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, 90435

Jennifer D. Gratz§, Sydney L. Schnulo¶, Eliot D. Aretskin-Hariton‖, Justin S. Gray**, and Robert D. Falck††
NASA Glenn Research Center, Cleveland, OH, 44135

In support of the Electrified Powertrain Flight Demonstrator and Advanced Air Transport Technologies projects at NASA, a new tool has been developed at NASA's Ames and Glenn Research Centers to enable coupled engine and airframe optimization and analysis. The new tool combines the engineering-level analysis methods and empirical models of the FORTRAN General Aviation Synthesis Program (GASP) with the Python-based OpenMDAO framework to provide a modular framework for efficient gradient-based optimization with the aim of incorporating new subsystem models for unconventional configurations. The tool has been verified against GASP analyses of several aircraft models and mission formulations. Preliminary efforts have been made to integrate pyCycle, a thermodynamic cycle analysis tool, to enable simultaneous optimization of hybrid propulsion system and vehicle parameters while taking full mission performance and constraints into account. This will improve current capabilities to assess impacts of electrified powertrain technologies on future aircraft designs.

I. Nomenclature

$\Delta(\cdot)$	=	change or increment in a variable
R	=	total mission range
θ	=	flight path angle
V	=	velocity (true airspeed unless otherwise specified)
W	=	vehicle or component weight
W/S	=	wing loading at takeoff
X_{LF}	=	takeoff load factor

II. Introduction

Assessment of state-of-the-art electric aircraft propulsion (EAP) designs is critical to the pursuit of ultra-efficient subsonic transports and achieving NASA's aggressive fuel, emissions, and noise targets. The process of analyzing these designs currently involves the use of trusted legacy tools relying on engineering-level methods for vehicle synthesis and mission analysis, together with tabular engine and aerodynamic performance data. While higher fidelity design tools can be coupled with these mission analysis tools, the resulting frameworks can be cumbersome. A tightly-integrated framework has been developed at NASA to perform gradient-based coupled engine and airframe optimizations, taking vehicle performance across an entire design mission into account rather than optimizing for

*Aerospace Engineer, Systems Analysis Office
†Aerospace Engineer, Systems Analysis Office
‡Aerospace Engineer, Systems Analysis Office
§Aerospace Engineer, Propulsion Systems Analysis Branch
¶Aerospace Engineer, Propulsion Systems Analysis Branch
‖Aerospace Engineer, Propulsion Systems Analysis Branch
**Aerospace Engineer, Propulsion Systems Analysis Branch
††Aerospace Engineer, Mission Architecture and Analysis Branch

particular flight conditions [1]. 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) and Advanced Air Transport Technologies (AATT) projects at NASA.

The new tool, called General Aviation Synthesis with Python (GASPy), is based on the FORTRAN General Aviation Synthesis Program (GASP) [2]. Initially developed at NASA Ames Research Center in the 1970s and further developed and enhanced at the Georgia Institute of Technology's School of Aerospace Engineering in the 1990s, GASP can size and estimate performance of fixed-wing aircraft ranging from regional turboprop to commercial transport designs. It continues to be updated to study impacts of advanced technology for the EPFD program. While recent extensions of the legacy code have added support for tabular engine data with and without electric augmentation power, it is limited to the use of a single engine design, motor size, and prescribed electrification schedule for a given analysis. Trade studies then require the generation of multiple sets of tables, typically by a different individual or team specializing in propulsion system design. If, for example, the vehicle-level system analyst would like to assess the impacts of a different engine parameter, effort is again required to generate more data.

To address this limitation and promote flexibly incorporating higher fidelity subsystem models directly, GASPy was developed as a nearly complete baseline reimplementation of GASP in Python using OpenMDAO [3]. Verification of subsystem models has been performed using a Boeing 737 MAX 8 model, with alterations to achieve improved test coverage for additional features not exercised by the MAX 8 model, such as a wing strut. An Embraer E190-E2 model was also created for both GASP and GASPy for further system-level testing. Detailed comparisons between GASP and GASPy results for these models are provided by Recine et al. [4]. In addition, an electrified geared turbofan thermodynamic cycle model implemented with pyCycle [5] has been included to perform coupled engine and airframe optimization. This paper describes the design, construction, and initial performance assessments of GASPy.

III. 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 (Fig. 1), 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 surfaces. 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 factors for parasitic drag, interference drag, and compressibility drag. 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, landing, and reserve segments of a specified mission are analyzed to compute the total range and fuel required. 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.

