Advancement of the General Aviation Synthesis Program Using Python to Enable Optimization-Based Hybrid-Propulsion Aircraft Design

Kenneth R. Lyons, Benjamin W. L. Margolis, and Jeffrey V. Bowles
NASA Ames Research Center, Moffett Field, CA, USA

Jennifer D. Gratz, Sydney L. Schnulo, Eliot D. Aretskin-Harriton, Justin S. Gray, and Robert D. Falck
NASA Glenn Research Center, Cleveland, OH, USA

In support of Electrified Powertrain Flight Demonstrator and Advanced Air Transport Technologies programs at NASA, engineers at NASA Ames and NASA Glenn Research Centers have developed a new tool for coupled engine and airframe optimization and analysis. The new tool combines the engineering-level analysis methods of the FORTRAN General Aviation Synthesis Program (GASP) with the OpenMDAO framework to handle highly coupled problems that legacy tools struggle to optimize. The tool has been verified to match GASP with good agreement on a 737 MAX8 baseline vehicle closure problem, and preliminary efforts have been made to integrate the pyCycle thermodynamic cycle analysis tool for electrified engine optimization in the context of a vehicle optimization problem.

I. Introduction

Assessment of state-of-the-art concepts for the next generation of aircraft currently involves the use of trusted legacy tools relying on engineering level analysis methods for the mission analysis, together with tabular engine and aerodynamic performance data. While higher fidelity design tools can be integrated with these mission analysis tools, the resulting frameworks can be cumbersome to work with. A new tightly-integrated framework focusing on gradient-based optimization has been developed at NASA to perform coupled engine and airframe optimizations, taking vehicle performance across an entire design mission into account. Under the Transformative Tools and Technologies (TTT) project, the effort aims to support advanced technology assessments and designs for the Electrified Powertrain Flight Demonstrator (EPFD) program and Advanced Air Transport Technologies (AATT) project at NASA.

The new tool, called General Aviation Synthesis with Python (GASPy), is based on the FORTRAN General Aviation Synthesis Program (GASP) [1]. Initially developed at NASA Ames Research Center in the 1970s and further developed and enhanced at Georgia Tech School of Aerospace Engineering in the 1990's, GASP can size and estimate performance of fixed wing aircraft ranging from regional turboprop to commercial transport designs. It continues to be updated and used to study impacts of advanced technology for EPFD.

GASPy contains a nearly complete port of GASP's subsystem models in Python using OpenMDAO [2] as well as modern trajectory analysis and optimization methods based on the GASP mission implementation. So far, a Boeing 737 MAX8 model has been used as a baseline vehicle for verification of the new tool with vehicle sizing and economic mission analyses. In addition, an electrified geared turbofan thermodynamic cycle model has been included for the capability to perform coupled engine and airframe optimization. This work outlines the methods used in constructing GASPy and progress made so far in assessing its performance.

II. Background: The General Aviation Synthesis Program

GASP performs configuration sizing and performance estimates associated with the conceptual phase of aircraft design. It uses engineering level analysis methods across all technical disciplines to perform configuration sizing and estimated vehicle performance characteristics. Utilizing modular discipline analysis construction and integrated into a computational flow, the focus is on capturing the interaction and synergistic effects of the various technical disciplines. GASP determines configuration size and weight estimates, assesses aircraft performance and economics, and is useful in performing tradeoff and sensitivity studies. By utilizing GASP, the impact of various aircraft requirements, design factors, and advanced technologies, either singularly or collectively infused into the configuration design, may be studied systematically. Benefits can be measured in terms of overall aircraft weights, dimensions, and mission performance.

Six "technology" sub-modules perform the various independent functions required in the design of fixed-wing aircraft. The six modules include geometry, aerodynamics, propulsion, weight and balance, mission performance, and economics. The geometry module calculates the dimensions of the synthesized aircraft components based on such input parameters as the number of passengers, aspect ratio, taper ratio, sweep angles, and the thickness to chord ratio of wing and tail surface. The aerodynamics module calculates the various lift and drag coefficients of the synthesized aircraft based on inputs related to the gross configuration geometry, flight conditions, and the type of high-lift devices. The impacts of advanced aerodynamics technology are assessed using component level parasitic drag reduction factors, interferences drag and compressibility drag reduction. The propulsion module determines the engine size and performance for the synthesized aircraft based on an input reference engine performance deck, with engine scaling to meet various cruise, takeoff, and climb requirements for the aircraft. This module can currently simulate turbojet, turbofan, turboprop, and reciprocating engines. A propeller module estimates propeller performance, weight and noise. Aircraft subsystem component weights are computed based on historically derived weight estimating relationships, with technology factors used to assess the impact of advanced technologies on aircraft structural weight. In the mission performance module, the taxi, takeoff, climb, cruise, descent, and landing segments of a specified mission are analyzed to compute the total range. Aircraft can also be sized to meet a required range. An off-design mission can also be specified. Economic performance characteristics are estimated using manufacturing cost buildup and operational costs related to fuel and crew costs.

