Optimization of Turbine Engine Cycle Analysis with Analytic Derivatives

Dr. Tristan Hearn,* Eric Hendricks,* Jeffrey Chin,* Justin Gray,†* Dr. Kenneth T. Moore,‡
NASA Glenn Research Center, Cleveland, OH

For many aircraft design tools the propulsion model is simply tabular engine data, interpolated to get data at specific altitudes, speeds, and throttle settings. The raw data is usually generated from a detailed propulsion model that is run over the full flight envelope in a pre-processing step. This method works when the design of the propulsion system is not coupled to the airframe, but such coupling is increasingly difficult to avoid as propulsion-airframe integration becomes more important. This has led to the need to integrate the full propulsion model directly into aircraft design tools. Existing tools, such as the NPSS propulsion modeling environment, have been successfully integrated into aircraft design models with some success. However, challenges encountered in these prior efforts have motivated the creation of new propulsion tool, Pycycle, designed specifically for integration into a multidisciplinary aircraft models. Pycycle is built with the OpenMDAO framework and provides analytic derivatives allowing for an efficient use of gradient-based optimization methods. This work presents a design optimization for a separate flow turbofan propulsion system with comparative results between analytic derivatives from PyCycle and finite-difference derivatives from NPSS. The results demonstrate a large reduction in computational cost when using Pycycle's analytic derivatives. The reduced cost of Pycycle will enable more detailed aircraft models, accounting for tight propulsion-airframe integration, that can be optimized efficiently with gradient based methods.

I. Introduction

Design of an aircraft is a complex, coupled, and increasingly multi-disciplinary problem. There are a large number of physical design variables to vary and many operating conditions to consider. An optimal configuration, particularly in the case of tight inter-disciplinary coupling, is often not intuitive. Given this, it is no surprise that numerical optimization has become a mainstream tool in conceptual design. Numerical optimization algorithms can generally be classified into two categories, gradient-based and gradient-free. Gradient-based optimization methods use derivatives of a model to iteratively perturb parameter values in a direction which improves the value of an objective function until a terminating criteria is reached (typically based on the KKT optimality conditions). This contrasts with gradient-free methods, which may use a variety of sampling and local search strategies to determine optimal parameter values for a model.[1] There are practical advantages and disadvantages to both types of optimization algorithms. Broadly speaking, gradient-based optimization, though it is susceptible to convergence onto local optima, uses far less function evaluations compared to gradient-free methods.[2] For large-scale computationally-expensive models, this is a significant advantage, particularly when combined with adjoint methods. As the name implies, gradient-based optimization algorithms rely on gradient information to achieve their reduced computational cost.

Finite-difference approximation is often used to compute derivatives, but this method scales poorly as the number of design variables increases. Finite-difference approximations are also sensitive to the step size used to estimate them, especially when a model has one or more internal solvers converging a nonlinear system of equations. For these reasons, tools that provide analytic derivatives in addition to their analysis outputs are particularly suitable for efficient optimization with gradient-based methods.

*Aerospace Engineer, Propulsion Systems Analysis Branch
†Doctoral Pre-Candidate, Department of Aerospace Engineering, University of Michigan
‡Senior Systems Engineer, Propulsion Systems Analysis Branch

There are an increasing number of aircraft design tools that do provide one or more forms of analytic derivatives for the purpose of gradient-based optimization. Many CFD and FEA codes now come with adjoint analytic derivative capabilities.[3–6] Recent work has begun on an adjoint-based aircraft optimization to perform aircraft mission analysis as well including a a new mission analysis tool, named PyMission, that includes adjoint derivatives for trajectory optimization.[7–9] However, to date, there exists no propulsion analysis tool that provides analytic derivatives. The current state of the art for propulsion analysis is the Numerical Propulsion System Simulation (NPSS) tool.[10] This tool provides a general framework to model a very wide variety of thermodynamics cycles, from simple turbojets to complex co-generation cycles.[11,12] Despite the effectiveness of NPSS as an analysis tool, it does present some challenges when being used in an optimization context. Geiselhart et. al. used NPSS as to model the propulsion system of a low-boom supersonic transport, but noted numerical stability as a primary motivation for using gradient free optimization methods.[13] Hendricks et. al. highlighted acute inaccuracy of finite difference gradients when optimizing a turbine design, using a mean-line turbomachinery model built in NPSS.[14] To address these challenges, a new cycle analysis tool, Pycycle, was written using OpenMDAO. Previous work presented a validation of the underlying thermodynamics modules in PyCycle.[15] In this paper we present the application of PyCycle to the analysis and optimization of a complete on-design turbofan engine cycle.

This paper is organized as follows. Section A provides brief summary of how analytical partial derivatives, provided by disciplinary components, can be automatically combined by the OpenMDAO into total derivatives across the model useful for optimization. Section B describes the structure of the Pycycle and how it differs conceptually from existing engine cycle tools (with supplementary graphics provided in Section V). Section II formulates the example turbofan engine optimization problem. Section III reports the results of applying gradient-based optimization to an two implementations of this model; one in Pycycle and one in NPSS. The performance of the optimization on each implementation is compared. Finally, comments and future applications are discussed in Section IV.

