Aviary: An Open-Source Multidisciplinary Design, Analysis, and Optimization Tool for Modeling Aircraft with Analytic Gradients

Jennifer Gratz*, John Jasa†, Eliot Aretskin-Hariton‡, Kenneth Moore§, Kiran Marfatia¶
Glenn Research Center, National Aeronautics and Space Administration

Jason Kirk‖
Langley Research Center, National Aeronautics and Space Administration

Carl Recine**
Ames Research Center, National Aeronautics and Space Administration

In recent years demands on aircraft design methods have begun to require higher degrees of coupling between disciplines and optimization in order to satisfy competing objectives involving large numbers of design parameters that define unconventional configurations. These expanding requirements have amplified a need for new and improved aircraft design, analysis, and optimization codes that are capable of performing coupled design and exploiting analytic gradients to perform gradient-based optimization where possible. To address this need, a new multidisciplinary design optimization and analysis tool called Aviary is presented. This tool, built on OpenMDAO, allows for tightly coupled, simultaneous aircraft and subsystem design using analytic gradients. Aviary includes methods from two NASA developed legacy aircraft analysis tools and provides native analytically differentiated calculations for five different disciplines (weights and sizing, aerodynamics, geometry, propulsion, and mission analysis), while also allowing external discipline analysis tools to be coupled, regardless of whether those tools can provide analytic gradients. Verification and preliminary examples and modeling efforts show Aviary's ability to effectively model novel concepts and explore large and non-intuitive design spaces. Additionally, a multi-level user interface in Aviary creates an easy entry point for users with any level of multidisciplinary design, analysis, and optimization experience.

*Aerospace Engineer, NASA Glenn Research Center, Cleveland, OH, USA
†Systems Engineer, NASA Glenn Research Center, Cleveland, OH, USA, Banner Quality Management Inc.
‡Aerospace Engineer, NASA Glenn Research Center, Cleveland, OH, USA, Member AIAA
§Senior Systems Engineer, NASA Glenn Research Center, Cleveland, OH, USA, Banner Quality Management Inc.
¶Student Intern, NASA Glenn Research Center, Cleveland, OH, USA, Student Member AIAA, Hackley School
‖Aerospace Engineer, NASA Langley Research Center, Hampton, VA, USA
**Aerospace Engineer, NASA Ames Research Center, Mountain View, CA, USA

Nomenclature

Acronyms

2DOF	Two Degrees of Freedom
EOM	Equations of Motion
FLOPS	Flight Optimization System
GASP	General Aviation Synthesis Program
MDAO	Multidisciplinary Design Analysis and Optimization
OAS	OpenAeroStruct
SNOPT	Sparse Nonlinear OPTimizer
T_2	Compressor inlet temperature
T_4	Turbine inlet temperature
TTBW	Transonic Truss-Braced Wing
VSPAERO	Vehicle Sketch Pad for AERODynamics
XDSM	eXtended Design Structure Matrix

I. Introduction

The design and analysis of aircraft models is a field that has changed and grown significantly over time. Until recently, the main method used for aircraft design and analysis at NASA was a moderately discrete approach where experts in different disciplines such as engine design, aerodynamic analysis, and performance analysis would independently design their portion or discipline of the aircraft, and then manually bring their designs together in an iterative process. This process required a human to drive the iteration loop between these different discipline experts, and at the time this was an effective and successful process. Eventually aircraft analysis tools began to emerge, and these tools improved this process. Flight Optimization System (FLOPS), a NASA developed aircraft configuration optimization program [1], improved the process significantly by providing a framework with which less human interaction was required in the design iterations, and some optimization was possible. FLOPS is used to perform conceptual design, though it still requires mass properties, aerodynamics, and propulsion analysis experts to design their subsystems separately and provide data if anything more than simple empirical relations are necessary.

Another example code which took a step in the right direction is the General Aviation Synthesis Program (GASP), a NASA developed preliminary aircraft design program that gives an aircraft analyst the ability to rapidly perform parametric studies [2]. The GASP software also advanced the state of the art in aircraft design, although GASP, similar to FLOPS, still relies heavily on aerodynamicists and propulsion analysts building data tables that are manually provided to the aircraft sizing process. These state of the art tools have several benefits, and throughout this paper they will be referred to as "legacy" tools because they have a long legacy of use and reliability (and continue to be used as the state of the art within NASA today). These legacy tools are relatively fast at performing their analyses, they provide good default values that produce reliable answers without the user specifying every single input, and their results are validated and are trusted within the category and class of aircraft these tools were built to model. These benefits make the legacy tools valuable for many applications. However, there are some applications that require capabilities the legacy tools do not have.

Over time, the demands on aircraft modeling software have expanded with the rising demand for unconventional aircraft configurations, such as electric aircraft [3], hybrid-electric aircraft [4], and urban air mobility vehicles [5]. Interest in these aircraft is primarily driven by a desire to decrease the environmental impact of aviation and minimize flight costs. These unconventional configurations raise a new problem: the disciplines of these aircraft are more tightly coupled than those of traditional subsonic transport type aircraft, and the traditional methods where subsystem experts manually iterate with aircraft modeling experts to build data tables need to be augmented. The new vehicles require a new vehicle-analysis paradigm, where discipline tools can be directly coupled to each other without a human in the loop. [5] [3] [6] A few examples of the need for and success of this coupled approach can be seen in Heath et al. [7], Li et al. [8], and Hendricks et al. [5] Heath et al. and Li et al. demonstrate the need for coupled disciplines in the high-speed regime of supersonics, while showing that coupled aerodynamic and propulsion optimizations can be done. Both Heath et al. and Li et al. performed coupled aero-propulsive design of a supersonic aircraft in order to reduce the sonic boom. [7] [8] In the work performed by Heath et al. they were specifically interested in the coupling effects caused by inlet spillage and nozzle jet exhaust of the engine on the sonic boom of the vehicle (typically an aerodynamic effect) as well as the viscous aerodynamic effects on the propulsion performance. [7] This coupling is necessary to include in the model in order to capture the engine installation effects on the sonic boom. [7] Li et al. studied several different situations that each included different specific coupled variables which capture the engine's impact on the vehicle aerodynamics and the aerodynamics' impact on engine performance.[8] On the other end of the speed spectrum, Hendricks et al. demonstrate the need for coupled disciplines in an Urban Air Mobility (UAM) environment. They compared the design of a baseline tilt-wing aircraft to several different re-designs that accounted for increasing amounts of coupling, and they showed that the fully coupled design produces the best aircraft (measured by increased overall efficiency and reduced power requirements). [5] The coupled effects that they studied were the impacts of propeller design, motor and generator speed, and wing design on the vehicle performance when the vehicle operation was allowed to be optimized. [5] These examples at the high-speed and low-speed ends of the aircraft design space demonstrate the importance of coupling between different disciplines in unconventional aircraft, and they also demonstrate that successful efforts have been made to find design improvements by exploiting these discipline couplings.

One significant benefit that has emerged from the need for tighter coupling between multiple disciplines is the ability to take advantage of these couplings by doing optimization. The complexity of the coupled interactions between disciplines in unconventional aircraft makes it difficult to know a priori what design choices will lead to the best overall vehicle as opposed to just the best individual subsystem. Optimizers can help with finding this ideal design by exploring a larger design space than can easily be visualized by an engineer. A significant part of optimization work is the use of accurate and efficient gradients, however, these can have a high computational cost. Multidisciplinary

Design Analysis and Optimization (MDAO) making use of analytic gradients has emerged as a new methodology for aircraft design and has quickly become a key ingredient for improving computational speed. The link between increased computational efficiency and the use of analytic gradients is explained by Gray et al.: "gradient-based optimization algorithms scale much better with the number of design variables than gradient-free methods. The computational efficiency of both Newton-type analysis methods and gradient-based optimization is, in large part, dependent on the cost and accuracy with which the necessary derivatives are computed." [9] Further support for this is provided by Martins and Ning: "Gradient-based optimization using the adjoint method is a powerful combination that scales well with the number of variables".[10] This indicates the computational efficiency benefits that can be achieved in MDAO by using computationally cheap and accurate derivatives. Due to the fact that computing analytic derivatives requires fewer function calls than obtaining derivatives via finite-differences, and that analytic derivatives give precise numerical results instead of estimates, analytic derivatives present themselves as computationally superior to finite-difference derivatives, all else being equal. The effects of this reality were shown by Hendricks et al. through a comparison of optimization computation time for engine cycle analysis codes with and without analytic gradients. [11] They found that derivative computation time was more than 300 times shorter when using an analytically differentiated code as opposed to a non-analytically differentiated code. Thus, as aircraft concepts become more tightly coupled between disciplines, they call for codes with a few specific features: the ability to couple together discipline and analysis tools, object-oriented building blocks which can be used for this coupling of disciplinary tools, the ability to perform efficient computation across an extensive number of coupled parameters, and accurate and efficient gradients to speed up this computational process.

One of the challenges with some coupling and optimization frameworks is that by nature optimization problems are not simple to execute, and they often require advanced knowledge of both the framework in use and optimization itself in order to implement something as complex as an aircraft optimization. While aircraft coupling and optimization is powerful, it is also challenging because it requires discipline expertise, niche expertise in optimization, and sometimes expertise in computer programming. In order to make coupled optimizations more common in aircraft design, effort needs to be made to develop a tool that allows users to perform aircraft optimization more simply. While optimization will never be simple, it is possible to make certain types of optimization simpler than in the past.

Aviary* is a new open-source software built on top of OpenMDAO and written in Python that advances the state of the art towards a fully-coupled and fully analytically differentiated aircraft design and analysis tool. The goal of Aviary is to provide an aircraft MDAO tool where disciplines and tools can be tightly coupled together in a computationally efficient manner; where aircraft analysts can run designs, analyses, and optimizations of novel and unconventional aircraft concepts without having to modify the underlying code architecture of the tool; and where aircraft analysts can run simpler analyses without needing any expertise in MDAO. In pursuit of these development goals, Aviary is an analytically differentiated aircraft design and analysis tool that provides a platform for performing aircraft optimization, includes the low-fidelity modeling techniques of both of NASA's existing legacy aircraft analysis tools GASP and FLOPS, and includes a defined interface for smoothly coupling external disciplinary tools into the optimization. It is important to note that several of the external disciplinary tools which users may couple to Aviary do not have analytic gradients. In these cases Aviary takes advantage of the finite difference and complex-step [12] capabilities of OpenMDAO to compute derivatives, but it simultaneously supports tools providing analytic derivatives if they are available. The support for analytic gradients in external tools will help Aviary to remain a tool of enduring relevance as the state of the art for aircraft design advances towards MDAO.

This paper presents Aviary as a multidisciplinary design optimization and analysis tool which allows for tightly coupled simultaneous aircraft and subsystem design using analytic gradients. Specifically, this paper will discuss the methods that are used in Aviary, the external tool integration that Aviary has created, the structure of the code within Aviary, the methods verification done on Aviary, the run time comparison of Aviary's analytic gradients to finite-difference gradients, and the methods of user interface which Aviary has distinguished.

