AIAA SciTech Forum
23-27 January 2023, National Harbor, MD & Online
AIAA SCITECH 2023 Forum

10.2514/6.2023-1162

A Parametric Design Process based on Optimization-Guided Incremental Design Decisions

Dongjoon Lee*, Cody Karcher†, Robert Haimes‡, and Marshall Galbraith§
Massachusetts Institute of Technology, Cambridge, MA, 02139

John F. Dannenhoffer¶
Syracuse University, Syracuse, NY, 13244

The traditional aircraft design process is typically split into three stages: conceptual, preliminary, and detailed design. This three-stage process usually proceeds sequentially from stage to stage, and major design decisions are frozen between stage transitions. But this model does not capture the more complex reality, where design decisions are subject to frequent iteration at all stages of the process. This paper presents the Engineering Sketch Pad (ESP) *Phasing* capability that captures this more complex design workflow by decomposing the process into atomic portions called *Phases*. Each phase is intended to branch from any completed phase and answer a specific design question, allowing the designer to make design decisions non-sequentially. The use of this *Phasing* capability is demonstrated with the sizing of an aircraft wing while simultaneously optimizing an airfoil with increasing aerodynamic and geometric model fidelity. A number of cases are presented, beginning with a low fidelity aerodynamic model and a NACA 24XX airfoil geometry, and culminating in a Kulfan CST4 representation of geometry with MSES to perform the airfoil analysis.

I. Nomenclature

A	=	aspect ratio
A_{prop}	=	propeller disk area
C_D	=	drag coefficient
C_{D_p}	=	profile drag coefficient
C_L	=	lift coefficient
$C_{L,\text{max}}$	=	lift coefficient
e	=	Oswald efficiency factor
f_{wadd}	=	wing added weight fraction
g	=	gravitational constant
h_{fuel}	=	fuel heating value
$\bar{I}_{\text{cap}}$	=	area moment of inertia per unit chord
$\bar{M}_r$	=	root moment per chord
P_{max}	=	maximum engine output power
p	=	$1 + 2\lambda$
q	=	$1 + \lambda$
R	=	Range
Re	=	Reynolds number
S	=	wing area
T	=	thrust force
T_{sprint}	=	thrust (sprint)
$\bar{t}_{\text{cap}}$	=	spar cap thickness per unit chord

*Graduate Student, MIT Aeronautics and Astronautics
†Postdoctoral Researcher, University of Michigan Aerospace Engineering; Former Graduate Student, MIT Aeronautics and Astronautics
‡Principal Research Engineer, MIT Aeronautics and Astronautics
§Research Engineer, MIT Aeronautics and Astronautics
¶Associate Professor, Syracuse Mechanical and Aerospace Engineering

V = flight speed
V_{sprint} = flight speed (sprint)
V_{stall} = stall speed
W = operating weight
W_{cap} = spar cap weight
W_{eng} = engine weight
W_{fixed} = fixed weight
$W_{\text{fuel,out}}$ = weight of fuel burned (outbound)
$W_{\text{fuel,ret}}$ = weight of fuel burned (return)
W_{MTO} = maximum takeoff weight
W_{out} = aircraft weight (outbound)
W_{pay} = payload weight
W_{web} = shear web weight
W_{wing} = wing weight
W_{zfw} = zero-fuel weight
$\tilde{W}$ = weight excluding wing
z_{bre} = Breguet parameter
η_0 = overall efficiency
$\eta_{0,\text{sprint}}$ = overall efficiency (sprint)
η_{eng} = engine efficiency
η_{i} = inviscid propeller efficiency
η_{prop} = propeller efficiency
η_{v} = viscous propeller efficiency
λ = wing taper ratio
ν = $(1 + \lambda + \lambda^2)/(1 + \lambda)^2$
ρ = air density
ρ_{sl} = sea level air density
μ = dynamic viscosity
τ = wing thickness ratio

II. Introduction

The traditional approach to engineering design often splits the workflow into sequential stages of conceptual, preliminary, and detailed design. Conceptual design translates a set of design requirements into a number of concepts and configurations that are sized and evaluated at a high level using engineering intuition, empirical models, and low-fidelity analysis. As the design matures, certain design decisions become frozen to focus on refining geometry and increasing analysis fidelity. But it is sometimes necessary to iterate on major design decisions in a more continuous workflow as the modeling and analysis fidelities increase. Concepts that were promising in early stages may present problems when higher fidelity geometry and analysis are used, and new information may be learned in later stages that change requirements or make other configurations more attractive. Existing methods cannot manage both phases of examining different concepts and phases of analyzing performance with differing fidelities, which brings up the question: How best can we handle the process complexity inherent in aircraft design?

One way to approach the design complexity problem is to view the design process as a decision tree where each node represents the work necessary to answer a specific design question. This view breaks the overall design process into manageable chunks – optimize the L/D of a wing at a specific flight condition, compare the fuel burn between two and four propulsor configurations using first-order aero-propulsive models. This process allows the designer to trace design decisions up to specific nodes and progress the design from any node in the tree, enable non-sequential design workflows.