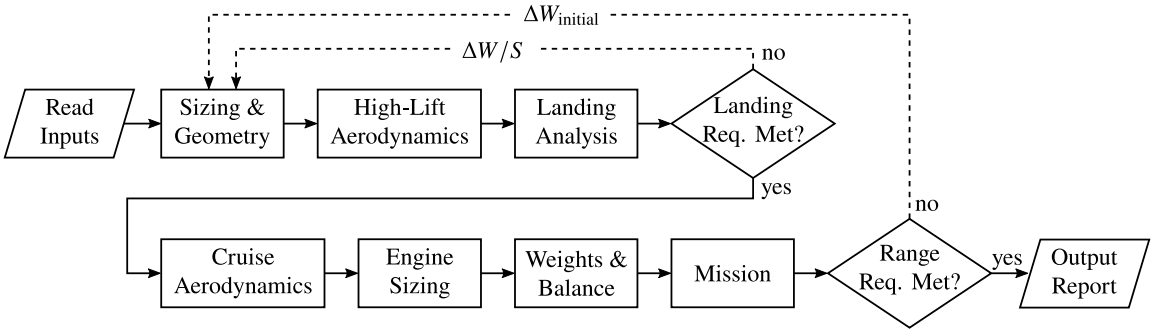

Fig. 1 GASP computational flow diagram, depicting the ordering of and feedback between subsystem analyses for a vehicle closure problem with engine sizing. Adapted from [2].

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 against 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 are determined.

IV. GASP Implementation with Python

A. Software Architecture

NASA's OpenMDAO framework [3] 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 algebraic loops in the model, interpolation methods for using tabular data (e.g. engine performance decks), and interfaces for a variety of optimization packages. The result is a modular system model where subsystems of varying fidelity can be interchanged as desired, with tooling to visualize the model structure, check for missing connections, and verify model derivatives. The resulting model can then be flexibly configured for analyses or gradient-based optimizations.

Two different trajectory modeling libraries have been used to support ordinary differential equation (ODE) integration for evaluating mission performance: Dymos [6] and SimuPy [7]. In both cases, the user provides OpenMDAO models for evaluating ODEs for each flight segment and specifies how segments are linked together. As the trajectory model is executed, derivatives of the performance metrics with respect to vehicle parameters are available to support incorporation into the overall model for gradient-based optimization.

Dymos, itself built with OpenMDAO, primarily focuses on trajectory optimization problems via collocation, where the state variables and any controls at a user-specified time grid become design variables in an optimization problem with constraints to enforce dynamics. 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. It can also be extremely efficient, requiring relatively few ODE evaluations that can even be run in parallel in some cases. Initial "guesses" for the states along the trajectory are required, however, which can be a challenge when exploring conceptual aircraft designs or changing mission specifications. Bounds on state and control variables throughout the trajectory are also often needed to ensure the optimizer doesn't drive the model into an ill-specified region.

SimuPy is more focused on simulation, used here as a shooting method alternative to collocation. Its author has recently developed methods to compute derivatives across discontinuous events such as initiation of flap deployment/retraction

and landing gear retraction [8]. 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 no additional design variables or constraints to the overall optimization problem, and initial guesses for state variables throughout the trajectory are not required. Using an initial value problem (IVP) solver with adaptive step size, the time points at which the ODEs are evaluated ultimately reflect the dynamics automatically, hence the time grid doesn't need to be specified and potentially tuned as in collocation.

In addition to the reimplementation of GASP's table-based engine performance scaling and evaluation, a geared turbofan engine model implemented with pyCycle [5] has been incorporated into the GASPy code. pyCycle offers physics-based thermodynamic cycle analysis with analytic derivatives such that engine design parameters may be incorporated into the overall design optimization problem. In the future, other tools may be substituted for GASP's models to increase fidelity as a vehicle concept develops to account for more complex interactions.