II. Background

The aircraft modeling capabilities of Aviary come from the GASP and FLOPS tools developed by NASA and mentioned above. The GASP tool was developed in the 1970s as a tool to aid in the preliminary design of small fixed-wing vehicles and their rapid parametric study [2]; over time, the program has evolved and expanded to include larger vehicles as well. GASP has been effectively used for several decades to perform aircraft design and analysis at NASA. The FLOPS tool was developed in the 1970s and 1980s to perform fast design of aircraft at the conceptual level

*https://github.com/OpenMDAO/Aviary

and to assess the impacts of advanced technology. [13] It has both analysis and design capabilities and has been used extensively for several decades to perform aircraft design and analysis at NASA.

Several efforts have been made to adapt the existing aircraft design tools such as FLOPS and GASP to the increased demands for coupling mentioned above. One framework that was developed, Environmental Design Space (EDS) [14], uses Numerical Propulsion System Simulation (NPSS) [15] as a driver for "multiple industry standard tools" integrated together in a single framework. [14] In addition, various studies such as a hybrid electric aircraft study out of the NASA Glenn Research Center built their own multidisciplinary drivers using FLOPS [4], and various updates were made to GASP to begin adapting to the new electric aircraft needs. However, across all of these efforts there are still elements that need to be added. Many of the independent frameworks built under FLOPS still require some sort of data table (e.g. for drag or engine performance) to be designed outside the framework. The updates to GASP model the electrical subsystems of the aircraft effectively but do not couple in other disciplines, and none of the studies mentioned include analytic gradients. In addition, none of the studies above are open-source, which means that the tools cannot leverage contributions and feedback from the public as effectively, and it also means that the tools will generally have higher barriers to entry. They are all stepping stones on the way to an aircraft design method that meets the needs of a coupled comprehensive design that makes use of an efficient gradient-based optimization in an open-source way, but there is still room for further development.

Outside of NASA, a multitude of conceptual-level aircraft design tools have been developed within both academia and industry. SUAVE [16] was created to analyze and optimize conventional and unconventional aircraft configurations. It is open-source and is also multi-fidelity in that it can combine different subsystem models at varying levels of fidelity. FAST-OAD [17] is an overall aircraft design tool that focuses on modularity and usability. It comes with empirical models for aerodynamics, propulsion, structures, and other disciplines based on commercial aircraft from the 90s and 2000s. AeroSandbox [18] is a conceptual-level open-source aircraft design tool that is fully differentiable, and it allows the use of efficient gradient-based optimization. AeroSandbox also allows for interfacing with user-defined external tools or additional disciplinary subsystems. Many other tools exist with similar capabilities and relatively limited information is available on tools internal to large aerospace design companies such as Boeing and Airbus. Although each of these aircraft design tools is useful, a single open-source tool that combines analytic derivatives, coupled aircraft-mission design optimization, and NASA-validated aircraft sizing equations did not exist prior to the work presented here.

The codes and tools discussed here lie on a spectrum of aircraft design frameworks that runs from a simple analysis tool with no analytic derivatives and requiring multiple data tables to be provided, all the way up to a fully integrated and completely differentiated tool which couples in every discipline and has analytic gradients for every discipline in a single framework. The ideal end of this spectrum has not yet been reached by the state of the art. OpenMDAO [9] [19] is a framework that makes the use of analytic gradients in a coupled optimization environment possible, but it only provides the bones for an MDAO aircraft tool, it does not actually provide any of the aircraft disciplines or an interface that aircraft analysts unfamiliar with an MDAO environment could use. Thus, OpenMDAO is not exactly a stepping stone on this spectrum, however, the capabilities of OpenMDAO can play a key role in advancing the state of the art in aircraft design tools by serving as a platform for those tools. Aviary is built on top of OpenMDAO and uses it to assist in applying the GASP and FLOPS methods to an MDAO tool. By leveraging OpenMDAO as well as the existing aircraft modeling capabilities of legacy tools, Aviary takes another step along this spectrum towards fully coupled and differentiated modeling capability.

III. Disciplines in Aviary

Aviary employs analytic gradients on legacy aircraft design methods for sizing, aerodynamics, geometry, propulsion, and mission analysis. Specifically, Aviary takes the legacy aircraft design methods from both GASP and FLOPS in the disciplines listed above, and implements them along with hand-calculated analytic gradients in the OpenMDAO framework. The section provides a brief discussion of these methods and how they are adapted and combined together in the Aviary tool.

A. Sizing and Weights

For sizing analysis, Aviary provides some built-in weight estimating relationships that have been fully differentiated. There are two different options for built-in weight estimating relationships in Aviary: one which has been ported over from the equations used in GASP that are documented by Hague et al. [20], and another which has been ported over from the equations used in FLOPS that are documented by Wells et al. [13]. The uniqueness of the weight and sizing

analysis in Aviary is not the origin of the equations, which are based on empirical relationships developed in the late 20th century. The uniqueness of the weight and sizing analysis comes from the combination of these two methods into one tool, and even more so from the hand differentiation of all the equations in both methods. Through this hand differentiation, Aviary allows analytic gradients across the entire weight and sizing routine.

The typical inputs to GASP for weight analysis are payload and weight trend coefficients, as well as geometry details. [2] Several simplifying assumptions were made in the development of the weight analysis in GASP, such as an assumption that component weights are proportional to certain vehicle dimensions. [20] The primary method of calculation of the aircraft weight in GASP is to compute the propulsion system weight from trend equations and scaling relationships (or to input it directly) and to compute the aircraft structural weight from regression/statistical equations of similar aircraft. [20]

The FLOPS weight equations are curve fits to the data of several transport and military aircraft (Aviary only includes the transport equations), and these curve fits were developed using optimization techniques to allow for predictive ability beyond the base data sets. [13] Several simplifying assumptions were made in the development of the FLOPS weight equations because they were based on existing aircraft data. However, FLOPS does include some routines to allow for calculation of the weight of unconventional wing types which fall outside of the training data set for the curve fit equations. FLOPS includes scaling factors for most computed component weights, which allows for a FLOPS model to be easily calibrated to produce a very close match to a specific aircraft with known component weights.

The equations from both legacy tools are isolated by subsystem (e.g., horizontal tail, avionics, etc.), and re-implemented in Aviary as OpenMDAO components. The subsystem decomposition allows the gradients to be implemented by hand calculation. In some cases the existing equations are modified slightly to use an activation function to switch between different calculation regimes smoothly instead of using a conditional branching statement like the legacy versions do. The purpose of these activation functions is to provide continuous derivatives across discontinuities or inflection points due to switching between calculation modes.

It should be emphasized that the equations for aircraft weight in GASP and FLOPS are not the same, and the weight modules from those codes are ultimately parameterized from a different set of inputs as well. In the process of bringing both weight estimation legacies together into one tool, Aviary adds the capability to use these different methods in harmony with each other. Aviary makes use of an option to select either a GASP-based or a FLOPS-based approach for weights, allowing for a combination of the two packages as needed. Either method can be used in conjunction with the Aviary code, allowing for combinations between the weight calculations of one legacy tool and other calculations (e.g., aerodynamics, mission) from another tool, an option which was not previously available. This and all the other subsystem methods provided in Aviary can also be augmented by external tools from outside of Aviary using the methods listed in Section IV.

B. Aerodynamic Analysis

Just like in the weight and sizing analysis, the aerodynamic equations from both GASP [21] and FLOPS [1] have been brought into Aviary, and they have been fully analytically differentiated to provide analytic gradients across the entire aerodynamics routine. Aviary has three different options for aerodynamic analysis: 1) user-input of tabular aerodynamic data, 2) user-connection of another aerodynamic analysis tool, and 3) built-in empirical aerodynamic equations. Option 1 is present because some types of modeling will perform aerodynamic design separate from aircraft design, and Aviary needs the ability to handle these situations. Option 2 is the primary mode which allows Aviary users to couple aerodynamic disciplinary design to aircraft design. Option 3 is the continuation of the legacy empirical aerodynamic estimating relationships from GASP and FLOPS.

GASP and FLOPS use very similar approaches to modeling the aerodynamic performance of the vehicle. In GASP, drag coefficients are calculated using empirical and experimental relationships as functions of vehicle geometry, and the base lift coefficient is calculated by assuming a constant lift curve slope. [21] Alternatively, a GASP user has the option to input lift and drag polars, which are used in place of the calculated values. In addition, GASP has the capability to add the effect of high lift devices into its aerodynamics calculations for low-speed flight. In FLOPS the user again has the option to input and scale drag polars, or FLOPS can alternatively calculate drag polars using a modification of the Empirical Drag Estimation Technique (EDET) [22] program. [23]

Aviary leverages the fact that both FLOPS and GASP have the capability to interpolate aerodynamic data based on provided polars and retains that as a core capability of Aviary aerodynamic analysis, taking advantage of the data interpolation techniques provided by OpenMDAO. Depending on whether a FLOPS-type coefficient buildup or a GASP-type coefficient buildup is used, Aviary will use either a third-order Lagrange polynomial or a second-order

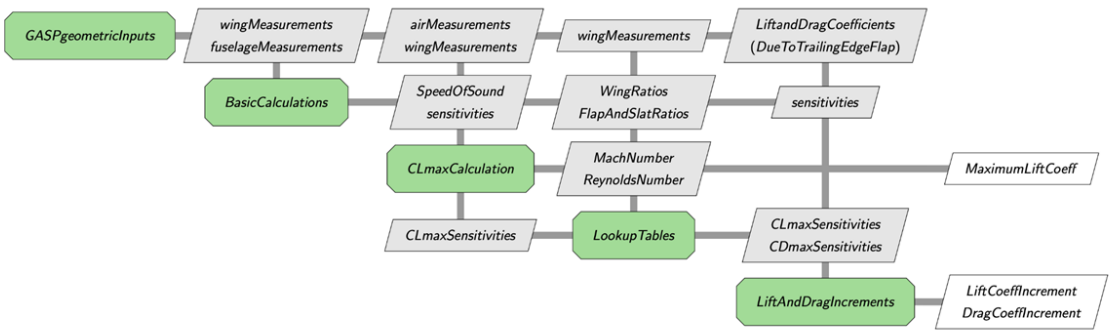


Fig. 1 The structure of the code for the high lift devices model gives a sampling of what many subsystem structures look like.

Lagrange polynomial respectively for the interpolation. Like the rest of Aviary, these OpenMDAO-provided Lagrange interpolation methods are fully differentiated.

The regression and empirical equations from FLOPS and GASP are re-implemented in Aviary as components with some modifications for smooth discontinuities as mentioned in the previous section. The analytic derivatives are hand-calculated and added to each component. For FLOPS, the fixed-iteration in the skin friction algorithm is implemented using a Newton solver from OpenMDAO. Again, similar to the weight analysis, the user selects either the GASP or FLOPS aerodynamic approach at the beginning of the analysis and that approach will be used throughout all flight phases.

One part of the aerodynamics module in Aviary that gives a good sampling of the internal Aviary subsystems is the capability to model high lift devices based on the legacy capability from GASP. The flaps model is a key component of the aerodynamics subsystem of the Aviary aircraft modeling tool. For this subsystem, geometric aircraft data, equations for 11 aerodynamic values, and equations for 19 sensitivity values are ported from GASP. These are used to develop a top-level flaps model in OpenMDAO with four components, which are connected by input and output variables to calculate final output values. There are three explicit components where inputs are added, partial derivatives are declared, and outputs are calculated using provided equations. Additionally, an interpolation component is added and an iterative loop between this interpolation and one of the other components is required (an OpenMDAO-based solver is used to converge this loop). This interpolation component uses linear interpolation and training data from GASP to calculate its outputs. As a result of the four components calculating values, three final outputs are computed: the increment in both the lift and drag coefficients and the maximum lift coefficient value. These variables show the effect that the flap has on the wing when it is deployed. The outputs of the model were verified using a unit test suite that will be discussed further in Section VI with GASP data for various flap types in various takeoff and landing configurations. Figure 1 provides a diagram of the data flow for this particular module. Figure 1 uses an eXtended Design Structure Matrix (XDSM) structure that follows the flow of variables through the program. For information on how to read an XDSM see Reference [24]. This closer look at an Aviary subsystem gives a representative example of how many of the subsystems were built up. Many features of this subsystem, such as the types of components used, the solver loop (the feedback loop between "LookupTables" and "CLmaxCalculations"), and the verification against legacy code data (not shown in figure), are common features of many Aviary subsystems.

