojet which moves a relatively small amount of mass through the engine and exhausts the mass at high velocity – providing a high specific thrust for a given thrust value.

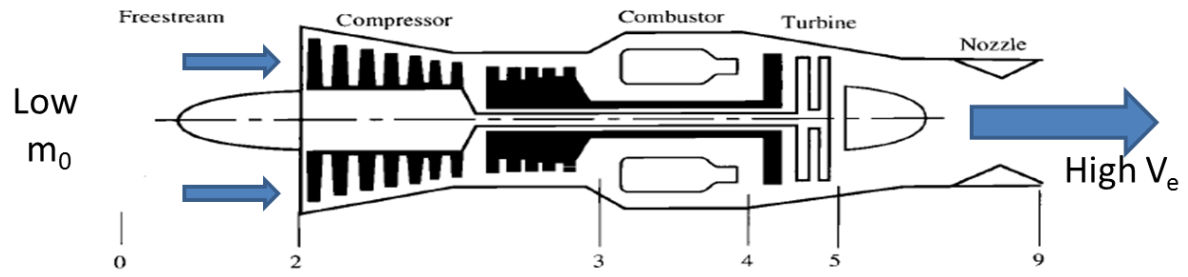

Figure 14: Typical Turbojet configuration [26]

The second approach is to move a relatively high amount of mass flow through the engine and exhaust it at a lower velocity. This is the approach taken by the turbofan.

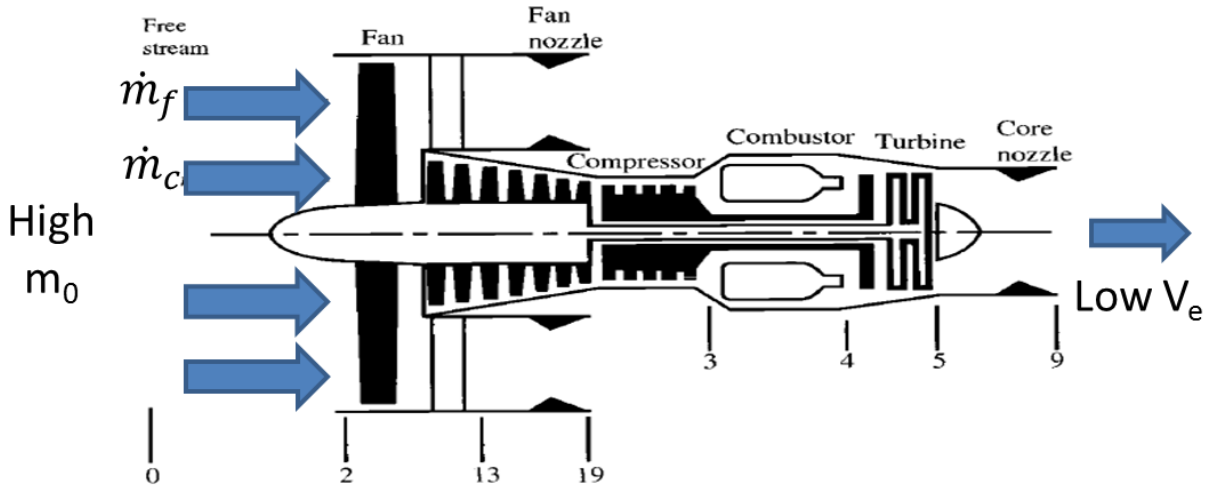

Figure 15: Typical Separate Flow Turbofan configuration[26]

Defining the Thrust Specific Fuel Consumption as

$$TSFC = \frac{\dot{m}_f}{F/\dot{m}_0} = \frac{f}{(1+f)u_e - u} \quad (8)\,.$$

it is easy to see why a high BPR turbofan has better fuel consumption characteristics than a

turbojet. Size is an issue for a military fighter aircraft for various reasons including weight and stealth. This directly implies a strict size limit on the engine and in some cases it isn't practical to use large mass flow to increase the thrust. That leaves option one – using exhaust velocity.

Substituting Eq. (5) into Eq. (7), and performing some rearrangement yields an equation for propulsive efficiency in terms of specific thrust and flight velocity:

$$\eta_p = \frac{1}{1 + \left(\frac{u}{2}\right)\left(\frac{F}{\dot{m}_0}\right)} \quad (9).$$

This shows that propulsive efficiency and specific thrust are inversely proportional so at first glance it appears that high specific thrust and high propulsive efficiency cannot be attained simultaneously.

Specific thrust for an SFTF follows from the same analysis of a generalized thrust producer, with the exception that there are now two separate flows. Thus Eq. (4) becomes

$$F = \dot{m}_F(u_{e,F} - u) + \dot{m}_C(u_{e,F} - u) \quad (10).$$

In Eq. (10), $\dot{m}_F$ is the air flow passing through the fan and $\dot{m}_C$ is the air flow passing through the core. Specific thrust and thrust specific fuel consumption are then

$$\frac{F}{\dot{m}_0} = \frac{a_0}{1 + BPR}\left[BPR\left(\frac{u_{e,F}}{a_0} - M_0\right) + \left(\frac{u_{e,C}}{a_0} - M_0\right)\right] \quad (11).$$

$$TSFC = \frac{\dot{m}_f}{F/\dot{m}_0} = \frac{f}{(1 + BPR)(F/\dot{m}_0)} \quad (12).$$

3.2.2 Cycle Solver: Modified Newton-Raphson Method

The function of the cycle solver is to drive the model to a self-consistent, or converged state. The solver selected as the basis for cycle balance and SBO is a modified form of the Newton-Raphson method. This is a commonly used technique for solving (finding roots for) systems of nonlinear equations and widely employed in many gas turbine modeling software packages.[27]

The common use of the Newton-Raphson method with engine cycle solvers, and its ability to produce much of the information required for optimization, makes it an excellent option to incorporate an optimization routine.

Gas turbine engines generally consist of a number of coupled equations which cannot be solved for explicitly, requiring an iterative approach in which an initial guess is iteratively refined until a satisfactory solution is found.[34] These coupled equations consist of a set of independent variables the values of which completely determine the state of a set of dependent conditions. The solution to this system of equations requires a transformation of the set of equations to the standard form

$$F(x, \propto, \beta, g) = 0 \tag{13}.$$

Where *F* is composed of the transformed cycle dependent parameters and is a function of the cycle independent parameters, design variables, design point operating conditions, and constraint parameters. Typically the cycle dependent parameters are transformed into dependent functions whose solutions are zero using a relative transformation

$$f_i(x, \propto, \beta, g) = \frac{\hat{y}_i(x, \propto, \beta, g) - y_i}{y_i} \tag{14}.$$

Cases where the target value is close to zero require a different type of transformation to prevent division by zero. In this case it is necessary to use an absolute transformation for the optimization dependents as given in Eq. (15)

$$f_i(x, \propto, \beta, g) = \hat{y}_i(x, \propto, \beta, g) - y_i \tag{15}.$$

where $\hat{y}_i$ is the calculated value of the dependent parameter and y_i is the target value. The solution is reached when the error term is reduced to a sufficiently low value $\boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is a vector of length (u+m+q) composed of individual tolerance values that the error terms much meet or surpass to reach convergence. The vector length represents the number of user defined equations (u), plus the number of optimization equations (q), plus the number of engine match

equations (m). The user defined parameters are defined by the analyst to meet the performance targets at each design point, and also include the optimization parameters selected to optimize the cycle. The engine match relations are cycle variables selected to ensure the laws of continuity and conservation of energy between the different engine components for each design point.

Successive calculations of the independent parameters are required to solve the system of equations or until the method is stopped at some maximum number of iteration. The independent parameters, in vector form, are given by a vector of length (u+m+q) where

$$X^U = \begin{bmatrix} x_1^U \\ x_2^U \\ \vdots \\ x_u^U \end{bmatrix} X^M = \begin{bmatrix} x_1^M \\ x_2^M \\ \vdots \\ x_m^M \end{bmatrix} X^Q = \begin{bmatrix} x_1^Q \\ x_2^Q \\ \vdots \\ x_q^Q \end{bmatrix} X = \begin{bmatrix} X^U \\ X^M \\ X^Q \end{bmatrix} \quad (16).$$

The Newton-Raphson method determines new iterations of the independent parameters by

$$X_{n+1} = X_n + F'(X_n)^{-1} F(X_n) \quad (17).$$

At each iteration a local model of the function F is created and the root of the local model is found, until the method converges to the solution. Given a multi-dimensional problem the Jacobian matrix is used to create the local model instead of F'

$$J(X) = \frac{\partial (F)_i}{\partial (X)_j}(X) \quad (18).$$

Constraints are treated identically as dependents. Once the constraint is identified it is must also be transformed. During each Jacobian matrix calculation the solver also calculates the partial derivative of each constraint as a function of the independent parameters and uses the results from the calculation of Eq. (19) when a constraint is active.

$$Z'(X) = \frac{\partial (Z)_i}{\partial (X)_j}(X) \quad (19).$$

Thus for an active constraint, the element of *Z* corresponding to that constraint is placed in *J* for the dependent to which it is linked.

3.2.3 Local Linear Approximation

Assuming the objective function is differentiable, derivatives can be used to approximate nonlinear functions by linear functions.[50] In the sense of a local linear model, differentiability can be described graphically. If a function is differentiable at a point, then a sufficiently magnified portion of the graph of the function centered at that point takes on the appearance of a straight line segment.[50] For this reason, such a function is said to be locally linear and this fact forms the basis for SBO and its use of a local linear approximation.

A state space model may be created of the form

$$\dot{x} = Ax + Bu \tag{20}.$$

$$y = Cx + Du \tag{21}.$$

where *x* is the vector of state variables, *y* is the vector of system output variables, and *u* is a vector of system input variables. Each contains all variables at all design points of interest.

A linear model is generated about a point of the non-linear model by determining the sensitivity of each derivative and output variable to small changes in the state and input variable.[34] This is done by perturbing each state and input variable in turn, determining the changes in the state derivative and output variables in response to these perturbations, and then dividing those changes by the change in the state or input value. The resulting sensitivity terms are gathered into four matrices.

$$A = \frac{\partial(state)_i}{\partial(state)_j}, B = \frac{\partial(states)}{\partial(inputs)}, C = \frac{\partial(outputs)}{\partial(states)}, D = \frac{\partial(outputs)}{\partial(inputs)}$$

The D matrix contains the information that will be useful for optimization of the engine model. It contains the sensitivities of the output variables to the changes in the input variables. For example take TSFC as the desired output (objective function) and VG main exhaust nozzle area

as the input (optimization parameter). For three design points SLS, TOC, and cruise, the D matrix would look like

$$D = \begin{bmatrix} \frac{\partial(TSFC_{SLS})}{\partial(A_{8,SLS})} & \frac{\partial(TSFC_{SLS})}{\partial(A_{8,TOC})} & \frac{\partial(TSFC_{SLS})}{\partial(A_{8,Cruise})} \\ \frac{\partial(TSFC_{TOC})}{\partial(A_{8,SLS})} & \frac{\partial(TSFC_{TOC})}{\partial(A_{8,TOC})} & \frac{\partial(TSFC_{TOC})}{\partial(A_{8,Cruise})} \\ \frac{\partial(TSFC_{Cruise})}{\partial(A_{8,SLS})} & \frac{\partial(TSFC_{Cruise})}{\partial(A_{8,TOC})} & \frac{\partial(TSFC_{Cruise})}{\partial(A_{8,Cruise})} \end{bmatrix}$$

3.2.4 Mathematical Treatment of Optimization Problem

The simplest optimization problem involves finding the values of *m* parameters that minimize a function of these parameters.

$$L(u_i, \ldots . u_m) = L(u) \tag{22}.$$

If there are no constraints on the possible values of $\boldsymbol{u}$ and $L(\boldsymbol{u})$ has first and second partial derivatives everywhere, necessary conditions for a minimum are

$$\frac{\partial L}{\partial u_i} = 0, i = 1, \ldots, m \tag{23}.$$

$$\frac{\partial^2 L}{\partial {u_i}^2} \geq 0 \tag{24}.$$

Where Eq. (24) is an *m* x *m* matrix which has components $\partial^2 L/\partial u_i \partial u_j$ and must be positive semidefinite (eigenvalues ≥ 0). Points that satisfy Eq. (23) are called stationary points. Sufficient conditions for a local minimum are Eq. (23) and that all eigenvalues must be positive,

$$\frac{\partial^2 L}{\partial {u_i}^2} > 0 \tag{25}.$$

If Eq. (23) is satisfied, but $\frac{\partial^2 L}{\partial {u_i}^2} = 0$ then the determinant of the matrix is zero. Such a point is called a singular point and more information is required to establish if the point is a minimum. If $\frac{\partial^2 L}{\partial {u_i}^2} = 0$ everywhere, then L is a linear function of u and in general a minimum does not exist.[47]

SBO is a method to tackle a more general class of optimization problems which attempts to find values of u that minimize L where L is a scalar function of $n + m$ parameters

$$L(x_1, \dots, x_n; u_1, \dots, u_m) = L(x, u)$$

Where n state parameters $\boldsymbol{x}$ are determined by the decision parameters $\boldsymbol{u}$ through a set of n constraint relations

$$f(x,u) = \begin{bmatrix} f_1(x_1, \dots, x_n; u_1, \dots, u_m) \\ \vdots \\ f_n(x_1, \dots, x_n; u_1, \dots, u_m) \end{bmatrix} = 0$$

Now a stationary point is one where $d\boldsymbol{L} = 0$ for arbitrary $d\boldsymbol{u}$ while holding $d\boldsymbol{f}=0$, thus

$$dL = L_x dx + L_u du = 0 \quad (26).$$

$$df = f_x dx + f_u du = 0 \quad (27).$$

Requiring $df=0$ and assuming f_x is nonsingular, Eq. (27) may be solved for dx and plugged into Eq. (26) leading to the necessary condition.[47]

$$\frac{dL}{du} = L_u - L_x f_x^{-1} f_u = 0 \quad (28).$$

3.2.5 Practical Treatment of Optimization Problem

In plain English, SBO relies on an elementary principle of calculus which states that a minimum or maximum of a function will be found when the derivative is exactly zero. It uses the Newton-Raphson method to find the solution and is essentially a root finder – finding the point where the derivative equals zero, to some tolerance.

SBO is attempting to find a stationary point, thus meeting the first necessary condition for a minimum, Eq. (23). In general, the method does not test for Eq. (24). As stated above, in each case either more information is required to find a minimum, or a minimum does not exist. This leads to an important assumption of SBO – since SBO does not test for Eq. (24) or Eq. (25) it assumes that the response function being optimized is convex within the design boundaries with respect to the optimization variables. A good physical example of this is TSFC as a function of BPR, where there is unique value of BPR giving a minimum TSFC. A function F(**X**) bounding a set is defined mathematically as convex if for any two points $\mathbf{X}_1$ and $\mathbf{X}_2$ contained in the set[39]

$$F[\theta \boldsymbol{X_1} + (1-\theta)\boldsymbol{X_2}] \leq \theta F(\boldsymbol{X_1}) + (1-\theta)F(\boldsymbol{X_2}), \;\; 0 \leq \theta \leq 1 \qquad (29).$$

If the function is linear, then by definition the minimum will exist on a boundary. [47] Given a convex function, SBO will find the inflection point and thus a minimum or maximum. It is left up to the designer to use knowledge of the propulsion system to know whether this inflection point is a min or max within the bounds of the problem.

SBO uses a local linear model to estimate the derivatives of the engine model by means of fast matrix multiplications rather than complete convergence of the non-linear thermodynamic model. The linear model will exactly agree with the non-linear model at the point where it was generated. The region where the linear model produces results sufficiently close depends on the non-linearity of the full model and the amount of difference deemed acceptable.

The *u* user defined equations come from the cycle analyst independent parameters specified by the cycle analyst to meet the performance targets of each of the design points. The *m* engine match equations are cycle variables selected to ensure that the laws of continuity and conservation of energy between the different components for each design point are satisfied by matching the different components. The *q* optimization equations come from the partial derivatives specified by the cycle analyst and provided by the linear model to optimize the cycle by driving the derivative to zero. These (*u*+*m*+*q*) equations form the system of nonlinear equations the Newton-Raphson method must solve. The beauty of SBO is that no modification to the solver is required to use the method as-is to optimize the cycle. One must only ensure that the absolute transform is used for the *q* optimization equation dependents. More importantly, no other software or looping structure is required to both optimize and balance the cycle.

Critical to any engine design problem is properly constraining the design space. Constraints are limits placed on the cycle which may not be exceeded and are usually fixed regardless of the flight conditions or design variables. The constraints follow from mechanical limits on design variables such as maximum exhaust nozzle area. Constraints also include technology limits, which are established by the technology level and cannot be exceeded. This would include items such as turbine inlet temperature limits. "The constraint relations are linked to one of the cycle design relations whose independent parameter will be used to meet the constraint should it be violated. In essence, the relations instruct the modified Newton-Raphson solver how to proceed when a constraint is violated."[27]

Constraints attached to a dependent are treated as inactive until it is violated. Once violated, the partial derivative for the constraint function is placed within the Jacobian matrix and the linked dependent function removed. If there is more than one constraint attached to a dependent being violated at the same time, the most severe will be used in the Jacobian. Once the constraint is inactive, the next most severe constraint function is used in the Jacobian if no priority is defined. When all constraints are inactive, the constraint function is removed from the Jacobian and the linked dependent function again takes its place.

3.3 Assumptions and Limitations

With any methodology solving a real world problem there come some assumptions to provide some scope to the problem at hand. There has already been mention of some important assumptions of SBO and a limitation of using the Newton-Raphson method has been hinted at. A major limitation of the solver is that the independent parameters must be unique and two dependent parameters cannot be linked to the same independent parameter. This means that only a single independent may be used to optimize multiple responses at the same time.

The first of the assumptions is that the response function chosen is convex on the interval of interest, described mathematically in Eq. (29). Thus if the objective and constraint functions are convex, only one optimum exists and this is a global optimum.[39] The second assumption was not plainly stated above, although it should be obvious. This assumption is that the response function is differentiable on the interval of interest. This is necessary to obtain the linear model.

The third assumptions follows directly from the assumption of differentiability. As a consequence, it is true that the response function may be closely approximated by a linear

function. Thus it can be assumed that the magnitude of the error in the linear approximation will be much smaller than the difference between the starting point and the starting point plus some increment, given that this increment in close to zero.[50] The assumption can be more clearly understood by example of a single variable function *h(t).* Suppose that *h(t)* is differentiable at $h=h_0$ and let

$$\Delta h = h(t_0 + \Delta t) - h(t_0) \tag{30}.$$

denote the change in *h* that corresponds to the change Δt in *t* from t_0 to $t_0+\Delta t$. Take that

$$\Delta h \approx h'(t_0)\Delta t \tag{31}.$$

provided that Δt is close to zero. In this case the error $\Delta h - h'(t_0)\Delta t$ in this approximation will have magnitude much smaller than that of Δt because[50]

$$\lim_{\Delta t \to 0} \frac{\Delta h - h'(t_0)\Delta t}{\Delta t} = \lim_{\Delta t \to 0} \left(\frac{h(t_0 + \Delta t) - h(t_0)}{\Delta t} - h'(t_0) \right) = h'(t_0) - h'(t_0) = 0 \tag{32}.$$

3.4 OMDP Optimization Strategies

There are conceivably two basic ways the D matrix can be used to optimize an engine model. The final five experiments aim to demonstrate each strategy and how OMDP is used to optimize an engine model. The addition of variables or responses to each method would not result in changing the methodology, setup, or strategy – it would only increase the number of independent/dependent combinations added to the solver and the number of inputs and outputs included in the LMG.

The first strategy type could be considered intra-point optimization. It uses variables at a design point to optimize a response at the same design point. The second strategy type could be considered cross-point optimization. It uses variables at a design point to optimize a response at a different design point. Cross-point optimization is the main advantage of combining SBO with MDP. It allows the designer to optimize across any design point while at the same time properly sizing the engine and meeting all requirements. Separate on design and off design iterations do

not need to take place as both spaces are solved for simultaneously. The addition of variables and/or responses to intra-point or cross-point would only be considered extensions of these two types. Thus each type may be multi-variable or multi-objective optimization, or both.

3.5 Procedure

A flowchart providing a general flow of information for SBO was given in Figure 12. A more detailed procedure is provided for implementing SBO on any type of engine architecture. Each of the experiments in this thesis will follow this procedure.

Step 1 - Select the objective function to optimize: The objective function represents the system performance parameter to be maximized or minimized and must be attainable from the cycle evaluation process.

Step 2 - Select optimization design point: The chosen objective function may exist at one or more design points of interest, and for MDP it must be specified at which design point the objective is being optimized, and which design points the optimization variables will respond at. For SDP, the objective function and optimization variables must be at the same design point.

Step 3 - Select variables available for optimization: The objective function will most likely be a function of many variables, and not all of them may be available to modify by the optimizer. This step involves identifying the additional degrees of freedom in the system above what is required to satisfy conservation of mass and energy and the variables in the model that correspond to these degrees of freedom. Then select one or more of these variables at the appropriate design points which will be used to optimize the objective.

Step 4 - Identify limits and constraints: There are many constraints on aircraft engine operation and some of these will most likely affect the optimization variables. For example, there are physical limits on how much an actuator may move a variable geometry exhaust nozzle. Also, there may be cycle limitations such as maximum turbine inlet temperature. If any limitations apply to the selected optimization variables or objective function, they must be linked to the corresponding dependents.

Step 5 - Set up linearization object: The linearization object is the key component of SBO and provides the solver with the desired derivative information. This object must be given the current state of the model at each solver iteration, including the settings of the optimization variables. The object then must output the derivative of the response with respect to the optimization variable to the solver. This information is accurate within a small region of the point where it was generated and is regenerated at each new iteration of the solver. A realistic perturbation size for each optimization variable must be selected.

Step 6 - Construct system of nonlinear equations: Identify the independent and dependent parameters from the cycle design, engine matching, optimization, and constraint relations, and assemble them into the solver independent and dependent vectors. This includes any constraint relations which must be linked to the dependent parameters.

Step 7 – Assign values to cycle design variables and an initial guess: Select values of cycle design variables, which represents a unique engine design. Assign values for an initial guess, which is required for MDP and SBO. The technique for setting these initial values will be discussed in Chapter 4.

Step 8 – Solver setup: Assign all values necessary for executing the solver. This includes error tolerance for the solver convergence, limits on the number of iterations, and convergence criteria for full generation of the Jacobian matrix. Also, step limits may be set for the solver independent parameters, which will in general will be different for the optimization in independents due to the absolute transformation used for the optimization dependent vectors.

Step 9 – Execute solver: Execute the solver to determine the solution for the chosen cycle design relations, which will also minimize the objective function. This step also includes execution of the linearization object at each iteration of the cycle solver to estimate the derivative values.

CHAPTER 4
IMPLEMENTATION

This chapter describes the environment selected for implementation of SBO. Ten research experiments have been identified to test the hypotheses put forward. The experiments are performed first on models of analytical test functions and then move on to a model of a notional high bypass ratio separate flow turbofan. The first five experiments are an incremental buildup of SBO to demonstrate how it works and prove the results against known phenomena in analytical functions and an SDP SFTF simulation.

The final five experiments establish a baseline for comparison, demonstrate the method working with MDP, and provides notional strategies for using OMDP. This section will describe the common pieces to the setup of each experiment. Chapter 5 will describe any implementation differences unique to each experiment in more detail. The solver setup for each experiment is very basic with the intent of providing a valid setup that can easily demonstrate the intricacies of SBO and OMDP and their utility.