B. Optimization Problem Formulation

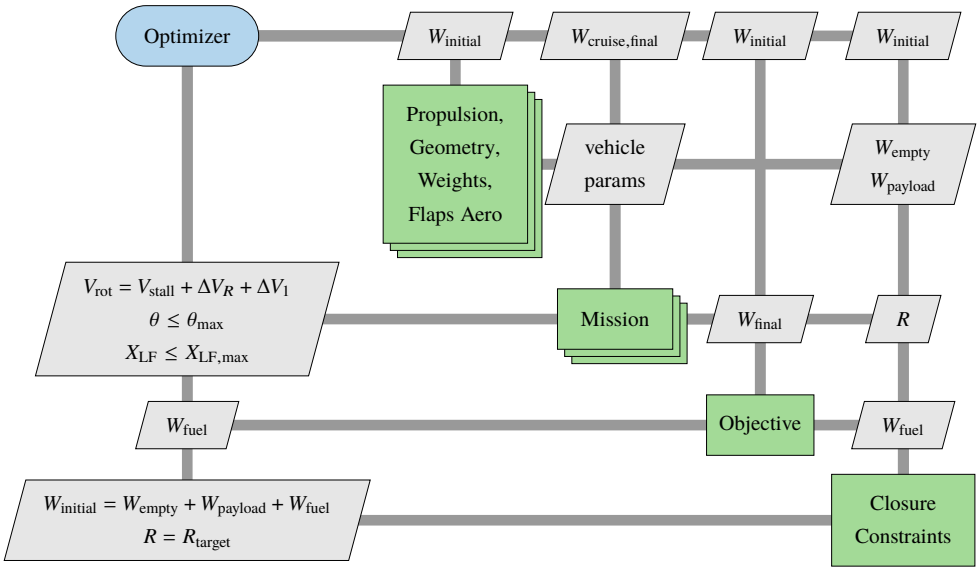

Fig. 2 Simplified design structure matrix showing the overall model structure and optimization variables for a vehicle closure problem. Computation generally flows between subsystems (green boxes) in a clockwise direction. Subsystems output variables to the right or left and take inputs from above or below. The optimizer at the outermost loop initiates model evaluation and drives design variables to a converged solution. Additional input parameters from the user (not shown), may feed into any subsystems directly.

Figure 2 is a design structure matrix illustrating the general structure of the GASPy model for a vehicle sizing problem, where the design gross takeoff weight, W_{initial}, is iterated on to achieve a range requirement and maintain weight closure. Controlling execution of the model, an optimizer (blue) drives variables that feed into four key top-level systems, depicted as green boxes along the diagonal. Inputs to the subsystems are aligned vertically and outputs are aligned horizontally. Additional inputs from the user, specifying fixed values such as wing aspect ratio, target range, number of passengers, etc. are not shown for simplicity. So far, GASPy has been tested using the sequential quadratic programming algorithm provided by SNOPT [9] and the interior point algorithm provided by IPOPT [10].

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: geometry, high-lift device aerodynamics, engine scaling, and weights. Of primary importance in the static subsystems for the overall problem is the fuel weight (W_{fuel}), which is determined iteratively. The mission group consists of a series of ODEs to evaluate state rates given vehicle parameters to compute aerodynamic performance and engine performance. The trajectory evaluation method (either collocation or shooting) then integrates these ODEs to determine the vehicle trajectory over time. With collocation, the

vehicle states — typically altitude, weight, and range — themselves become additional design variables with constraints to enforce the dynamics and continuity between phases. The final weight at the end of the trajectory is compared to the initial gross takeoff weight driven by the optimizer, after which a reserve fuel weight is added to evaluate block fuel, which can be used as an objective 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 combined with the 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 are the same or equivalent to those in GASP, this overall structure for achieving vehicle closure is more general than GASP's custom solver loop for range balance. 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. OpenMDAO would then provide the derivatives of the objective function and constraints with respect to the additional design variable to the optimizer. The optimizer would then attempt to maintain vehicle and range closure while potentially finding a fuel-saving design. In addition, other mission types can be implemented via simple changes in problem formulation. Performance of a given aircraft in a short-range/economic mission may be evaluated by fixing the design gross takeoff weight to that obtained in a vehicle closure run and varying the actual initial fuel load to achieve the specified range. The maximum range of a fixed vehicle with a specified payload may be evaluated by fixing the takeoff weight and maximizing the range rather than constraining it to a fixed value.

C. Model Subsystems

Static subsystems in GASPy are nearly direct ports of the GASP code to Python with OpenMDAO. Various geometry parameters are calculated from user inputs, then component weights are computed. Within the weights subsystem, a solver iteratively converges the wing and fuel weights, 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 trajectory integration.

A notable departure from GASP is how engine size is determined when it is not specified by the user, though the *scaling* of table-based engine performance variables is the same. When using tabular engine data, scaling is based on an assumption that, for a given flight condition and power setting, the specific thrust and percent corrected airflow of the reference and scaled engines are equal [11]. The sea level static (SLS) airflow is used as the scaling variable and in GASP, is computed directly given SLS thrust provided by the user or thrust required to meet one or more predicted aircraft performance criteria such as a minimum rate of climb at top of climb or maximum takeoff field length. So far in GASPy, a simpler automatic engine sizing approach has been taken, where the rate of climb at top of climb as flown in the design mission is constrained by a lower bound, which the optimizer satisfies. Additional criteria may be added in the future, such as 14 CFR Part 25 climb gradient requirements, requiring a separate trajectory integration from the design mission. While it is somewhat more efficient to predict weight at the top of climb hence estimating the corresponding achievable rate of climb before performing mission analysis, the GASPy approach gives a more accurate minimal engine size without needing to manually adjust the weight at top of climb. With a full thermodynamic cycle analysis supplying engine performance via pyCycle, the "size" of the engine is set by changing the airflow at the chosen design point for the engine. Rather than becoming a simple scaling factor on thrust and fuel flow like in the tabular engine data case, the design point cycle is analyzed within the static analysis group and parameters are passed to off-design points within the mission ODEs.