C. Geometry

For the aircraft geometry definition, Aviary provides two built-in options: one based on the equations used in GASP, documented by Hague et al. [25], and one based on the equations used in FLOPS, documented by McCullers et al. [1] Similar to the weight and aerodynamic methods above, the geometry method in Aviary is unique because it combines two NASA-developed legacy methods into a single tool and employs hand differentiation to give analytic gradients.

In the development of GASP, several simplifying assumptions about goemetry were made, the most notable of which are a circular fuselage cross section and trend equations taken from aircraft existing in the 1970s that were used to determine tail surface geometry. [2] GASP uses two types of calculations, the first of which is to break up components into simple geometric shapes where feasible, such as approximating the main cabin as a cylinder. [25] For this type of analysis, the vehicle geometry is found using basic geometric equations and simplifying assumptions about wall thicknesses, etc. [25] However, some portions of the components can not be simplified into basic geometric shapes, and in these situations empirical coefficients and equations are used. [25] Similar to the GASP approach, the geometry in FLOPS is determined through a combination of simplifying assumptions, basic geometric relationships where possible, and empirical equations when basic geometry is insufficient.

The Aviary user can select to use the geometry analysis from GASP, FLOPS, or both. If both are selected then the user will need to decide which calculation method should take precedence for the few circumstances where both methods calculate the same variable. Sometimes it is necessary to select both methods based on what aerodynamic or weight method was selected and what inputs that method needs.

D. Propulsion Analysis

Aviary's propulsion subsystem is unique among the included subsystems because the way the propulsors are modeled is largely method-agnostic. At the highest level, propulsion is responsible for organizing all of the engines on an aircraft in the OpenMDAO problem. At this level, each engine model is a black box with a common interface which provides the engine performance and weight needed to size the system and model engine performance during the mission. These engine models are only responsible for modeling a single, physical instance of a propulsor on the aircraft. If the aircraft contains multiple interchangeable propulsors that use the same model, Aviary will automatically handle summing relevant values to system-level totals, such as mass, thrust, fuel flow, and others.

The default engine modeling method included with Aviary is a tabular-based engine model, where steady-state performance data created externally is provided. This capability is present because, despite the value often added by capturing the coupled nature of engine design with other disciplines, there are times when coupling the full engine cycle design with the aircraft analysis is out of scope for the design being performed.

Both FLOPS and GASP support engine modelling through externally provided data tables. The tables used by both codes are extremely similar - both formats consist of a three-dimensional table that provide engine performance data (such as thrust and fuel flow) at a given flight condition. [1] [26] This data is interpolated during mission analysis to cover the entire flight envelope. The independent variables for interpolation are always altitude, Mach number, and a control parameter. In the case of FLOPS, the control parameter is power code, sometimes referred to as pilot lever angle. Power code is a scale from 0-50 with specific values referencing important flight conditions. For example, 0 is engine idle, while 50 is max continuous power. For GASP, the control parameters most often used are power code which ranges from 0-50 or percent thrust which ranges from 0-100. T_4/T_2, or the ratio of turbine inlet temperature (T_4) to compressor inlet temperature (T_2), is also sometimes used as a control parameter in GASP, as is T_4, the turbine inlet temperature. Aviary supports conversion of these GASP and FLOPS data tables into a native Aviary format. The Aviary format consits of a column-based data table including a header. FLOPS tables are directly reformatted into this new data file with no numerical massaging necessary, but GASP values undergo some conversion to find and normalize T_4. For GASP engine files, Aviary normalizes T_4 into the throttle control parameter, however this value is not directly given in the data table and must be computed. It is computed by finding T_2 via the freestream stagnation temperature from current flight conditions in Aviary's atmospheric model and by using the known temperature ratio provided in the GASP engine data table to compute T_4. This conversion is represented by Equation 1. The normalized T_4 ranges in value from 0-1, where 0 is either idle or minimum thrust provided, and 1 is thrust at $T_{4,\,max}$. The actual T_4 values are also preserved in the converted data file. GASP engine decks are regular, structured grids, which means they typically contain extrapolated data. This data is often non-physical, such as points with non-physical negative thrust or turbine inlet temperatures over $T_{4,\,max}$. These points are removed during conversion to Aviary format, which assumes that all provided data is physically valid.

$$T_4 = T_2 * \frac{T_4}{T_2} \tag{1}$$

Aviary supports additional variables that are not included in one or both legacy code formats. Additional dependent (output) variables include electric power, shaft power, and rotor RPM, and an additional independent (input) variable is introduced: hybrid throttle. Hybrid throttle, as the name suggests, is a control parameter for hybrid-electric propulsors. In a hybrid engine, electricity and jet fuel are independent sources of energy whose rate of utilization can be independently varied. Therefore, a second control parameter is required to accurately model these types of propulsors. While hybrid throttle was created for this specific use case, its implementation is fully generic. Hybrid throttle can be used as an arbitrary second degree of control for any unconventional propulsor(s) the user wishes to model.

These engine performance tables are used by Aviary during mission analysis to estimate engine performance. This is done with a simple linear interpolation; however, users have the option to select any supported interpolator for OpenMDAO SemiStructuredGrid components on a per-table basis. Linear interpolation was selected as the default for a balance of speed and low average error across a variety of tested data tables, including four-dimensional tables that model hybrid-electric engines utilizing the hybrid throttle control. As explained earlier, OpenMDAO's interpolation schemes are fully differentiated, which means that Aviary's propulsion modeling method is also fully differentiated.

One of the original use cases of Aviary was to incorporate external engine design tools in order to perform coupled aeropropulsive optimizations. Therefore, it is also possible for users to provide their own engine models using the propulsion system interface. An example of using Aviary to couple to an external engine cycle analysis tool is a Transonic Truss-Braced Wing (TTBW) study performed in Aviary by researchers at NASA. [27] Aviary can accept a user-provided engine model wrapped using the subsystem builder interface described in Section IV. Engine models are essentially a special case of subsystem builder, and use an inherited class that performs some additional input checks specific to engine modeling. Engine models are loaded into Aviary separately from other user-provided subsystems. The propulsion subsystem requires engines to be loaded in this way so that subsystem outputs can be summed together to represent the aircraft system as a whole. User-provided engine models require the use of the more advanced Aviary interface layers which will be discussed further in in Sec VIII.

E. Mission Analysis

For mission analysis, Aviary retains the dynamic equations from the legacy aircraft analysis tools, but employs different numerical integration schemes on those equations in order to provide optimal control capabilities as well as analytic gradients. There are two options for Equations of Motion (EOM) in Aviary, including one based on an energy method approach [28] used in FLOPS and one based on a Two Degrees of Freedom (2DOF) approach used in GASP. The EOMs listed in Francisco et al. [28] and Hague et al. [29] are implemented in Aviary, but the integration approaches mentioned in those works are not employed. Instead, Aviary uses two different integration schemes which the user can select between and apply to either set of dynamic equations. The first integration scheme Aviary offers is a collocation scheme built on top of OpenMDAO that takes advantage of the capability for analytic gradients and is implemented in a software called Dymos. [30] The second integration scheme Aviary offers is a shooting with gradients method that also takes advantage of the capability for analytic gradients, and is implemented in a software called Simupy [31]. There are tradeoffs between both types of dynamic equations as well as between both types of numerical integration that will be discussed later in this section.

There are two options for EOMs. The first option is based on the energy method, which is an approach that models the dynamics of a vehicle without solving the full EOMs by modeling the vehicle as a point mass (no rotational freedom) and using a single parameter called energy-height which is the weight normalized sum of the kinetic and potential energies of the vehicle. [28] This is the approach that was used in FLOPS, and it is the lowest fidelity option for EOMs in Aviary. The second option is the 2DOF approach and is slightly higher fidelity than the energy method. It uses a typical two-dimensional free body diagram to derive the standard 2DOF velocity equations for vehicle motion, and these velocity equations are used as the EOMs. This 2DOF approach is the approach used in GASP. [29] Both approaches assume steady-level flight for cruise (although GASP has options for non-steady-level flight if the user chooses). [28] [29]

As mentioned above, the numerical integration methods available in GASP and FLOPS for these EOMs are slightly different from the integration methods used in Aviary. GASP uses a basic altitude or time stepping scheme and FLOPS uses a table generation approach to find the available conditions. [28] Aviary takes advantage of Dymos and Simupy as well as the ability to analytically differentiate equations in OpenMDAO and provides two different avenues for trajectory

modeling and optimization during design and analysis. The Dymos collocation approach approximates the trajectory with a set of polynomials that the optimizer drives to be coincidental and tangent at each end, and this approach allows for a more dynamic trajectory for each state variable but also adds more complexity than a simple integration approach. The Simupy shooting approach uses a gradient-based shooting method to approximate the trajectory. This method has the advantage that each optimizer iteration is physically valid and that specific stopping conditions can be more easily specified, but for larger models it is more computationally expensive than Dymos because it does not allow for the parallelization of computations on different points in the trajectory, a capability that Dymos provides. Aviary allows the user to have the option of selecting Dymos or Simupy as the integration method for their EOMs, and includes examples of both. Which method should be used depends on the problem being solved. As mentioned above, the collocation approach with Dymos makes the most sense for large or computationally expensive problems because it can be parallelized, but for situations where an optimization is not desirable or where ease of convergence is a priority the shooting approach with Simupy would be a better choice.

IV. Integration of External Tools

In addition to providing calculations and analytic gradients for the disciplines listed above, Aviary enables the integration of external disciplinary tools into its framework, and it smoothly handles any gradients that the tool provides. This allows for tight coupling to detailed disciplinary models. The integration of external tools is accomplished in Aviary using a standard interface developed to provide new variables and analyses to the underlying Aviary structure. The standard builder interface expects the external tool to be provided as an OpenMDAO system. This does not mean that the tool needs to be written in OpenMDAO, as long as it is wrapped as an OpenMDAO component so that it can provide its output and derivatives. Individual calculation groups can be specified for one or more of: pre-mission, mission, and post-mission calculations. The builder interface also allows the user to incorporate their subsystem into the overall design problem by specifying design variables or constraints, defining data connections between pre-mission and mission, and declaring controls or dynamic states to be used in the ODE. Figure 2 gives a visual representation of how an external subsystem can be plugged into OpenMDAO, and how a builder can use that tool to provide Aviary with the necessary information for execution. Apart from the standard builder that allows Aviary to call external tools, Aviary also includes a capability to extend its base variable set in case those external tools need to define additional variables that are not included inside of the Aviary codebase. Core Aviary has a set of variables that it uses, and these variables are defined as attributes of a class structure, with details described in variable metadata. Aviary itself only has a core set of aircraft variables used in the source code, and any variables beyond those are unknown to the software. External subsystems often require variables other than those provided in the core set, and so Aviary includes a provision for extending that core set to encompass the external variables. The Aviary user defines all of their new variables using the same type of class structure as the Aviary core, and similarly creates metadata for their new variables. Then they use built-in Aviary functions to combine their new definitions with the definitions in the Aviary core. That combined result gets passed to Aviary, where it is handled appropriately, thus smoothly integrating the variables of the new external subsystem.