4.1 Modeling Environment

The modeling environment chosen for this thesis is the Numerical Propulsion System Simulation v1.65. This tool was created through a joint United States industry and NASA effort to develop a state of the art aircraft engine cycle analysis simulation tool (GT2010-22350).[18] It is based on C++ object oriented framework, which makes it efficient at dealing with different design points with many variables and constraints. NPSS includes all the features necessary to create almost any engine model with the flexibility to create any required modifications to existing elements. It also includes a solver that finds steady state solutions subject to flow continuity, shaft power balance, and user defined constraints. Additionally it has been extensively verified against proprietary tools, validated against existing engine performance data, and is widely accepted as the US industry standard.[1]

NPSS contains two key artifacts which make it ideal for SBO. The first is a robust Newton-Raphson cycle solver. The NPSS solver is extremely robust and an excellent root finder. It is

already a tested and trusted functionality used in NPSS for finding a converged state of a model. The second artifact is the Linear Model Generator (LMG). The LMG is another tested and trusted NPSS artifact that is useful for estimating the model state derivative values.[34] It provides a means of estimating derivatives by means of simple and fast matrix multiplications rather than complete convergence of the full engine model.

4.2 SBO Setup for NPSS

This section describes the generic setup for SBO. Use of SBO on any model differs very little and this generic setup can for the most part be followed for any model type in NPSS. For SBO to work, there must be 2 separate run files. The flow of information between these two run files is provided in Figure 16. Information is passed by means of a simple text file.

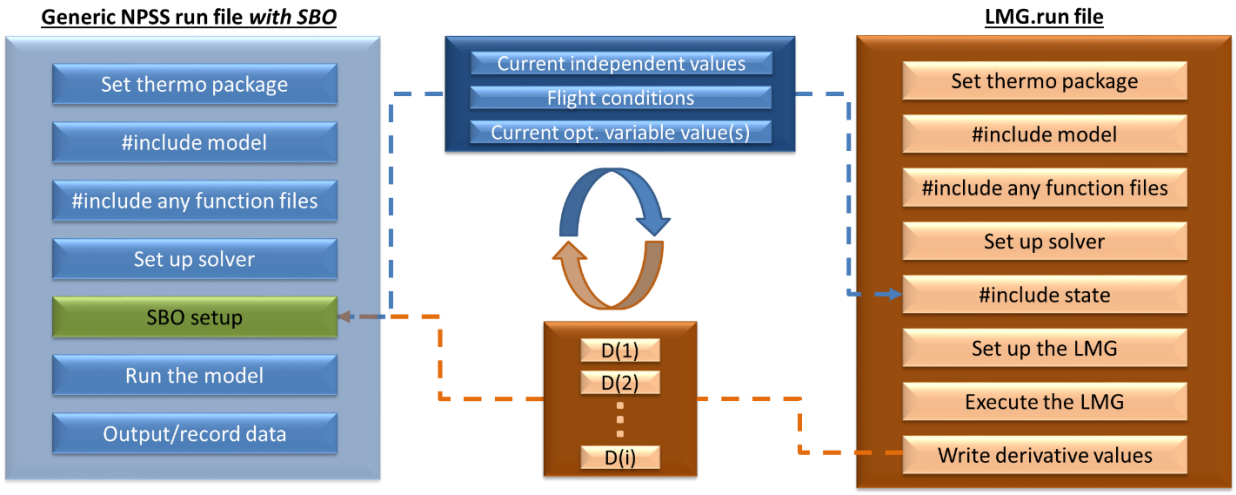

Figure 16: Information Flow between Main Run File and LMG Run File

The first run file is the main run file, which follows the generic form of any NPSS run file. The second run file is the LMG run file which follows a similar form to the generic file setup. The difference is that there is no run command in this second file. Instead it executes the LMG. SBO is implemented by adding 4 main features to any normal NPSS run file setup. These features should be added after the main solver setup, and before the run command.

1) Add the "call_LMG" function to the main run file directly after the main solver is set up.

The actual placement of this function only needs to be anywhere before step 2, but it's best to keep all the additional code grouped together for easier debugging. "call_LMG" runs the separate run file and parses the information produced from that file.

2) Append the function just created to the main solver. This causes NPSS to execute the function at the end of normal solver execution.
3) Add the independent/dependents to the solver that are to be used to optimize the cycle. The independent is the variable used to optimize the response. The independent is adjusted to drive the dependent left-hand side to equal the dependent right-hand side. In this case, the right-hand side is zero, and the left-hand side is the derivative of the response with respect to the independent.
4) Add any constraints to the dependent.

The "call_LMG" function does three main things. It first writes any variables needed by the LMG.run file to a text file. This is necessary since the LMG.run file cannot access any information defined in, or produced from, the main run file. This text file is written once for each iteration of the solver. call_LMG then executes LMG.run. All independent values in the solver should be passed, the current value of the optimization independents, as well as the flight conditions for each design point. Finally, it parses the information from LMG.run in the form of a text file so it can be used by the main run file. Specifically, it reads in the derivative information produced by the LMG.

The LMG.run file is set up in a similar way to any generic NPSS run file. The model is included and solver is set up as normal. The text file created by the call_LMG function is included. Next, the LMG is set up. This step is relatively simple. The LMG is defined, and given the input values and output values. The input is the independent used for optimization and the output is the response. The perturbation value used by the LMG can be altered from the default if desired and the type of perturbation can be changed between fractional or absolute. The LMG can then be executed. The output from this step is the desired derivative. The derivative information is then written to a text file. Again, this is the only way the information can be communicated back to the main run file.

To quickly summarize how the information is passed between the two run files, the call_LMG function which is appended to the solver writes the solver independent values to a text

file, which is included in the LMG.run file. Once that file is finished running, it writes the derivative information to a text file, which the call_LMG function parses. The solver uses this information to simultaneously drive the derivative to zero while converging the model. The LMG *must* be placed in a separate run file. Inclusion of the LMG step in the main run file will result in an infinite loop because of the way the LMG works.

4.3 Selection of Objective Function and Optimization Variables

The objective function must be one obtainable from the model. For the analytical test functions to be described in the next section, this objective is simply the function value. The objective is a function of two variables and must be minimized to a known minimum value. There are obviously only two possible variables available to be used to minimize the objective. Both will be used in the first three experiments described in the next chapter.

The objective for an engine model again must be one obtainable from the cycle analysis. There are several parameters often used for determining the performance of an engine cycle. Typically the performance objective at most part power points inside a flight envelope, shown as a notional example in Figure 17, is to minimize TSFC while maintaining a desired thrust. Thus TSFC is an ideal parameter to be used as the objective function for experiments one through ten.

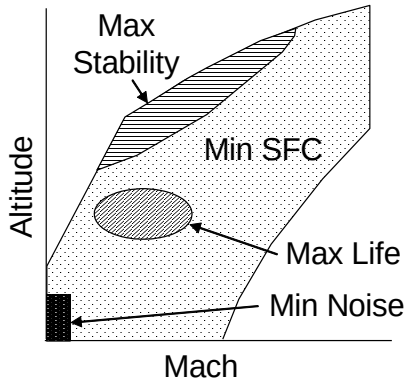

Figure 17: Part Power Performance Optimization Objectives throughout Flight Envelope[19]

TSFC is a function of a number of variables including fuel flow, exhaust velocity, ambient conditions, airflow, and BPR in a turbofan engine. Several variables are available for use to

optimize TSFC in a turbofan. One obvious one is BPR, which will be used since it can be shown theoretically that there is a unique BPR giving a minimum TSFC. BPR is not considered a variable geometry feature but it is a cycle design variable which the user has the ability to select. For an SDP model it can be used to minimize TSFC at the on design point only. However in MDP it may be used to minimize TSFC at any design point, which will be shown in the following experiments.

Other variables which can be used are airflow and exhaust velocity. Operationally, these variables can both be controlled to some extent by a variable geometry exhaust nozzle. This obviously is a VG component and thus is an excellent variable to choose to demonstrate how SBO can be utilized to set component geometries. SBO can be given control of the variable geometry exhaust nozzle area at any design point to minimize TSFC

4.4 Model Concepts

This section will describe the models used for the ten experiments identified to investigate the research questions put forward in Chapter 2. It will also describe the general solver setup for both models, which is identical for the most part. Any differences for a specific experiment will be described in Chapter 5.

4.4.1 Analytical Model Description and Solver Setup

A full engine model need not be created for the proof of concept. Instead, a simple dummy element can be created with the analytical test function embedded (see APPENDIX A). NPSS will function normally with the automatic solver setup simply being empty since there is no conservation of mass or energy to satisfy. Independent/dependent combinations are then added by the user to drive the derivatives of the function to zero. Thus there must be one independent for each optimization variable and one dependent for each objective function. The dependent right hand side is then the derivative value and the left hand side is exactly zero. Any constraints can be placed directly on the dependents.

The derivative information required is provided by the LMG. In order for the derivative information to be available, the LMG is placed in a separate run file which is appended to the solver. In this way the solver calls this run file as part of its normal operation, which in turn runs the LMG. The derivative information is written to a text file and then parsed to read the

derivatives into the solver dependents. The LMG *must* be placed in a separate run file. Inclusion of the LMG step in the main run file will result in an infinite loop because of the way the LMG works. However, this requires that the solver information from the main run file must be passed to the LMG run file explicitly by means of a text file since it cannot access any information contained in, or produced by, the main run file.

The first test function, given by Eq. (33) and shown in Figure 18, is the function used by Brown to test the internal gradient and gradient integration methods. Brown successfully showed his methods found the optimum more rapidly than two other optimization methods.

$$f_B(x_1, x_2) = {x_2}^3 - 8x_1x_2 + (x_2 - 2)^2 + 4(x_1 - 4)^2 \quad (33).$$

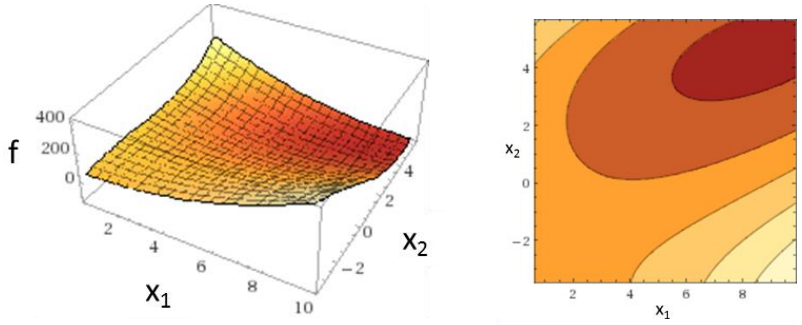

Figure 18: Brown's Analytical Test Function

The first test problem will use SBO to find the minimum of Brown's test function. It can be stated mathematically as:

Minimize: $f_B(x_1, x_2) = {x_2}^3 - 8x_1x_2 + (x_2 - 2)^2 + 4(x_1 - 4)^2$

Subject to:

$$0 \leq x_1 \leq 12$$

$$0 \leq x_2 \leq 8$$

The second test problem for a proof of concept for SBO involves the Rosenbrock function, given by Eq. (34) and shown in Figure 19. This is a common function used to test the ability of

an optimization algorithm to find an optimum and demonstrate its efficiency. Use of a common function such as this will better illustrate the efficacy of SBO and will make it easy to provide a basis for comparison of SBOs utility compared to other optimization algorithms. It has a global optimum at $\boldsymbol{X}^* = \{1,1\}$.

$$f_R(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \quad (34).$$

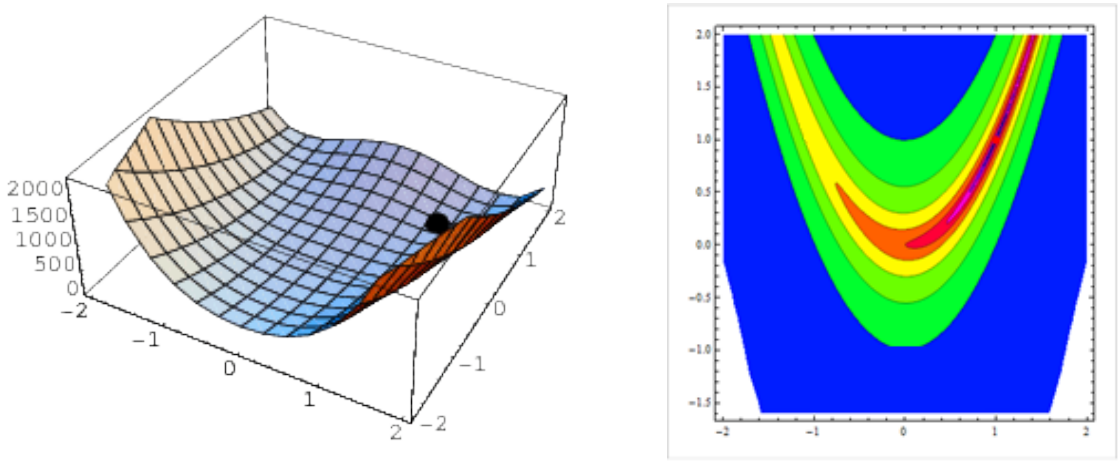

Figure 19: Rosenbrock Function

The second test problem can be stated mathematically as:

Minimize: $f_R(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$

Subject to:

$$-2 \leq x_1 \leq 2$$

$$-1 \leq x_2 \leq 4$$

The general solver setup for the analytical function experiments is identical. As shown in Table 2 there is one independent/dependent combination for each optimization variable. The dependent lhs is always the derivative value and the rhs is always exactly zero. Each dependent in Table 2 has two constraint dependents attached, shown in Table 3 which constrain the independent values by the user-defined minimum and maximum values.

Table 2: Solver Setup for Analytical Functions of Two Variables

Independent		***Dependent***		
Name	**varName**	**Name**	**lhs**	**rhs**
x1_Indep	x1	x1_dep	d1	0.0
x2_Indep	x2	x2_dep	d2	0.0

Table 3: Constraint/Dependent Pairings

Constraint	**Min/Max**	**Dependent Name**	**lhs**	**rhs**
con_x1_max	Max	x1_dep	x1	x1max
con_x1_min	Min	x1_dep	x1	x1min
con_x2_max	Max	x2_dep	x2	x2max
con_x2_min	Min	x2_dep	x2	x2min

4.4.2 SFTF Engine Model Description and Solver Setup

The engine cycle model is a mathematical model of the aerothermodynamics of the internal engine flow which strictly adheres to the principles of mass and energy conservation. It is this model that is used to determine performance data for a given flight condition. A mathematical representation of each component is created and thermodynamic properties are measured at stations between each component such that each component's thermodynamic parameters are calculated based on upstream conditions.

The architecture chosen to demonstrate OMDP is a high bypass ratio separate flow turbofan. Figure 20 provides a schematic of the NPSS model. The low pressure system is composed of a single stage fan and multi-stage LPC driven by a multi-stage LPT on a single shaft. The high pressure system consists of a multi-stage HPC and HPT on a single shaft. The NPSS model elements include ambient flight conditions, inlet, fan, splitter, LPC, HPC, fuel start, burner, HPT, LPT, core and fan nozzle, LP and HP shafts. No cooling flow is accounted for in this model and thus is not shown in the schematic, although the model contains the required elements to add cooling if so desired in future work.

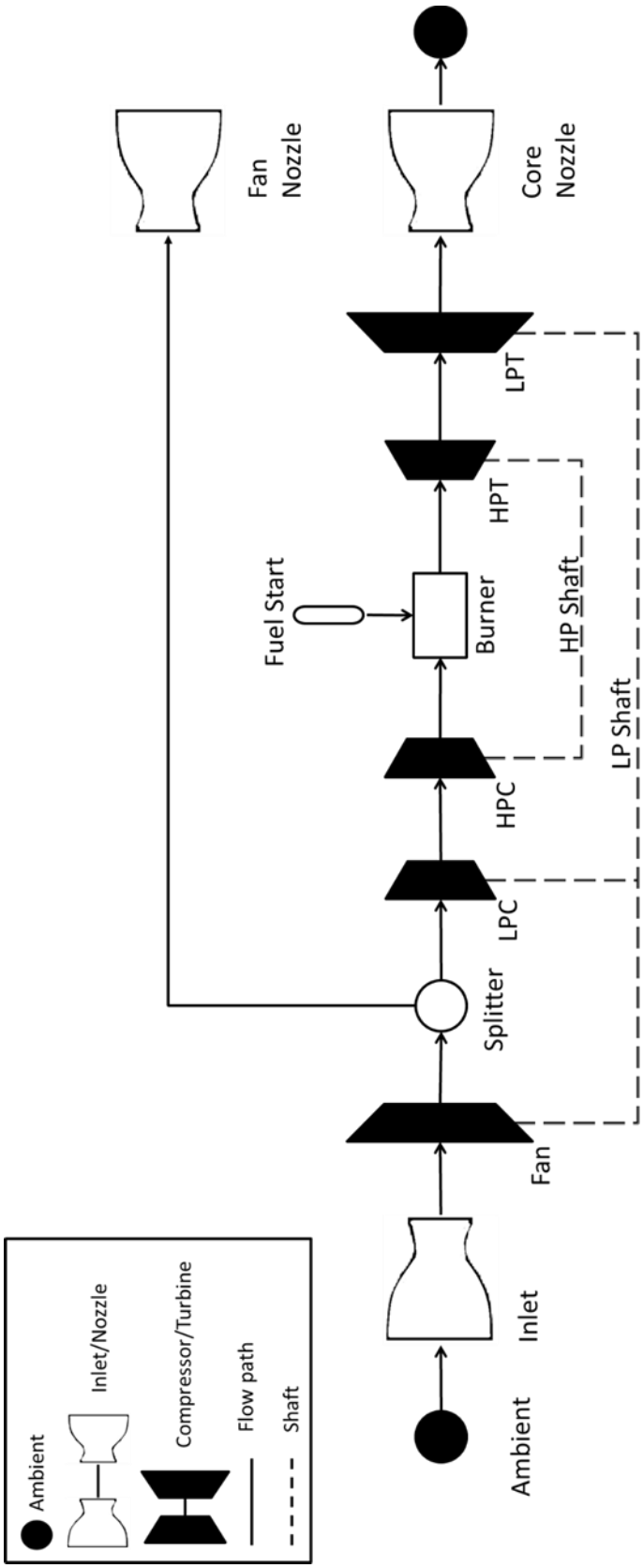

Figure 20: NPSS Model Schematic of SFTF

4.4.2.1 Solver Setup for SBO with SDP

The solver setup is the most critical and complex part of the setup process. It constructs the system of nonlinear equations from the cycle design relations, engine matching relations, and constraint relations.[27] This is done through sets of independent and dependent parameters so that the solver can manipulate the independent parameter to achieve the required value of one of the dependent parameters. The independents must be unique and there must be an equal number of independents and dependents. In this case, TSFC will be the model output variable used as the objective function and the optimization variable be bypass ratio. One user added independent/dependent combination will need to be included by the user to drive the derivative of TSFC found by the LMG to zero by adjusting the bypass ratio. Additionally, the solver will now have several engine match relations which are automatically included to satisfy conservation of mass and energy. This final proof of concept test case for SBO is important. First, it will prove that SBO is capable of being scaled up to a full engine model. More importantly, it will demonstrate that it can work with the complex design space represented by a turbofan engine.

Experiment 4 uses a simple SFTF model. SBO varies the BPR to find the minimum TSFC for a chosen design FPR value. It is expected that as FPR is increased the optimum BPR will decrease. It is also expected that there is a unique BPR for an altitude and Mach, and selected FPR and HPC PR which will give the minimum TSFC. The solver setup follows exactly the same concept as the analytical test cases. The chosen independent is adjusted to drive the derivative to zero. For this case only one independent was used for optimization ("BPR_indep" in Table 4).

Table 4: SFTF Model Solver Setup

	Independent		*Dependent*		
Class	**Name**	**varName**	**Name**	**lhs**	**rhs**
User	Ind_W1	Ambient.W	Dep_W2R	Fan.Fl_I.Wc	zw2r
Engine Match	HPT.S_map.ind_parmMap	parmMap	Integrate_Nmech	trqNet	0
Engine Match	LPT.S_map.ind_parmMap	parmMap	Integrate_Nmech	trqNet	0
User Opt.	BPR_Indep	Splitter.BPR	BPR_dep	d1	0

4.4.2.2 Solver Setup for OMDP

The MDP experiments all have a common setup for the most part and will be given here. Any deviations from this common setup will be noted for each experiment. The solver setup for MDP is typically more complex than for SDP where now the system of nonlinear equations may be a coupled system across multiple design points.

The final 5 experiments will all the same basic setup. Two points will be used for each experiment – one at TOC and one at TKO. TOC is at Mach 0.8 and 30,000ft altitude. TKO is at Mach 0.3, standard day sea level conditions. Typically TOC is a key sizing condition and sets the maximum corrected airflow through the fan and the maximum corrected engine speed. TKO generally sets the maximum T4. The solver setup (Table 5) for this experiment includes the auto solver variables for conservation of mass and energy. User defined variables to meet the requirements will include FAR at TKO, FAR at TOC, and mass flow at TOC. These variables will be used to meet the thrust requirements at TKO and TOC as well as hit a maximum T4. The setup is such that a maximum fan speed is hit at either TOC or TKO subject to max T4 constraints at TOC and TKO. The constraint relations are given in Table 6. The result is that a max fan speed and T4 will always be hit at the most constraining design point no matter how the thrust requirements or flight conditions change.

A subsonic cruise point will also be included in MDP, which is at Mach 0.8 and 28,000ft altitude. At this point the engine is expected to be at a part power setting which should not constrain the design space at all. It is being included as the point at which to optimize. The solver setup at this point will again include the auto solver setup variables and fuel flow to meet thrust. The solver tolerance is set to 1e-9. The default maximum allowable change for an independent

variable on any single iteration, the defaultDxLimit, is set to 0.1.

An initial guess will always be given as the starting point for optimization. This initial guess contains values for all solver independent values and may be obtained in two ways. First, the model may be run in SDP and the values at the converged points may be taken as the initial guess. The second method uses MDP and is often only possible after getting at least one good initial guess from SDP. From there, MDP may be run with different input values and still converge. This converged point can then provide the initial guess for other points.