The six technology modules are integrated into a single system by a control module. This integrated approach ensures that the results from each module contain the effect of design interactions among all the modules. Starting from a set of simple input quantities concerning aircraft type, size, and performance requirements, the synthesis is extended to the point where all of the important aircraft characteristics are analyzed quantitatively.

GASP has been applied to a wide range of vehicle concepts, ranging from general aviation aircraft, thin haul and regional turboprop and turbofan designs, and to commercial transport designs, focusing primarily on the assessment of requirements and/or technology impacts. The assessment process typically begins with establishing a reference baseline design based on existing aircraft configurations or study designs, with calibration of the code to model published performance characteristics of the design. Trade studies are conducted for the baseline design to assess sensitivity of the vehicle characteristics to changes in various disciplinary performance levels in order to define the key areas of technology development that will have the biggest impact on the overall vehicle performance, and help define the required technology portfolio for further development. With the optimal set of technologies defined for a particular design and mission, the impact of the combined candidate advanced technologies is assessed and overall performance improvements resulting from the application of these advanced technologies relative to the baseline configuration determined.

III. GASP Implementation with Python

A. Software Architecture

NASA's OpenMDAO framework [2] was chosen as a basis for constructing the new GASPy tool, offering the ability to provide or estimate subsystem derivatives and assemble them into total derivatives across the entire model. OpenMDAO also provides solvers for resolving feedback loops in the model, interpolation methods for using tabular data (e.g. engine performance decks), and interfaces for a variety of optimizers. The result is a modular system model where subsystem models of varying fidelity can be interchanged as desired, with tooling to visualize the model structure, check for missing connections, and verify model derivatives.

Two different trajectory modeling libraries have been used to support ordinary differential equation (ODE) integration for evaluating mission perfomance: Dymos [3] and SimuPy [4]. In both cases, the user provides OpenMDAO systems for evaluating ODEs for each flight segment, segments are connected together, and derivatives of any boundary values or path integrals are computed by the framework to support optimization.

Dymos, itself built with OpenMDAO, is primarily suited for trajectory optimization via collocation, where the state variables and any controls are design variables in an optimization problem. This is a convenient option in cases where there are free dynamic variables to optimize directly, such as the angle of attack in a minimum-time-to-climb problem where a feedback controller design isn't the goal. Having this trajectory integration method as an option may in the future enable powerful optimization capabilities in situations where a fixed control schedule or control algorithm are not desired, such as an electric assist schedule for hybrid aircraft.

SimuPy is more focused on simulation, used here as a shooting method alternative to collocation. Its author has

recently developed methods to support computation of derivatives across the trajectory, supporting discontinuous events such as initiation of flap and gear retraction [5]. As opposed to collocation, which generally only produces a valid trajectory at the converged solution, shooting methods produce physically valid trajectories at each optimizer iteration. This method also introduces far fewer design variables and constraints to the overall optimization problem, and initial guesses for state values throughout the trajectory are not needed. However, it is typically more expensive per optimizer iteration, involving more evaluations of the ODE than collocation, which usually can use a sparser time grid.