One important feature of Aviary which allows for external tools to provide some but not all of the calculations otherwise provided by Aviary's built-in subsystems is variable overriding. This feature allows external subsystems to compute variables that would otherwise have been internally computed in Aviary, and to provide those variables to Aviary to be used instead of Aviary's internally computed variables. The internal workings of the Aviary code have been designed such that if an external subsystem provides an output which was previously calculated by an internal Aviary subsystem, the externally calculated value takes precedence and the internally calculated one is ignored. This allows for a partial adaptation of an Aviary internal subsystem to meet the needs of an unconventional type of vehicle. For example, when modeling a TTBW vehicle in Aviary, the user might prefer to provide an external value for the weight of the wing because it is an unconventional wing type, but they might want to use internal Aviary calculations for all of the other variables. In this case the user could connect an external subsystem that calculates only the wing weight and use the output of that subsystem to override the wing weight computed by the internal Aviary subsystem, while still keeping all the other weight computations from the internal Aviary weight subsystem. This type of overriding is possible with any Aviary variable and allows greater flexibility in the use of external subsystems. Below are some practical examples of prior cases where external subsystems have been integrated into Aviary to enhance aircraft models and take advantage of Aviary's derivative handling and variable overriding capabilities.

The coupling of Aviary with external subsystems has been demonstrated by performing a sizing and trajectory optimization on a TTBW. [27] The external geometry is modeled in Open Vehicle Sketch Pad (OpenVSP) [32] [33],

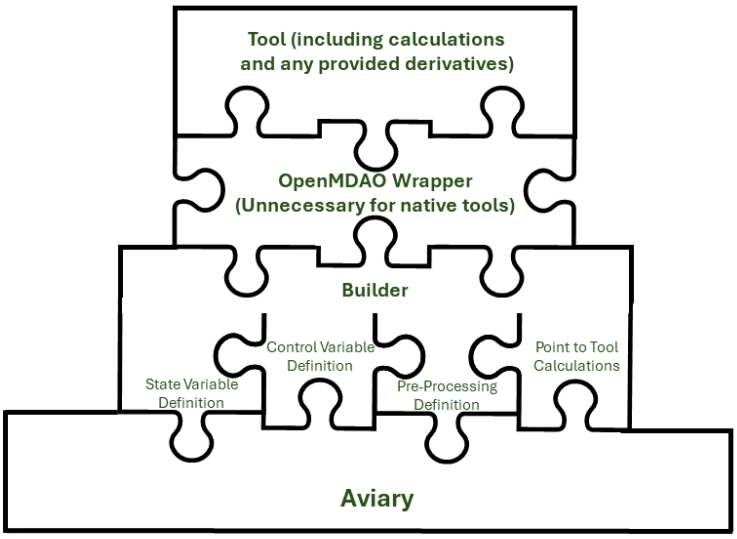


Fig. 2 A visual depiction of the connections that allow a builder to provide Aviary with the necessary information (what variables are states and controls, what preprocessing is required, and where the external tool calculations can be found in the OpenMDAO code) to run an external subsystem.

which is wrapped using its Python Application Programming Interface (API) as an external builder. This adds a subsystem to the pre-mission that runs OpenVSP to compute detailed geometry parameters based on a simpler parameterization. These parameters take precedence over Aviary's internally-computed parameters through the use of Aviary's overriding system. For aerodynamics (lift and drag) along the trajectory, Vehicle Sketch Pad for AERODynamics (VSPAERO) [34] is also wrapped in an external builder that adds a subsystem to the pre-mission. This subsystem includes an OpenMDAO subproblem that can generate a lift and drag polar by running VSPAERO over a fixed grid of mach numbers, altitudes, and angles of attack. The polars are passed as OpenMDAO variables into the mission phases, where the internal tabular aerodynamic interpolation capabilities are used on these polars to produce the lift and drag coefficients at each point in the trajectory.

A further demonstration of how external subsystems have been integrated into Aviary is in the same TTBW model, and an external propulsion system and a motor plus battery are integrated. The propulsion system is designed in pyCycle [35] [11], a Python-based cycle analysis code that is already written in OpenMDAO (which makes integration even easier). This system's direct integration allows the engine to be re-sized during the optimization by changing the mass-flow rate at cruise, rather than re-scaling an engine table. pyCycle also has analytic gradients throughout, and Aviary is able to use these analytic gradients in the optimization to take advantage of the speed gains addressed above. Similarly, the motor plus battery model is a Python-based OpenMDAO component, not natively packaged with Aviary but including analytic gradients. It containes several maps and differential equations to describe how the motor and battery system draw down battery charge over time and inject power onto the low-pressure turbine shaft. Both pyCycle and the motor plus battery models are incorporated into Aviary by creating individual Aviary builders for each of those subsystems. Those builders, which expose all the inputs and outputs of those systems, enable Aviary to understand how to connect them to the larger TTBW model. The coupling of external subsystems into this TTBW model in Aviary is not simply an academic exercise but also demonstrates that Aviary can be used to gain new insight about a vehicle design. The most notable discovery that Aretskin-Hariton et al. found was to identify the exact mission defined sizing point of the engine. [27] They found that a constraint on the minimum required available rate of climb during cruise is the driving force behind the minimum size of the engine in the hybrid electric aircraft, and they identified that allowing the engine to draw electric power at this point to satisfy this requirement can reduce the overall required engine size. [27] While both of these subsystems are Python-based, the method for incorporating non-Python-based external components is the same (with the potential added step of wrapping the subsystems in OpenMDAO).

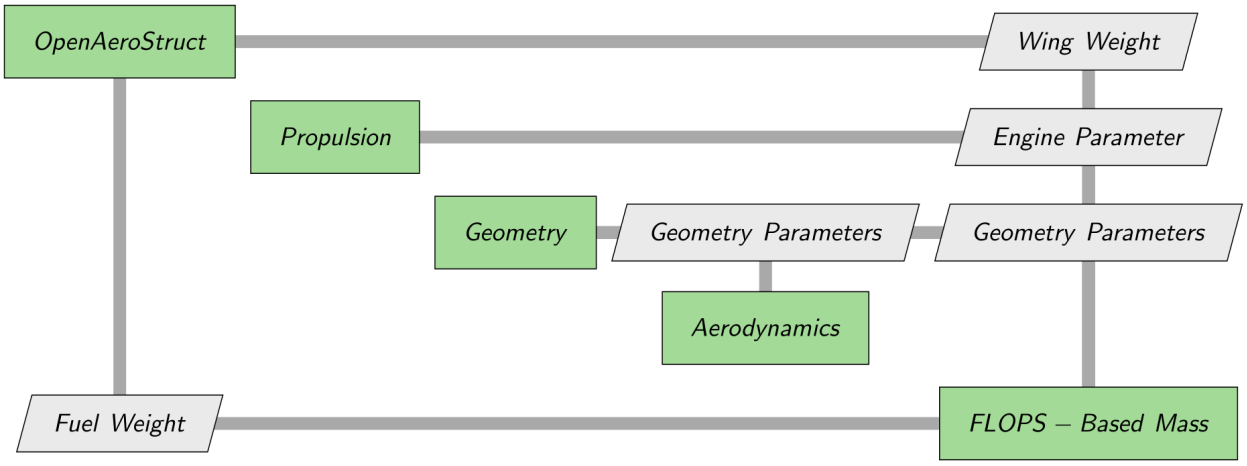


Fig. 3 The data flow of the pre-mission subsystem for an example of integrating an external wing weight subsystem into an Aviary model.

A final example of coupling external subsystems to Aviary is a sample problem which lives on the Aviary github website and is accessible a sample to the public for the purpose of demonstrating how to integrate an external model into Aviary. This example is another instance where a higher fidelity structural analysis code is integrated as an external subsystem in Aviary to introduce higher fidelity into the model. For this particular example the aero-structural code OpenAeroStruct (OAS) [36] provides more detailed calculations for the wing-weight based on the weight of the fuel that the wing was carrying. In this setup the external subsystem builder is configured to call OAS during pre-mission analysis (see Section V, and provides the Aviary variable of fuel weight as an input to OAS which calculates the weight of the wing necessary to carry that fuel and returns that weight to Aviary for use in the remainder of the analysis. See Figure 3 for an XDSM of the pre-mission subsystem in this example. This is a somewhat simple implementation of the external subsystem capability in Aviary, but it demonstrates the vital role that external subsystems can play in Aviary models when an unconventional aircraft such as a TTBW or blended wing body (BWB) is being modeled.

There are several other tools beyond these examples which have been used to couple external subsystems to Aviary. A few of the tools which have been used are Toolkit for the Analysis of Composite Structures (TACS) [37], Aircraft NOise Prediction Program 2 (ANOPP2) [38], and Probabilistic Technology Investment Ranking System (PTIRS) [39], which perform structural analysis, acoustic analysis, and cost analysis respectively.

V. Structure of the Disciplines

Aviary is structured into three main blocks that allow the disciplines listed above and/or any external subsystems to execute at the proper time. Figure 4 shows a graphical characterization of how the subsystems are arranged within the blocks. In observing Figure 4 it is important to note that the mission block does not fully show the Ordinary Differential Equation (ODE) and integration methods, and instead it is just notional to show subsystem names and variable identities. The first block represents pre-mission analysis, and this section executes before the mission analysis portion of the code. This section contains all the elements of analysis which are necessary for the mission analysis but do not depend on time (and thus are fixed during the mission analysis). In a standard Aviary problem with no external subsystems, pre-mission typically includes weight analysis, generation or setup of provided aerodynamic data, geometry analysis, and setup of provided propulsion data. The second block is mission analysis; this is the portion of the code that models everything that changes in time while flying the mission, primarily the dynamic motion of the vehicle. Here a standard Aviary problem with no external subsystems would typically include dynamic calls to the aerodynamics and propulsion data, as well as the analysis and numerical integration of the equations of motion that make up the mission analysis. The final block of Aviary is post-mission analysis, and this is somewhat of a counterpart to pre-mission analysis. It too contains analyses which do not change in time, but it typically only involves calculations that do not affect the mission analysis, such as cost estimation, acoustic analysis, and other types of post-processing. This is not a strict hierarchy because there are situations where results from post-mission analysis are fed back into mission analysis in a coupled manner, but that usually occurs in a more complex optimization and the feedback usually occurs through constraints as opposed to direct coupling, which would require a solver loop around the mission. The standard Aviary model with