A. Model Optimization With Analytic Derivatives

Following the methodology of Martins et. al.,[16] it is informative to first illustrate how discipline-level derivative information (i.e. partial derivatives) can be used to compute total derivative quantities across a model (even in the presence of tight inter-disciplinary coupling). Let

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) : \mathbb{R}^{k+n} \rightarrow \mathbb{R}^m \tag{1}$$

represent the objective function and all constraints of an MDAO problem, with $\mathbf{x} \in \mathbb{R}^k$ being a vector of k design variables, and $\mathbf{y} \in \mathbb{R}^n$ being a vector of n state variables, returning $\mathbb{R}^m$ where m is the total number of constraints plus the objective. Figure 1 illustrates an example model possessing this structure, with data passed between three discipline-level modeling components, C_1, C_2, and C_3. Here, the components C_1 and C_2 are coupled together in a manner that must be solved numerically.

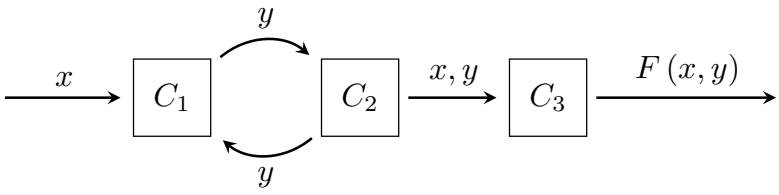

Figure 1. Illustration of a coupled model, with input variables x, coupling variables y, and output $F(x, y)$.

Next, let

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) : \mathbb{R}^{k+n} \rightarrow \mathbb{R}^n \tag{2}$$

represent a series of n residual equations that describe the multidisciplinary coupling between C_1 and C_2, such that for a feasible set of values for $\mathbf{x}, \mathbf{y}$, (i.e., when data passes from discipline components C_2 to C_3) we have

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = 0. \tag{3}$$

To compute a gradient for an optimization iteration we will need to compute the derivative of the quantities of interest $\mathbf{F}$ with respect to the design variables $\mathbf{x}$,

$$\underbrace{\frac{d\mathbf{F}}{d\mathbf{x}}}_{m\times k} = \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{x}}}_{m\times k} + \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{y}}}_{m\times n}\underbrace{\frac{d\mathbf{y}}{d\mathbf{x}}}_{n\times k}. \tag{4}$$

The residual and state equations may be combined to give an expression for the elements of the total derivative matrix in terms of partial derivatives of the objective functions, constraints, and residual equations.[16] In plain terms, the total derivative matrix is constructed one column at a time (known as the forward form) as

$$\underbrace{\frac{d\mathbf{F}}{d\mathbf{x}_i}}_{m\times 1} = \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{x}_i}}_{m\times 1} - \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{y}}}_{m\times n}\underbrace{\left(\frac{\partial \mathbf{R}}{\partial \mathbf{y}}\right)^{-1}\frac{\partial \mathbf{R}}{\partial \mathbf{x}_i}}_{n\times 1} \tag{5}$$

for each $\mathbf{x}_i$. Note that $(\cdot)^{-1}(\cdot)$ does not necessarily denotes a actual matrix inversion, and could instead be the numerical solution of a linear system by a suitable numerical method (either direct or iterative). Indeed, in a large distributed model the matrix $\frac{\partial \mathbf{R}}{\partial \mathbf{y}}$ may not be explicitly assembled at all, making iterative methods such as GMRES[17] a good choice for solving the linear system in Equation 5.

Similarly, the adjoint form of the derivatives may be computed as

$$\underbrace{\frac{d\mathbf{F}_i}{d\mathbf{x}}}_{1\times k} = \underbrace{\frac{\partial \mathbf{F}_i}{\partial \mathbf{x}}}_{1\times k} - \underbrace{\left(\left(\frac{\partial \mathbf{R}}{\partial \mathbf{y}}^T\right)^{-1}\frac{\partial \mathbf{F}_i}{\partial \mathbf{y}}^T\right)^T}_{1\times n}\underbrace{\frac{\partial \mathbf{R}}{\partial \mathbf{x}}}_{n\times k}, \tag{6}$$

for each $\mathbf{F}_i$. In the adjoint form, the total derivative Jacobian is computed 1 row at a time. Depending on which is larger, the number of design variables or the number of constraints plus the objective, you can use either the forward or adjoint form. In other words, if the Jacobian matrices $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$, $\frac{\partial \mathbf{F}}{\partial \mathbf{y}}$, $\frac{\partial \mathbf{R}}{\partial \mathbf{y}}$, and $\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$ can be provided within the model, then the total derivatives can be computed efficiently without re-evaluation of the non-linear model.

OpenMDAO implements this total derivatives calculation automatically by collecting partial derivatives provided by individual model components and solving for the total derivative across a multidisciplinary model in either forward or adjoint mode. Model components which do not provide any derivative information can individually have their derivatives approximated using finite-difference, allowing for an OpenMDAO model to automatically make use of a mix of analytic and finite-difference derivatives when total derivative quantities are needed.