Table 5: Independent/Dependent Combinations for Experiments 5-10

Class	Independent	Dependent
User	TOC_ind_FAR	TKO_dep_fntarget
User	TOC_ind_W1	TOC_dep_fntarget
User	TKO_ind_FAR	TKO_dep_pcn2max
User	CR_ind_FAR	CR_dep_fntarget
Engine Match	Cruise.Ambient.ind_W	Cruise.Core_Nozzle.dep_Area
Engine Match	Cruise.Fan.S_map.ind_RlineMap	Cruise.Fan.S_map.dep_errWc
Engine Match	Cruise.HPC.S_map.ind_RlineMap	Cruise.HPC.S_map.dep_errWc
Engine Match	Cruise.HPT.S_map.ind_parmMap	Cruise.HPT.S_map.dep_errWp
Engine Match	Cruise.HP_SHAFT.ind_Nmech	Cruise.HP_SHAFT.Integrate_Nmech
Engine Match	Cruise.LPC.S_map.ind_RlineMap	Cruise.LPC.S_map.dep_errWc
Engine Match	Cruise.LPT.S_map.ind_parmMap	Cruise.LPT.S_map.dep_errWp
Engine Match	Cruise.LP_SHAFT.ind_Nmech	Cruise.LP_SHAFT.Integrate_Nmech
Engine Match	Cruise.Splitter.ind_BPR	Cruise.Fan_Nozzle.dep_Area
Engine Match	TKO.Ambient.ind_W	TKO.Core_Nozzle.dep_Area
Engine Match	TKO.Fan.S_map.ind_RlineMap	TKO.Fan.S_map.dep_errWc
Engine Match	TKO.HPC.S_map.ind_RlineMap	TKO.HPC.S_map.dep_errWc
Engine Match	TKO.HPT.S_map.ind_parmMap	TKO.HPT.S_map.dep_errWp
Engine Match	TKO.HP_SHAFT.ind_Nmech	TKO.HP_SHAFT.Integrate_Nmech
Engine Match	TKO.LPC.S_map.ind_RlineMap	TKO.LPC.S_map.dep_errWc
Engine Match	TKO.LPT.S_map.ind_parmMap	TKO.LPT.S_map.dep_errWp
Engine Match	TKO.LP_SHAFT.ind_Nmech	TKO.LP_SHAFT.Integrate_Nmech
Engine Match	TKO.Splitter.ind_BPR	TKO.Fan_Nozzle.dep_Area
Engine Match	TOC.HPT.S_map.ind_parmMap	TOC.HPT.S_map.dep_errWp
Engine Match	TOC.LPT.S_map.ind_parmMap	TOC.LPT.S_map.dep_errWp

Table 6: Constraint/Dependent Pairings for Experiments 5-10

Constraint	Min/Max	Dependent Name	lhs	rhs
TKO_MAX_T4	Max	TKO_dep_pcn2max	TKO.HPT.Fl_I.Tt	3250
TOC_MAX_T4	Max	TKO_dep_pcn2max	TOC.HPT.Fl_I.Tt	3250

The DPMM (Table 7) succinctly summarizes the design points, design alternatives, performance requirements, component performance estimates, and technology limits. It also shows that TOC is designated as the component map scaling point. "At this design point, the component performance maps will be scaled to attain the necessary mass flow."[27]

Table 7: Design Point Mapping Matrix for Experiments 5-10

Design Points		**TKO** Mn 0.3 | 0ft	**Cruise** Mn 0.8 | 28kft	**TOC** Mn 0.80 | 30 kft
Comp. Map Scaling Point				**x**
Design Variables	FPR			1.68
	LPCPR			1.45
	HPCPR			11.6
	OPR			28
	Fan R line			2
	LPC R line			2
	HPC R line			2
	Fan Nc Map			105%
	LPC Nc Map			100%
	HPC Nc Map			100%
	T4	3250 R		
Performance Requirements	TKO Fn	35000 lbf		
	Cruise Fn		10000lbf	
	TOC Fn			11000 lbf
Component Performance Estimation	Fan Adiabatic eff.			0.878
	LPC Adiabatic eff.			0.890
	HPC Adiabatic eff.			0.863
	HPT Adiabatic eff.			0.891
	LPT Adiabatic eff.			0.939
	Burner eff.			1.0
	Bypass nozzle Cv			Function of PR
	Bypass nozzle CdTh			Function of PR
	Core Nozzle Cv			Function of PR
	Core Nozzle CdTh			Function of PR
Technology Limits	T4max			3250 R

CHAPTER 5
RESULTS

This chapter examines the results from two series of experiments designed to test the hypotheses put forth in Chapter 2. The first series of experiments are a proof of concept for solver based optimization. The second series of experiments investigates the application of the method and its flexibility. These experiments will prove the SBO methodology developed and:

- Demonstrate the setup and implementation of SBO and its use with a Newton-Raphson cycle solver on a multivariable function.
- Prove that the optimization method can efficiently find known optimums of both analytical test functions and an example aircraft engine model, while investigating any limitations.
- Demonstrate that the method can simultaneously find a balanced and optimum cycle design for an aircraft engine.
- Illustrate the extension of SBO to MDP and demonstrate the functionality and advantages this brings to cycle analysis.

5.1 Experiments 1-3, Solver Based Optimization Proof of Concept

The purpose of this section is to test the ability of the proposed method of Solver Based Optimization to find an optimum. SBO is intended to be used with engine models and in the end be combined with MDP. As a proof of concept however, it is reasonable to use simple analytical test functions to prove the method. Use of an analytical function for which the optimal solution is known serves several purposes. First, it will demonstrate the implementation and setup of the method in an easy to understand problem. It will also help work out any issues that may crop up. Finally, it will prove that the method can find the optimum.

5.1.1 Setup and Implementation

The first test function is implemented in NPSS by coding Eq. (33) into a dummy element. This

element simply calculates the value of Eq. (33). Two independent/dependent combinations are added to the solver. The first adjusts the independent x_1 so that the derivative of $f_B(x_1, x_2)$ with respect to x_1 equals zero to within some tolerance, subject to being bounded by 0 and 12. The second adjusts the independent x_2 so that the derivative of $f_B(x_1, x_2)$ with respect to x_2 equals zero, subject to being bounded by 0 and 8. The optimum found should be relatively close to the answer found by Brown, where the optimum can be found analytically to be $\boldsymbol{X}^* = \{8.60555, 4.60555\}$.

In the same way as the previous function the second test function is implemented in NPSS by coding Eq. (34) into a dummy element. Two independent/dependent combinations are added to the solver. The first adjusts the independent x_1 so that the derivative of $f_R(x_1, x_2)$ with respect to x_1 equals zero to within some tolerance, subject to being bounded by -2 and 2. The second adjusts the independent x_2 so that the derivative of $f_R(x_1, x_2)$ with respect to x_2 equals zero, subject to being bounded by -1 and 4. The optimum found should be at $\boldsymbol{X}^* = \{1,1\}$; and built in functionality in NPSS will show how many model passes were required to converge to the optimum.

5.1.2 Results

SBO's ability to find a known optimum of several analytical functions has been shown. A dummy model was set up and each test function embedded so that the function is executed on each pass through the model. The solver setup in each case allows the solver to use both x_1 and x_2 to drive the derivative to zero. Specifically, the solver adjusts x_1 and x_2 so that $\frac{\partial(f)}{\partial(x_1)} = 0$ and $\frac{\partial(f)}{\partial(x_2)} = 0$ respectively. The solver independent and dependent combinations were given in Table 2. The default solver setup was used except that the toleranceType for the dependents was changed to absolute and the dxLimitType for independents was changed to absolute.

Brown's test function was successfully minimized by the SBO method with the independent and function values for the final iteration shown in Table 8, where a (*) represents the optimum. The first thing to notice is that SBO did not approach any of the constraint values. Thus this effectively results in an unconstrained optimization problem. Second, SBO found a solution much closer to the analytical optimum than the gradient integration method used by Brown. He was able to attain an accuracy to within about 2.7%. While this is an excellent result, SBO was

able to reduce this to a negligible 0.00012%. SBO postulated that the error would be much smaller than the allowable step size. The perturbation for the linear model generator was set for x_1 at $(x_{1,max} - x_{1,min}) * 0.001 = 0.012$ and for x_2 at $(x_{2,max} - x_{2,min}) * 0.001 = 0.008$. Therefore this accuracy is not surprising, and is indeed much less than the step size as put forward in the assumptions.

Table 8: SBO Results – Unconstrained Analytical Functions

Function	**x1***	**x2***	**f***
Brown (Analytical)	8.605551	4.605551	-225.06644
Brown (published values)	8.718	4.640	-231.11296
SBO	8.605548	4.605548	-225.06616
Rosenbrock (Analytical)	1	1	0
SBO	0.996871	0.993752	9.79E-06

Experiment 2 focused on the Rosenbrock function – a common test function for optimization algorithms. Again the experiment resulted in unconstrained optimization. SBO never attempted to violate the constraints placed on the dependents so none of them became active. The accuracy of the method is approximately 0.000098%. The perturbation size for the LMG was set in a similar manner such that set for x_1 it was set at $(x_{1,max} - x_{1,min}) * 0.001 = 0.004$ and for x_2 it was set at $(x_{2,max} - x_{2,min}) * 0.001 = 0.005$. Again, this accuracy falls well within the assumption. This is also less than the accuracy obtained for Brown's function, which follows from using a smaller limit on the perturbation size for the LMG.

The ability of SBO to find an optimum when there are active constraints is an important feature of the method and one which was possible to test once the previous results were obtained. As was shown, neither analytical functions reached a constraint boundary. With these results in hand it was possible to set limits that SBO would naturally attempt to break in order to test the ability of the method to handle constraints. To do this, experiment 1 and 2 also tested SBO against both functions with active constraints. For each function, a constraint which would definitely become active was placed on one of the dependents which constrained the independent value. The results are shown in Table 9.

Table 9: SBO Results (Constrained)

Function	x1*	x2*	f*	Constraint
Brown	6	3.843319	-108.311	x1 <= 6.0
Rosenbrock	0.996808	0.993626	1.02E-05	x2 >= 0.2

For Brown's function the constraint on x_1 was placed such that the optimum lies in the constrained region. This test is to ensure that SBO can reach a suboptimum value without failing when the optimum lies in the constrained region of the design space. In that case SBO reaches a constraint and can go no further, finally stopping at a sub-optimum. As can be seen in Table 9, SBO stopped on the constraint boundary. This can be seen visually from Figure 22 which shows the path SBO takes to reach the final solution. It takes almost exactly the same path in both the constrained and unconstrained cases as one would expect, however stops on the boundary in the constrained case.

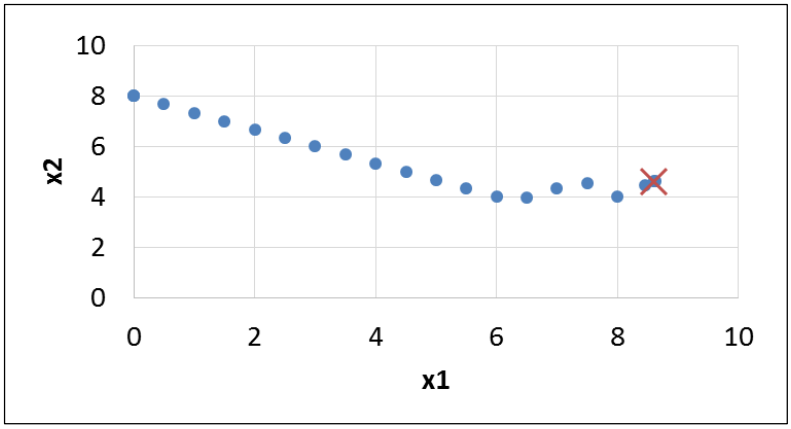

Figure 21: Brown's Test Function Optimization Path, Unconstrained

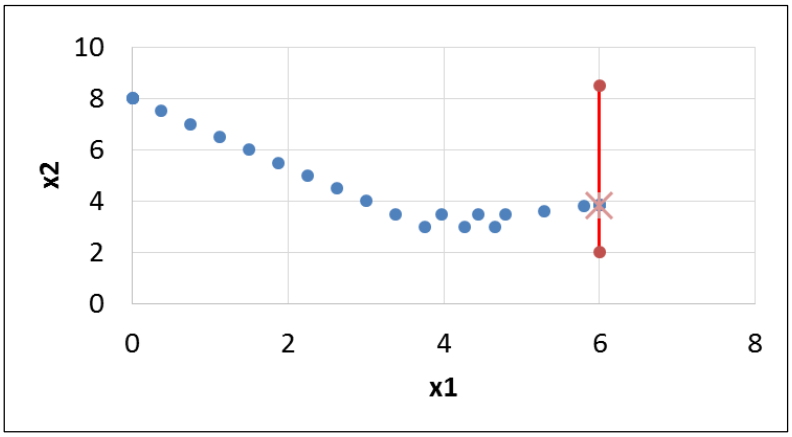

Figure 22: Brown's Test Function Optimization Path, Constrained

The constraint for the Rosenbrock function leaves the optimum in the unconstrained space but cuts off the path SBO would take to reach the solution if left unconstrained. A gradient based method naturally attempts to follow the "valley" in the Rosenbrock function. The constraint in this case cuts off the path to make sure SBO can still handle such a constraint.

The path SBO takes to find the optimum in the unconstrained case is relatively smooth and follows the valley in the Rosenbrock function as expected from a gradient based optimization algorithm. The constrained case follows a slightly more erratic path but still finds an optimum. These paths can be seen in Figure 23. Since NPSS only activates a constraint once it has been violated there are several points that are in the constraint region. Once NPSS reads a value past its constraint, it moves the value back towards the unconstrained region and deactivates the constraint once it is again unconstrained.[34] The same default settings were used for the solver as in the unconstrained case at first, but it was found that NPSS would allow the independents to move too far, making SBO unstable. It was necessary to adjust the dxLimit manually to find a limit suitable to allow convergence. If this becomes a problem in the future, Brown proposes a way to dynamically set the limit to ensure convergence which could be used here.[43]

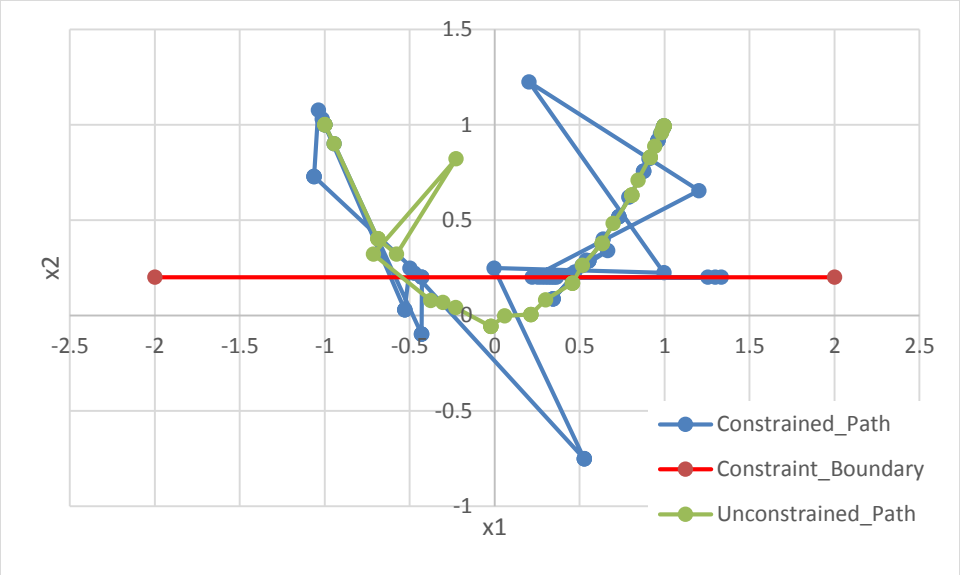

Figure 23: Rosenbrock Function Optimization Path

There are a number of methods and optimization algorithms that could be used to optimize an engine model. One of the main advantages gained by using SBO comes from the fact that no nested loop structure or other software is needed to incorporate the optimization – considerably reducing the effort required to build the model and optimize it. However, any of these advantages may be lost if SBO cannot efficiently arrive at an optimum. Experiment 3 is a comparison of SBO against known optimization algorithms for the Rosenbrock function. It proves that SBO is able to attain approximately the same accuracy and efficiency as seven other methods. The result of running seven optimization algorithms, shown in Table 10, were obtained through an open source software developed by Wolfram. It is a lighter version of Wolfram *Mathematica* which allows *Mathematica* code to be run on a free player. The code (Reference 46) runs seven known optimization algorithms to find the optimum of the Rosenbrock function. SBO is more efficient in terms of number of function evaluations than 2 methods. The accuracies it achieves are comparable to all of the methods. Not surprisingly, the number of evaluations SBO takes is closest to Newton's method. This is expected since the NPSS solver is a modified Newton method.

Table 10: SBO Comparison for Rosenbrock Function

	Starting Point			*Ending Point*	
Solvers for Rosenbrock	**x1**	**x2**	***number of evaluations***	**x1**	**x2**
Levenberg Marquardt	-1	1	14	1.004	0.992
Principal Axis	-1	1	15	1.005	0.998
Interior Point	-1	1	20	1.002	0.994
Newton	-1	1	21	1.01	0.988
SBO	-1	1	26	0.997	0.994
QuasiNewton	-1	1	35	1.018	0.985
Conjugate Gradient	-1	1	37	1.006	0.994

5.2 Experiment 4, SDP Engine Model Optimization

Now that SBO is proven to efficiently be able to find an optimum, it can be proven that it works for an engine model. Finally, SBO will be used with a simple SFTF engine model that has one variable geometry feature. The incorporation of SBO into a working engine model will effectively be the same as the analytical test function cases. Given flight conditions T_0 and M_0, it can be analytically demonstrated that there exists an optimum bypass ratio for a chosen HPC pressure ratio and FPR for a turbofan engine.[26] This optimum bypass ratio gives the minimum TSFC for all other parameters fixed. The FPR – BPR – TSFC design space is also convex; following the assumption that guarantees SBO will find a minimum.

For an idealized separate flow turbofan, TSFC is given by

$$TSFC = \frac{\dot{m}_f}{F/\dot{m}_0} = \frac{f}{(1+BPR)(F/\dot{m}_0)} \quad (35).$$

Taking the partial derivative of Eq. (35) with respect to BPR, setting equal to zero, and solving for BPR will provide the optimum bypass ratio for a given flight condition, FPR, and HPC pressure ratio. A full derivation of this can be found in Reference 26 and the results of this can be shown in Figure 24 for and ideal SFTF and one with losses.

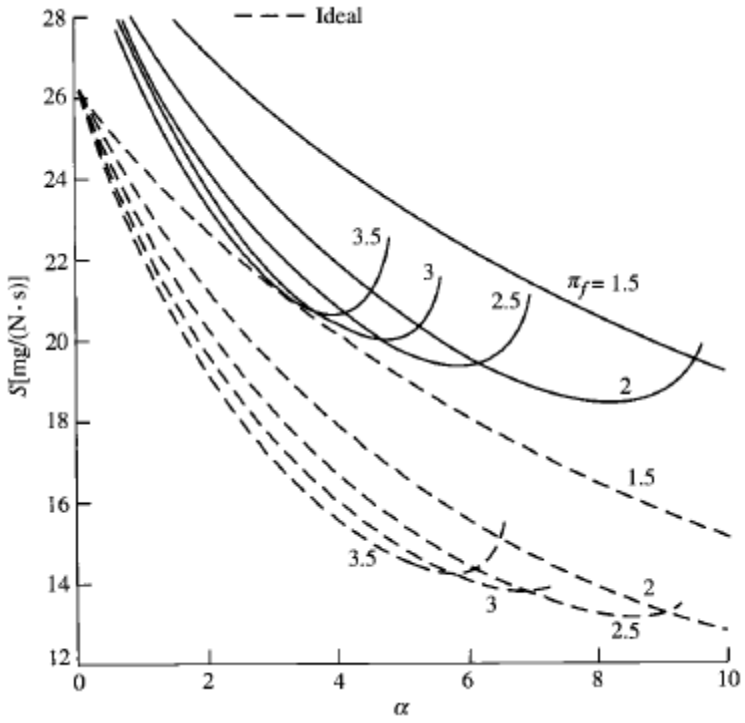

Figure 24: Real and Ideal Turbofan TSFC vs FPR for HPCPR=24, M_0=0.9[26]

SBO is a gradient based method and as such the assumption inherent in the method is that the design space of an engine is fairly smooth and continuous with the objective function being differentiable. Brown[43] and Roth[19] have both already shown that a gradient based method will work for a VCE type design.[43] As Figure 24 shows, it can be analytically proven that there exists an optimum BPR for a given flight condition and selection of HPCPR and FPR. SBO should be able to find this optimum. More importantly SBO will also simultaneously balance the cycle. Balancing the cycle while optimizing the cycle removes the need for any type of nested optimization structure or outside algorithms which must separately optimize and balance the cycle

5.2.1 Setup and Implementation

The design point chosen for this experiment is TOC at 30,000 feet altitude and Mach 0.8. SBO varies the BPR to find the minimum TSFC for a chosen design FPR value. It is expected that as FPR is increased the optimum BPR will decrease. It is also expected that there is a unique BPR for an altitude and Mach, and selected FPR and HPC PR which will give the minimum TSFC.

The solver setup follows exactly the same concept as the analytical test cases. The chosen independent is adjusted to drive the derivative to zero. For this case only one independent was used for optimization ("BPR_indep" in Table 4).

5.2.2 Results

The objective here is to verify that SBO finds a balanced cycled design and finds the BPR which gives the minimum TSFC, as predicted by analysis of (33). Once the model was built it was possible to mimic Figure 24 and produce Figure 25 produced by plotting the TSFC results of running the model in a normal (non-SBO) configuration while driving the BPR to a desired value in 0.5 increments. This was done for two separate design values of FPR. As expected this produces a curve where the optimum BPR (and thus minimum TSFC) for a design FPR is at the inflection point of the curve. Not surprisingly, the curves presented in Figure 25 match what is seen in the example shown in Figure 24 and in the process validate the model is working properly.

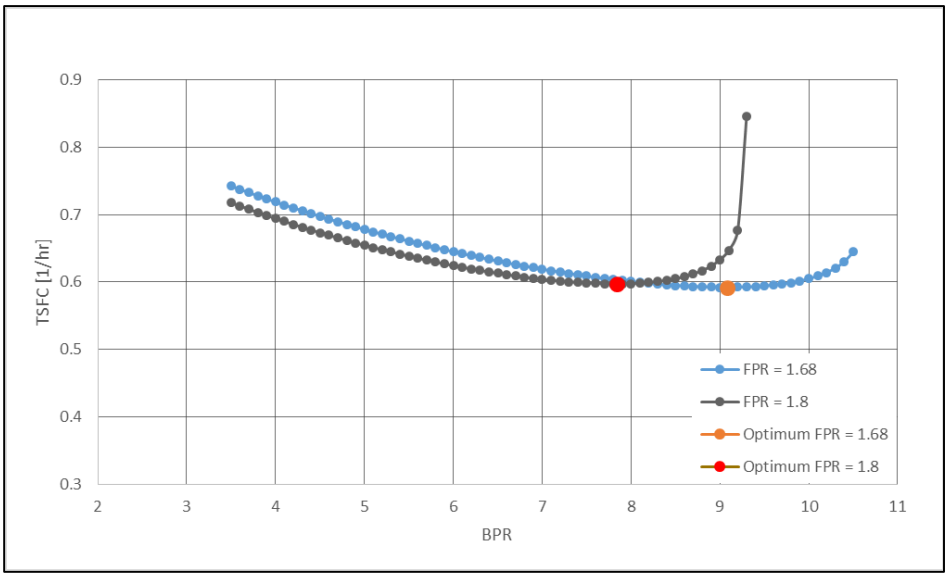

Figure 25: SFTF Optimization Using BPR, HPCPR = 11.6, M_0 = 0.8

The results of running SBO are overlayed on these curves as single points. The SBO results correspond exactly with the inflection points of the two curves demonstrating for the first time

that SBO is capable of optimizing and simultaneously balancing a cycle. As further validation of the method, the optimum BPR found by SBO decreases for increasing FPR again corresponding the expected results exemplified in Figure 24. As a further note on the efficiency of SBO, convergence to the reported optimums took 42 model passes for each FPR case and took on the order of 30 seconds on a Windows 8 OS using a 2.6 GHz Intel(R) Core (TM) i5-3230M CPU. It should be emphasized that the model was converged to a balanced *and* optimum cycle.

These results prove that SBO can efficiently converge to an optimal solution for a design space represented by a turbofan engine model. It is expected that the complexity of the model will not adversely affect the convergence speed. However, incorporating SBO with MDP is expected to have a noticeable impact on the convergence speed. This conclusion comes from previous experience with MDP and is due to a much larger number of independents included in the solver, plus those that are now required for optimization.