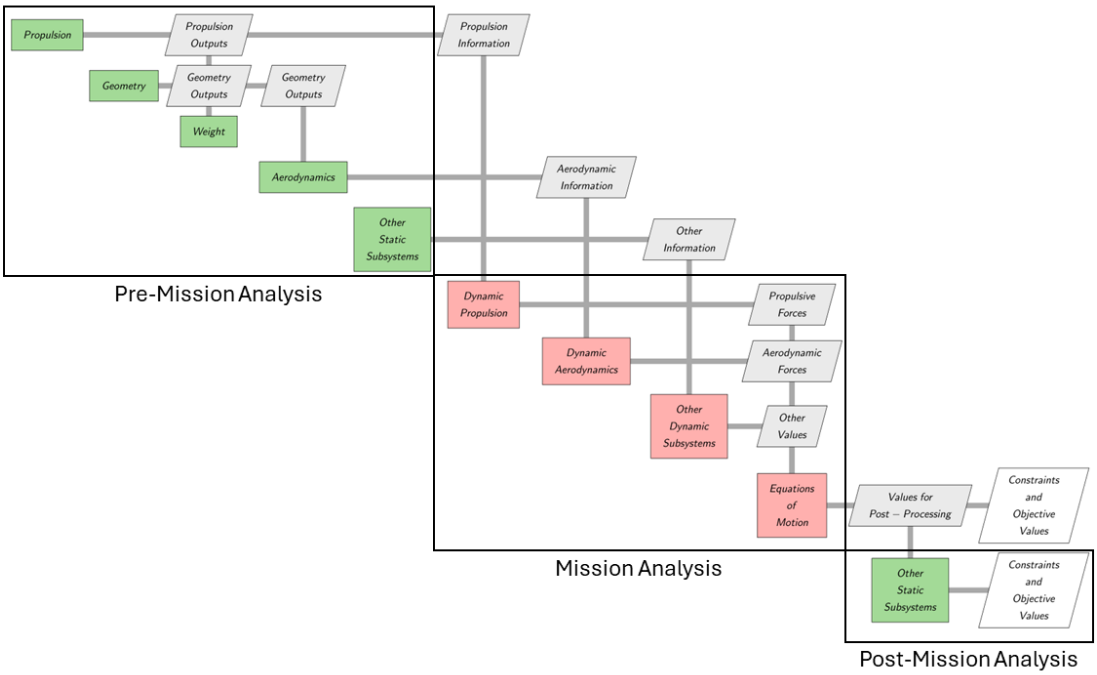


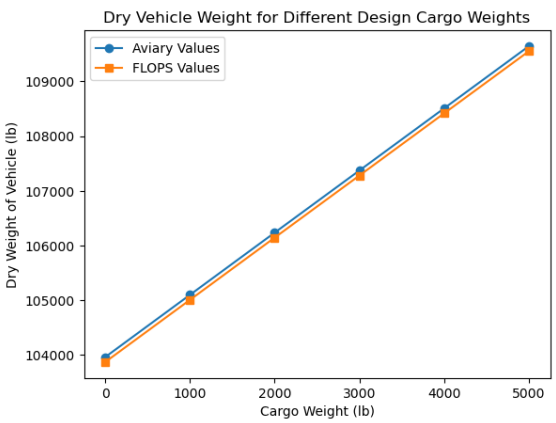
Fig. 4 The general layout of subsystems within the pre-mission, mission, and post-mission sections of Aviary.

no external subsystems doesn't have any computations in the post-mission analysis block, so this block exists purely for the use of external subsystems. An external subsystem can be inserted into any of these blocks or can provide contributing subsystems to multiple blocks (e.g., both internal aerodynamics and propulsion subsystems are used in both the pre-mission and mission blocks).

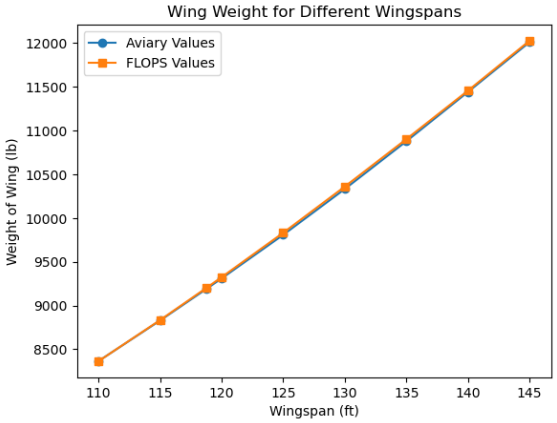
VI. Verification

Verification of Aviary has been performed in a few different ways. The primary method of verification in Aviary is continuous unit and integration testing on the code. The tests can be divided into unit tests and benchmark tests. Unit tests can be further subdivided into method, component, and subsystem unit tests, each of which verifies that the method, component, or subsystem is functioning as expected. Integration testing (known in Aviary as benchmark testing) is a higher level of testing that is conducted on full vehicle models, and the results from those models are compared both with previous Aviary results as well as with results from runs of the same vehicle in GASP and FLOPS. No code is permitted to be contributed to Aviary unless all unit and benchmark tests are passing. Tables 1 and 2 summarize the tests and tolerances which the new code is required satisfy. Where a computation is involved, the truth values are generated using the legacy codes GASP and/or FLOPS in most cases and generated by hand or by regression analysis for the portions of the code that were not included in GASP or FLOPS. In a few cases the original truth data from GASP and FLOPS has been manually updated to account for an intentional change in the code. These cases are all minor updates to modernize the code, such as updating the flight attendant weight estimation to be the weight of the average flight attendant instead of the weight of the average woman. In addition to value computations the derivatives for every component are also tested versus complex-step derivatives using built-in capability in OpenMDAO. The N3CC aircraft mentioned in the table is a commonly used NASA aircraft model that represents a single-aisle aircraft with advanced technology assumptions estimated for a 2030 entry into service.

Verification has also been performed to verify that Aviary behavior matches GASP and FLOPS for a growing selection of aircraft models. While the differences in Aviary's implementation – in particular, the mission integration methods, table interpolation algorithms, and trajectory optimization formulation – make it more difficult to compare the optimization results, subsystem values can be directly compared through subsystem unit testing. In particular, the mass, aerodynamics, geometry, and propulsion values can be checked against the legacy codes. Additionally, design studies can be compared through benchmark testing and relative trends should match even if specific values such as fuel burn are different.



(a) Dry Vehicle Weight



(b) Wing Weight

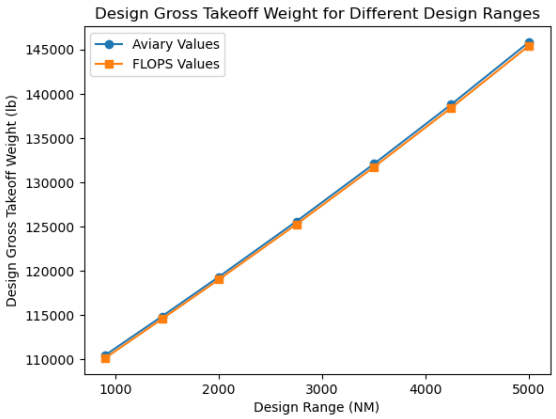
Fig. 5 A comparison of the weight calculations in Aviary (calculated during the pre-mission portion) as a function of cargo weight and wingspan shows that Aviary has good agreement with the legacy FLOPS code.

As an example of the verification performed in Aviary, Figures 5 and 6 show a comparison between Aviary and FLOPS from a few trade studies where an aircraft was independently sized and optimized over a series of different parameter values. This is an example of the benchmark testing performed on Aviary. Figure 5 shows a sweep of the aircraft dry weight while the weight of the cargo (not including passenger baggage) is varied, as well as a sweep of the dry weight while the wingspan is varied, for both FLOPS and Aviary. The slight differences observed in the case of the dry vehicle weight as a function of the cargo weight are primarily due to the differences in the mission integration method between the two tools (Aviary vs. FLOPS). Likewise, Figure 6 shows a comparison of four primary tool outputs as a function of the design range of the vehicle. All of the values in Figure 6 are design values, and again, the slight differences observed are primarily due to differences in the way that the two tools perform their mission analysis integration. To reproduce the legacy results, Aviary was artificially constrained to use a fixed cruise altitude at 35000 feet which mirrors the FLOPS case. In reality, Aviary is capable of optimizing these missions for a different cruise altitude with better performance.

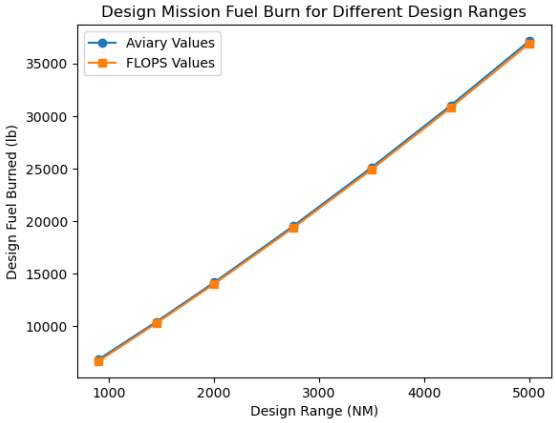
Table 1 A summary of the benchmark test cases included in Aviary that are compared against legacy tools' results for verification. Comparison method is similar to the example detailed above.

Aircraft Type Used	**Mass Estimation Method**	**Mission Analysis Method**	**Test Case Notes**
Single-aisle transport	GASP-based	FLOPS-based	Regression data only, no matching legacy case exists.
Single-aisle transport	FLOPS-based	FLOPS-based	
Single-aisle transport	GASP-based	GASP-based	Initial version based on GASP data, code updates led to this now being regression data only.
Single-aisle transport	FLOPS-based	GASP-based	Regression data only, no matching legacy case exists.
N3CC	FLOPS-based	FLOPS-based	Balanced field length analysis
N3CC	FLOPS-based	FLOPS-based	Sizing mission, regression data only.
N3CC	FLOPS-based	FLOPS-based	Detailed takeoff simulation
N3CC	FLOPS-based	FLOPS-based	Detailed landing simulation

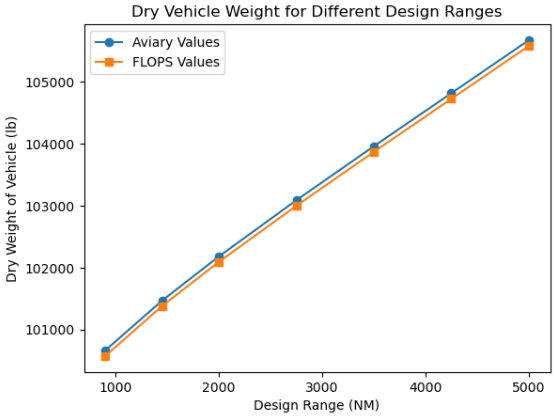
A summary of the benchmark test cases included in the Aviary repository is shown in Table 1. The tolerances used in the benchmark test cases vary slightly because, as was seen in Figure 6 and Figure 5, there are some differences between Aviary and the legacy tools that are especially apparent at the vehicle level. In addition, some cases in Aviary cannot be run in the legacy codes, like the cases where a combination of GASP- and FLOPS-based subsystems are used. Because of these differences, the tolerances used on the benchmark test cases listed in Table 1 are around 1 percent (with



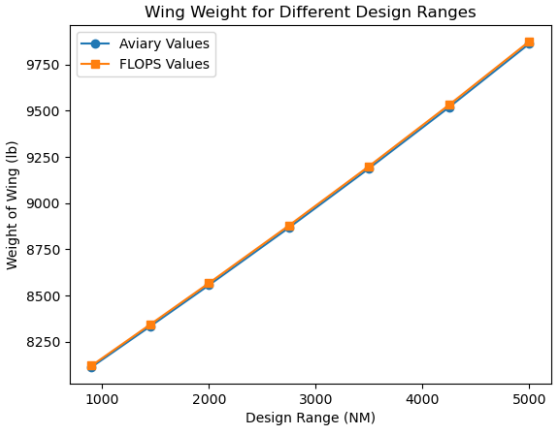
(a) Gross Takeoff Weight



(b) Design Fuel Burn



(c) Dry Vehicle Weight



(d) Wing Weight

Fig. 6 A comparison of several outputs of Aviary over different design ranges to verification data from FLOPS shows that there is good agreement between Aviary and the legacy tool. These results were obtained in Aviary at a fixed cruise altitude that was different from the optimal altitude, resulting in the slightly heavier weights in Aviary.

slight variation due to the differences mentioned), and in some cases (noted in the table) the truth data used is regression data instead of legacy data. Additional cases were run during the development of Aviary but cannot be released with the open-source tool as they contain proprietary information.