B. Structure Of Pycycle

Pycycle is a new thermodynamic cycle analysis tool developed using the OpenMDAO framework. Like NPSS, Pycycle provides a means to perform single dimensional (1D) steady-state engine cycle analysis. For a cycle analysis performed in "on-design" mode, the initial engine cycle reference (or "design" point calculation) used to size engine geometry can be viewed as a four-step process. To begin, the equilibrium composition of the working fluid is computed by solving a system of nonlinear equations to minimize a Gibbs free energy function. Next, the thermodynamic state variables (temperature T, pressure P, density ρ, entropy S, and enthalpy h) of the fluid mixture is computed. The implementation of these first two steps in Pycycle are outlined in detail in previous works by Gray et. al.[15] Third, the analysis of each thermodynamic cycle stage (the corresponding to physical engine elements) is executed, and the corresponding velocities and areas are calculated. Finally, the system is brought to convergence by varying engine geometry and operating parameters until all boundary conditions reach agreement, and power imbalances are driven to zero (in the case of steady-state) or to a time-dependent integrated state (in the case of transient). Following the successful convergence of an on-design reference case, a single or series of "off-design" cases may then be computed. In this mode, elements sized using the reference on-design case compute their operating characteristics at a specified flight Mach number, throttle setting, and altitude. Aggregating the results

from a sufficiently large number of off-design cases allows an analyst to identify design choices which are sufficiently robust over a large range of operational flight conditions.[10]

From a high level, the design of Pycycle was heavily influenced by NPSS. Like NPSS, Pycycle uses a heavily object-oriented structure to represent the physical components (typically referred to as elements) of an engine cycle. Object orientation is an inherent aspect of models built within OpenMDAO, as it provides a reasonable encapsulation of separable calculations, while enabling a modular data flow representation of an overall model. As illustrated in Section A, it is this model component compartmentalization that allows for analytical total derivatives (in either the forward or adjoint formulation) to be implemented and tested in a user-friendly and highly maintainable fashion. Aside from the addition of analytical derivatives to the chemical equilibrium and engineering calculations, the main difference between Pycycle and NPSS is in how the required thermodynamics calculations are organized within their respective elements. In Pycycle, thermodynamic chemical equilibrium calculations (and the resulting flow properties) are handled in their own components that are exposed directly to the framework. This contrasts with NPSS, where thermodynamic calculations are done with function calls inside the the individual engine elements and are thus hidden from the rest of framework.

There are practical advantages to having the flexibility to treat the thermodynamic calculations on the same level as the engineering calculations. For instance, the derivative calculations can be compartmentalized, and therefore implemented and tested more easily. This also allows for some of these variables to be converged at a higher level, or even handed to an optimization algorithm to perform simultaneous analysis and design.[18]

For reference, Section V provides an illustration of each of the Pycycle engine elements which were used to construct the turbofan engine model for the optimization described in the following sections.

II. Application to a gradient-based optimization

To demonstrate the advantage of the analytic derivatives provided by Pycycle, an optimization problem was set up in such a way that a direct comparison to an NPSS-based model was possible.

For this, a single point on-design turbofan engine model was constructed within both Pycycle and NPSS. Figure 2 illustrates the structure of this model. Generally speaking, this is a type of open Brayton engine cycle with a core stream passing through a fan, compressor, burner, and turbines, with a second stream of air compressed by the fan but bypassing the core. There are a number common design parameters to most turbofan engines which may be selected by an engine cycle analyst. These are the overall pressure ratio (OPR), combustor temperature (T_4), ratio of mass flow between the bypass and core streams (BPR), and pressure ratios of the fan and engine compressor (FPR and CPR). These variables have the greatest impact on the typical quantities of interest, such as thrust-specific fuel consumption (TSFC).[19]

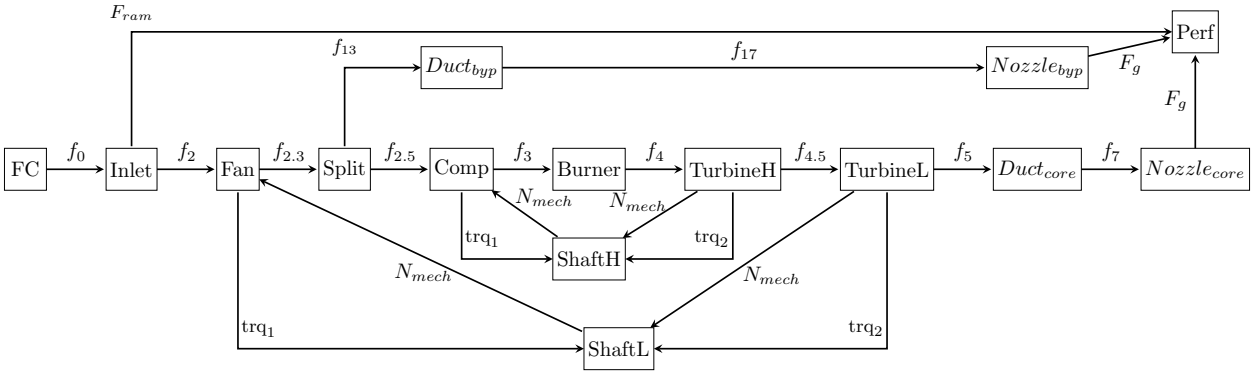

Figure 2. Directed-graph structure of a turbofan engine model. Each rectangle represents an analysis component, and the arrows represent data relationships and required executing order. Each label represents a variable or series of variables being passed between the elements (e.g. f_i represents a flowstation, carrying a series of variables representing computed flow properties.)