B. Optimization Problem Formulation

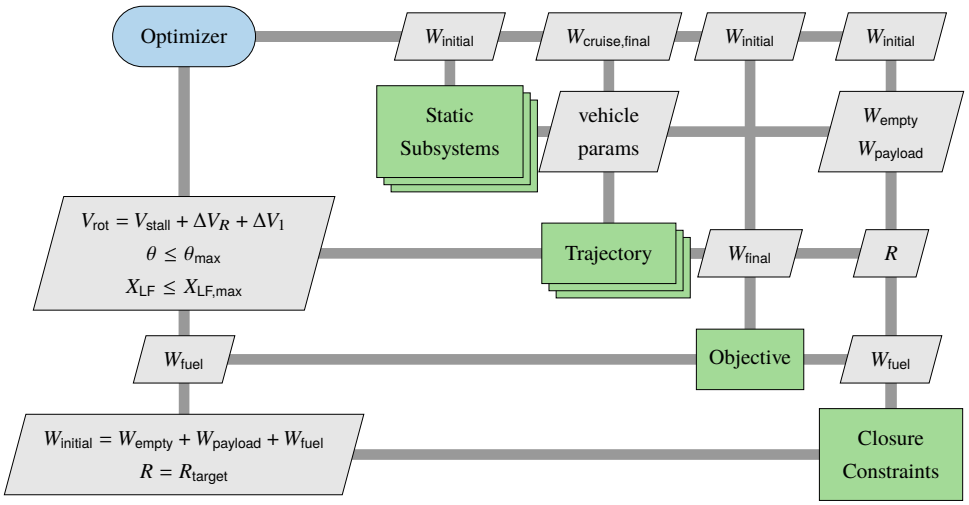

Fig. 1 Design structure matrix showing the overall model structure for the vehicle closure problem.

Figure 1 illustrates the general structure of the GASPy model for a vehicle sizing problem, where the initial gross takeoff weight, W_{initial}, is iterated on to achieve a range requirement and maintain weight closure. Controlling execution of the model, an optimizer drives design variables that feed into four key top-level systems. So far, GASPy has used the sequential quadratic programming algorithm provided by SNOPT [6], though other optimizers can easily be swapped in through the OpenMDAO programming interface. First, a static analysis group takes design variables as well as fixed user inputs and computes a variety of vehicle parameters that remain fixed throughout the mission (for a given optimizer iteration). The subsystems making up this static analysis group follow closely the methods and equations in GASP to compute vehicle component weights and dimensions. The trajectory group consists of a series of ODEs to evaluate state rates given vehicle parameters to compute aerodynamic performance and engine performance tables, and the trajectory integration method (either Dymos or SimuPy) integrates these ODEs to determine the vehicle trajectory over time. With Dymos, the vehicle states themselves become additional design variables, and constraints are added to link flight segments together. The final weight at the end of the trajectory compared to the initial gross takeoff weight driven by the optimizer gives the mission fuel used, and a reserve fuel weight is added to evaluate block fuel, forming the objective function to minimize. In a fixed target range mission, the range computed from the trajectory is constrained by an equality constraint and the computed block fuel, W_{fuel}, is made to combine with empty weight and payload weight to match the initial gross takeoff weight set by the optimizer via another equality constraint.

While the methods used within the static subsystems and the equations of motion underlying the trajectory integration are the same or equivalent to those in GASP, this overall structure for achieving vehicle closure is a major departure from GASP's ad hoc solver for balancing range. The advantage of this structure is that it can readily be extended for more complex performance analyses and optimizations by including additional design variables and constraints without major structural changes to the model. For example, a single line of code could add wing aspect ratio as a design variable and OpenMDAO would then provide the derivatives of the objective function and constraints with respect to the additional design variable to the optimizer to maintain vehicle and range closure while potentially finding a fuel-saving design.

C. Model Subsystems

Static subsystems are nearly direct ports of the GASP code in OpenMDAO. Various geometry parameters are calculated from user inputs, then component weights are computed. A solver iteratively converges the wing and fuel weight, since the wing structure and fuel capacity are interrelated. High lift devices are then sized and lift and drag increments determined for takeoff and landing. Once the static vehicle characteristics have been determined, model execution progresses to determining the mission trajectory.

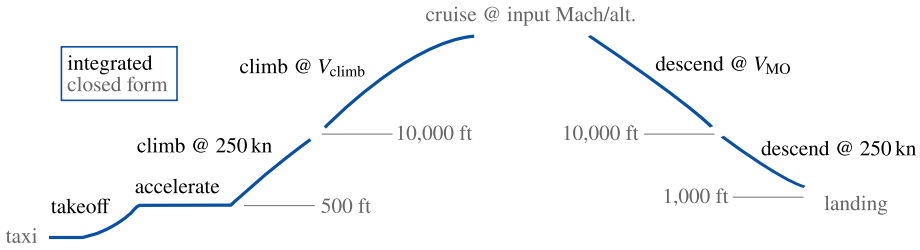

Fig. 2 Baseline mission trajectory specification. Segment distances/heights are not drawn to scale.