A summary of the individual subsystem unit tests is shown in Table 2. Table 2 summarizes the unit tests performed on the pre-mission subsystems within Aviary as well as the unit tests performed on the individual EOMs for the mission portion of Aviary. The table shows that some of the tests are run to tighter tolerances than others. The reason for the tighter tolerances in some cases such as the EOMs is that these systems are less complicated and involve less departure from the legacy code (departures would be things such as variation in interpolation methods, different solver tolerances, etc.). Conversely, the reason for looser tolerances in the aerodynamics subsystems is that these systems include more deviations from the legacy codes, especially in the interpolation methods. In the case of FLOPS-based geometry subsystem the loose tolerance is primarily due to roundoff error in the legacy code data. It is impossible to include all of the component and method unit tests in the Aviary codebase here because Aviary contains around 220 components and each individual component is tested, resulting in hundreds of tests. However, component level tests are also similarly performed on each component that supports the subsystems listed in this table. In general the component unit tests will have tighter tolerances than the subsystem tests because they do not have to account for differences between Aviary and GASP or FLOPS from sources such as solver tolerance discrepancies. Most of the component unit tests in Aviary have

Table 2 A summary of the subsystem unit test cases for each subsystem included in Aviary that are compared against legacy tools' results for verification.

Subsystem Tested	Unit Test Tolerance	Derivative Test Tolerance	Test Case Notes
GASP-based aerodynamics	1.5e-3	N/A	Derivatives only checked in the component unit tests (same for FLOPS aerodynamics).
FLOPS-based aerodynamics	5e-3	N/A	Some extrapolated points require looser tolerance due to difference in interpolation methods.
GASP-based geometry	5e-4	1e-9	Some values slightly altered to account for intentional changes in the code.
FLOPS-based geometry	1e-2	1e-6	Does not test canard features. Loose tolerance primarily due to roundoff error in FLOPS data.
GASP-based mass	5e-4	3e-9	Some values slightly altered to account for intentional changes in the code.
FLOPS-based mass	1e-4	1e-10	
Propulsion	1e-10	1e-10	This tests the scaled sea-level static thrust from pre-mission.
Height Energy EOMs	1e-9	1e-12	
2DOF EOMs	1e-6	1e-12	Takeoff EOM not tested against GASP due to numerical integration differences. These cases are covered in integration testing. Limited climb, cruise, and descent data points are available from GASP so these points were calculated by hand and are now regression tested.

tolerances in the range of 1e-4 to 1e-7, where 1e-4 is for cases where there is not very precise legacy data and 1e-7 is for more standard cases. A few of the component unit tests have tolerances as tight as 1e-12 in situations where legacy data is not available and the test data was calcualted by hand.

VII. Computational Speed of Derivatives

One of the most important features of Aviary is the inclusion of analytically-defined derivatives throughout the entire tool. These derivatives enable efficient gradient-based optimization of complex aircraft systems by providing accurate design sensitivities at a low computational cost. If Aviary were to use gradient-based optimization methods but not provide analytic derivatives of the problem, it would be necessary to compute the total derivatives of the problem using an approximation method such as finite differencing, complex step, or automatic differentiation. To quantify the impact of these derivatives, below is a timing and accuracy study for a nominal Aviary mission-design problem. The derivatives computed by Aviary are compared to those obtained by finite-differencing across the entire optimization problem.

Although finite differencing is very low effort for the developer, it suffers from being computationally expensive (especially for problems with large numbers of design variables) and having lower accuracy than other forms of differentiation [10]. The complex-step method is much more accurate but still scales linearly with the number of design variables and requires the entirety of the code to be complex-safe [10], which is not always feasible. Within Aviary, there are multiple uses of NumPy and SciPy methods that are not complex-safe and would require rewriting portions of the code to allow the use of top-level complex-step.

Automatic differentiation techniques that use tools such as JAX [40] could help obtain derivatives, though it may require special consideration when designing the tool to make it possible to be automatically differentiated correctly. Specifically within Aviary, arrays are often modified in-place, there are hierarchical solver loops, and there are other computational techniques that are not immediately safe for automatic differentiation. Because of these considerations and the complexity of the Aviary codebase (it is not complex-safe or able to be easily automatically differentiated), this study focused on a comparison the analytic derivatives to finite-differencing. In the future automatic differentiation may

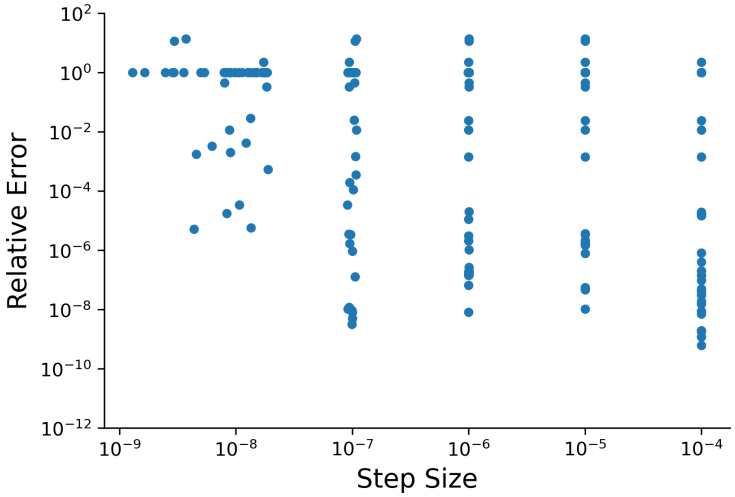


Fig. 7 The relative errors for the entries of the total derivatives (the problem-level Jacobian) vary based on the finite difference step size used. The accuracy is problem-dependent and for this particular problem the optimal finite difference step size appears to be between 10^{-4} and 10^{-7}.

be able to be added to this study and would be a more efficient option than finite difference.

The optimization problem used in this study minimizes the fuel burn for a single-aisle commercial transport aircraft flying a three-phase (climb, cruise, and descent) mission where the mission definition can be optimized. This aircraft is a generic example included in Aviary and has 169 passengers, a gross mass of 175,400 lbm, and a wingspan of 118 ft. The Sparse Nonlinear OPTimizer (SNOPT) optimizer was used for this case [41]. This example is typical of a standard Aviary problem using no external subsystems and it includes all the standard Aviary subsystems listed in Section III (FLOPS-based origins were used for all). In addition to the mission definition being optimized, the optimizer can control the gross mass of the aircraft, the wing aspect ratio, and a parameter that scales the size of the engine. This combination of aircraft design variables and trajectory optimization is representative of a typical Aviary problem, and these design variables show a simple notion of the multidisciplinary trade-offs for which Aviary can optimize.

Before using finite differencing in a production environment, it is important to perform a step-size study to determine the step size that provides the most accurate results [10]. Figure 7 shows the results of a step-size study for the problem described in the prior paragraph. The Aviary model was queried using a range of step sizes from 10^{-4} to 10^{-9} and a plot of the relative error (compared to the analytic derivatives) was generated for each entry in the total derivative Jacobian needed for the gradient-based optimization method. Figure 7 shows that the best step size is between 10^{-4} and 10^{-7} and that a large amount of accuracy is lost at step sizes below 10^{-7}. This accuracy loss at very small step sizes is due to the Aviary model not being sensitive to such small perturbations; the bunching of total derivatives with a relative error of 1.0 show cases where the approximated derivatives are erroneously 0. For the remaining studies in this section, a step size of 10^{-6} was used for consistency.

The time taken to compute the total derivatives of the problem – sensitivities of all functions of interest (objective and constraints) with respect to the design variables – for both the analytic and approximated derivative computation methods was recorded for two points in the optimization. The first point is the initial optimization iteration, and the second point is the design point found by the optimizer (although this could only be recorded when using analytic derivatives because the optimizer did not converge with finite difference derivatives). The reason for the comparison of computation speed at two different points in the optimization process is that there may be different computational costs associated with the derivative calculations depending on where in the design space the optimizer is. To ensure properly representative results this process was repeated 10 times for each method and design point, and the results are presented below.

Figure 8 compares the time to compute the total derivatives at a single point in the design space for both the analytic and approximated derivative computation methods. It shows that Aviary computes the analytic total derivatives of this optimization problem 13x faster than the finite-differenced approximation method. The relatively large cost of the finite difference approximation method is because this method repeatedly calls the full Aviary model with different perturbations in the design space whereas the analytically-computed derivatives do not need to do this. These comparisons were executed on a workstation with a 3.6 GHz AMD 3700X processor with 16GB of RAM. The results

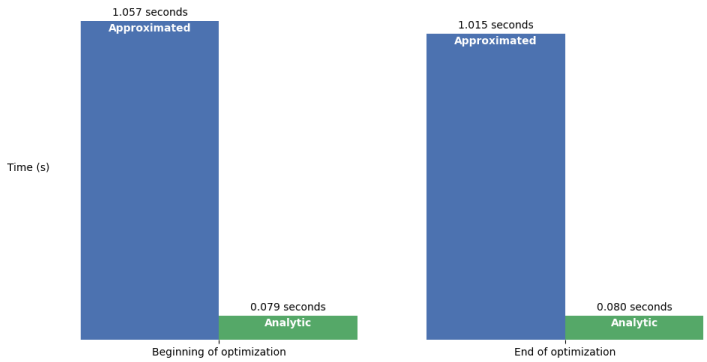


Fig. 8 The cost of computing the total derivatives for a nominal problem using the analytic derivatives provided by Aviary is 13x less than the cost of using finite differencing. This cost does not change greatly when comparing different points in the design space for this particular problem.

also show that computational costs do not change meaningfully between the initial and final points in the optimization. The subsystems included in this optimization problem are low fidelity empirical-based analyses; with higher fidelity analyses included, such as physics-based aerodynamics or propulsion subsystems, the difference in wall clock time would be more stark, with potential implications for the number of included disciplines or other factors due to computational resource limitations. Additionally, the timing values shown in Fig. 8 are for one total derivative computation. A complete optimization computes the total derivatives at dozens or hundreds of points in the design space, leading to a dramatic speedup when using the analytically-provided derivatives.

Beyond just computational cost improvements, analytic derivatives can offer substantial accuracy improvements relative to finite-differenced derivatives. This is because the finite difference approximation suffers from truncation error at relatively large step sizes and subtractive cancellation at small step sizes [10]. (Again it is worth noting that complex step and automatic differentiation derivatives are also more accurate than finite difference, however, as mentioned above those methods cannot be easily employed here because they would require all calculations to be complex-safe and safe for automatic differentiation.)