As mentioned in Section B, each element (such as a compressor or turbine) needs to compute the total and static flow properties at its exit boundaries (and sometimes at interior points) by satisfying the steady-state thermodynamic equilibrium equations. For most elements, this involves an iterative solution of the thermo

equations coupled with some element-specific performance and sizing calculations. At a higher level in the model, the power on the two shaft spools must be properly balanced by varying the pressure ratio of the low-pressure turbine (LPT) and high-pressure turbine (HPT) so that the net power is zero (no acceleration). This introduces a cyclic data connection containing the compressor, HPT, LPT, and shaft elements. When this coupling is in a properly converged state, the quantities $\text{ShaftL}_{\text{net pwr.}}$ and $\text{ShaftH}_{\text{net pwr.}}$ are each zero. This is an implicit condition that is satisfied using a numerical solver. In Pycycle an analytic derivative based Newton-Krylov solver from the OpenMDAO standard library is used. NPSS uses a Newton-Raphson solver based on finite-difference Jacobian approximations with Broyden updates.[20] For both the NPSS and Pycycle models, the non-linear solvers were configured to have a tolerance of 10^{-6}.

An optimization problem was then formulated to minimize thrust-specific fuel consumption (TSFC), with respect to fan pressure ratio (FPR), compressor pressure ratio (CPR), bypass ratio (BPR), and mass-flow rate (W). Engine burner temperature (T_4) was constrained to be less than 3000 degrees Rankin, and a net thrust (F_n) target of 25,000 lbf was set. Overall pressure ratio (OPR) was also constrained to be equal to 30. The optimization problem therefore has the formulation shown in Table 1.

Minimize:			
		TSFC	
With respect to:			
	$1 \leq$	FPR	≤ 2
	$1 \leq$	CPR	≤ 30
	$1 \leq$	BPR	≤ 12
	$1 \leq$	W	$\leq 2000 \ \frac{\text{lbm}}{\text{s}}$
Such That:			
		OPR	$= 30$
		F_n	$= 25,000$ lbf
		T_4	$\leq 3000^\circ$ R

Table 1. The formulation of the turbofan optimization problem.

This problem formulation has 4 design variables, 1 objective and 3 constraints. Since there are four each of design variables and quantities of interest, there is no direct preference for forward or adjoint mode when computing analytic total derivatives.

III. Results

The optimization problem was solved using the SNOPT[21] optimizer from within OpenMDAO. The identically defined NPSS turbofan model was also optimized using SNOPT via an OpenMDAO wrapper, using derivatives approximated via forward-form finite-difference. Several separate wrapped optimizations were performed on this NPSS implementation with multiple finite-difference step sizes. An optimality tolerance of 10^{-3} on the objective function and feasibility tolerance of 10^{-5} on the constraints was specified to the SNOPT optimizer for both optimizations. Table 2 compares the parameter and objective values between the baseline and optimized configurations, for both turbofan model implementations. The baseline and optimized objective and constraint values computed by the Pycycle model were also verified using the NPSS implementation, as a check on the accuracy of the Pycycle thermodynamic calculations. Broadly speaking, the results in Table 2 show agreement between the Pycycle and NPSS implementations of the on-design turbofan model, as they each achieve nearly the same result when optimized using SNOPT.

Table 2 also shows that the Pycycle convergence achieved the tightest absolute tolerance on both the internal shaft power balances as well as the three optimization constraints. The NPSS model did consistently achieve a configuration with 1.7% greater TSFC reduction over baseline, however when the optimization solution achieved by Pycycle is verified afterwards using NPSS, it does agree with the TSFC value of 0.320, indicating a small potential mismatch in the final performance calculations between NPSS and Pycycle, but consistency is the establishment of the parameterization as a feasible minimizing point. The mismatch in

	Baseline	Optimized (Pycycle)	Optimized (NPSS)		
FD step size	-	-	10^{-5}	10^{-4}	10^{-3}
FPR	1.5	2.0	2.0	2.0	2.0
CPR	10.3	15.0	15.0	15.0	15.0
BPR	5.0	12.0	12.0	12.0	12.0
W	500.0	1069.2	1032.41	1032.39	1032.40
TurbL_{PR}	1.611	9.096	9.090	9.091	9.090
TurbH_{PR}	1.935	2.356	2.356	2.356	2.356
$\text{ShaftL}_{\text{net pwr.}}$	$1.64 \cdot 10^{-6}$	$4.530 \cdot 10^{-9}$	-0.023	-0.022	-0.022
$\text{ShaftH}_{\text{net pwr.}}$	$6.11 \cdot 10^{-8}$	$9.555 \cdot 10^{-9}$	$2.823 \cdot 10^{-6}$	$2.825 \cdot 10^{-6}$	$2.826 \cdot 10^{-6}$
TSFC	0.612	0.331	0.320	0.320	0.320
F_n	13723.128	25000.000	25000.003	24999.649	25000.001
OPR	15.450	30.000	30.000	30.000	30.000
T_4	2762.254	2913.355	2913.690	2913.690	2913.690

Table 2. Comparison of design variables, objectives, and constraint values between the baseline and optimized configuration for the two turbofan models.

mass flow rates, W, may be related to this, or due to tighter tolerances achieved on the constraints and internal balances in Pycycle. In any case, the difference in solutions returned by the four optimizations is interpreted to be small enough to consider the results as being in mutual agreement. The three optimizer-selected pressure ratios and the internally solved turbine pressure ratios were all very consistent across the four optimizations.