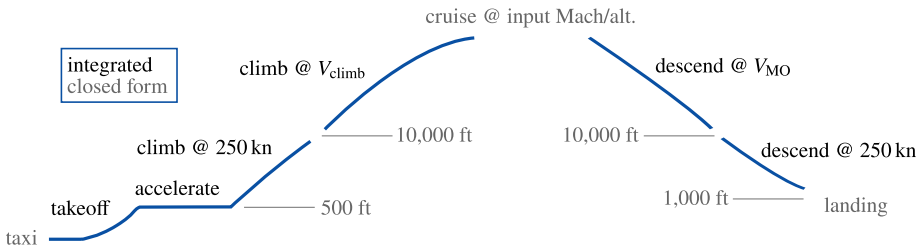

Fig. 3 Baseline mission trajectory specification. Segment dimensions are not drawn to scale.

The trajectory specification implemented in GASPy is shown in Fig. 3. 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 engine at idle power at airport altitude for a user input duration to evaluate weight change. Landing computes glide, flare, touchdown, and groundroll characteristics via analytic expressions [2], 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 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. During 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 the load factor constraint $X_{\mathrm{LF}} \leq X_{\mathrm{LF,max}}$ (typically 1.1).

In the collocation 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 several seconds or more, each collocation node can also be evaluated in parallel to greatly reduce runtime for a given optimizer iteration. In the shooting implementation, segments are executed in series and final/initial state values between segments are matched by construction.

V. Baseline Vehicle Comparison

A Boeing 737 MAX8 vehicle model with tabular performance data for 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. The design mission specification used for verification is shown in Table 1. The vehicle has also been used as a reference for exploring impacts of advanced technology in the EPFD program [4].

Table 1 737 MAX 8 design mission vehicle closure problem specification

Payload	36,000 lb (180 PAX)
Range	3,675 NM
Cruise Mach	0.8
Cruise Altitude	37,500 ft
Reserve fuel	4,998 lb
SLS thrust	28,690 lbf

A summary comparison of the outputs from GASP and GASPy for this vehicle is shown in Table 2. Numerical values for GASPy are from the collocation-based mission implementation, though the outputs from shooting are nearly identical. Figure 4 shows the overall trajectory for the case described above, showing overall good agreement between the GASP and GASPy mission implementations. GASPy is shown to climb faster than GASP, caused by the lack of a small acceleration term in the climb equations of motion accounting for the gradual change in true airspeed with a constant equivalent airspeed climb. GASP uses an ad hoc finite difference approximation for this term, whereas the GASPy implementations will require more careful treatment, accounting for change in air density with altitude and the transition to cruise Mach or altitude, whichever occurs first. Equations of motion are shared between the collocation and shooting implementations, so this discrepancy is shown in both and the shooting trajectory essentially passes through the collocation nodes. Since performance at cruise dominates overall mission performance for the long-range design

mission, the impact to the overall vehicle closure problem is relatively small. Regarding computational performance, the SimuPy shooting implementation gives similar results in typically 3–4 iterations for a vehicle sizing problem, whereas the Dymos collocation implementation takes roughly 20–30 iterations in part due to the additional design variables and constraints. Run time with tabular engine performance data is similar in most cases without extensive tuning of initial guesses for the collocation implementation. More thorough testing is needed to evaluate the tradeoffs between iteration duration, total number of iterations, and optimization stability.

Table 2 Summary comparison of GASP and GASPy results for 737 MAX 8 closure

	GASP	GASPy	error
GTOW (lb)	176,016	175,678	-0.19%
OEW (lb)	96,928	96,547	-0.39%
Block fuel (lb)	43,086	43,132	0.11%
Cruise L/D	18.66	18.47	-1.04%
Cruise TSFC	0.5487	0.5484	-0.05%