5.2.2.1 Selection of Linear Model Perturbation Size

One of the key assumptions of SBO is that the error in the linear model is proportional to the perturbation size. Choosing a perturbation size very small guarantees a low error about the point at which the model was generated but does not guarantee that the derivative truly captures the slope of the function. Too small or too large of a perturbation can result in poor derivative calculations. Thus the question must be asked: How can the perturbation size be chosen to accurately represent the derivative of a possibly highly nonlinear function? To start it must be chosen such that the error in the approximation is limited to a reasonable value. Then the perturbation must also be chosen such that it is representative of the chosen objective function and optimization variables. Examination of the results of various perturbation sizes for this SDP SFTF model will help in leading to a common method for determining a perturbation size for any optimization variable.

The perturbation size for the SDP model just used for this experiment was varied from 0.003 to 5.0, keeping track of the number of model passes, the final TSFC value, and final BPR value. It is apparent from Figure 26 that there is a “sweet spot” where the number of model passes is minimized while still producing almost exactly the same value of TSFC. Too low (below 0.003) and the derivative is not good enough to be used at all and optimization fails altogether. Too high (above approximately 0.9) and the error in the approximation results in highly inaccurate values

of TSFC. The reported derivative is still zero, but obviously there is a large error in this calculation at high perturbations.

It was found that the variation in TSFC in this sweet spot region was at most 0.004%. For experiment 4, the default LMG perturbation of 0.005 was used. Now it can be seen that this value still produces accurate results, but at the cost of 42 model passes. Better selection of the perturbation size would still have produced acceptable accuracy but with 62% fewer model passes. 42 model passes is somewhat trivial for a SDP model with so little in the solver, however 62% can be significant when moving to larger and more complex models.

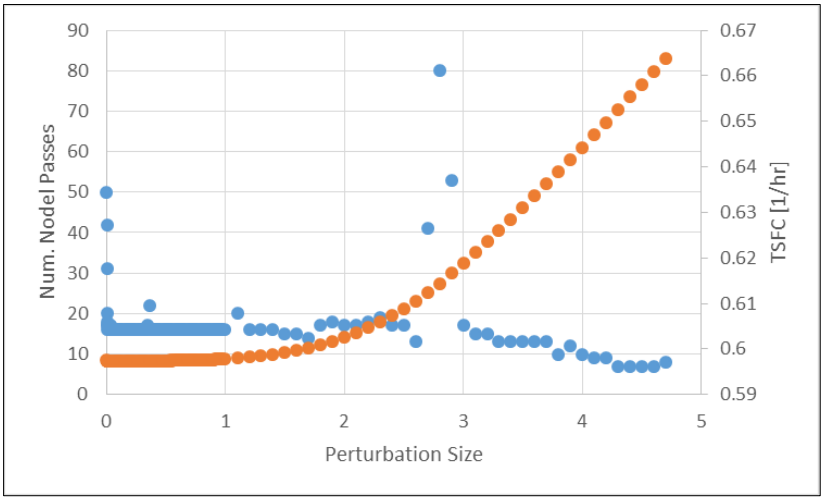

Figure 26: Effect of Linear Model Perturbation Size on Objective Function

Examination of Figure 27 reveals that there is a similar sweet spot for the optimization variable BPR lying in the same region of perturbation sizes. It can be seen that BPR however is slightly more sensitive to perturbation size than TSFC, with BPR values varying by up to 1.9%. This is still a small variation, but again the same result as experiment 4 could have been obtained in far fewer model passes.

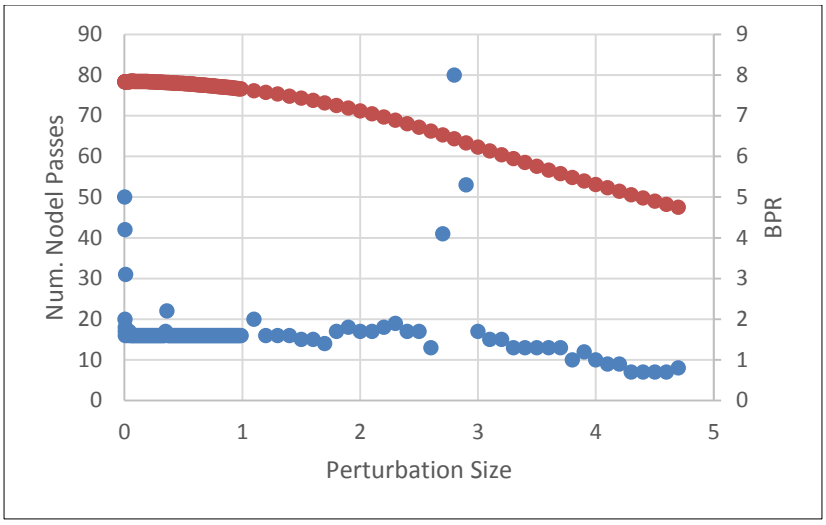

Figure 27: Effect of Linear Model Perturbation Size on Optimization Variable

This leads to a general methodology for selection of a perturbation size. When determining the optimization variables to use, limits should also have been established on how far the variable can realistically vary within the design space. These limits can be used to set the perturbation size in an intelligent manner. Take a small number called x_{bound} and use

$$x_{perturb} = \left(x_{i,max} - x_{i,min}\right) * x_{bound} \quad (36).$$

x_{bound} should be representative of the difference in the maximum and minimum values identified. In general, x_{bound} should start at 0.01 for differences on the order of 1, and decrease by an order of 10 for every order of 10 increase in the difference. So for a difference of 10 to 100 x_{bound} should be 0.001. This in fact was the method used for experiments 1 through 3. Used for experiment 4, it produces a perturbation value of $x_{perturb} = (9 - 4.5) * 0.01 = 0.045$. This results in a minute difference in TSFC and BPR from the previous $x_{perturb}$ of 0.005, but reduces the number of model passes to 16. This method for selecting the perturbation size will be used for the remaining experiments.

5.3 Experiments 5-10: Engine Model Optimization with SBO and OMDP

Hypothesis 4 addresses the fact that what is considered as the optimum can vary based on the

intentions of the designer. In any case, it must be demonstrated that OMDP can produce fully feasible designs no matter the choice of optimization point. For the following experiments, a baseline was created for comparison of OMDP results. This baseline represents an SFTF as represented in the DPMM given in Table 7 with the solver setup as described in the previous chapter. The baseline values are given below in Table 11.

Table 11: Baseline Values for SFTF MDP Model

	Variable	Base Results
Responses	TOC TSFC [1/hr]	0.6548
	TKO TSFC [1/hr]	0.5141
	Cruise TSFC [1/hr]	0.6531
Optimization Variables	TOC BPR	5.0
	Cruise A8 [sq.in.]	574.2
	Cruise A16 [sq.in.]	1941.4
Limits	TOC T4 [R]	2925.7
	TKO T4 [R]	3250
	Cruise T4 [R]	2787.6
Requirements	TOC Fn [lbf]	11000
	TKO Fn [lbf]	35000
	Cruise Fn [lbf]	10000

This comparison will serve to highlight how each design point is affected by the choice of design point or points at which to optimize. And, while MDP ensures that the cycle at each design point will be feasible and meet all requirements, it does not mean that it will do so in a manner that is efficient. Attempting to globally optimize the engine model at every design point or optimization at any subset of points may result in a severely suboptimal solution at one design point. If this point is cruise for instance, and the aircraft is expected to spend the majority of time in cruise, it would not be desirable to have a suboptimal cycle at this point even if overall the cycle is more efficient. The selection of optimization point(s) can be thought of as differing optimization "strategies". The comparison of results from using different strategies will demonstrate how OMDP performs based on the desirements of the designer.

5.3.1 Experiment 5: Intra-point Optimization, Single Variable, Single Objective

Experiment 5 is intended to demonstrate the ability of MDP with SBO (OMDP) to minimize TSFC at a single design point using a single optimization variable. Both TSFC and the chosen optimization variable will be at the same design point. This experiment will in essence act as a proof of concept for OMDP. It is designed to be as simple as possible to demonstrate the use of SBO with MDP. The objective is to minimize TSFC at a single design point using a variable available for optimization while still meeting all performance requirements at all design points.

The variable used for optimization is Cruise Fan Nozzle Area (A16) and the response to optimize is Cruise TSFC. The LMG input variable is cruise A16 and the output variable is cruise TSFC. The resulting D matrix looks like

$$D = \left[\frac{\partial(TSFC_{Cruise})}{\partial(A_{16,Cruise})}\right]$$

Thus, the LMG writes D[0][0] to a text file which is then parsed by the main run file as the derivative left hand side. It is expected that TSFC at the cruise condition from running OMDP will be less than or equal to the result using MDP and the requirements will be met at all design points. It can be shown that fan nozzle area at cruise will be modified by the solver to find minimum cruise TSFC. The typical MDP setup must be modified for this case and A16 is taken out of the normal set of pre-included variables for cruise. Without this step, the solver will not be able to alter the value of A16.

5.3.1.1 Results

The evaluation of OMDP begins here with experiment 5. It is expected that there is a unique value of A16 for which TSFC is minimum. To demonstrate this, the MDP model was run for multiple values of A16, keeping all else constant. A16 was driven to specified values at 10sq.in. increments producing a curve with an obvious inflection point at which the TSFC is minimum at cruise. Additionally, the LMG was run at each point to give the derivative, shown in Figure 28 as the right hand axes. The inflection point corresponds to the point at which the derivative is zero, as expected. Finally, the MDP model was then run once with SBO included.

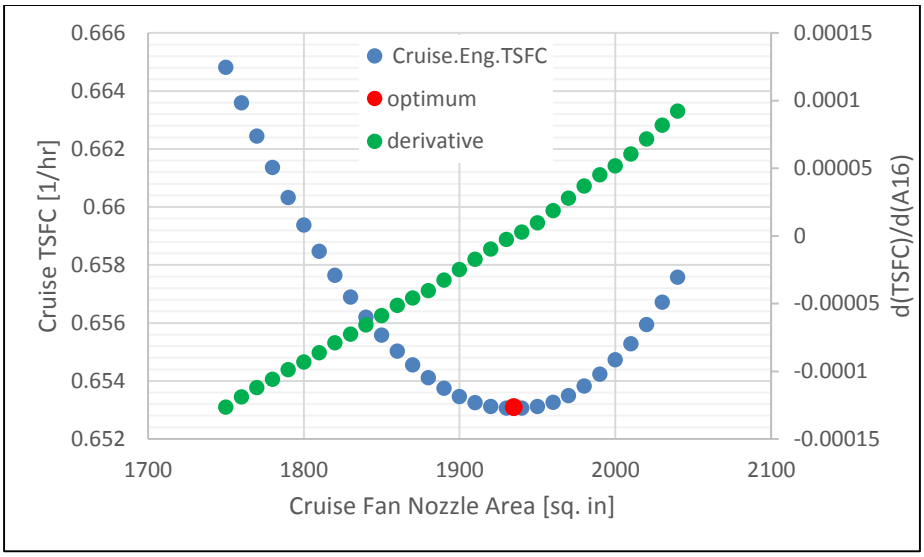

Figure 28: Cruise TSFC vs A16 and d(TSFC)/d(A16)

The final result of OMDP is the red point in Figure 28. It is located where expected at the inflection point of the curve. The main advantage of OMDP is its ability to find the optimum while simultaneously finding a balanced cycled for each design point included. A16 represents a VG features, showing that SBO can be used to schedule variable geometry features for optimum performance. Table 12 is a side-by-side comparison of OMDP with the baseline MDP results. The baseline A16 was already close to the optimum so the model did not have far to go, since the initial guess already gave MDP a converged solution. The cruise TSFC then did not noticeably drop, however it still met the criteria of being equal to or less than the baseline result. No constraints were violated and all requirements were met. It should also be pointed out that the results at design points other than cruise were not affected by optimization. This is as expected since the solver is only given control of A16 at cruise. A16 at all other design points should be, and is, the same as the baseline.

Table 12: Experiment 5 – Baseline vs OMDP Results

	Variable of Interest	Base Results	Exp. 5	Perturb 0.3
Responses	TOC TSFC [1/hr]	0.6548	0.6548	0.6548
	TKO TSFC [1/hr]	0.5141	0.5141	0.5141
	Cruise TSFC [1/hr]	0.6531	0.6531	0.6531
Optimization Variables	TOC BPR	5.0	5.0	5.0
	Cruise A8 [sq.in.]	574.192	574.2	574.2
	Cruise A16 [sq.in.]	1941.445	1934.7	1939.4
Limits	TOC T4 [R]	2925.655	2925.7	2925.7
	TKO T4 [R]	3250	3250	3250
	Cruise T4 [R]	2787.601	2787.3	2787.5
Requirements	TOC Fn [lbf]	11000	11000	11000
	TKO Fn [lbf]	35000	35000	35000
	Cruise Fn [lbf]	10000	10000	10000
Derivatives	d1	N/A	6.5E-14	-6.60E-14
	d2	N/A	-3.5E-10	-3.5E-10

This experiment was run twice. First, the default perturbation size was used for the LMG of 0.005. Then the method given in the last section for computing the perturbation size was again used. The calculated perturbation size for this experiment is $(2050 - 1750) * 0.001 = 0.3$. Figure 29 shows the effect of perturbation size required for convergence on number of model passes and fan nozzle exhaust area. Again, the more intelligent selection of perturbation size resulted in a decrease in the number of model passes from 55 to 30 – a reduction of 45%. It is also apparent that this resulted in a slightly different answer for nozzle area. However, this answer is different by only 0.2%, easily small enough to be acceptable for this application. The last column in Table 12 gives the results using the calculated perturbation size of 0.3. The A16 for minimum TSFC was only slightly higher and the majority of other results did not change at all.

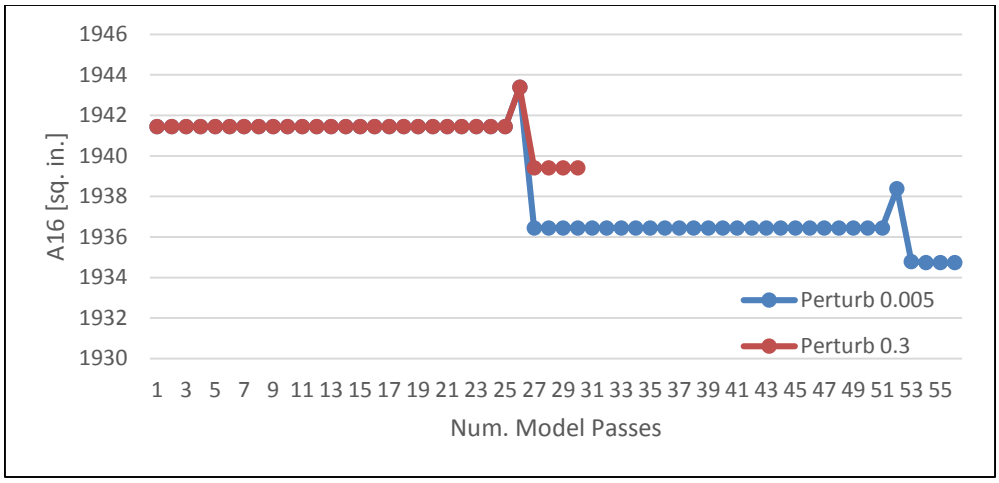

Figure 29: Effect of Perturbation Size on A16

5.3.2 Experiment 6 Cross-point Optimization, Single Variable, Single Objective

Experiment 6 is intended to demonstrate the ability of OMDP to minimize TSFC at a single design point using a single optimization variable. TSFC and the chosen optimization variable will be at separate points. The variable used for optimization is TOC BPR and the response to optimize is cruise TSFC. BPR is a cycle design variable selected at the on design point and is not considered variable geometry. However, it is still a user selectable parameter which SBO can be given control of. This experiment will demonstrate the flexibility of SBO to use any user selectable parameter, further investigating hypotheses 1, 3, and 4. It will also highlight the ability of OMDP to optimize across design points where the optimization variable is at a completely different point than the objective function.

The LMG input variable is TOC BPR and the output variable is cruise TSFC. The resulting D matrix looks like

$$D = \left[\frac{\partial(TSFC_{Cruise})}{\partial(BPR_{TOC})}\right]$$

Thus, the LMG writes D[0][0] to a text file which is then parsed by the main run file as the derivative left hand side. It is expected that TSFC at the cruise condition from running OMDP will be less than or equal to the result using MDP and the requirements will be met at all design points. It can be shown that BPR at TOC will be modified by the solver to find minimum cruise

TSFC. All requirements will be met. All constraints will not be violated.

5.3.2.1 Results

To create the BPR vs TSFC curve the MDP model was run for multiple values of BPR, keeping all else constant. BPR was driven to specified values at 0.5 increments producing the plot in Figure 30. Again, this mimics Figure 24 and shows a unique BPR giving a minimum TSFC at the inflection point of the curve. Finally, the MDP model was then run once with SBO included. The final result of OMDP is the red point in Figure 30. It is located where expected at the inflection point of the curve.

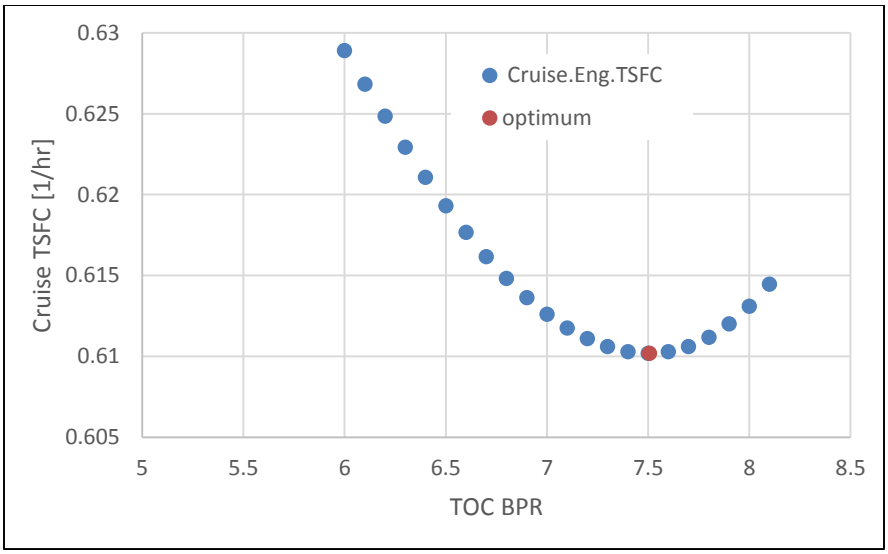

Figure 30: Cruise TSFC vs TOC BPR

Figure 31 and Figure 32 below should be viewed together. This provides a good insight into how the solver is operating to simultaneously balance the cycle and minimize TSFC. For Figure 31 the red points are ones which the solver is modifying only the engine match and user defined independents. The blue points are ones which the solver is also modifying BPR to minimize TSFC, and the green points are on the secondary right hand axis and are the computed derivative values at each model pass. In the first flat region in Figure 31 there is no movement on BPR and yet in the corresponding region in Figure 32 there are slight changes in TSFC. This region

corresponds to the initial pass of the solver through the model. The first sloped region the solver is attempting to shift the model towards convergence which includes modifying BPR towards a zero derivative and meeting; the error criteria for both the engine match relations and the user defined cycle relations. The solver again moves the BPR independent in the second sloped region, finally reaching the minimum TSFC and altering the remaining independents to reach convergence. It can be seen in the last flat region in Figure 31 that the final balancing of the cycle actually moves the derivative away from zero slightly towards a more negative value while the TSFC in Figure 32 increases very slightly. However this does not affect the derivative enough and the value stays close enough to zero to be within the specified tolerance of 1E-9.

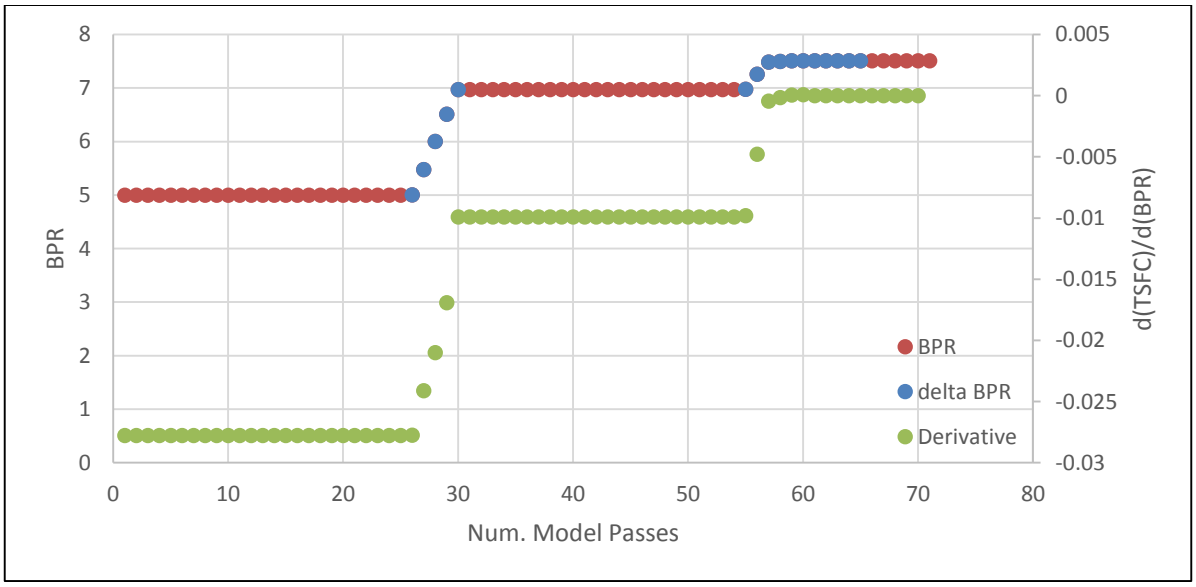

Figure 31: BPR_{TOC} vs Num. Model Passes

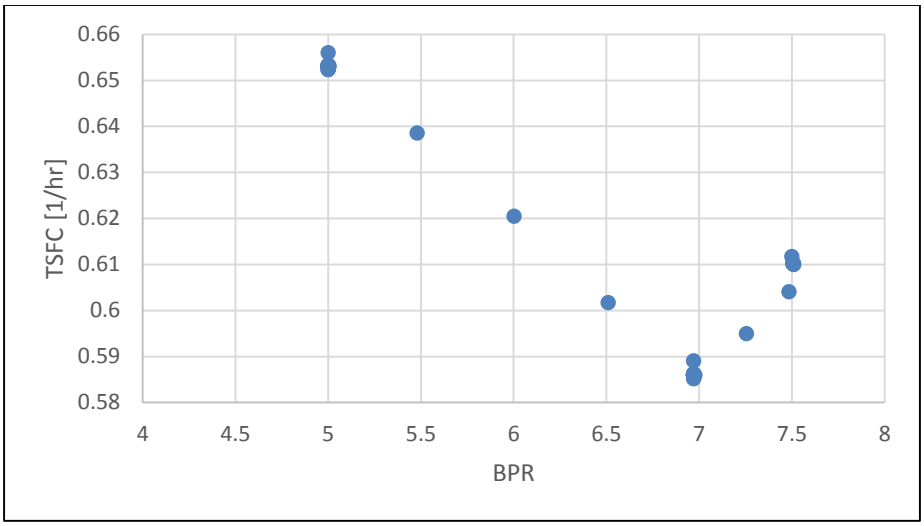

Figure 32: Calculated $TSFC_{cruise}$ vs BPR_{TOC}

Table 13 is a side-by-side comparison of OMDP with the baseline MDP results. The BPR at TOC was successfully varied by the solver to reach a minimum cruise TSFC without breaking any constraints, and while meeting the performance requirements. This cross-point optimization used BPR at TOC which would be expected to affect performance at other points and is indeed shown in the table below. Increasing the on design BPR also had the unintended, although not unexpected, result of reducing TOC and TKO TSFC.