The trajectory specification implemented in GASPy is shown in Fig. 2. As in GASP, there are both closed-form and integrated flight segments. Taxi and landing are direct reimplementations of GASP in OpenMDAO. Taxi simply runs the idle engine model at airport altitude for a user input duration to evaluate weight change. Landing computes glide, flare, touchdown, and groundroll characteristics via analytic expressions [1], with engine and aerodynamic performance computed similarly to integrated flight segments. Cruise uses the Breguet range formulation, where the optimizer varies the final cruise weight such that, together with the other flight segments, the change in vehicle weight over the entire trajectory plus reserve fuel matches the fuel available.

The rest of the flight segments use either Dymos or SimuPy to integrate planar equations of motion. The 1976 U.S. Standard Atmosphere model, provided by Dymos, determines ambient conditions from the altitude at each point along the trajectory. The ambient conditions are then used in both the engine and aerodynamic models. GASPy supports GASP-formatted engine tables, parameterized by Mach number, altitude, and T_4/T_2 ratio or power code. There is also preliminary support for an N+3 geared turbofan engine model with electric augmentation, implemented with pyCycle. The throttle setting for each flight segment is specified, and the engine tables are interpolated or the pyCycle model is executed to evaluate thrust output and fuel flow rate at each time point. The GASP aerodynamic models are also replicated in GASPy to compute lift and drag at each time point from ambient conditions and vehicle parameters. In takeoff, lift and drag increments due to flaps, landing gear, and ground effects are accounted for and dynamically tapered to emulate flap and gear retraction. In level flight (the accelerate segment), the vehicle angle of attack is solved for so that lift balances the vehicle weight at each time, and thrust provided by the engines compared to overall drag determines the acceleration. Climb and descent use a quasi-steady model with constant equivalent airspeed and a path constraint on the flight path angle $\theta \leq \theta_{\max}$ (typically 15 degrees). The initial ascent portion of the takeoff segment also imposes load factor constraint $X_{\mathrm{LF}} \leq X_{\mathrm{LF,max}}$ (typically 1.1).

In the Dymos implementation, flight segments are linked together by equality constraints between final/initial state values of adjacent segments that the optimizer works to satisfy. This construction decouples each segment from one another such that they can be evaluated in parallel on a given optimizer iteration. When the pyCycle model is used for propulsion, where a single evaluation may take on the order of one minute, each collocation node can be evaluated in parallel to greatly reduce runtime. In the SimuPy implementation, segments are executed in series and final/initial state values are matched by construction.

IV. Baseline Vehicle Comparison

A Boeing 737 MAX8 vehicle model with a CFM International LEAP-1B engine was used as the baseline vehicle for both testing subsystem implementations against GASP as well as integration testing the vehicle closure solution. This vehicle has also been used as a reference for exploring impacts of advanced technology in the EPFD program [7]. The vehicle carries 180 passengers for a target range of 3,675 nautical miles. A comparison of the outputs from GASP and GASPy for this vehicle shown in Table 1. In the GASP case, the range is determined by using all fuel available (minus reserve). This range was then used as a target range for the GASPy vehicle sizing problem, which varies the gross takeoff weight to achieve the target range and minimize block fuel.

Table 1 Comparison of 737 MAX8 vehicle sizing results from GASP and GASPy

	GASP	GASPy	error
GTOW (lb)	175,400	174,562	-0.48%
OEW (lb)	96,348	95,805	-0.56%
Block fuel (lb)	43,050	42,756	-0.68%
Flight time (hr)	8.128	8.110	-0.22%
Cruise L/D	18.61	18.58	-0.16%