Figure 9 shows the percentage error between the approximated derivatives computed by finite differencing and the analytic derivatives provided by Aviary. For simplicity, only the Jacobian pairs where there is a non-zero error in the approximation are shown. In many cases, the finite difference approximation provides adequate accuracy predicting the derivatives. However, derivatives associated with the throttle constraints, engine scaling parameter, and mass defect constraints are subject to unacceptable inaccuracy. Specifically, the finite-differenced derivatives associated with the throttle constraints incorrectly show no sensitivity to the engine scaling parameter or phase durations. This is because the perturbations used in the finite difference method are too small to register any effect on the throttle constraints with respect to the phase duration length or scaled engine thrust parameter. Step size investigations may have revealed an appropriate step size where these sensitivities could be captured, however, the purpose here is simply to demonstrate the issues with finite difference derivatives. These inaccurate derivatives lead to a poorly behaved optimization problem, and in fact the optimizer (SNOPT [41]) does not successfully converge when using the approximated derivatives. In this case, the optimizer takes approximately three major steps before being unable to proceed due to the inaccurate Hessian approximation used to determine the next design to query.

These studies comparing the accuracy and speed when using the analytic and approximated derivatives show the benefit of using the provided analytic derivatives within Aviary. By providing partial derivatives for each of the subsystems included in Aviary, OpenMDAO is able to efficiently compute total derivatives of complex mission-design problems. The accuracy and relatively low computational cost of these derivatives allow for more robust exploration of complex aircraft design spaces. The results shown here are for a relatively simple design and trajectory optimization case, but when using higher fidelity analyses within the optimization loop, these derivative cost savings are much more pronounced and allow Aviary to solve problems that would be intractable without accurate and efficient derivative information.

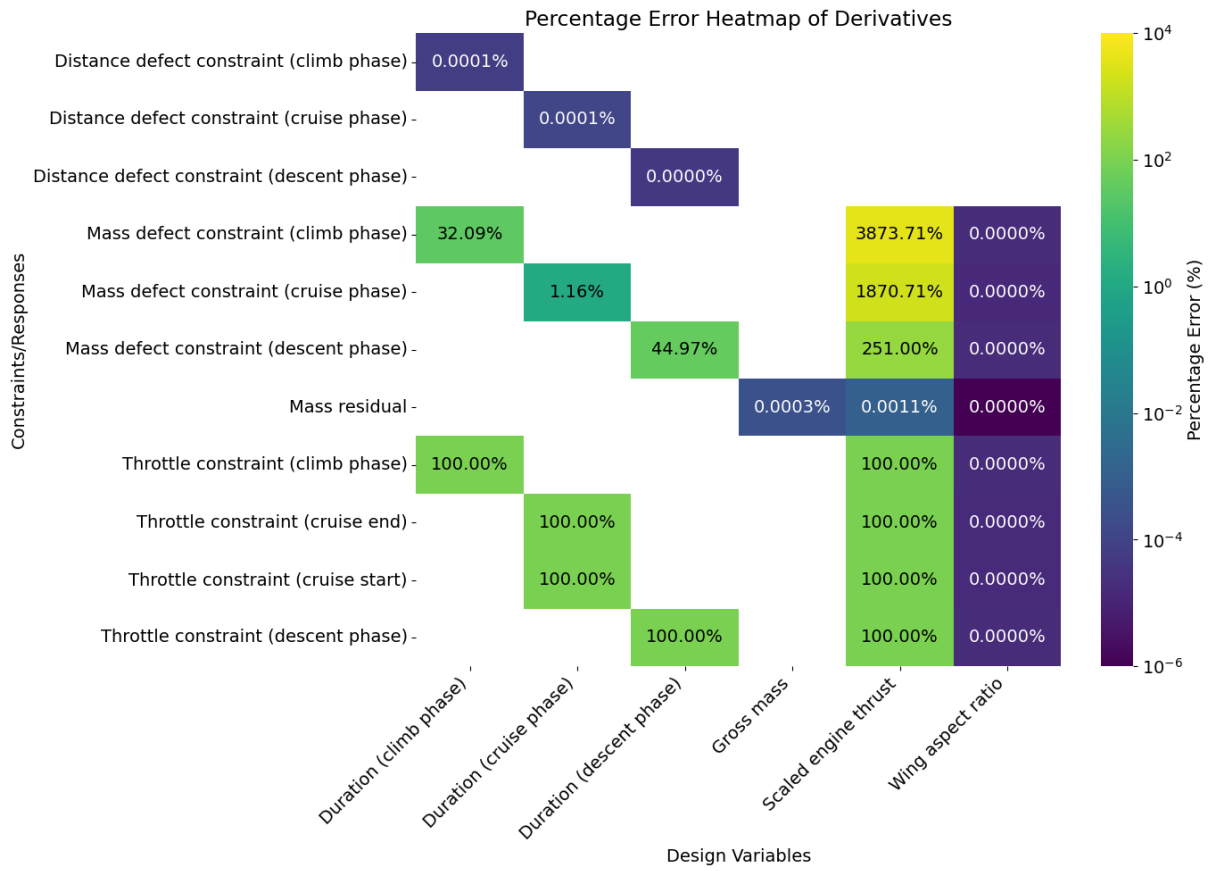


Fig. 9 The finite-difference approximated derivatives are inaccurate for some Jacobian entries which leads to poor optimization convergence as the optimizer does not know how to progress through the design space.

VIII. User Experience and User Entry

Aviary distinguishes different methods of user entry in order to enable flexible use of the tool. As mentioned in Section I, the goal of Aviary is to provide an aircraft MDAO tool that allows for computationally efficient tight coupling of disciplines in a relatively accessible manner. A significant part of how Aviary meets that goal is by providing different levels at which an Aviary user can access the code based on their level of expertise with MDAO and the various underlying software as well as based on the needs of their model. This section discusses how Aviary has broken the user-interface up into three different levels (level 1, 2, and 3) which can be selected from based on the complexity of the model being developed. The higher the level chosen the more complex of a model the user can build, but the interface to build that model will be more complex as well.

Level 1 is specifically targeted towards users who are new to OpenMDAO and/or aircraft design, users of legacy tools such as GASP and FLOPS, and users who do not need to couple external subsystems into Aviary. These users will only need the basic functionality of Aviary in order to replicate the abilities of the legacy tools they are used to. In level 1 users will define the aircraft and mission properties through a text-based csv file, instead of requiring Python code. A command line interface is the primary way to access Aviary at this level, although it is also possible to use a call to a singular Python function for slightly more flexibility. Level 1 is the simplest way to use Aviary, but it is also the most limited.

Users of legacy FLOPS and GASP tools may want to convert their previously designed aircraft into Aviary models. To assist with this, Aviary contains a *fortran_to_aviary* converter. This adds the ability to convert FLOPS and GASP input decks (FORTRAN Namelists) into Aviary csv input files using the mappings found in the *historical_name* section of the *variable_meta_data*. The resulting csv is automatically sorted into three sections: Input Values, Initial Guesses, and Unconverted Values. Input Values are any FORTRAN variables that were mapped to input variables in Aviary components. Initial Guesses are used to initialize the trajectory to improve the reliability of convergence. These are displayed separately from the Input Values because they will not be passed directly to components. Finally, Unconverted

Values are any remaining FORTRAN variables that could not be mapped to an Aviary variable. Some variables are expected here because they are unused by Aviary, such as flags that would affect the behavior of the FLOPS or GASP code execution and intermediate variables that are not needed in Aviary, while others are necessary values that do not have a one-to-one match with an Aviary input variable, which is common for mission definition inputs. It is recommended that users check this section after conversion to ensure that there aren't any unexpected variables here. This ability to convert GASP and FLOPS input decks significantly reduces the time needed for analysts to convert and use legacy models. This capability is most often used for level 1, however, the resulting csv input files can be imported and used at any level of the Aviary interface.

Level 2 is an intermediate level that allows more flexibility than level 1 as well as requiring slightly more familiarity with Aviary and/or OpenMDAO than level 1. This level is typically used for situations where an external subsystem needs to be coupled to Aviary or the user needs greater control over which processes are executed in what order in Aviary. At this level the user will be developing run scripts using a coding interface instead of simply a csv and command-line interface, but the specific coding interface they use has been formatted to abstract away as much as possible of the OpenMDAO-related code and to simply provide the necessary information. At this level users can perform operations such as control the initial and final states of each flight phase more easily, as well as add external subsystems. This level is fairly flexible but is limited to standard Aviary behaviors. Non-standard behaviors may require a switch to level 3.

Level 3 is the most flexible of all the access levels, but this flexibility requires the most user knowledge. At this level the user again will be building their run scripts in Python as opposed to simply using Aviary's command-line interface, and here the user has ultimate control. They can manually modify the ways that pre-mission, mission, or post-mission groups are set up, and they can also make under the hood modifications to allow external subsystems to interact with Aviary in new and non-standard ways. This level of access is not required for most Aviary problems except for the very most advanced, however, it is a good option for novel concepts that Aviary does not have the capability to accomplish by default. A quote from the Aviary documentation reads, "At this level, users have full access to the Python and OpenMDAO methods that Aviary calls. They can use the complete set of Aviary's methods, functionalities, and classes to construct and fine-tune their aircraft models. Level 3 enables users to have supreme control over every aspect of the model, including subsystems, connections, and advanced optimization techniques."[42]

One of the most important resources for Aviary users at any level is the extensive documentation written by Aviary's developers and available online. This documentation covers in detail how to interact with the Aviary code as a user, and a suite of docs is being developed for interacting with the code as a developer as well. Reference [42] points directly to this documentation, and is the starting point for all Aviary users. Several examples of the different user interface levels exist in the Aviary code and documentation, and an onboarding guide within the documentation walks through a use of each of the three levels.[†]

IX. Summary

Two legacy codes, General Aviation Synthesis Program (GASP) and Flight Optimization System (FLOPS) are preserved in Aviary by retention of their calculation methods and analytic differentiation of their equations. Aviary provides the option to select from either of their calculation methods for weights, aerodynamics, geometry, and mission analysis, and also retains their capability to interpolate propulsion data, while adding on the capability to have additional independent engine control parameters. Aviary has updated GASP and FLOPS mission analysis methods for use in a gradient based optimization environment by retaining their equations of motion and physics while using new integration techniques.

Aviary creates the ability to integrate external analysis tools into the aircraft design and analysis through the use of the external subsystem builder, and allows these external tools to provide any gradients they may have developed. Aviary structures the external tools and the internal computations alike into a three block structure of pre-mission, mission, and post-mission analysis, and uses this structure to model the aircraft, allowing for a smooth combination of Aviary subsystems and external subsystems.

The capabilities provided by Aviary need to be checked against the legacy tools that Aviary is built on. To accomplish this the calculations in Aviary are verified against GASP and FLOPS with unit testing on individual code blocks and by integration testing (benchmark testing) on entire models. The results are matched to GASP and FLOPS data, and discrepancies are explained to be the result of known intentional changes in Aviary (new interpolation and integration methods, etc.). It is also important to examine the usefulness of the analytic differentiation capability that Aviary offers, and a comparison of these derivatives to finite difference demonstrates that the analytic derivative calculations in

[†]https://openmdao.github.io/Aviary/getting$_s tartedonboarding.html$

Aviary perform significantly faster than finite difference methods, although complex step and automatic differentiation methods cannot be employed on some parts of the code base as it stands now. Finally, the capabilities of Aviary must be accessible to users in order to have any usefulness, so Aviary includes a three level interface that allows users to interact with the tool in the way that best fits the needs of their model. This interface allows for the complexity to fully capture the model physics but also simplifies away pieces of the interface that are not needed.