Next, Figure 3 illustrates the logarithm of both the overall SNOPT model feasibility and optimality measures for the Pycycle and NPSS optimizations, as a function of iteration number. These show that both optimizations experiences a slow evolution from the baseline configuration and then a rapid improvement at a further point in the design space, though the Pycycle optimization reached this point more quickly (at the 44^{th} iteration, noted above). This would indicate that the analytic derivatives provided by Pycycle are precise enough for SNOPT to converge more efficiently in terms of required iterations than for any of the finite-difference driven optimizations. Note also that the evolution of the feasibility and optimality metrics are far smoother for Pycycle than for any of the finite-difference driven optimizations, whose progressions are seen to exhibit far more noise as they progress through the design space. This is interpreted to be due to the precision inherent to the analytic derivatives.

Several summary metrics were also recorded for each optimization, including total number of SNOPT optimizer iterations, percent improvement in objective function, maximum constraint violation at the optimized configuration, and execution times. These metrics are tabulated in Table 3, and compared with the optimization performed using Pycycle with analytic derivatives. As an example from this tabulated data, it is seen that the NPSS-based optimization with a step size of 10^{-5} required 120 major iterations of the SNOPT optimizer, while the Pycycle optimization required only 44. Given that the default step size for finite-difference calculation within OpenMDAO is 10^{-6} (and even smaller within optimizers such as SNOPT and SLSQP used alone outside of OpenMDAO), this highlights how computationally difficult even this simple optimization problem would have been without at least some a priori experimentation with step size selection. Note that the term here "major iteration" refers to the number of feasible designs considered by the optimizer where derivatives are needed. The number of major iterations is distinct from the total number of function evaluations used to solve the optimization problem. In addition to function evaluations at each major iteration, there are additional evaluations needed by the optimizer during line-searches (where no derivatives are computed) and when finite-differences are used there are also additional function evaluations used to compute the derivatives at each major iteration.

It is also clearly seen that the performance of the finite-difference based optimizations here is seen to be very dependent on step size, with the total number of SNOPT iterations required ranging from 11 for a

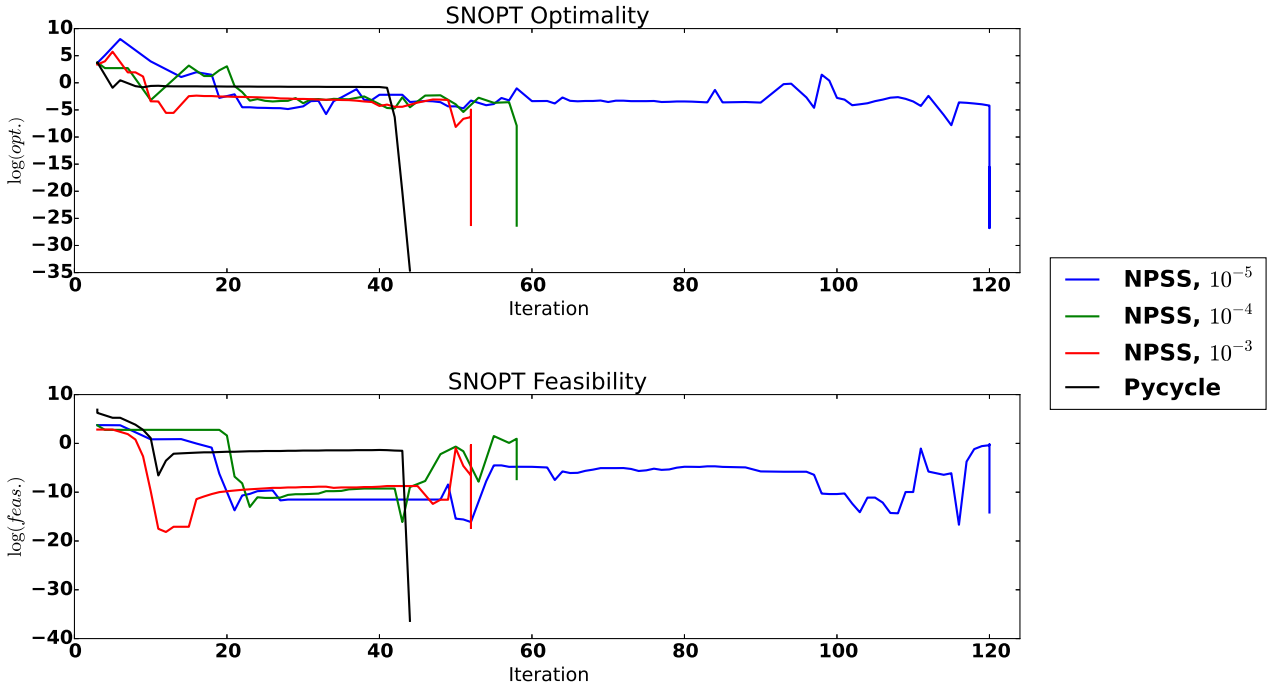

Figure 3. Convergence history of the Pycycle and NPSS turbofan optimization. Here the logarithm of both the feasibility and optimality metrics are plotted with respect to iteration number.

finite-difference step size of 10^{-4}, to 721 iterations for a very similar finite-difference step size of $0.99 \cdot 10^{-3}$. This represents a notably broad range of both computational effort and overall stability of performance in optimizations performed on the turbofan engine model when finite-difference approximations are the only option available. As highlighted before, there is no such careful selection of step size required when optimizing the engine cycle using Pycycle's analytic derivatives.