Table 13: Experiment 6 – Baseline vs OMDP Results

	Variable of Interest	Base Results	Experiment 6
Responses	TOC TSFC [1/hr]	0.6548	0.6008
	TKO TSFC [1/hr]	0.5141	0.4504
	Cruise TSFC [1/hr]	0.6531	0.6102
Optimization Variables	TOC BPR	5.0	7.5
	Cruise A8 [sq.in.]	574.2	905.9
	Cruise A16 [sq.in.]	1941.4	2693.7
Limits	TOC T4 [R]	2925.7	2915.7
	TKO T4 [R]	3250	3250
	Cruise T4 [R]	2787.6	2800.2
Requirements	TOC Fn [lbf]	11000	11000
	TKO Fn [lbf]	35000	35000
	Cruise Fn [lbf]	10000	10000
Derivatives	d(TSFC)/d(BPR)	N/A	-2.2E-10

5.3.3 Experiment 7: Intra-point Optimization, Multi-variable, Multi-objective

Experiment 7 is intended to demonstrate how one variable at each design point can be used to optimize a single response at an equal number of design points. TOC BPR is used to optimize TOC TSFC and cruise A8 is used to optimize cruise TSFC. The typical MDP setup must be modified for this case and A8 is taken out of the normal set of pre-included variables for cruise. Without this step, the solver will not be able to alter the value of A8.

The LMG input variables are cruise A8 and TOC BPR the output variables are cruise and TOC TSFC. The resulting D matrix has more terms than are being used, with the highlighted items being used for optimization.

$$D = \begin{bmatrix} \boldsymbol{\frac{\partial(TSFC_{TOC})}{\partial(BPR_{TOC})}} & \frac{\partial(TSFC_{TOC})}{\partial(A_{8,Cruise})} \\ \frac{\partial(TSFC_{Cruise})}{\partial(BPR_{TOC})} & \boldsymbol{\frac{\partial(TSFC_{Cruise})}{\partial(A_{8,Cruise})}} \end{bmatrix}$$

Thus, the LMG run file writes D[0][0] and D[1][1] to a text file which is then parsed by the main

run file as the left hand side of each derivative. It is expected that TSFC at cruise and TOC condition from running OMDP will be less than or equal to the result using MDP. The requirements will be met all design points without breaking any constraints.

5.3.3.1 Results

Multiple variables may be used to simultaneously optimize multiple responses. In this case, the variable and response it is attempting to optimize are at the same design point. This experiment provides further proof for hypothesis 4. It is clear from Table 14 that both responses were reduced from their baseline values by allowing the solver to modify BPR at TOC and A8 at cruise. Again as a consequence of a higher on design BPR the TKO TSFC is reduced from its baseline value.

This same consequence would apply to cruise TSFC without the addition of A8 as an optimization variable. However, adding A8 allows further optimization at the cruise point. One of the key points of OMDP is that it removes the need to find an on design cycle and then in series find an optimal off design cycle. In this experiment an optimal and balanced on design cycle was found while simultaneously finding an optimal off design cycle. Contrary to any previous cycle optimization methods discussed earlier, no additional manual work was required, nor was any type of nested structure or outside optimization algorithms. The on design changes are automatically captured and used in off design while all requirements were met at all design points. This can be seen in the slightly higher BPR value than in experiment 6 which reflects the change in main exhaust nozzle area at cruise.

Table 14: Experiment 7 – Baseline vs OMDP Results

	Variable of Interest	Base Results	Experiment 7
Responses	TOC TSFC [1/hr]	0.6548	0.6002
	TKO TSFC [1/hr]	0.5141	0.4479
	Cruise TSFC [1/hr]	0.6531	0.6058
Optimization Variables	TOC BPR	5	7.7
	Cruise A8 [sq.in.]	574.2	831.7
	Cruise A16 [sq.in.]	1941.4	2781.2
Limits	TOC T4 [R]	2925.7	2909.9
	TKO T4 [R]	3250	3250
	Cruise T4 [R]	2787.6	2812.8
Requirements	TOC Fn [lbf]	11000	11000
	TKO Fn [lbf]	35000	35000
	Cruise Fn [lbf]	10000	10000
Derivatives	d(TOC_TSFC)/d(TOC_BPR)	N/A	-7E-12
	d(Cruies_TSFC)/d(Cruise_A8)	N/A	5.7E-12

5.3.3.2 Results with Added Constraints

Constraint handling is an important aspect of the solver for engine balance and optimization. This was explored in earlier experiments, however not yet with a multi-variable and multi-objective optimization case. To further explore this, a constraint was added to BPR. Suppose that it was determined that the maximum allowable BPR for this engine is 6.5. A constraint was added to the derivative dependent for d(TOC_TSFC)/d(TOC_BPR) constraining BPR to a maximum of 6.5 and then experiment 7 was rerun. The results shown in Figure 33 is the result after 72 model passes, at which point the simulation was manually halted. Once BPR reaches a value of 6.5, the constraint becomes active and the derivative value is no longer in the dependent. The value of TSFC at this point is 0.6163 – less than the baseline but still greater than what is possible. All independents have reached a value which allows convergence except for the second optimization independent value of A8. This is the reason the simulation must be halted manually and represents a limitation of SBO as it currently stands.

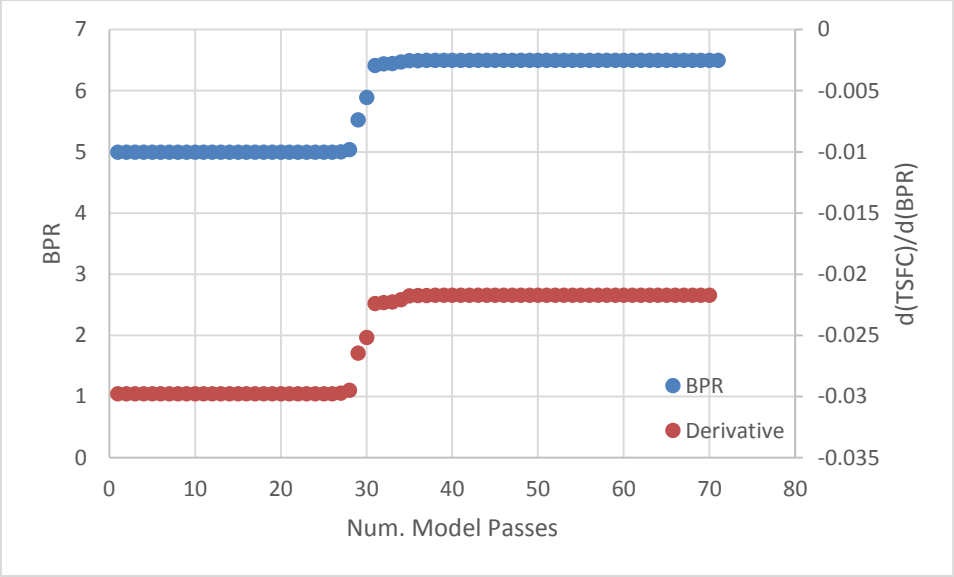

Figure 33: Num. Model Passes vs BPR and $d(TSFC_{TOC})/d(BPR_{TOC})$

Figure 34 shows that the derivative of TSFC at cruise with respect to cruise A8 gets very close to zero at around 1E-8, but not close enough to be within the specified tolerance of 1E-9. A zero derivative cannot be reached in this case due to the constraint on BPR. It is obvious then that a current limitation of the method is that it does not incorporate some way for the optimization independents to be aware of active constraints on other optimization dependents. Since the second derivative dependent value here can never be reached, the simulation continues to run until the maximum number of iterations are reached unless stopped prematurely.

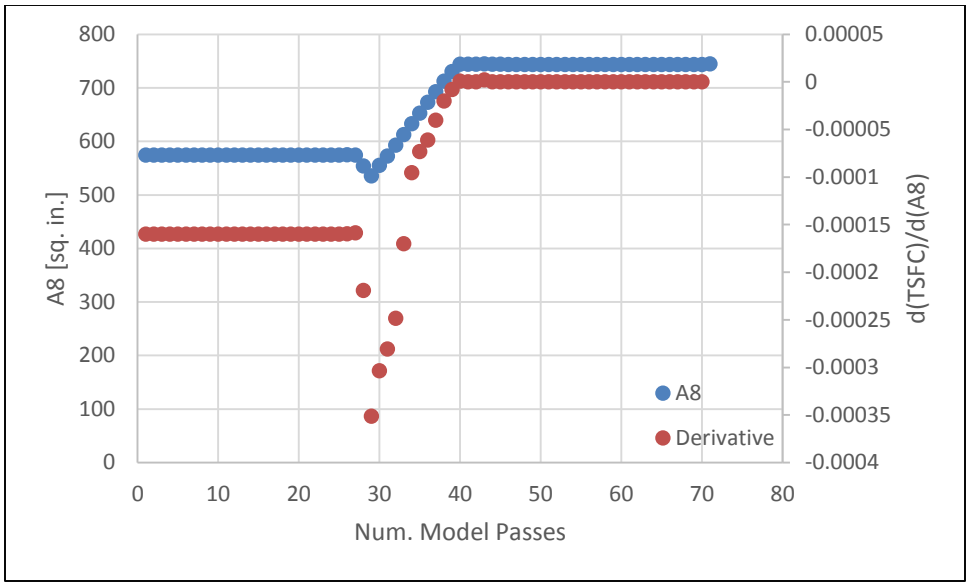

Figure 34: Num. Model Passes vs $A8_{Cruise}$

It would notionally be possible to incorporate this check into SBO. A flowchart for how this might be done is provided in Figure 35. This could be conceivable incorporated in two places. The most obvious is to create a function which does the checking and outputs a result which is then incorporated into a dependent. A dependent to check for these active constraints would then be added to every optimization dependent. The second solution would be to create a similar function which is added appended to the solver in the same way that the linear model generator is added.

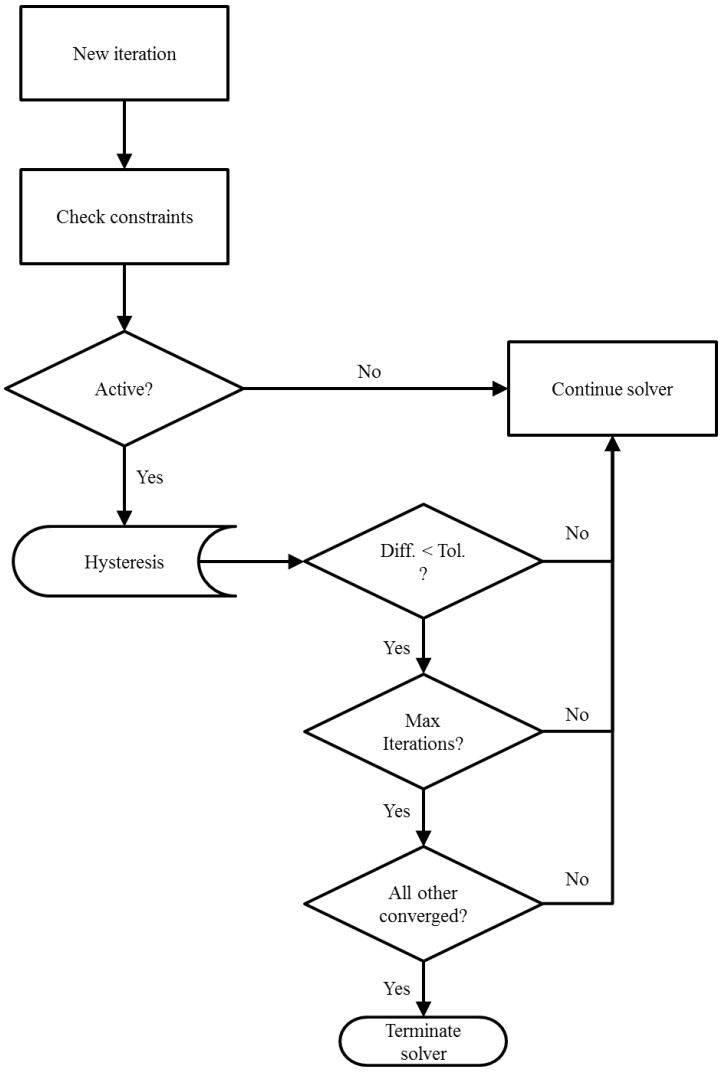

Figure 35: Active Optimization Constraint Check Process Flowchart

Following Figure 35, at the end of each solver iteration a check is performed to check for active constraints on optimization objective. If none are active, the solver continues unhindered. If a constraint is active, then the hysteresis would track the change in the other active optimization dependents at each iteration and the number of consecutive iterations with the same active constraints. If the change is less than a certain amount for a specified number of solver iterations, then a check would be performed on all other solver variables. If the solver is otherwise converged, then the simulation should be allowed to exit. All other paths lead to continuation of the solver.

5.3.4 Experiment 8: Cross-point Optimization, Multi-variable, Single Objective

Experiment 8 is intended to demonstrate how one variable at each of multiple design points can be used to optimize a single response at a single design point. TOC BPR and cruise A16 are used to optimize cruise TSFC. The typical MDP setup must be modified for this case and A16 is taken out of the normal set of pre-included variables for cruise. Without this step, the solver will not be able to alter the value of A16.

The LMG input variables are cruise A16 and TOC BPR and the output variable is cruise TSFC. The resulting D matrix is

$$D = \left[\frac{\partial(TSFC_{Cruise})}{\partial(BPR_{TOC})} \quad \frac{\partial(TSFC_{Cruise})}{\partial(A_{16,Cruise})}\right]$$

Thus, the LMG run file will write D[0][0] and D[0][1] to a text file which is then parsed by the main run file as the left hand side of each derivative. It is expected that TSFC at cruise from running OMDP will be less than or equal to the result using MDP. The requirements will be met all design points without breaking any constraints.

5.3.4.1 Results

Before examining the final results of this experiment, there were several lessons learned from this experiment which should be noted. They followed from the first few attempts to run OMDP for this experiment which were unsuccessful. To better understand where the problem was, MDP was run through many variations of TOC BPR and cruise A16 to get a better idea of what the design space looks like. As Figure 36 shows, the minimum TSFC lies somewhere in the neighborhood of a 7.5 BPR and 2600 sq. in. fan nozzle area.

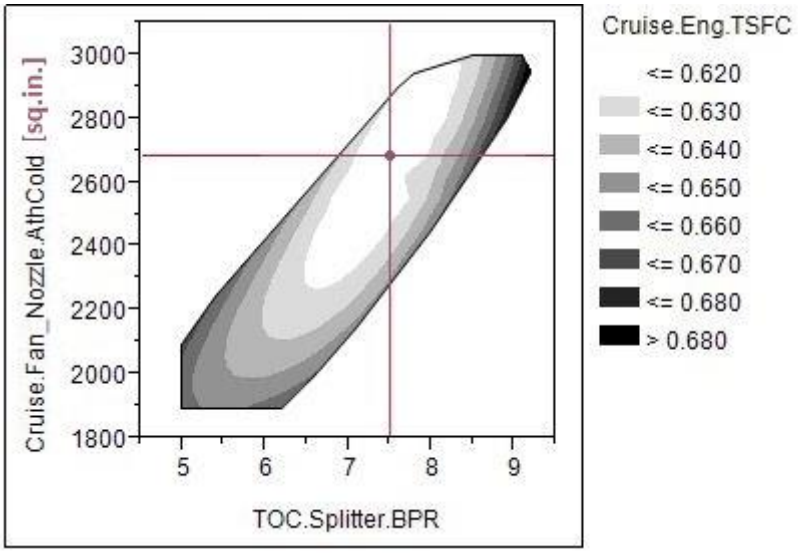

Figure 36: Contour Plot of Cruise A16 and TOC BPR vs Cruise TSFC

After further examination of the iteration details it became apparent that SBO was getting stuck in a relatively shallow region of the design space with regards to TSFC. Raising the dxLimit on the optimization dependent allowed the solver to move out of this shallow region towards the true optimum. Thus, the first lesson is that the dxLimit for the dependent does have an effect on whether the model converges. Initially the limit was set at 5 for A16, which does not allow A16 to vary enough to get out of sub-minimum shallow regions of the design space.

However leaving the dxLimit unset also stalls convergence as the default dxLimit used by the NPSS solver is too low. With dxLimits of 10 and 100 the model converged and an optimum was found. These limits allowed the solver to break out of sub-optimal points without allowing it to step too far into a region that it cannot recover from. It was found that in this case there is a wide margin for selection of the dxLimit. After repeated attempts it became obvious when the limit was too low. In this case the solver would often move an optimization independent the maximum allowed for many iterations in a row. Without sufficient improvement in the derivative the solver would then backtrack and the independent would move backwards, exactly as shown in Figure 37 with a dxLimit of 5 on cruise A16.

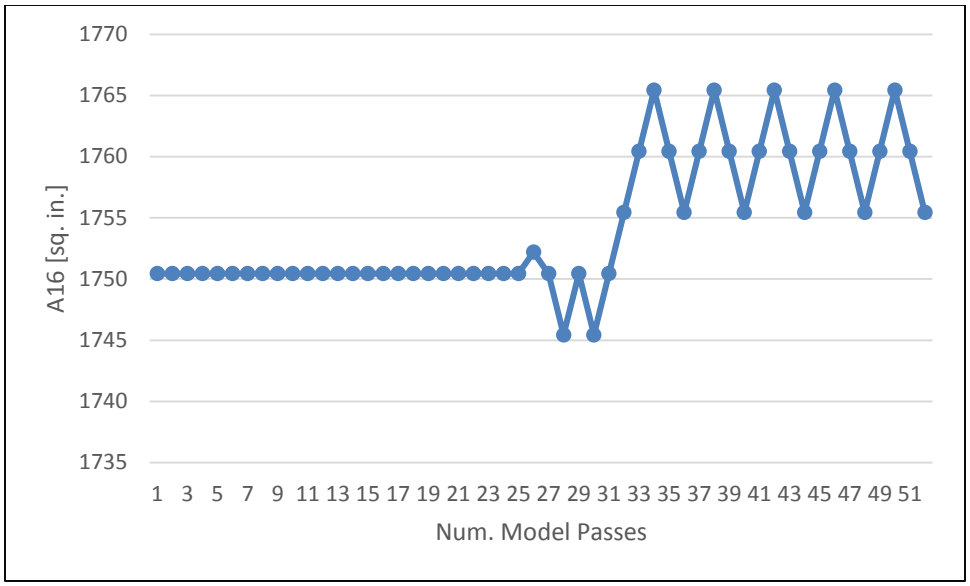

Figure 37: Insufficient dxLimit Oscillatory Behavior

For this experiment, increasing the dxLimit had the effect of reducing the number of model passes required for convergence as seen in Figure 38. Below a step size of 10, the solver could not move the independent enough and experienced the oscillatory behavior seen in Figure 37. Setting a limit of 1000 moved the limit well beyond the maximum the solver ever tried to step the A16, which was about 200 square inches. Thus in this experiment, there is no upper limit on A16 step size but in general a reasonable upper limit should be specified on step size to help with highly nonlinear objectives.

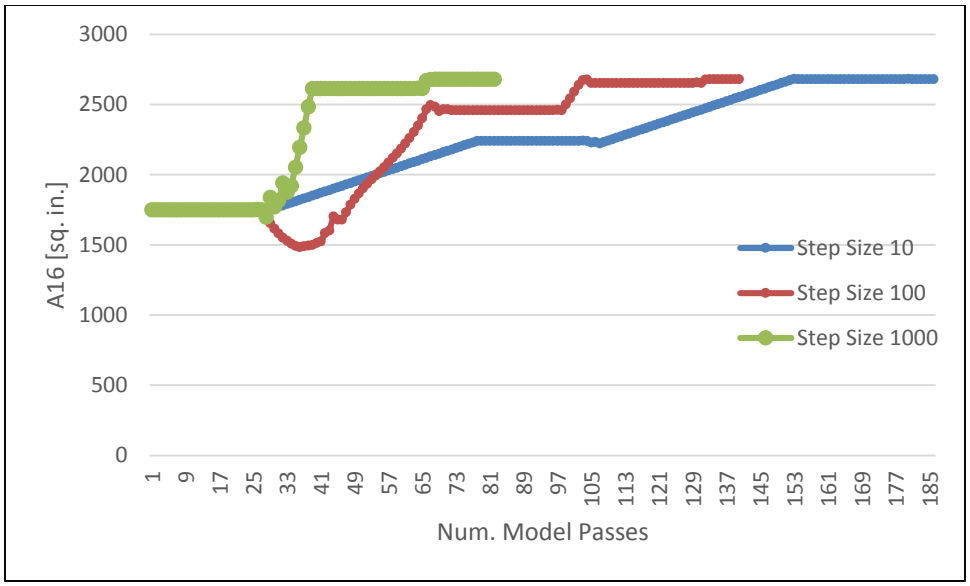

Figure 38: Effect of Step Size on Number of Model Passes

While trying different variations of this experiment, it became apparent that the LMG typically will only take about 10 or fewer model passes to converge. Once past that, the LMG never will converge. This occurs when the main solver changes the state of the model such that the LMG cannot converge. But, within several passes through the model the solver moves the state to a place where the LMG can again converge. The second lesson learned is then to set the number of passes allowed for the LMG low. It was not uncommon for the LMG to be unable to converge for one or two model passes. Leaving the solver to allow too many model passes for the LMG significantly increased the overall time to convergence even in cases where the LMG only fails a handful of times. It was found that 50 passes for the LMG was sufficient to greatly speed up the method, while still being conservative enough to handle tough points.

With more reasonable limits placed on the dependent A16, OMDP was able to converge to an optimum solution at the expected point in the design space as shown by the crosshairs in Figure 36. Comparison of the results against the baseline shows that the cruise TSFC was reduced. All requirements were met without breaking any constraints. These results are with a dxLimit on A16 of 10 sq.in. Increasing the limit to 100 produced nearly identical results but at fewer model passes and slightly faster time to overall convergence. This highlights the effect the dxLimit may have on not just the ability to converge, but time to convergence. In these experiments there are only 3 design points and time to convergence is on the order of minutes.

However as the number of solver variables increases, this time increases and it may be advantageous to select a limit that allows convergence and decreases convergence time.

Table 15: Experiment 8 – Baseline vs OMDP Results

	Variable of Interest	Base Results	Experiment 8
Responses	TOC TSFC [1/hr]	0.6548	0.6008
	TKO TSFC [1/hr]	0.5141	0.4503
	Cruise TSFC [1/hr]	0.6531	0.6102
Optimization Variables	TOC BPR	5	7.5
	Cruise A8 [sq.in.]	574.2	906.7
	Cruise A16 [sq.in.]	1941.4	2680.6
Limits	TOC T4 [R]	2925.7	2915.6
	TKO T4 [R]	3250	3250
	Cruise T4 [R]	2787.6	2799.9
Requirements	TOC Fn [lbf]	11000	11000
	TKO Fn [lbf]	35000	35000
	Cruise Fn [lbf]	10000	10000
Derivatives	d(Cruies_TSFC)/d(Cruise_A16)	N/A	1.04E-13
	d(Cruies_TSFC)/d(TOC_BPR)	N/A	-5.3E-10

5.3.5 Experiment 9: Cross-point Optimization, Single Variable, Multi-objective

Experiment 9 is intended to demonstrate that a single variable at a single design point can be used to optimize a response at multiple design points. It was already shown that TOC BPR has an effect on both TOC and cruise TSFC. Thus the solver setup here will be done in such a way as to attempt to use TOC BPR to optimize both TOC TSFC and cruise TSFC.