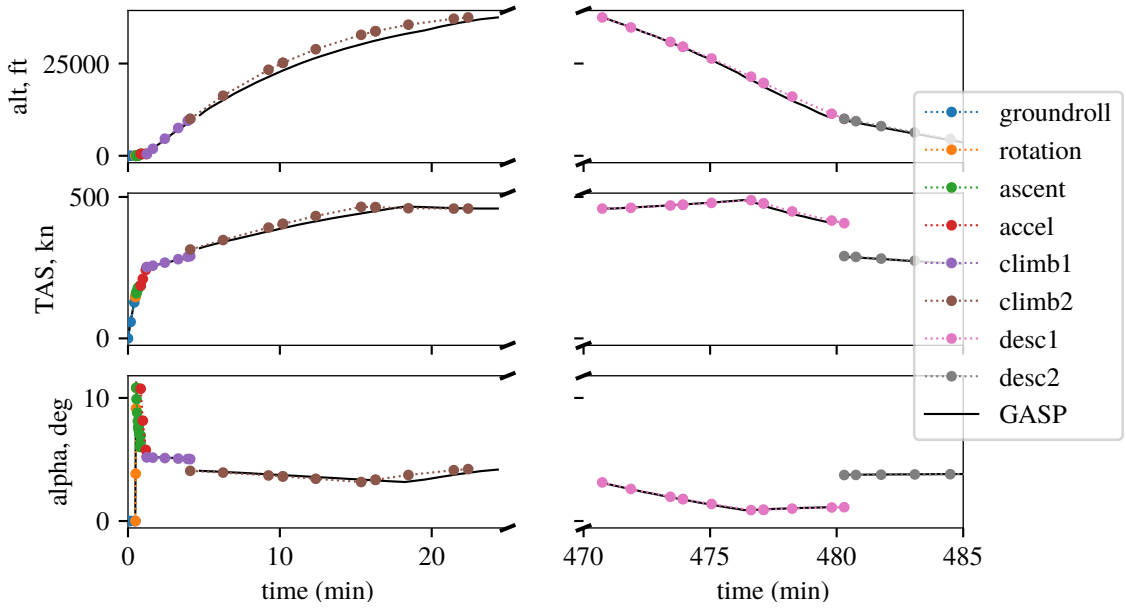

Fig. 3 Comparison of final GASP and GASPy trajectories following vehicle closure.

Figure 3 shows the overall trajectory for the case described above, showing good agreement between the GASP and GASPy mission implementations. GASPy is shown to climb faster than GASP, which is caused by a minor oversight in the climb equation of motion that is currently being addressed. Table 1 and Fig. 3 both correspond to Dymos implementations of the mission, though the SimuPy results are similar. Regarding performance, the SimuPy shooting implementation gives similar results in far fewer iterations (typically ~ 4), but each iteration takes longer to run. More thorough testing is needed to evaluate the tradeoffs between iteration duration, total number of iterations, and optimization stability.

V. Conclusion

The new GASPy tool shows promise in offering trusted methods and models for fixed wing vehicle synthesis with new capabilities in integrating detailed thermodynamic cycle analysis for coupled engine and airframe optimization. It has been shown to provide similar results to GASP, but its full capabilities have yet to be tested.

In the near future, the tool will be developed further to offer improved stability and robustness for vehicle and engine sizing problems, tested more thoroughly with pyCycle providing engine performance, and extended to support coupling with higher fidelity aerodynamic performance analysis. There are also efforts in progress to merge the GASPy tool with a reimplementation of LEAPS [8], bringing together methods from FLOPS and GASP into a single modern tool for vehicle analysis and optimization.

References

[1] Hague, D., "GASP – General Aviation Synthesis Program. Volume 1: Main Program. Part 1: Theoretical Development," Tech. Rep. NASA-CR-152303-VOL-1-PT-1, National Aeronautics and Space Administration, Jan. 1978.

[2] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. https://doi.org/10.1007/s00158-019-02211-z.

[3] Falck, R., Gray, J., Ponnapalli, K., and Wright, T., "dymos: A Python package for optimal control of multidisciplinary systems," *Journal of Open Source Software*, Vol. 6, No. 59, 2021, p. 2809. https://doi.org/10.21105/joss.02809.

[4] Margolis, B. W. L., "SimuPy: A Python framework for modeling and simulating dynamical systems," *The Journal of Open Source Software*, Vol. 2, No. 17, 2017, p. 396. https://doi.org/10.21105/joss.00396.

[5] Margolis, B. W. L., "Variational Derivatives for Ordinary Differential Equations with Events," *Journal of Optimization Theory and Applications*, 2023. Submitted for publication.

[6] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, Vol. 47, No. 1, 2005, pp. 99–131. https://doi.org/10.1137/s0036144504446096.

[7] Recine, C., Pham, D., Bowles, J. V., Lyons, K. R., Margolis, B. W. L., and Garcia, J. A., "Analysis and Optimization of Baseline Single Aisle Aircraft for Future Electrified Powertrain Flight Demonstrator Comparisons," *AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, San Diego, CA, 2023. Submitted for publication.

[8] Capristan, F. M., Caldwell, D., Condotta, R., and Petty, B., "Aircraft Analysis Using the Layered and Extensible Aircraft Performance System," Tech. Rep. NASA-TM-2020-220558, National Aeronautics and Space Administration, 2020.