	Pycycle	NPSS				
FD step size	-	10^{-5}	10^{-4}	$0.99 \cdot 10^{-3}$	10^{-3}	$1.01 \cdot 10^{-3}$
TSFC reduction	45.9 %	47.6%	47.6%	47.6%	47.6%	47.6%
Number of SNOPT iterations	44	120	58	721	11	98
Maximum constraint violation	$3.5 \cdot 10^{-15}$	0.003	0.351	0.102	$1.2 \cdot 10^{-3}$	0.102
Run time (seconds)	3753	30912	12796	131581	1071	18788

Table 3. Comparison of benchmarks between the Pycycle optimization with analytic derivatives and the NPSS optimization with finite-difference approximated derivatives.

The last row of Table 3 shows total execution time (or "wall time") required for each of the four performed optimizations. For consistency, all optimizations of record (including the Pycycle optimization) were performed within the same computing environment, a Linux computing cluster. The Pycycle based optimization using analytic derivatives took 3,753 seconds, while the NPSS based optimization took anywhere from 1,071 seconds to 131,581 seconds, depending on the finite-difference step sizes.

Inferences between the relative execution wall times of the Pycycle and NPSS-based optimizations may be more difficult to make than the more direct comparisons of SNOPT iteration requirements and total number of model executions. The Pycycle optimization clearly performed faster than all but one of the NPSS runs in terms of total execution time, number of iterations required, and fidelity to the model constraints. However, it is worth noting that there is no canonically accepted manner of performing an optimization around an NPSS model. The simple file-wrapped approach used for this work was very straightforward and generalizable,

but not carefully designed to improve run-time performance specifically for this model. There are a number of possible wrapper implementations that could have favorable run-time behavior relative to the Pycycle optimization, such as a direct in-memory interface, or a file based wrapper that parallelized the executions required for finite-difference gradient estimations. However, improvement of the run-time performance of the NPSS wrapper alone would not change the other results (number of iterations, solution precision, etc.) or conclusions drawn from this work for the comparison of finite-difference to analytic derivatives for engine cycle optimization from a stability or precision standpoint.

Likewise, it can be imagined that in a model sufficiently complex to exhibit variability in finite-difference sensitivity in different areas of the design space, analytic derivatives would be a practical necessity for the increase in stability that they provide.

IV. Conclusion

This paper has presented the design optimization, using a gradient based SQP optimization algorithm, of a separate flow turbofan engine with the newly developed Pycycle propulsion analysis tool. Pycycle was developed specifically to support gradient based optimization with analytic derivatives via the OpenMDAO framework. The addition of analytic derivatives was the main main motivation for its development, and represents its major technical contribution. The modular breakdown of propulsion system parts into elements and the underlying engineering equations in those elements were borrowed from the NPSS propulsion modeling tool. However, unlike in NPSS the calculations for each propulsion element were further subdivided into smaller components to facilitate the calculation of the partial derivatives across each part of the analysis.

For comparison and validation purposes the same turbofan model was built in both Pycycle and NPSS and the optimization was run on each model. The results in Section III show that the the analytic derivatives from Pycycle give a significant computational cost savings over the finite-differenced derivatives of NPSS. While it would be possible to reduce the wall time of the NPSS optimization by using a parallel finite-differencing scheme, there would not be any decrease in computational cost since the wall time would drop by the same factor as the number of parallel processes running.

The results presented in this work are for an optimization of just 4 design variables for a single-point, on-design, separate flow turbofan. Even for this simple problem, the computational savings are significant. Multi point engine design methods and advanced propulsion technologies including third streams, variable cycle technology, and hybrid electric systems will introduce even more design variables into the problem and make finite-difference even less feasible. If we consider integrated aircraft design problems involving the coupling of engine performance cycle analysis to a more computationally expensive discipline (such as CFD, or trajectory analysis involving a large series of off-design points), we can infer from these results how the availability of analytical adjoint derivatives can enable far more ambitious types of optimization problem formulations.

Future work will extend the capabilities of Pycycle first to off-design propulsion simulation for traditional propulsion models and second to hybrid electric propulsion systems. Follow on work from there will involve integration of Pycycle with an aircraft mission analysis tool to build a fully integrated aircraft design capability with analytic derivatives to support efficient gradient based optimization.

V. Appendix

Figures 4 through 8 illustrate the structure of the Pycycle engine modeling elements used in the turbofan MDP optimization problem. As per the OpenMDAO 1.0 API, each of the figures collectively represents a Group object, while the rectangular graphics represent Component objects.

More specifically, the rectangular components indicated an engineering calculation, while the elliptical components indicate a chemical equilibrium-based thermodynamic calculation, which is itself is a sub-group illustrated in figure 9. Input parameters and output values are the labels shown without a shape outline.

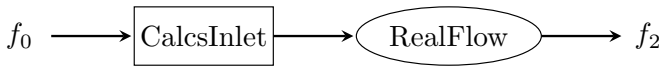

Figure 4. Structure of an inlet element.