The LMG input variables is TOC BPR the output variables are cruise and TOC TSFC. The resulting D matrix has more terms than are being used, with the highlighted items being used for optimization.

$$D = \begin{bmatrix} \frac{\partial(TSFC_{TOC})}{\partial(BPR_{TOC})} \\ \frac{\partial(TSFC_{Cruise})}{\partial(BPR_{TOC})} \end{bmatrix}$$

Thus, the LMG run file writes D[0][0] and D[1][0] to a text file which is then parsed by the main run file as the left hand side of each derivative.

5.3.5.1 Results

The aim of this experiment was to see if it is possible to use a single variable to optimize multiple responses. This, however, is problematic for SBO due to how the solver is implemented. The solver only allows one dependent to be active per independent. There is a single derivative value per response, thus attempting to add both derivatives as independents results in having one too many dependents.

Several attempts were made to try and combine the two derivatives into one dependent. However, combinations of the two derivatives gives no clear way for the solver to alter the independent value to drive the derivatives to zero. Another attempt was made to use features of the solver itself. In that case, the second derivative dependent was added twice as a constraint on the first derivative independent – once as a minimum and once as a maximum. The idea was to trick the solver into alternate between using BPR_{TOC} to drive both derivatives to zero. However for an unknown reason the constraints never became active. This resulted in one derivative being driven to zero but the other left at a value well away from zero.

Due to the way the NPSS solver works, it does not currently appear that SBO can handle the case of single variable, multi-objective optimization. More research should be done though before this is completely ruled out. There may still be a way to utilize some built in features of the solver to realize this goal.

5.3.6 Experiment 10: Optimization Starting Point Study

The OMDP method follows from a typical MDP setup where it is given an initial guess derived from a converged point in the solution space. This holds true for any starting point of the optimization variable for the variable examined – in this case TOC BPR.

The very first initial iterate for MDP came from running an SDP model with TOC as the map

scaling point, exactly as was used for experiment 4. The cruise point is close enough to TOC that the off design cruise engine match independent values were identical to TOC and a user defined independent was selected to drive thrust to a target value using burner fuel flow. Finally, a TKO point was run in off design and the independent values captured. The only user defined relation at TKO used FAR to meet a thrust target. Once these values were obtained and input for the initial iterate for MDP, the model converged. Then the converged values were used as the initial iterate causing the solver to converge in a single pass when re-run with this new initial iterate. This initial iterate was used to create all other initial iterates for various starting values of BPR in this experiment.

5.3.6.1 Results

Using the initial guess from experiments 5-9 and starting BPR at each of the values listed in Table 16 OMDP failed to converge to an optimum. Good initial iterates were then obtained by using the same methodology as described above. The same initial iterate was used as experiments 5-9 except a normal MDP setup was used and incrementally worked down or up to the desired BPR value. Once the model converged with the new BPR, the independent values were captured and used as the initial iterates for each of OMDP sub-experiments here. With a good initial iterate for each BPR_{TOC} starting point, the method was able to converge to the same BPR and TSFC values for starting point values of BPR_{TOC} from 5 to 9. The path taken can be seen in Figure 39. In general, the closer the starting point is to the optimum BPR, the fewer model passes required to converge.

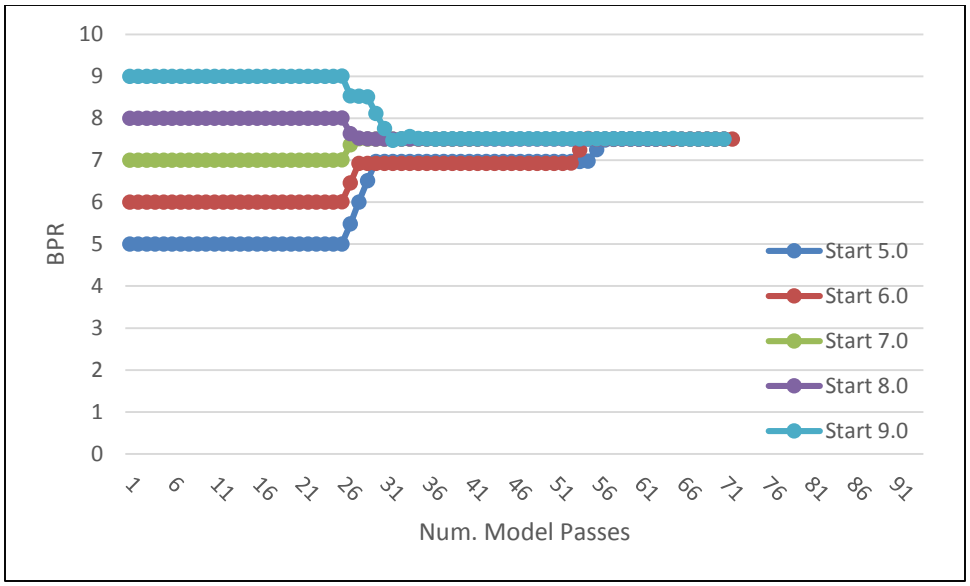

Figure 39: Effect of Starting Point on Num. Model Passes and End Point

As seen in Table 16 are from BPR 0 to 2 the model failed to converge. In this region, the normal MDP model will not converge to a solution. Thus OMDP cannot move itself from a region well outside of the normal converged region of the design space to an optimum. In this example, it is obvious where this region lies and why the model cannot converge. A zero value for BPR produces a singularity and too low of BPR values choke the flow and the solver cannot move the model state to convergence. The next exceptions are at BPRs of 3 and 4, as can be seen in Table 16. These anomalies will be explained momentarily.

Table 16: OMDP Starting Point Results

BPR_{init}	BPR*	TSFC*	Num. Model Passes
0	N/A	N/A	N/A
1	N/A	N/A	N/A
2	N/A	N/A	N/A
3	4.2	0.6784	96
4	4.1	0.6785	422
5	7.5	0.6102	71
6	7.5	0.6102	72
7	7.5	0.6102	42
8	7.5	0.6102	63
9	7.5	0.6102	71

It is often not possible to ensure that the absolute optimum design will be found in the application of optimization techniques to design problems of practical interest. This may be due to a variety of reasons, but from a practical standpoint the best option is often to choose a number of initial vectors to start the optimization process. [39] However, in Chapter 3 a restriction was placed on the method to ensure that an optimum was found. This restriction, given mathematically as Eq. (29), states that the function being optimized must be convex. While this restriction may be ignored, it opens up the possibility of SBO finding a sub-optimum. This is exactly what happens for BPR values below approximately 4.2. A practical explanation of a convex set is that for any two points in the set, a straight line drawn between them will not fall outside the set. This is obviously not true in Figure 40 where there exists a very flat region. At a starting point of 3 for BPR, OMDP converged to the suboptimum value of 4.2. For starting points closer to this flat region such as a BPR of 4, the method was extremely unstable and failed it took 422 model passes and still was unable to converge to the true optimum.

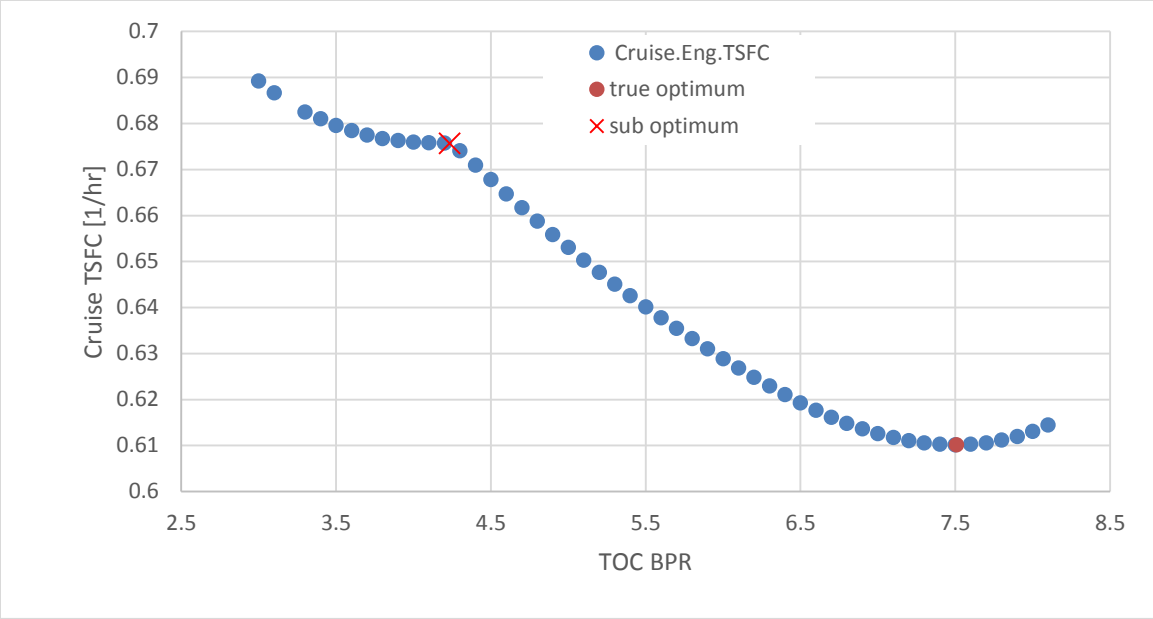

Figure 40: Non-Convex Design Space

CHAPTER 6
CONCLUSIONS

This work develops a method for simultaneously balancing and optimizing an engine cycle using a Newton-Raphson solver. This method is demonstrated using an SFTF model which is implemented in NPSS. Combination of this method with MDP results in a method for designing gas turbine engine cycles that simultaneously meet requirements and constraints at multiple design points while also optimizing an objective function. The method has been proven against analytical functions with known optimum and against engine models. It is also left general enough to be used with our without MDP and with almost any conceivable engine architecture and allows optimization with any user selectable parameter.

6.1 Research Questions and Hypotheses

The first research question of this thesis was how can optimal settings be found for engines incorporating variable geometry in a way that is efficient and robust enough to handle many configurations quickly? The first hypothesis concerning this question stated that the solver and a local linear model can be combined to produce an optimization method. Experiments 1-3 were able to confirm this hypothesis by successfully finding a known minimum of several analytical functions – reaching comparable accuracy and model passes to a number of known optimization algorithms. The second hypothesis stated that the method would be able to optimize an engine cycle while simultaneously balancing the cycle. This was proven by experiment 4 with an SFTF engine model. Theoretically, for an SFTF, there is a unique BPR giving a minimum TSFC. The method was successfully able to find this point and balance the cycle with minimum run time.

The second question asked if the on design and associated off design search space of an engine can be optimized while simultaneously balancing the cycle at all design points while including all desired cycle performance requirements and constraints. The third hypothesis addressed this question and stated that combining SBO with MDP would produce an optimization method capable of simultaneously optimizing the cycle while meeting all performance requirements and constraints across all included design points. The fourth

hypothesis stated that the derivative information produced by the linear model can be used to optimize one or multiple responses with one or many variables. Experiments 5-9 were designed to investigate the capabilities of the method and the variety of ways that the derivative information can be used. These experiments successfully proved hypothesis 3 by optimizing the SFTF model and meeting all desired performance requirements and constraints at all design points. This included single and multi-variable/objective optimization within and across design points, thus proving hypothesis 4. Three design points were included in each experiment at TOC, TKO, and cruise.

Hypothesis 5 stated that since the MDP method includes an initial guess from a previously converged point, the method will be insensitive to starting point. Experiment 10 proved this and also showed why a restriction on convexity is placed on the objective function to guarantee an optimum. These experiments showed that SBO can easily be combined with OMDP to further explore the design space and take into account multiple design points at the same time.

6.2 Research Contributions

A number of contributions are being made to the field of propulsion system design with this thesis. The first is the SBO method itself. While originally developed for use with VCEs, it is widely applicable to any engine design having additional degrees of freedom which can be used to optimize the cycle. SBO directly incorporates the derivative information available from a local linear approximation into a Newton-Raphson solver to find an optimum engine cycle for an architecture incorporating variable geometry. Combination of these two pieces results in a gradient based optimization method which greatly simplifies optimization of an engine as compared to previous methods. Additionally, the general nature of SBO allows it to be used with almost any architecture.

An added bonus of SBO is the ability to use it directly with MDP. This further speeds up design, optimization, and analysis by combining on design, off design, and optimization into a single, general, and simultaneous implementation. The end product of OMDP is an engine design that is optimum while still being feasible at all design points, meets all performance requirements, and does not exceed any constraints.

6.3 Recommendations for Future Work

The initial motivation for SBO came from the optimization problem inherent in VCE architectures. Thus the natural next step is to use the method on a VCE model. This may start out with the optimization of the model set up with a single design point. However multiple design points may size a VCE, thus OMDP is ideal for VCE designs and would provide the most benefit in this application. The general nature of the method however lends itself to any engine architecture and it would be informative to demonstrate the method on other classes of problems.

Another natural progression of SBO is to include it in the Environmental Design Space (EDS), as described in Reference 2 (AIAA 2012-3812). MDP has already been included in the EDS and is capable of analyzing a variety of engine architectures and properly matching the engine to aircraft. SBO does not require any direct modification to the MDP method itself, thus incorporating it into EDS with MDP should be relatively straightforward.

Finally, the constraint handling features of SBO should be more fully vetted as that is an extremely important aspect of engine systems and optimization. All experiments used constraints, however only the analytical test functions and part of experiment 7 included constraints on the objective function. The constraints were easily handled in the analytical cases, but experiment 7 demonstrated that there are currently some limitations on constraint handling for SBO in certain cases. More robust constraint handling would be important for constraints such as a max fan diameter constraint on BPR. Investigation of constraint handling and updating the method to reflect this would be an important and useful avenue for future research.

APPENDIX A: NPSS ANALYTICAL MODEL FILES

The NPSS files listed here were used in experiments 1-3:

- A1: NPSS Main Run File
- A2: NPSS LMG Run File
- A3: NPSS Dummy Model File for Analytical Function
- A4: NPSS Setup Function File

A1: NPSS Main Run File

```
#include <basic_functions.npss>
#include <print_macros.fnc>
#include <Hack.cs> //dummy element
#include <test_func.mdl> //dummy model
#include <parseFunctions.npss>

real x1; //function variable
real x2; //function variable
real y_current; //holds current value of y at each iteration

////// for Rosenbrock test function
real x1max=2;
real x1min=-2;
real x2max=4;
real x2min=-1;
x1 = -1; //current x1 value for each iteration
x2 = 1; //current x2 value for each iteration
////////////////////////////

real d1 = 0; //derivative value for x1_dep
real d2 = 0; //derivative value for x2_dep

OutFileStream vars_list;
OutFileStream metrics;
metrics.open("metrics.csv");

void call_me()
  {

      //write out variables needed by LMG to text file
    vars_list.open("vars.in");
      vars_list << "x1 " << " = " << toStr(x1,10)    <<        ";\n" ;
      vars_list << "x2 " << " = " << toStr(x2,10)    <<        ";\n" ;
      vars_list << "x1max " << " = " << toStr(x1max,10)    <<        ";\n" ;
      vars_list << "x1min " << " = " << toStr(x1min,10)    <<        ";\n" ;
      vars_list << "x2max " << " = " << toStr(x2max,10)    <<        ";\n" ;
      vars_list << "x2min " << " = " << toStr(x2min,10)    <<        ";\n" ;
      vars_list << "d1 " << " = " << toStr(d1,10)    <<        ";\n" ;
      vars_list << "d2 " << " = " << toStr(d2,10)    <<        ";\n" ;
      vars_list.close();

      //run the LMG
      system("run LMG.run");

      y_current = (1-x1)**2 + 100*(x2-x1**2)**2; //store computed function value

      //parse derivative values
      d1 = toReal(parse("Dmatrix", 3, 1, 0, "LMG_vars.in"));
      d2 = toReal(parse("Dmatrix", 4, 1, 0, "LMG_vars.in"));
      close_parse();

      //output variables of interest at each iteration
```

```
    //alternative to using viewers
    metrics << toStr(y_current,10)      <<      ", " ;
    metrics << toStr(x1,10)     <<      ", " ;
    metrics << toStr(x2,10)     <<      ", " ;
    metrics << toStr(d1,10)     <<      ", " ;
    metrics << toStr(d2,10)     <<      ", " ;
    metrics << "\n";

}
 //append call_me() to the solver
    solver.executionSequence.append ("call_me");

    #include <setup.cs> //solver variable definitions

    //solver setup
    solver.clear();
    solver.maxIterations = 2000;
    solver.maxJacobians = 20;
    autoSolverSetup();

    solver.addIndependent("x1_Indep");
    solver.addDependent("x1_dep");

    solver.addIndependent("x2_Indep");
    solver.addDependent("x2_dep");

    // ********diagnostics stuff*************
    // solver.solutionMode="ONE_PASS";
    // solver.debugLevel = "ITERATION_DETAILS";
  // solver.diagnosticFile = "solver_Iteration.Output";
    // ********diagnostics stuff*************

    run();
    printPride();
    metrics.close();
```

A2: NPSS LMG Run File

```
//
//-------------------------------------------------------------------------
//                                                      |
//   File Name:     LMG.run                                  |
//   Date(s):       Feb 3 2014                             |
//   Author:        Sean T Ford                                 |
//                                                      |
//   Description:   LMG run file for optimization of main model               |
//                                                      |
//-------------------------------------------------------------------------

#include <Hack.cs> //dummy element

real x1;
real x2;
real deltaBound = 0.001;
real deltax1;
real deltax2;
real x1max;
real x1min;
real x2max;
real x2min;
real d1;
real d2;

// real x1start;
// real x2start;
real D[][]; //2D variable to store LMG D matrix

//give the model the state it was in from the last iteration of the main run file
#include <vars.in>

//set bounds on how much the LMG can perturb each input variable
deltax1 = (x1max-x1min)*deltaBound;
deltax2 = (x2max-x2min)*deltaBound;

//include the model
#include <test_func.mdl>

//set current state of model (comes from vars.in)
dummy.x1 = x1;
dummy.x2 = x2;

//setup LMG. See User Guide for more details on setup
//linearized model generator ------------------------------------
LinearModelGenerator Test_LMG
{
      inputVars = {"dummy.x1", "dummy.x2" }
      outputVars = {"dummy.y"}
      setPerturb("dummy.x1", deltax1);
      setPerturb("dummy.x2", deltax2);
      setPerturbType("dummy.x1","ABSOLUTE");
      setPerturbType("dummy.x2","ABSOLUTE");
      reportFileName = "junk";
}
```

```
//clear out the solver
solver.clear();
autoSolverSetup();

//generate derivatives around given point
Test_LMG.generate();

//output D matrix for debugging
cout << "D matrix \n " << Test_LMG.D << endl;

//store D matrix from LMG
D = Test_LMG.D;

//get derivatives from D matrix that must be written out to be used by main run file
real D1 = D[0][0];
real D2 = D[0][1];

//output derivative info from LMG to text file to read by main run file
//**note, must write 1 more values than is used
//otherwise parse() won't be able to read the variables in the main run file.
//I don't know why this is, just know this is a problem from experience
OutFileStream LMG_vars_list;
      LMG_vars_list.open("LMG_vars.in");
      LMG_vars_list << "Dmatrix " << "= " << toStr(D1,10) << " " << toStr(D2,10);
      LMG_vars_list.close();
```

A3: NPSS Dummy Model File for Analytical Function

```
class Hack extends Element {
//dummy element to compute function value
  real x1;
  real x2;
  real y;
  void calculate() { //calculate function value

       y = (1-x1)**2 + 100*(x2-x1**2)**2; //Rosenbrock function
  }

  void postexecute() {
     cout<<"y = "<<y<<endl;
  }
} //end Hack
```

A4: NPSS Setup Function File

```
//constraint, independent, dependent definitions

Dependent con_x1_max    {eq_lhs = "x1"; eq_rhs = "x1max";}
Dependent con_x1_min    {eq_lhs = "x1"; eq_rhs = "x1min";}
Dependent con_x2_max    {eq_lhs = "x2"; eq_rhs = "x2max";}
Dependent con_x2_min    {eq_lhs = "x2"; eq_rhs = "x2min";}

Independent x1_Indep {
       varName = "x1";
       dxLimit = .5;
       dxLimitType = "ABSOLUTE";
       }

Dependent x1_dep{
                eq_lhs = "d1";
                eq_rhs = "0.0";
                constraintNameList = {"con_x1_max","con_x1_min"};
                limitTypes = {"MAX","MIN"};
                toleranceType = "ABSOLUTE";
     }

     Independent x2_Indep {
       varName = "x2";
       dxLimit = .5;
       dxLimitType = "ABSOLUTE";
       }


Dependent x2_dep{
                eq_lhs = "d2";
                eq_rhs = "0.0";
                constraintNameList = {"con_x2_max","con_x2_min"};
                limitTypes = {"MAX","MIN"};
                toleranceType = "ABSOLUTE";
     }
```

APPENDIX B: NPSS OMDP MODEL FILES

The NPSS files listed here were used in experiments 5-10:

- B1: Main Run File
- B2: LMG Run File
- B3: NPSS SFTF Model File
- B4: Setup Function File
- B5: Assembly Setup File
- B6: MDP Scalars File
- B7: SFTF Function File