This paper has shown that Aviary is a multidisciplinary design optimization and analysis tool which allows for tightly coupled simultaneous aircraft and subsystem design using analytic gradients. As aircraft design moves into a new regime where vehicles have higher amounts of coupling, gradient based optimization is an increasingly attractive way to model these vehicles. Having fast and accurate derivative calculations is an important part of gradient based optimization, and Aviary has provided a way to perform analytic gradient based optimization on air vehicles.

Acknowledgements

The work of developing Aviary would not be possible without the contributions of many different individuals. The authors want to acknowledge and recognize the significant contributions of the other members (past and present) of the Aviary development team, including: Christopher Bennett (AMA[‡]), Darrell Caldwell (AMA), Xun Jiang (AMA), Erik Olson (formerly), and Gregory Wrenn (AMA) of NASA Langley Research Center; Kenneth Lyons and Dahlia Pham of the NASA Ames Research Center; and Robert Falck, Bret Naylor (BQMI[§]), Kaushik Ponnapalli (HX5), Herbert Schilling, and Sydney Schnulo (formerly) of the NASA Glenn Research Center. These individuals have all contributed large amounts of dedication, time, and talent to developing the Aviary code base and this work is greatly indebted to them. The authors also want to thank those who have served in advisory or management roles in support of the Aviary work, including: Jeffryes Chapman, Eric Hendricks, and Justin Gray (formerly) of the NASA Glenn Research Center, Joseph Garcia and Ben Margolis of the NASA Ames Research Center, and Ben Phillips of the NASA Langley Research Center. The authors would also like to thank the Model-Based Systems Analysis and Engineering Team led by Irian Ordaz of the NASA Langley Research Center for their feedback on the Aviary tool and for Irian's generation of the data used in the verification plots in this paper.

The work presented in this paper and the Aviary tool itself were developed with support from NASA's Transformational Tools and Technologies (TTT), Advanced Air Transport Technology (AATT), and Electric Powered Flight Demonstration (EPFD) Projects. These projects are funded by Transformational Aeronautic Concepts Program (TACP), Advanced Air Vehicles Program (AAVP), and Integrated Aviation Systems Program (IASP) respectively. Funding for those Programs is provided by the Aeronautics Research Mission Directorate (ARMD).

[‡]Analytical Mechanics Association

[§]Banner Quality Management Inc.

References

[1] McCullers, L. A., "Aircraft Configuration Optimization Including Optimized Flight Profiles," *Recent Experiences in Multidisciplinary Analysis and Optimization*, 1984, pp. 395–412. URL https://ntrs.nasa.gov/citations/19870002310.

[2] Hague, D., "Main Program," *GASP - General Aviation Synthesis Program*, Vol. 1, 1978. URL https://ntrs.nasa.gov/api/citations/19810010562/downloads/19810010562.pdf.

[3] Falck, R., Chin, J., Schnulo, S., Burt, J., and Gray, J., "Trajectory Optimization of Electric Aircraft Subject to Subsystem Thermal Constraints," *AIAA Aviation*, Denver, Colorado, 2017. URL https://arc.aiaa.org/doi/pdf/10.2514/6.2017-4002.

[4] Antcliff, K. R., Guynn, M. D., Marien, T., Wells, D. P., Schneider, S. J., and Tong, M. J., *Mission Analysis and Aircraft Sizing of a Hybrid-Electric Regional Aircraft*, 2016. https://doi.org/10.2514/6.2016-1028.

[5] Hendricks, E., Falk, R., Gray, J., Aretskin-Hariton, E., Ingraham, D., Chapman, J., Schnulo, S., Chin, J., Jasa, J., and Bergeson, J., "Multidisciplinary Optimization of a Turboelectric Tiltwing Urban Air Mobility Aircraft," *American Institute of Aeronautics and Astronautics*, 2019. https://doi.org/https://doi.org/10.2514/6.2019-3551.

[6] Schnulo, S., "Overview of NASA GRC Electrified Aircraft Propulsion Systems Analysis Methods," URL: https://ntrs.nasa.gov/api/citations/20180001332/downloads/20180001332.pdf, 10 2017. EnergyTech 2017 Presentation.

[7] Heath, C., Seidel, J., and Rallabhandi, S. K., *Viscous Aerodynamic Shape Optimization with Installed Propulsion Effects*, 2017. https://doi.org/10.2514/6.2017-3046.

[8] Li, W., and Geiselhart, K., *Propulsion-Airframe Integration for Conceptual Redesign of a Low-Boom Supersonic Transport*, 2023. https://doi.org/10.2514/6.2023-1511.

[9] 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.

[10] Martins, J. R., and Ning, A., *Engineering design optimization*, Cambridge University Press, 2021.

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

[12] Martins, J. J., Sturdza, P., and Alonso, J. J., "The complex-step derivative approximation," *ACM Transactions on Mathematical Software*, Vol. 29, 2003, pp. 245 – 262. https://doi.org/10.1145/838250.838251, URL https://hal.science/hal-01483287.

[13] Wells, D. P., Horvath, B. L., and Mccullers, L. A., "The Flight Optimization System Weights Estimation Method," 2017. URL https://ntrs.nasa.gov/api/citations/20170005851/downloads/20170005851.pdf.

[14] Nunez, L. S., Tai, J. C., and Mavris, D. N., *The Environmental Design Space: Modeling and Performance Updates*, 2021. https://doi.org/10.2514/6.2021-1422.

[15] Lytle, J. K., "The Numerical Propulsion System Simulation: A Multidisciplinary Design System for Aerospace Vehicles," 1999. URL https://ntrs.nasa.gov/api/citations/19990062672/downloads/19990062672.pdf.

[16] Lukaczyk, T. W., Wendorff, A. D., Colonno, M., Economon, T. D., Alonso, J. J., Orra, T. H., and Ilario, C., "SUAVE: an open-source environment for multi-fidelity conceptual vehicle design," *16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2015, p. 3087.

[17] David, C., Delbecq, S., Defoort, S., Schmollgruber, P., Benard, E., and Pommier-Budinger, V., "From FAST to FAST-OAD: An open source framework for rapid Overall Aircraft Design," *IOP Conference Series: Materials Science and Engineering*, Vol. 1024, IOP Publishing, 2021, p. 012062.

[18] Sharpe, P. D., "AeroSandbox: A Differentiable Framework for Aircraft Design Optimization," Master's thesis, Massachusetts Institute of Technology, 2021.

[19] Heath, C., and Gray, J., "OpenMDAO: Framework for Flexible Multidisciplinary Design, Analysis and Optimization Methods," 2012. https://doi.org/10.2514/6.2012-1673.

[20] Hague, D., "Weights," *GASP - General Aviation Synthesis Program*, Vol. 5, 1978. URL https://ntrs.nasa.gov/api/citations/19810010566/downloads/19810010566.pdf.

[21] Hague, D., "Aerodynamics," *GASP - General Aviation Synthesis Program*, Vol. 3, 1978. URL https://ntrs.nasa.gov/api/citations/19810010564/downloads/19810010564.pdf.

[22] Feagin, R., and Morrison, W. J., "Delta Method, An Empirical Drag Buildup Technique," *NASA Contractor Report*, 1978. URL https://ntrs.nasa.gov/api/citations/19790009630/downloads/19790009630.pdf.

[23] NASA, "Flight Optimization System Release 9.0.0 User's Guide," Available with public distribution of software, 2016. NASA.

[24] Lambe, A. B., and Martins, J. R. R. A., "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes," *Structural and Multidisciplinary Optimization*, Vol. 46, No. 2, 2012, pp. 273–284. https://doi.org/10.1007/s00158-012-0763-y.

[25] Hague, D., "Geometry," *GASP - General Aviation Synthesis Program*, Vol. 2, 1978. URL https://ntrs.nasa.gov/api/citations/19810010563/downloads/19810010563.pdf.

[26] Hague, D., "Propulsion, Part I - Theoretical Development," *GASP - General Aviation Synthesis Program*, Vol. 4, 1978. URL https://ntrs.nasa.gov/api/citations/19810010564/downloads/19810010564.pdf.

[27] Aretskin-Hariton, E., Gratz, J., Jasa, J., Moore, K., Falck, R., Kuhnle, C., Hendricks, E., Kirk, J., Olson, E., Caldwell, D., Recine, C., and Lyons, K., "Multidisciplinary Optimization of a Transonic Truss-Braced Wing Aircraft using the Aviary Framework," *AIAA Scitech 2024 Forum*, 2024. https://doi.org/10.2514/6.2024-1084.

[28] Capristan, F. M., and Welstead, J. R., *An Energy-Based Low-Order Approach for Mission Analysis of Air Vehicles in LEAPS*, 2018. https://doi.org/10.2514/6.2018-1755.

[29] Hague, D., "Performance," *GASP - General Aviation Synthesis Program*, Vol. 6, 1978. URL https://ntrs.nasa.gov/api/citations/19810010567/downloads/19810010567.pdf.

[30] Falck, R., Gray, J. S., 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.

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

[32] Hann, A., "Vehicle Sketch Pad: A Parametric Geometry Modeler for Conceptual Aircraft Design," 2010. https://doi.org/10.2514/6.2010-657.

[33] McDonald, R. A., and Gloudemans, J. R., *Open Vehicle Sketch Pad: An Open Source Parametric Geometry and Analysis Tool for Conceptual Aircraft Design*, 2021. https://doi.org/10.2514/6.2022-0004.

[34] NASA, "Software Package, Version 3.31.1," URL: http://openvsp.org/, 2022. NASA.

[35] Gray, J., Chin, J., Hearn, T., Hendricks, E., Lavelle, T., and Martins, J. R. R. A., "Chemical-Equilibrium Analysis with Adjoint Derivatives for Propulsion Cycle Analysis," *Journal of Propulsion and Power*, Vol. 33, No. 5, 2017, pp. 1041–1052. https://doi.org/10.2514/1.B36215.

[36] Jasa, J. P., Hwang, J. T., and Martins, J. R. R. A., "Open-source coupled aerostructural optimization using Python," *Structural and Multidisciplinary Optimization*, 2018. https://doi.org/10.1007/s00158-018-1912-8.

[37] Kennedy, G. J., and Martins, J. R., "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. https://doi.org//10.1016/j.finel.2014.04.011.

[38] Lopes, L., and Burley, C., "Design of the Next Generation Aircraft Noise Prediction Program: ANOPP2," *American Institute of Aeronautics and Astronautics*, AIAA, 2011.

[39] Frederic, P., Bezos-O'Connor, G. M., and Nickol, C., *Cost Analysis Approach in the Development of Advanced Technologies for Green Aviation Aircraft*, John Wiley and Sons, Ltd, 2016, pp. 1–13. https://doi.org//10.1002/9780470686652.eae1078.

[40] Frostig, R., Johnson, M. J., and Leary, C., "Compiling machine learning programs via high-level tracing," *Systems for Machine Learning*, Vol. 4, No. 9, 2018.

[41] 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.

[42] NASA, "Aviary Online Documentation," https://openmdao.github.io/Aviary/intro.html, 2024. Accessed: 2024-5-20.