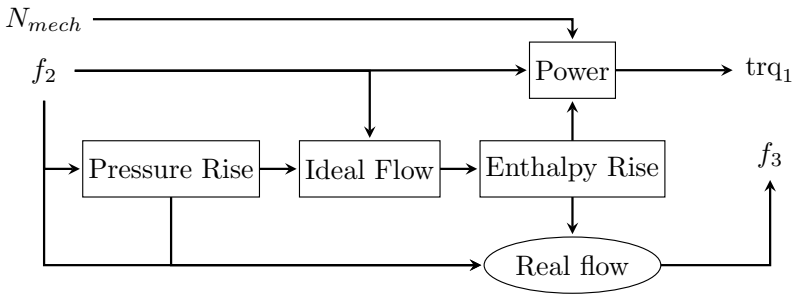

Figure 5. Structure of a compressor element.

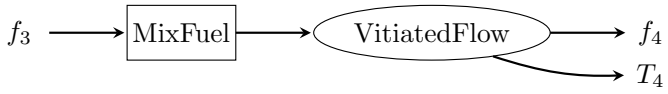

Figure 6. Structure of a burner element.

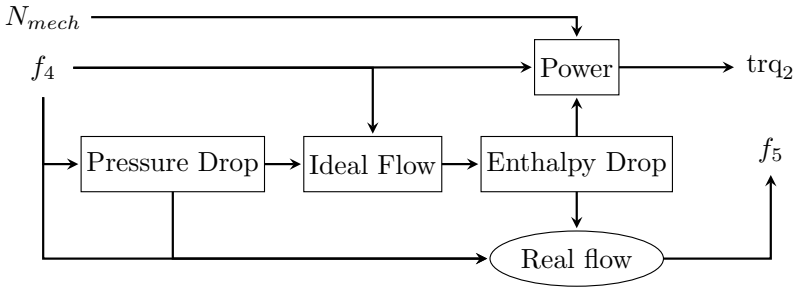

Figure 7. Structure of a turbine element.

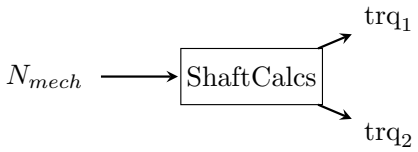

Figure 8. Structure of a shaft element.

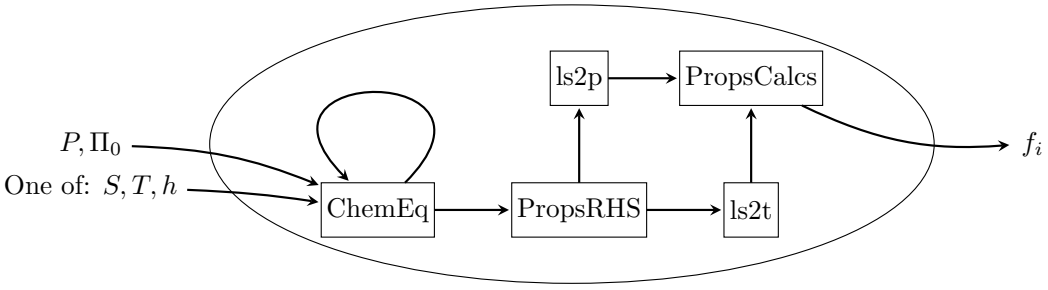

Figure 9. Structure of a thermodynamic chemical equilibrium component.

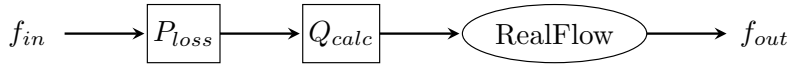

Figure 10. Structure of a duct element.

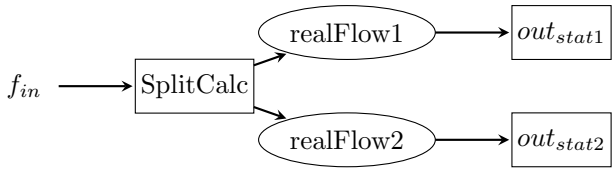

Figure 11. Structure of a splitter element.

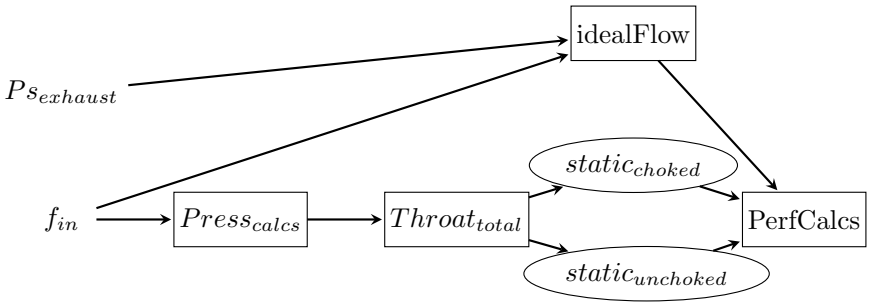

Figure 12. Structure of a nozzle element.