B1: NPSS Main Run File

```
//-------------------------------------------------------------------------
//                                                      |
//   File Name:     runDesign.run                              |
//   Date(s):       Feb 3 1988                           |
//   Author:        Sean T Ford                             |
//   Description:   Design point for SFTF                        |
//                                                      |
//-------------------------------------------------------------------------

cout << "\n\nSTART RUN:  TIME = "<< timeOfDay << "\n\n";

setThermoPackage("GasTbl");
#include <basic_functions.npss>
#include <parseFunctions.npss>

#define TOCdef SET
#define TKOdef SET
#define Cruisedef SET
#define optimize SET //for easy switching between OMDP and MDP

#include <print_macros.fnc>
#include <SFTF_functions.cs>
#include <guessIRP_SFTF.fnc>

//-------------------------------------------------------------------------
//                        Output Data Viewers
//-------------------------------------------------------------------------
#include <sftf_page_164.view>
#include <sftf_164_row.view>  // RKD 12-12-07
#include <SFTF.view_flops>

#include <shaft1.int>
#include <Duct1.int>
#include <Burner1.int>
#include <CDTH.int>
#include <CVELOCITY.int>
#include <COMPRESSOR_REYNOLDS_EFFECTS.int>
#include <TURBINE_REYNOLDS_EFFECTS.int>

//-------------------------------------------------------------------------
//      Design Case
//-------------------------------------------------------------------------
real d1 = 0; //storage for derivative
real d2 = 0;

solver.resetConstraints();
#include <assembly_setup.cs> //set up assemblies
#include <solver_setup.cs> //set up all solver variables
#include <guess.cs> //give model a good starting point

TOC.Splitter.BPR = 9.0;
```

```
//create output streams for LMG vars and metrics.csv
OutFileStream vars_list;

#ifdef optimize
//call me function used to pass LMG the model state and call the LMG
void call_LMG()
 {
    //write model variables needed by LMG
  vars_list.open("vars.in");
            //design point conditions
            vars_list << "      TOC.Ambient.alt
   "        <<      " = " << toStr(TOC.Ambient.alt,10)          <<      ";\n" ;
            vars_list << "      TOC.Ambient.MN
   "        <<      " = " << toStr(TOC.Ambient.MN,10)          <<      ";\n" ;
            vars_list << "      TKO.Ambient.alt
   "        <<      " = " << toStr(TKO.Ambient.alt,10)          <<      ";\n" ;
            vars_list << "      TKO.Ambient.MN
   "        <<      " = " << toStr(TKO.Ambient.MN,10)          <<      ";\n" ;
            vars_list << "      Cruise.Ambient.alt                                                  "
   <<       " = " << toStr(Cruise.Ambient.alt,10)          <<      ";\n" ;
            vars_list << "      Cruise.Ambient.MN
   "        <<      " = " << toStr(Cruise.Ambient.MN,10)          <<      ";\n" ;
            //solver independents
            vars_list << "      TOC.HPT.S_map.parmMap
   "        <<      " = " << toStr(TOC.HPT.S_map.parmMap,10)          <<      ";\n" ;
            vars_list << "      TOC.Ambient.W
   "        <<      " = " << toStr(TOC.Ambient.W,10)          <<      ";\n" ;
            vars_list << "      TOC.LPT.S_map.parmMap
   "        <<      " = " << toStr(TOC.LPT.S_map.parmMap,10)          <<      ";\n" ;
            vars_list << "      TOC.Burner.FAR
   "        <<      " = " << toStr(TOC.Burner.FAR,10)          <<      ";\n" ;
            vars_list << "      TKO.Ambient.W
   "        <<      " = " << toStr(TKO.Ambient.W,10)          <<      ";\n" ;
            vars_list << "      TKO.Fan.S_map.RlineMap
   "        <<      " = " << toStr(TKO.Fan.S_map.RlineMap,10)          <<      ";\n" ;
            vars_list << "      TKO.HPC.S_map.RlineMap
   "        <<      " = " << toStr(TKO.HPC.S_map.RlineMap,10)          <<      ";\n" ;
            vars_list << "      TKO.HPT.S_map.parmMap
   "        <<      " = " << toStr(TKO.HPT.S_map.parmMap,10)          <<      ";\n" ;
            vars_list << "      TKO.HP_SHAFT.Nmech
   "        <<      " = " << toStr(TKO.HP_SHAFT.Nmech,10) <<      ";\n" ;
            vars_list << "      TKO.LPC.S_map.RlineMap
   "        <<      " = " << toStr(TKO.LPC.S_map.RlineMap,10)          <<      ";\n" ;
            vars_list << "      TKO.LPT.S_map.parmMap
   "        <<      " = " << toStr(TKO.LPT.S_map.parmMap,10)          <<      ";\n" ;
            vars_list << "      TKO.LP_SHAFT.Nmech
   "        <<      " = " << toStr(TKO.LP_SHAFT.Nmech,10) <<      ";\n" ;
            vars_list << "      TKO.Splitter.BPR
   "        <<      " = " << toStr(TKO.Splitter.BPR,10)          <<      ";\n" ;
            vars_list << "      TKO.Burner.FAR
   "        <<      " = " << toStr(TKO.Burner.FAR,10)          <<      ";\n" ;
            vars_list << "      Cruise.Ambient.W
   "        <<      " = " << toStr(Cruise.Ambient.W,10)          <<      ";\n" ;
            vars_list << "      Cruise.Fan.S_map.RlineMap
   "        <<      " = " << toStr(Cruise.Fan.S_map.RlineMap,10)          <<      ";\n" ;
            vars_list << "      Cruise.HPC.S_map.RlineMap
```

```
    "       <<      " = " << toStr(Cruise.HPC.S_map.RlineMap,10)    <<      ";\n" ;
            vars_list << "      Cruise.HPT.S_map.parmMap
    "       <<      " = " << toStr(Cruise.HPT.S_map.parmMap,10)     <<      ";\n" ;
            vars_list << "      Cruise.HP_SHAFT.Nmech
    "       <<      " = " << toStr(Cruise.HP_SHAFT.Nmech,10)        <<      ";\n" ;
            vars_list << "      Cruise.LPC.S_map.RlineMap
    "       <<      " = " << toStr(Cruise.LPC.S_map.RlineMap,10)    <<      ";\n" ;
            vars_list << "      Cruise.LPT.S_map.parmMap
    "       <<      " = " << toStr(Cruise.LPT.S_map.parmMap,10)     <<      ";\n" ;
            vars_list << "      Cruise.LP_SHAFT.Nmech
    "       <<      " = " << toStr(Cruise.LP_SHAFT.Nmech,10)        <<      ";\n" ;
            vars_list << "      Cruise.Splitter.BPR
    "       <<      " = " << toStr(Cruise.Splitter.BPR,10)      <<      ";\n" ;
            vars_list << "      Cruise.Burner.Wfuel
    "       <<      " = " << toStr(Cruise.Burner.Wfuel,10)      <<      ";\n" ;

            //optimization variables
            vars_list << "      TOC.Splitter.BPR                                                    "
    <<      " = " << toStr(TOC.Splitter.BPR    ,10)     <<      ";\n" ;
    vars_list.close();

    //run the LMG
    system("run LMG.run");

    //update the rowviewer
    rowSheet.update();

    //parse the derivative value(s)
    d1 = toReal(parse("Dmatrix", 3, 1, 0, "LMG_vars.in"));
    close_parse();
    cout << "\n\n *****d1 = " << d1 << " ******* \n\n";
 }

 //append call_me() to the solver
    solver.executionSequence.append ("call_LMG");

    //optimization independent/dependent definition
    Independent opt_Indep { varName = "TOC.Splitter.BPR"; dxLimit = 5; dxLimitType = "ABSOLUTE"; }
  Dependent opt_Dep{eq_lhs = "d1"; eq_rhs = "0.0"; toleranceType = "ABSOLUTE";}

    //add optimization variables to the solver
    solver.addIndependent("opt_Indep");
    solver.addDependent("opt_Dep");

#endif
//***************diagnostics stufff**************************
// cout << "\n\n DEPENDENTS \n" << solver.dependentNames << endl;
// cout << "\n\n INDEPENDENTS \n" << solver.independentNames << endl;
// cout << "\n\n Constraints \n" << solver.constraintNames << endl;
// setOption("solutionMode", "ONE_PASS");
solver.debugLevel = "ITERATION_DETAILS";
solver.diagnosticFile = "solver_Iteration.cs";
// solver.debugLevel = "MATRIX_DETAILS";
// solver.diagnosticFile = "solver_matrix.cs";
//************************************************************
```

```
                run();
                rowSheet.update();

pv.display();
printPride();
write_indeps();
// write_scalars();
#ifdef TOCdef
      TOC.pointTOC.display();
#endif
#ifdef TKOdef
      TKO.pointTKO.display();
#endif
#ifdef Cruisedef
      Cruise.pointCR.display();
#endif



cout << "COMPLETED RUN:  TIME = "<< timeOfDay << endl;
```

B2: LMG Run File

```
//-------------------------------------------------------------------------
//                                                      |
//   File Name:     LMG.run                                  |
//   Date(s):       Feb 3 2014                          |
//   Author:        Sean T Ford                              |
//   Description:   LMG run file for optimization of main model              |
//                                                      |
//-------------------------------------------------------------------------

//include thermo package
setThermoPackage("GasTbl");

//include macros. don't currently use any but may be useful for debugging later
#include <print_macros.fnc>

//include function file and SFTF model
#define TOCdef SET
#define TKOdef SET
#define Cruisedef SET
#include <SFTF_functions.cs>

#include <shaft1.int>
#include <Duct1.int>
#include <Burner1.int>
#include <CDTH.int>
#include <CVELOCITY.int>
#include <COMPRESSOR_REYNOLDS_EFFECTS.int>
#include <TURBINE_REYNOLDS_EFFECTS.int>

#include <assembly_setup.cs> //set up assemblies
#include <solver_setup.cs> //set up all solver variables

real deltax1 = 10; //max amount LMG can perturb model by

solver.maxIterations = 50;

real D[][]; //2D variable to store LMG derivative matrix

//give the model the state it was in from the last iteration of the main run file
#include <vars.in>


//setup LMG. See User Guide for more details on setup
LinearModelGenerator Test_LMG
{
      inputVars = {"TOC.Splitter.BPR"}
      outputVars = {"Cruise.Eng.TSFC"}
      reportFileName = "junk";
}

//execute the LMG
Test_LMG.execute();
```

```
//this output mostly for debugging to get a visual that everything is working in LMG.run
cout << "D matrix \n " << Test_LMG.D << endl;

//save off D matrix
D = Test_LMG.D;

//get derivatives from D matrix that must be written out to be used by main run file
real D1 = D[0][0];
real D2 = 0;


//this output mostly for debugging to get a visual that everything is working in LMG.run
cout << "D1 \n " << D1 << endl;

//output derivative info from LMG to text file to read by main run file
//**note, must write at least 2 values even if the second one is not used for anything
//otherwise parse() won't be able to read the variables in the main run file.
//I don't know why this is, just know this is a problem from experience
OutFileStream LMG_vars_list;
    LMG_vars_list.open("LMG_vars.in");
    LMG_vars_list << "Dmatrix" << " = " << toStr(D1,10) << " " << toStr(D2,10);
    LMG_vars_list.close();
```

B3: NPSS SFTF Model File

```
//-------------------------------------------------------------------------
//                                                      |
//   File Name:     SFTF.mdl                           |
//   Date(s):       January 21, 2012                   |
//   Author:        Russ Denney
//                          modified for MDP by STF 25 Apr 2014                |
//   Description:   Separate Flow Turbofan engine model              |
//                                                      |
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
//                  User-Defined Tables and Functions
//-------------------------------------------------------------------------
Table PercentPower( real PC ) {
  PC =      {  50.0, 21.0,  20.0 }  // R. Denney 6-21-05
  PartPwr = {   1.0, 0.10,  0.05 }
}

//-------------------------------------------------------------------------
//                        Model Definition
//-------------------------------------------------------------------------
MODELNAME = "2-Spool Separate Flow Turbofan";
AUTHOR = "R. Denney";

Element FlightConditions Ambient {
   //alt = 30000;
   //MN = 0.80;
   //W = 270.0;
}

Element Inlet Inlet {
   Fl_O.MN = 0.5;
   Subelement ramRecovery S_rec {  // this gives Mil Spec ram recovery vs. Mach
      eRam = 1.0;  }
//   eRamBase = 1.00;
}

Element Duct IGVDuct {
   Fl_O.MN = 0.4;
   dPqPbase = 0.000;
}

Element Compressor Fan {
   #include <CMGENFan.map>

   Fl_O.MN = 0.4;
   S_map.effDes = 0.878;
   S_map.PRdes = 1.68;
   S_map.RlineMap = 2.00;
   S_map.NcDes = 1.0;    // set to 1.0 in ncp file R. Denney 6-21-05
}

Element Splitter Splitter {
```

```
  BPR = 5.92;
  Fl_01.MN = 0.45; // swan neck duct inlet Mach number
  Fl_02.MN = 0.4;  // bypass duct inlet Mach number
}

Element Compressor LPC {
  #include <ncp13.map>

  Fl_O.MN = 0.4;
  S_map.effDes = 0.890;
  S_map.NcDes = 1.0;
  S_map.PRdes = 2.43/1.68;
  S_map.RlineMap = 2.0;
}

Element Duct SwanNeckDuct {
  Fl_O.MN = 0.4;
  dPqPbase = 0.0;
}

Element Compressor HPC {
  #include <CMGENHPC.map>

  Fl_O.MN = 0.4;
  S_map.effDes = 0.863;
  S_map.NcDes = 1.0;
  S_map.PRdes = 11.56;
  S_map.RlineMap = 2.0;

  InterStageBleedOutPort  Cool3 {    // LPT inlet cooling flow
    fracBldWork = 0.4; //0.3500;
    fracBldP = 0.16;   //0.1465;
    fracBldW = 0.0;
  }
  InterStageBleedOutPort  Cool4 {    // LPT exit cooling flow
    fracBldWork = 0.4; //0.3500;
    fracBldP = 0.16;   //0.1465;
    fracBldW = 0.0;
  }

  InterStageBleedOutPort  CustomerBld {
    fracBldWork = 0.57;
    fracBldW = 0.0000;
  }

  void preexecute() {
    CustomerBld.fracBldW = 0.0/Fl_I.W;

  }
}

Element Bleed CDPBld {
  Fl_O.MN = 0.35; // OGV inlet Mach number
  WrefName = "HPC.Fl_I.W";

  BleedOutPort  Cool1 {
```

```
    fracW = 0.1300;
  }

  BleedOutPort  Cool2 {
    fracW = 0.1300;
  }

}

Element Duct OGVduct {
  Fl_O.MN = 0.2;  // burner inlet Mach number
  dPqPbase = 0.0;
}

Element FuelStart Fuel {
  LHV = 18400;
}

Element Burner Burner {
  TtCombOut = 3090.0;
  FAR = 0.03576;
  Wfuel = 2.;
  effBase = 1.0;
  dPqPBase = 0.045;
  Fl_O.MN = 0.10; // required to get a Ps

  //switchBurn = "TEMPERATURE";
}

Element Turbine HPT {
  #include <ncp04.map>

  FlowStation FS41;

  Fl_O.MN = 0.4;

  S_map.parmMap = 3.996;    // was 2.80 R. Denney 11-20-07
  S_map.effDes = 0.891;
  S_map.parmNcDes = 100;
  S_map.parmGeomMap = 1.0;

  InterStageBleedInPort  Non_ChargeableBld {
    Pfract = 1.0;
  }

  InterStageBleedInPort  ChargeableBld {
    Pfract = 0.0;
  }
}

Element Duct ITTduct {  // Duct used to connect the two turbines for WATE - RKD 4-22-08
  Fl_O.MN = 0.4;   // Same as HPT
  dPqPbase = 0.0;
}

Element Turbine LPT {
```

```
  #include <ncp05.map>

  Fl_O.MN = 0.4;   // TEGV inlet Mach number

  S_map.parmMap = 2.4944;  // was 2.20 R. Denney 11-20-07
  S_map.effDes = 0.939;
  S_map.parmNcDes = 100.;
  S_map.parmGeomMap = 1.0;

  FlowStation FS49;

  InterStageBleedInPort  Non_ChargeableBld {
    Pfract = 1.0;
  }

  InterStageBleedInPort  ChargeableBld {
    Pfract = 0.0;
  }
}

Element Duct TEGVduct {
  Fl_O.MN = 0.25;
  dPqPbase = 0.00;
}

Element Duct Tailpipe {
  Fl_O.MN = 0.25;
  dPqPbase = 0.00;
}

Element Duct BPduct {
  Fl_O.MN = 0.40;
  dPqPbase = 0.00;
}

Element Nozzle Core_Nozzle {
  switchType = "CONIC";
  switchCoef = "CFG";
  Cfg = 0.992;

  PsExhName = "Ambient.Ps";
} // end Core_Nozzle

Element Nozzle Fan_Nozzle {
  switchType = "CONIC";
  switchCoef = "CFG";
  Cfg = 0.979;

  PsExhName = "Ambient.Ps";
} // end Fan_Nozzle

Element FlowEnd Core_Nozz_End {
}

Element FlowEnd Fan_Nozz_End {
}
```

```
Element FlowEnd OBsink1 { }  // flow end for customer bleed

Element Shaft1 HP_SHAFT {
  ShaftInputPort HPC, HPT;

  HPX = 85.0;
  Nmech = 100.0;

}

Element Shaft1 LP_SHAFT {
  ShaftInputPort FAN, LPC, LPT;

  Nmech = 100.0;
}

Element EngPerf Eng {

  real FnFullPower, ThrustTarget, EPR, PC;
  real EPRtarget; real pcn2max;

  void postexecute() {
    EPR = LPT.Fl_O.Pt / Fan.Fl_I.Pt ;          // changed to Pt5 from Pt7 R. Denney 6-21-05
    OPR = HPC.Fl_O.Pt / Fan.Fl_I.Pt ;          // OPR is Pt3 / Pt2 R. Denney 6-21-05
  }
}


//--------------------------------------------------------------------------
//      linkPorts
//--------------------------------------------------------------------------

linkPorts( "Ambient.Fl_O"       , "Inlet.Fl_I"          , "FS0"   );
linkPorts( "Inlet.Fl_O"         , "IGVDuct.Fl_I"        , "FS1"   );
linkPorts( "IGVDuct.Fl_O"       , "Fan.Fl_I"            , "FS2"   );
linkPorts( "Fan.Fl_O"           , "Splitter.Fl_I"       , "FS21"  );
linkPorts( "Splitter.Fl_01"     , "LPC.Fl_I"            , "FS23"  );
linkPorts( "LPC.Fl_O"           , "SwanNeckDuct.Fl_I"   , "FS24"  );
linkPorts( "SwanNeckDuct.Fl_O"  , "HPC.Fl_I"            , "FS25"  );
linkPorts( "HPC.Fl_O"           , "CDPBld.Fl_I"         , "FS3"   );
linkPorts( "CDPBld.Fl_O"        , "OGVduct.Fl_I"        , "FS31"  );
linkPorts( "OGVduct.Fl_O"       , "Burner.Fl_I"         , "FS32"  );
linkPorts( "Fuel.Fu_O"          , "Burner.Fu_I"         , "FS36"  );
linkPorts( "Burner.Fl_O"        , "HPT.Fl_I"            , "FS4"   );
linkPorts( "HPT.Fl_O"           , "ITTduct.Fl_I"        , "FS45"  );
linkPorts( "ITTduct.Fl_O"       , "LPT.Fl_I"            , "FS48"  );
linkPorts( "LPT.Fl_O"           , "TEGVduct.Fl_I"       , "FS5"   );
linkPorts( "TEGVduct.Fl_O"      , "Tailpipe.Fl_I"       , "FS6"   );
linkPorts( "Tailpipe.Fl_O"      , "Core_Nozzle.Fl_I"    , "FS7"   );
linkPorts( "Core_Nozzle.Fl_O"   , "Core_Nozz_End.Fl_I"  , "FS9"   );

// BYPASS linkPorts
linkPorts( "Splitter.Fl_02"     , "BPduct.Fl_I"         , "FS14"  );
linkPorts( "BPduct.Fl_O"        , "Fan_Nozzle.Fl_I"     , "FS17"  );
linkPorts( "Fan_Nozzle.Fl_O"    , "Fan_Nozz_End.Fl_I"   , "FS19"  );
```

```
// BLEED linkPorts
linkPorts("CDPBld.Cool1" ,          "HPT.Non_ChargeableBld", "C_FS41" );
linkPorts("CDPBld.Cool2"   ,        "HPT.ChargeableBld"    , "C_FS42" );
linkPorts("HPC.Cool3"      ,     "LPT.Non_ChargeableBld"   , "C_FS48" );
linkPorts("HPC.Cool4"      ,       "LPT.ChargeableBld"    , "C_FS49" );
linkPorts("HPC.CustomerBld"        , "OBsink1.Fl_I"         , "OB_Cust");

// SHAFT linkPorts
linkPorts( "Fan.Sh_O"          , "LP_SHAFT.FAN"        , "FANwork" );
linkPorts( "LPC.Sh_O"          , "LP_SHAFT.LPC"        , "LPCwork" );
linkPorts( "LPT.Sh_O"          , "LP_SHAFT.LPT"        , "LPTwork" );
linkPorts( "HPT.Sh_O"          , "HP_SHAFT.HPT"        , "HPTwork" );
linkPorts( "HPC.Sh_O"          , "HP_SHAFT.HPC"        , "HPCwork" );

//-------------------------------------------------------------------------------
//  Solver Sequence
//-------------------------------------------------------------------------------
```

B4: Setup Function File

```
//******solver setup for all design points**************

#ifdef TOCdef
    TOC{setOption ("switchDes","DESIGN"); }
#endif
#ifdef TKOdef
    TKO{setOption ("switchDes","OFFDESIGN"); }
#endif

#ifdef Cruisedef
    Cruise{setOption ("switchDes","OFFDESIGN"); }
#endif

CASE = 1;
solver.clear();
autoSolverSetup();

solver.defaultTolerance = 0.000000001;
solver.defaultToleranceType = "FRACTIONAL";  //ABSOLUTE
solver.maxJacobians = 600;
solver.maxIterations = 600;

solver.defaultDxLimit = 0.1;

#ifdef TOCdef
    TOC.Ambient.alt = 30000;
    TOC.Ambient.MN = 0.8;

    TOC.Burner.switchBurn = "FAR";
    TOC.Eng.ThrustTarget = 11000;

    solver.addIndependent( "TOC_ind_W1" );
    solver.addDependent( "TOC_dep_fntarget" );

    solver.addIndependent("TOC_ind_FAR"); //vary to meet TKO thrust constrained by most constraining T4
    solver.addDependent("TKO_dep_fntarget");

    TOC.Fan.S_map.alpha=0.;              //set IGV full open
    TOC.LPC.S_map.alpha=0.;               //set vgv full open
    TOC.HPC.S_map.alpha=0.;               //set stp full open

#endif

#ifdef TKOdef
    TKO.Ambient.alt = 0;
    TKO.Ambient.MN = 0.3;

    TKO.Burner.switchBurn = "FAR";
    TKO.Eng.ThrustTarget = 35000;

    solver.addIndependent("TKO_ind_FAR"); //vary to target fan corrected speed.
    solver.addDependent( "TKO_dep_pcn2max" );
```

```
#endif

#ifdef Cruisedef
    Cruise.Ambient.alt =28000;
    Cruise.Ambient.MN = 0.8;
    Cruise.Burner.switchBurn = "FUEL";
    Cruise.Eng.ThrustTarget = 10000;

    solver.addIndependent("CR_ind_zwf36");
    solver.addDependent( "CR_dep_fntarget" );
#endif
```

B5: Assembly Setup File

```
//****************** define assemblies ******************

#ifdef TOCdef //ON DESIGN point
    Element Assembly TOC{
        #include <SFTF.mdl>
    OutFileStream pointTOCStream { filename = "pointTOC.viewOut"; }
        DataViewer PageViewer pointTOC {
        #include <pointTOC.view>
        outStreamHandle = "pointTOCStream";}
    }

#endif


#ifdef TKOdef
    Element Assembly TKO{
        #include <SFTF.mdl>

        OutFileStream pointTKOStream { filename = "pointTKO.viewOut"; }
        DataViewer PageViewer pointTKO {
        #include <pointTKO.view>
        outStreamHandle = "pointTKOStream";}

        void preexecute(){
            #include <mdp_scalars.int>
        }
    }

#endif


#ifdef Cruisedef
    Element Assembly Cruise{
        #include <SFTF.mdl>

        OutFileStream pointCRStream { filename = "pointCR.viewOut"; }
        DataViewer PageViewer pointCR {
        #include <pointCR.view>
        outStreamHandle = "pointCRStream";}

        void preexecute(){
            #include <mdp_scalars_Cruise.int>
        }
    }

#endif
```