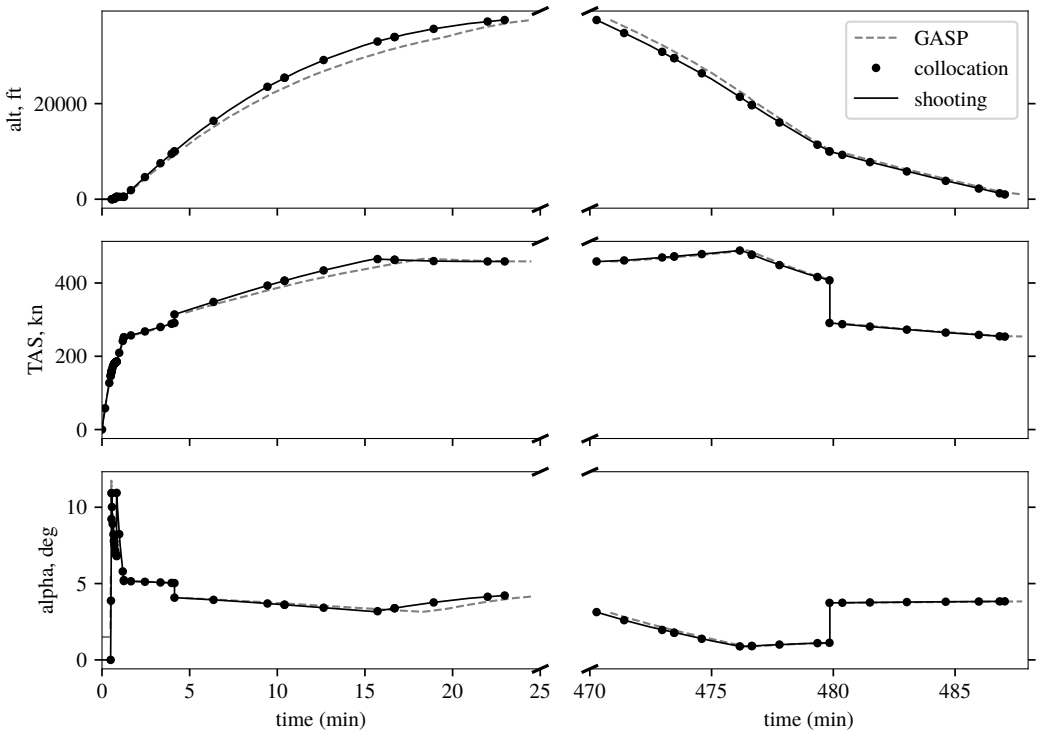

Fig. 4 Comparison of converged GASP and GASPy trajectories for the 737 MAX 8 design mission. Cruise at constant Mach and altitude is not shown, taking place between the two halves of each plot.

Convergence with the 737 MAX 8 model with the tabular engine data replaced by a notional N+3 electrified geared turbofan high-bypass pyCycle engine model has been achieved, demonstrating progress toward coupled engine and airframe optimization. The engine model with a top-of-climb design point runs within the static analysis group and passes design parameters to equivalent engine models configured for off-design evaluation at arbitrary flight conditions within the ODEs. The 737 MAX 8 vehicle closure problem with the design point engine fixed was run on NASA's Pleiades cluster, taking approximately 25 minutes to converge with 42 processes (one for each collocation node). Additional work is ongoing to support automatic sizing of the design point engine to meet mission constraints (e.g. rate of climb at top of climb), however the lack of detailed engine weight estimation is a barrier to achieving meaningful results.

VI. 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 thoroughly. 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 analyses. There are also efforts in progress to merge the GASPy tool with a reimplementation of LEAPS [12], bringing together methods from FLOPS and GASP into a single modern tool for vehicle analysis and optimization. Ultimately, in order to address the needs of the NASA programs to assess both conceptual and high technology readiness level designs, additional tooling around the core of a tool such as GASPy will be needed to incorporate estimation of emissions and community noise.

References

[1] Adler, E., and Martins, J. R., "Aerostructural wing design optimization considering full mission analysis," *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022. https://doi.org/10.2514/6.2022-0382.

[2] 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.

[3] 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.

[4] 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.

[5] Hendricks, E. S., and Gray, J. S., "pyCycle: A Tool for Efficient Optimization of Gas Turbine Engine Cycles," *Aerospace*, Vol. 6, No. 8, 2019, p. 87. https://doi.org/10.3390/aerospace6080087.

[6] 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.

[7] 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.

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

[9] 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.

[10] Wächter, A., and Biegler, L. T., "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, Vol. 106, No. 1, 2005. https://doi.org/10.1007/s10107-004-0559-y.

[11] Hague, D., "GASP – General Aviation Synthesis Program. Volume 4: Propulsion. Part 1: Theoretical Development," Tech. Rep. NASA-CR-152303-VOL-4-PT-1, National Aeronautics and Space Administration, Jan. 1978.

[12] 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.