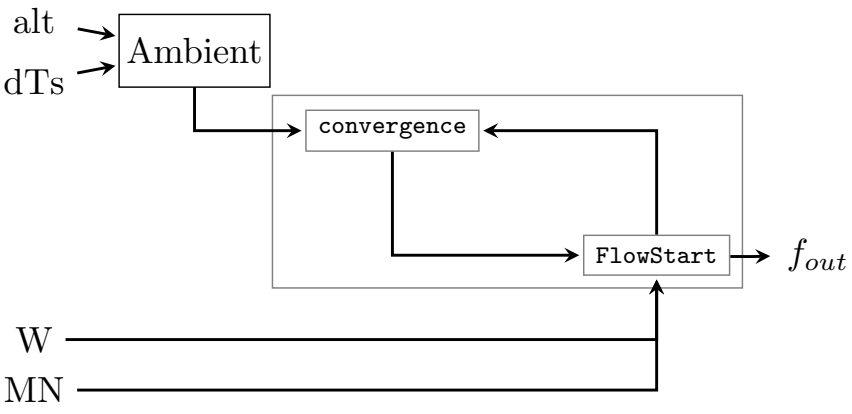

Figure 13. Structure of a Flight Conditions element.

References

[1]Rios, L. M. and Sahinidis, N. V., "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, Vol. 56, No. 3, 2013, pp. 1247–1293.

[2]Conn, A. R., Scheinberg, K., and Vicente, L. N., *Introduction to derivative-free optimization*, Vol. 8, Siam, 2009.

[3]Nielsen, E. J. and Diskin, B., "Discrete Adjoint-Based Design for Unsteady Turbulent Flows on Dynamic Overset Unstructured Grids," *AIAA Journal*, Vol. 51, No. 6, June 2013, pp. 1355–1373.

[4]Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations," *AIAA Journal*, Vol. 52, 2014, pp. 935–951.

[5]Palacios, F., Economon, T. D., Aranake, A. C., Copeland, S. R., Lonkar, A. K., Lukaczyk, T. W., Manosalvas, D. E., Naik, K. R., Padrón, A. S., Tracey, B., et al., "Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows," *AIAA paper*, Vol. 243, 2014, pp. 13–17.

[6]Kennedy, G. J. and Martins, J. R. R. A., "A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures," *Finite Elements in Analysis and Design*, Vol. 87, 2014, pp. 56 – 73.

[7]Kao, J. Y., Hwang, J. T., Martins, J. R. R. A., Gray, J. S., and Moore, K. T., "A modular adjoint approach to aircraft mission analysis and optimization," *56th AIAA SDM Conference*, Kissimmee, FL, 2015.

[8]Hwang, J. T., Roy, S., Kao, J. Y., Martins, J. R. R. A., and Crossley, W. A., "Simultaneous aircraft allocation and mission optimization using a modular adjoint approach," *56th AIAA SDM Conference*, Kissimmee, FL, 2015.

[9]Hwang, J. T. and Martins, J. R. R. A., "Parallel allocation-mission optimization of a 128-route network," *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Dallas, TX, 2015.

[10]Jones, S., *An Introduction to Thermodynamic Performance Analysis of Aircraft Gas Turbine Engine Cycles Using the Numerical Propulsion System Simulation Code*, NASA, 2007, TM-2007-214690.

[11]Felder, J. L., Kim, H. D., and Brown, G. V., "Turboelectric distributed propulsion engine cycle analysis for hybrid-wing-body aircraft," *47th AIAA Aerospace Sciences Meeting, Orlando, FL, January*, 2009.

[12]Freeh, J. E., Pratt, J. W., and Brouwer, J., "Development of a solid-oxide fuel cell/gas turbine hybrid system model for aerospace applications," *ASME Turbo Expo 2004: Power for Land, Sea, and Air*, American Society of Mechanical Engineers, 2004, pp. 371–379.

[13]Geiselhart, K. A., Ozoroski, L. P., Fenbert, J. W., Shields, E. W., and Li, W., "Integration of multifidelity multidisciplinary computer codes for design and analysis of supersonic aircraft," *49th AIAA Aerospace Sciences Meeting*, No. 2011-465, 2011.

[14]Hendricks, E. S., Jones, S. M., and Gray, J. S., "Design Optimization of a Variable-Speed Power-Turbine," *50TH AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA, Cleveland, Ohio, July 2014, AIAA-2014-3445.

[15]Gray, J. S., Chin, J. C., Hendricks, E. S., Hearn, T. A., and Lavelle, T., "Thermodynamics For Gas Turbine Cycles With Analytic Derivatives in OpenMDAO," *2016 AIAA SciTech Conference*, American Institute of Aeronautics and Astronautics, Janurary 2016.

[16]Martins, J. R. R. A. and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models," *AIAA Journal*, Vol. 51, No. 11, November 2013, pp. 2582–2599.

[17]Saad, Y. and Schultz, M. H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[18]Haftka, R. T., "Simultaneous analysis and design," *AIAA journal*, Vol. 23, No. 7, 1985, pp. 1099–1103.

[19]Jones, S. M., "Steady-State Modeling of Gas Turbine Engines Using The Numerical Propulsion System Simulation Code," *ASME Turbo Expo 2010: Power for Land, Sea, and Air*, American Society of Mechanical Engineers, 2010, pp. 89–116.

[20]Consortium, N. P. S. S. et al., "NPSS User Guide, Software Release: NPSS 2.3. 1, Revision 2," *User Documentation*.

[21]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM journal on optimization*, Vol. 12, No. 4, 2002, pp. 979–1006.