B6: MDP Scalars File

```
Inlet.eRam=TOC.Inlet.eRam;
Inlet.eRamBase=TOC.Inlet.eRamBase;
Fan.S_map.s_effDes=TOC.Fan.S_map.s_effDes;
Fan.S_map.s_NcDes=TOC.Fan.S_map.s_NcDes;
Fan.S_map.s_PRdes=TOC.Fan.S_map.s_PRdes;
Fan.S_map.s_WcDes=TOC.Fan.S_map.s_WcDes;
HPC.S_map.s_effDes=TOC.HPC.S_map.s_effDes;
HPC.S_map.s_NcDes=TOC.HPC.S_map.s_NcDes;
HPC.S_map.s_PRdes=TOC.HPC.S_map.s_PRdes;
HPC.S_map.s_WcDes=TOC.HPC.S_map.s_WcDes;
LPC.S_map.s_effDes=TOC.LPC.S_map.s_effDes;
LPC.S_map.s_NcDes=TOC.LPC.S_map.s_NcDes;
LPC.S_map.s_PRdes=TOC.LPC.S_map.s_PRdes;
LPC.S_map.s_WcDes=TOC.LPC.S_map.s_WcDes;
HPT.S_map.s_dPqP=TOC.HPT.S_map.s_dPqP;
HPT.S_map.s_Np=TOC.HPT.S_map.s_Np;
HPT.S_map.s_eff=TOC.HPT.S_map.s_eff;
HPT.S_map.s_effRe=TOC.HPT.S_map.s_effRe;
HPT.S_map.s_Wp=TOC.HPT.S_map.s_Wp;
HPT.S_map.s_parmGeom=TOC.HPT.S_map.s_parmGeom;
HPT.S_map.s_parmMap=TOC.HPT.S_map.s_parmMap;
LPT.S_map.s_effRe=TOC.LPT.S_map.s_effRe;
LPT.S_map.s_eff=TOC.LPT.S_map.s_eff;
LPT.S_map.s_Np=TOC.LPT.S_map.s_Np;
LPT.S_map.s_dPqP=TOC.LPT.S_map.s_dPqP;
LPT.S_map.s_Wp=TOC.LPT.S_map.s_Wp;
LPT.S_map.s_parmMap=TOC.LPT.S_map.s_parmMap;
LPT.S_map.s_parmGeom=TOC.LPT.S_map.s_parmGeom;
Ambient.Fl_O.Aphy=TOC.Ambient.Fl_O.Aphy;
Inlet.Fl_I.Aphy=TOC.Inlet.Fl_I.Aphy;
Inlet.Fl_O.Aphy=TOC.Inlet.Fl_O.Aphy;
IGVDuct.Fl_O.Aphy=TOC.IGVDuct.Fl_O.Aphy;
Fan.Fl_O.Aphy=TOC.Fan.Fl_O.Aphy;
Splitter.Fl_01.Aphy=TOC.Splitter.Fl_01.Aphy;
Splitter.Fl_02.Aphy=TOC.Splitter.Fl_02.Aphy;
SwanNeckDuct.Fl_O.Aphy=TOC.SwanNeckDuct.Fl_O.Aphy;
HPC.Fl_O.Aphy=TOC.HPC.Fl_O.Aphy;
LPC.Fl_O.Aphy=TOC.LPC.Fl_O.Aphy;
CDPBld.Fl_O.Aphy=TOC.CDPBld.Fl_O.Aphy;
OGVduct.Fl_O.Aphy=TOC.OGVduct.Fl_O.Aphy;
Burner.Fl_O.Aphy=TOC.Burner.Fl_O.Aphy;
HPT.Fl_O.Aphy=TOC.HPT.Fl_O.Aphy;
ITTduct.Fl_O.Aphy=TOC.ITTduct.Fl_O.Aphy;
LPT.Fl_O.Aphy=TOC.LPT.Fl_O.Aphy;
TEGVduct.Fl_O.Aphy=TOC.TEGVduct.Fl_O.Aphy;
BPduct.Fl_O.Aphy=TOC.BPduct.Fl_O.Aphy;
Tailpipe.Fl_O.Aphy=TOC.Tailpipe.Fl_O.Aphy;
Core_Nozzle.AthCold=TOC.Core_Nozzle.AthCold;
Fan_Nozzle.AthCold=TOC.Fan_Nozzle.AthCold;
CDPBld.Cool1.fracW=TOC.CDPBld.Cool1.fracW;
CDPBld.Cool2.fracW=TOC.CDPBld.Cool2.fracW;
HPC.Cool3.fracBldW=TOC.HPC.Cool3.fracBldW;
HPC.Cool4.fracBldW=TOC.HPC.Cool4.fracBldW;
```

Fan.NcDes=TOC.Fan.NcDes;
LPC.NcDes=TOC.LPC.NcDes;
HPC.NcDes=TOC.HPC.NcDes;
Fan.Nc=TOC.Fan.Nc;
LPC.Nc=TOC.LPC.Nc;
HPC.Nc=TOC.HPC.Nc;
HPT.Nc=TOC.HPT.Nc;
LPT.Nc=TOC.LPT.Nc;
HPT.Np=TOC.HPT.Np;
LPT.Np=TOC.LPT.Np;
HPT.NpDes=TOC.HPT.NpDes;
LPT.NpDes=TOC.LPT.NpDes;
HPT.NpqNpDes=TOC.HPT.NpqNpDes;
LPT.NpqNpDes=TOC.LPT.NpqNpDes;
Fan.Ndes=TOC.Fan.Ndes;
LPC.Ndes=TOC.LPC.Ndes;
HPC.Ndes=TOC.HPC.Ndes;
HP_SHAFT.Ndes=TOC.HP_SHAFT.Ndes;
LP_SHAFT.Ndes=TOC.LP_SHAFT.Ndes;
Fan.S_map.NcDes=TOC.Fan.S_map.NcDes;
HPC.S_map.NcDes=TOC.HPC.S_map.NcDes;
LPC.S_map.NcDes=TOC.LPC.S_map.NcDes;
Fan.S_map.WcDes=TOC.Fan.S_map.WcDes;
Fan.S_map.ReDes=TOC.Fan.S_map.ReDes;
HPC.S_map.WcDes=TOC.HPC.S_map.WcDes;
HPC.S_map.ReDes=TOC.HPC.S_map.ReDes;
LPC.S_map.WcDes=TOC.LPC.S_map.WcDes;
LPC.S_map.ReDes=TOC.LPC.S_map.ReDes;
LPT.S_map.PtMap=TOC.LPT.S_map.PtMap;
LPT.S_map.effDes=TOC.LPT.S_map.effDes;
LPT.S_map.parmMapDes=TOC.LPT.S_map.parmMapDes;
LPT.S_map.parmNcDes=TOC.LPT.S_map.parmNcDes;
LPT.S_map.effMap=TOC.LPT.S_map.effMap;
LPT.S_map.parmGeomMap=TOC.LPT.S_map.parmGeomMap;
HP_SHAFT.HPX=TOC.HP_SHAFT.HPX;
HPT.S_map.PtMap=TOC.HPT.S_map.PtMap;
HPT.S_map.effDes=TOC.HPT.S_map.effDes;
HPT.S_map.parmMapDes=TOC.HPT.S_map.parmMapDes;
HPT.S_map.parmNcDes=TOC.HPT.S_map.parmNcDes;
HPT.S_map.effMap=TOC.HPT.S_map.effMap;
HPT.S_map.parmGeomMap=TOC.HPT.S_map.parmGeomMap;

B7: SFTF Function File

```
//*************************************************************************
//Title:       SFTF model function file
//Author:      STF
//Modified:    by STF for MDP
//
//Description: This file sets up the variables and functions
//             for a simple SFTF model.
//*************************************************************************

real zfnDesign {
  value       = 25000.0;
  units       = "lbf";
  description  = "SLS std day uninstalled design thrust";
  iDescription = " ";
}

//Maximum limits ------------------------------------------------------

real pcn2max {
  value       = 100.0;   // run to 100% corrected fan speed unless over-ridden
  units       = "none";
  description  = "Max percent corrected low spool RPM";
  iDescription = "";
}
real t4max {
  value       = 4500.;  // set out of the way
  units       = "R";
  description  = "Max allowable turbine inlet temperature";
  iDescription = "Max allowable average temperature at stn 4";
}
real t41max {
  value       = 3000.+459.67;
  units       = "R";
  description  = "Max allowable turbine inlet temperature";
  iDescription = "Max allowable average temperature at stn 41";
}

//Upper and lower bounds on independents ------------------------------------

real zwf36min {
  value       = 0.15;
  units       = "lbm/sec";
  description  = "Min allowable fuel flow rate";
  iDescription = "Usually set by fuel sys hrdwr & pilot stability";
}
real zwf36max {
  value       = 10.0;            // was 5.0 R. Denney 6-15-04
  units       = "lbm/sec";
  description  = "Max allowable fuel flow rate";
  iDescription = "Usually set by fuel system hardware";
}
real za8min {
  value       = 600.0;
```

```
  units       = "in2";
  description  = "Min allowable nozzle throat area";
  iDescription = "Usually set by nozzle mechanical limits";
}
real za8max {
  value       = 800.0;
  units       = "in2";
  description  = "Max allowable nozzle throat area";
  iDescription = "Usually set by nozzle mechanical limits";
}

#ifdef TOCdef

    Independent TOC_ind_W1 { varName = "TOC.Ambient.W";}
    Independent TOC_ind_BPR {varName = "TOC.Splitter.BPR";}
    Independent TOC_ind_FAR { varName = "TOC.Burner.FAR"; }

    Dependent TOC_dep_fntarget { eq_lhs="TOC.Eng.Fn"; eq_rhs="TOC.Eng.ThrustTarget";}
    Dependent TOC_MAX_T4 { eq_lhs = "TOC.HPT.Fl_I.Tt"; eq_rhs = "3250.0"; }

#endif

#ifdef TKOdef
    Independent TKO_ind_FAR { varName = "TKO.Burner.FAR"; dxLimitType="ABSOLUTE";}

    Dependent TKO_MAX_T4 { eq_lhs = "TKO.HPT.Fl_I.Tt"; eq_rhs = "3250.0"; }
    Dependent TKO_dep_fntarget { eq_lhs="TKO.Eng.Fn"; eq_rhs="TKO.Eng.ThrustTarget";}
    Dependent TKO_dep_pcn2max { eq_lhs="TKO.Fan.NcqNcDesPct"; eq_rhs="TOC.Fan.NcqNcDesPct";
                    constraintNameList = {"TKO_MAX_T4","TOC_MAX_T4"};
                    limitTypes={"MAX","MAX"};}
#endif

#ifdef Cruisedef
    Independent CR_ind_zwf36 { varName = "Cruise.Burner.Wfuel"; indepRef= "zwf36max-zwf36min";
            dxLimit=0.02*(zwf36max-zwf36min);
            dxLimitType="ABSOLUTE";}

    Independent CR_ind_FAR { varName = "Cruise.Burner.FAR"; dxLimitType="ABSOLUTE";}

    Dependent CR_dep_fntarget { eq_lhs="Cruise.Eng.Fn";       eq_rhs="Cruise.Eng.ThrustTarget";}
#endif

void write_indeps()
{

    OutFileStream current_guess;
    current_guess.open("current_guess.cs");

    current_guess << "          TOC.HPT.S_map.parmMap
    "          <<       " = " << toStr(TOC.HPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          TOC.Ambient.W
    "          <<       " = " << toStr(TOC.Ambient.W,10)          <<       ";\n" ;
    current_guess << "          TOC.LPT.S_map.parmMap
    "          <<       " = " << toStr(TOC.LPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          TOC.Burner.FAR
    "          <<       " = " << toStr(TOC.Burner.FAR,10)          <<       ";\n" ;
```

```
    current_guess << "          TKO.Ambient.W
    "          <<       " = " << toStr(TKO.Ambient.W,10)          <<       ";\n" ;
    current_guess << "          TKO.Fan.S_map.RlineMap
    "          <<       " = " << toStr(TKO.Fan.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          TKO.HPC.S_map.RlineMap
    "          <<       " = " << toStr(TKO.HPC.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          TKO.HPT.S_map.parmMap
    "          <<       " = " << toStr(TKO.HPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          TKO.HP_SHAFT.Nmech
    "          <<       " = " << toStr(TKO.HP_SHAFT.Nmech,10) <<       ";\n" ;
    current_guess << "          TKO.LPC.S_map.RlineMap
    "          <<       " = " << toStr(TKO.LPC.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          TKO.LPT.S_map.parmMap
    "          <<       " = " << toStr(TKO.LPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          TKO.LP_SHAFT.Nmech
    "          <<       " = " << toStr(TKO.LP_SHAFT.Nmech,10) <<       ";\n" ;
    current_guess << "          TKO.Splitter.BPR
    "          <<       " = " << toStr(TKO.Splitter.BPR,10)          <<       ";\n" ;
    current_guess << "          TKO.Burner.FAR
    "          <<       " = " << toStr(TKO.Burner.FAR,10)          <<       ";\n" ;

    current_guess << "          Cruise.Ambient.W
    "          <<       " = " << toStr(Cruise.Ambient.W,10)          <<       ";\n" ;
    current_guess << "          Cruise.Fan.S_map.RlineMap
    "          <<       " = " << toStr(Cruise.Fan.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          Cruise.HPC.S_map.RlineMap
    "          <<       " = " << toStr(Cruise.HPC.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          Cruise.HPT.S_map.parmMap
    "          <<       " = " << toStr(Cruise.HPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          Cruise.HP_SHAFT.Nmech
    "          <<       " = " << toStr(Cruise.HP_SHAFT.Nmech,10)          <<       ";\n" ;
    current_guess << "          Cruise.LPC.S_map.RlineMap
    "          <<       " = " << toStr(Cruise.LPC.S_map.RlineMap,10)       <<       ";\n" ;
    current_guess << "          Cruise.LPT.S_map.parmMap
    "          <<       " = " << toStr(Cruise.LPT.S_map.parmMap,10)       <<       ";\n" ;
    current_guess << "          Cruise.LP_SHAFT.Nmech
    "          <<       " = " << toStr(Cruise.LP_SHAFT.Nmech,10)          <<       ";\n" ;
    current_guess << "          Cruise.Splitter.BPR
    "          <<       " = " << toStr(Cruise.Splitter.BPR,10)          <<       ";\n" ;
    current_guess << "          Cruise.Burner.Wfuel
    "          <<       " = " << toStr(Cruise.Burner.Wfuel,10)          <<       ";\n" ;
    current_guess << "          Cruise.Fan_Nozzle.AthCold
    "          <<       " = " << toStr(Cruise.Fan_Nozzle.AthCold   ,10)       <<       ";\n" ;
    //current_guess << "          Cruise.Core_Nozzle.AthCold                                                  "
    <<       " = " << toStr(Cruise.Core_Nozzle.AthCold ,10)       <<       ";\n" ;
    current_guess.close();
}
```

REFERENCES

1 Simmons, Ronald J. "Design and Control of a Variable Geometry Turbofan with an Independently Modulated Third Stream." Thesis. Ohio State University, 2009. Web. 4 Jan. 2013.

2 Schutte, Jeff, Jimmy C. Tai, and Dimitri N. Mavris. "Multi-design Point Cycle Design Incorporation into the Environmental Design Space." *AIAA Joint Propulsion Conference* (2012)

3 Denney, Russel, Jimmy Tai, and Brian Kestner. "Variable Cycle Optimization for Supersonic Commercial Applications." (2005): n. pag. Print.

4 Kurzke, Joachim. "The Mission Defines the Cycle: Turbojet, Turbofan and Variable Cycle Engines for High Speed Propulsion." N.p., n.d. Web. 20` Dec. 2012.

5 Warwick, Graham. "AFRL Backs New Type of Combat-Aircraft Engine." *Aviation Week and Space Technology* 2012: n. pag. Web. <http://www.aviationweek.com/Article.aspx?id=/article-xml/AW_09_24_2012_p31-497914.xml>.

6 Denney, Russel K., Jimmy C. Tai, Brian K. Kestner, and Dimitri N. Mavris. "Variable Cycle Optimization for Supersonic Commercial Applications." *SAE Technical Papers* (2005): n. pag. Print.

7 Schutte, Jeffrey, Jimmy Tai, Jonathan Sands, and Dimitri Mavris. "Cycle Design Exploration Using Multi-design Point Approach." *ASME Turbo Expo* (2012): n. pag. Web.

8 Giffin, Rollin G., III, E. Woltmann, and P. McHugh. Gas Turbine Engine with Augmentor and Variable Area Bypass Injector. General Electric Company, assignee. Patent 4187378. 4 Apr. 1989. Print.

9 Giffin, Rollin G., III. Variable Specific Thrust Turbofan Engine. General Electric Company, assignee. Patent 5261227. 16 Nov. 1993. Print.

10 Giffin, Rollin G., John J. Ciokajlo, and Wayne Dunbar. Compressor Splitter for Use with a Forward Variable Area Bypass Injector. General Electric Company, assignee. Patent 5680754. 28 Oct. 1997. Print.

11 Johnson, James E., Tom Foster, and Roy D. Allan. Variable Cycle Gas Turbine Engines. The USA as Represented by the Administrator of NASA, assignee. Patent 4064692. 27 Dec. 1977. Print.

12 Johnson, James E., Elmore V. Sprunger, and John R. Simmons. Variable Cycle Turbofan-Ramjet Engine. General Electric Company, assignee. Patent 5694768. 9 Dec. 1997. Print.

13 Johnson, James E. Turbofan Engine with a Core Driven Supercharged Bypass Duct and Fixed Geometry Nozzle. General Electric Company, assignee. Patent 5806303. 15 Sept. 1998. Print.

14 Willis, Edward. "Variable-Cycle Engines for Supersonic Cruise Aircraft." *Nasa Technical Reports Server*. N.p., 1976. Web.

15 Hoffman, Sherwood, and Mary C. Varholic. "Contracts, Grants, and Funding Summary of Supersonic Cruise Research and Variable-Cycle Engine Technology Programs 1972-1982." *Nasa Technical Reports Server*. N.p., Sept. 1983. Web.

16 Johnson, J. E. "Variable Cycle Engine Concepts." (1988): n. pag. Web.

17 Corbett, Michael W. "Large-Scale Transient Loading of a Three Stream Variable Cycle Engine." Proc. of AIAA Joint Propulsion Conference, Atlanta. N.p.: n.p., 2012. Print.

18 Jones, Scott M. "Steady-State Modeling of Gas Turbine Engines Using the Numerical Propulsion System Simulation Code." *Power for Land, Sea, and Air*. Proc. of ASME Turbo Expo, UK, Glasgow. N.p.: n.p., n.d. N. pag. Print.

19 Roth, Bryce, Jimmy Tai, Patrick Biltgen, and Russell Denney. "Creation of a Parametric Variable Cycle Engine Model for Supersonic Business Jet Applications." NASA Glenn Research Center, 11 June 2004

20 Grieb, N., W. Weiler, and G. Weist. *Variable-Cycle Engines for Fighter Aircraft - Advance in Performance and Development Programs*. Rep. no. AGARD CP241. N.p.: n.p., 1978. Print.

21 *A Review of United States Air Force and Department of Defense Aerospace Propulsion Needs*. Washington, D.C.: National Academies., 2006. N. pag. 86. Web. http://www.nap.edu/openbook.php?record_id=11780&page=86

22 AIAA. "The Versatile Affordable Advanced Turbine Engines (VAATE) Initiative." AIAA, Jan. 2006. Web.

23 Tai, Jimmy, Russell Denney, and Brian Kestner. "Creation of a Parametric Variable Cycle Engine Model for Supersonic Business Jet Applications." 26 May 2005. Lecture.

24 Johnson, J. E. "Variable Cycle Engine Concepts." N.p., n.d. Web.

25 Mavris, Dimitri. "Engine Design Space Exploration for a Multi-Mission Fighter Under Evolving Requirements." 12 May 2011. Lecture.

26 Mattingly, Jack D. *Elements of Propulsion*. [Colorado Springs, Colo.?]: Dept. of Aeronautics, U.S. Air Force Academy, 1985. Print.

27 Schutte, Jeffrey S. *Simultaneous Multi-Design Point Approach to Gas Turbine On-Design Cycle Analysis for Aircraft Engines*. Thesis. Georgia Institute of Technology, 2009. N.p.: n.p., n.d. Print.

28 Johnson, J. E. "Propulsion." *Variable Cycle Engines - The Next Step in Propulsion Evolution?* Proc. of AIAA/SAE Propulsion Conference, Palio Alto, California. N.p.: n.p., n.d. N. pag. Print.

29 Hill, Philip G., and Carl R. Peterson. *Mechanics and Thermodynamics of Propulsion*. Reading, MA: Addison-Wesley, 1992. Print.

30 Krebs, J. N., and R. D. Allan. "Supersonic Propulsion - 1970 to 1977." *AIAA/SAE Joint Propulsion Conference* (1977): n. pag. Web.

31 Curnock, B. "Gas Turbine Engine Transient Behavior." Lecture. Von-Karman Institute for Fluid Dynamics. Web. May 1993.

32 "IHPTET Advanced Concept Common Core Engine." *IHPTET Advanced Concept Common Core Engine*. N.p., n.d. Web. 02 Apr. 2013. http://csat.au.af.mil/2025/concepts/810021.HTM

33 *NPSS Reference Sheets*. 12 Mar. 2008. Software Release: NPSS_1.6.5. John H. Glenn Research Center, Cleveland.

34 *NPSS User Guide*. 12 Mar. 2008. Software Release: NPSS_1.6.5. John H. Glenn Research Center, Cleveland.

35 Aleid, Louay. *Variable Cycle Propulsion Systems for a Supersonic Civil Transport*. Thesis. Cranfield University, 1997. N.p.: Cranfield University

36 Mavris, Dimitri N., and Mark Lewis. *Engine Design Space Exploration for a Multi-Mission Fighter Under Evolving Requirements*. 16 May 2011.

37 "Aciturri | Aerostructures TIER 1 | Aciturri." *Aciturri | Aerostructures TIER 1 | Aciturri*. N.p., n.d. Web. 21 Jan. 2014. <http://www.aciturri.com>

38 "GasTurb." *GasTurb*. N.p., n.d. Web. 21 Jan. 2014. < http://www.gasturb.de/>

39 Vanderplaats, Garret N. *Multidiscipline Design Optimization: Textbook*. Colorado Springs: Vanderplaats, 2009. Print.

40 Raymer, Daniel P. *Aircraft Design: A Conceptual Approach*. Washington, D.C.: American Institute of Aeronautics and Astronautics, 2006. Print.

41 Myers, Raymond H., and Douglas C. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. New York: J. Wiley, 2002. Print.

42 Sivanandam, S. N., and S. N. Deepa. *Introduction to Genetic Algorithms*. Berlin: Springer, 2007. Print.

43 Brown, Harold. *Multi-Variable Cycle Optimization by Gradient Methods*. Proc. of AIAA 18th Aerospace Sciences Meeting, Pasadena, California. Pasadena: AIAA, 1980.

44 Denner, Brett W., Brent N. McCallum, and Phil P. Truax. *CFD Prediction of Inlet Spill Drag Increments*. Tech. no. AIAA 98-3566. Fort Worth: Lockheed Martin

45 Beckey, Carl, Capt, Roy Hartfield, Dr., and Mark Carpenter, Dr. *Compressor Modeling for Engine Control and Maintenance*. July 2011. Air Force Flight Test Center, Edwards AFB, California.

46 "Minimizing the Rosenbrock Function" from the Wolfram Demonstrations Project? http://demonstrations.wolfram.com/MinimizingTheRosenbrockFunction/

47 Bryson, AA.E., Ho, Y.C., *Applied Optimal Control*, Taylor & Francis, 1975.

48 *Aircraft propulsion system performance station designation and nomenclature*, in *Aerospace Recommended Practice (ARP) 755B*. 1994, Society of Automotive Engineers Warrendale, PA.

49 "Experimental Feature." *Wolfram|Alpha: Computational Knowledge Engine*. N.p., n.d. Web. 05 Aug. 2014. <http://www.wolframalpha.com/>

50 Anton, Howard, Irl Bivens, and Stephen Davis. *Calculus: Early Transcendentals*. 8th ed. Hoboken, NJ: John Wiley & Sons, 2005. Print.