In order to provide a software infrastructure that could be used as the basis for a design system, the Engineering Sketch Pad (ESP) open-source project [1] contains a parametric geometry system integrated with the ability to specify process workflows as described. In ESP, each node on the decision tree is called a *Phase* and has the following attributes. First, any *Phase* can be a stepping stone to another with internal object models and values stored. The designer can take the result of any node in the design tree and start a new *Phase* to answer a different design question. Second, a completed *Phase* can branch into multiple new *Phases*, which allows the designer to explore and compare multiple

solutions for a particular design question. The best performing branch can be selected and the others can be pruned. Lastly, each *Phase* can be driven by differing workflow and geometry scripts, which allows each *Phase* to be streamlined in answering the design question of interest.

ESP provides multiple capabilities in addition to *Phasing* which facilitates the design workflow. Direct connections to a diverse collection of variable fidelity and multidisciplinary analysis suites allows users to seamlessly move data throughout the system. The parametric geometry build is either differentiated by hand or the code itself is differentiated using operator overloaded automatic differentiation [2–5] which provides analytic derivatives for efficient gradient-based optimization. To enable a collaborative workflow, a browser-based user interface to ESP can be simultaneously accessed from various locations. These capabilities are provided by various ESP components which can be accessed as individual APIs or through the integrated ESP system. The CAPS component connects geometry to analysis with Analysis Interface Modules (AIMs), which provide an abstraction and plug-in technology that facilitates dealing with connecting solvers [6–9]. Most CAPS workflow is performed using pyCAPS [10], a Python c-types wrapper to provide a Python object interface, enabling similar workflow to Python-based MDO frameworks such as OpenMDAO [11]. The rest of the paper demonstrates a multi-fidelity design process, using ESP to enable connection between the optimization and the underlying geometry, as well as *Phasing* capability to break up the design process into manageable incremental design decisions.

III. The Aircraft Design Optimization Problem

The *Phasing* demonstration builds upon a design optimization problem originally proposed in Hoburg and Abbeel [12]. The problem seeks to size a UAV flying an outbound cruise segment, a second return cruise segment, and a third sprint segment used to size the engine. The goal is to minimize the total weight of consumed fuel,

$$W_{\text{fuel,out}} + W_{\text{fuel,ret}} \tag{1}$$

subject to the following constraints (which are classified for readability):

Steady level flight relations:

$$W = \frac{1}{2}\rho V^2 C_L S \qquad T \geq \frac{1}{2}\rho V^2 C_D S \qquad Re = \frac{\rho V S^{1/2}}{A^{1/2}\mu} \tag{2}$$

Landing flight condition:

$$W_{\text{MTO}} \leq \frac{1}{2}\rho_{\text{sl}} V_{\text{stall}}^2 C_{L,\max} S \qquad V_{\text{stall}} \leq 38 \tag{3}$$

Sprint flight condition:

$$P_{\max} \geq \frac{T_{\text{sprint}} V_{\text{sprint}}}{\eta_{0,\text{sprint}}} \qquad V_{\text{sprint}} \geq 150 \tag{4}$$

Drag model:

$$\begin{aligned} C_D \geq & \frac{0.5}{S} + C_{D_p} + \frac{C_L^2}{\pi e A} \\ 1 \geq & 2.56\frac{C_L^{5.88}}{\tau^{3.32} Re^{1.54} C_{D_p}^{2.26}} + 3.80\times 10^{-9}\frac{\tau^{6.23}}{C_L^{0.92} Re^{1.38} C_{D_p}^{9.57}} + \quad 2.20\times 10^{-3}\frac{\tau^{0.03} Re^{0.14}}{C_L^{0.01} C_{D_p}^{0.73}} + \\ & 1.19\times 10^{4}\frac{C_L^{9.78}\tau^{1.76}}{Re^{1.00} C_{D_p}^{0.91}} + 6.14\times 10^{-6}\frac{C_L^{6.53}}{\tau^{0.52} Re^{0.99} C_{D_p}^{5.19}} \end{aligned} \tag{5}$$

Propulsive efficiency:

$$\eta_0 \leq \eta_{\text{eng}}\eta_{\text{prop}} \qquad \eta_{\text{prop}} \leq \eta_i \eta_v \qquad 4\eta_i + \frac{T\eta_i^2}{\frac{1}{2}\rho V^2 A_{\text{prop}}} \leq 4 \tag{6}$$

Range constraints:

$$R \geq 5000 \times 10^3 \qquad z_{\text{bre}} \geq \frac{gRT}{h_{\text{fuel}}\eta_0 W} \qquad \frac{W_{\text{fuel}}}{W} \geq z_{\text{bre}} + \frac{z_{\text{bre}}^2}{2} + \frac{z_{\text{bre}}^3}{6} + \frac{z_{\text{bre}}^4}{24} \tag{7}$$

Weight relations:

$$\begin{aligned} W_{\text{pay}} &\geq 500g & \frac{W_{\text{wing}}}{f_{\text{wadd}}} &\geq W_{\text{web}} + W_{\text{cap}} \\ \tilde{W} &\geq W_{\text{fixed}} + W_{\text{pay}} + W_{\text{eng}} & W_{\text{out}} &\geq W_{\text{zfw}} + W_{\text{fuel,ret}} \\ W_{\text{zfw}} &\geq \tilde{W} + W_{\text{wing}} & W_{\text{MTO}} &\geq W_{\text{out}} + W_{\text{fuel,out}} \\ W_{\text{eng}} &\geq 0.0372 P_{\text{max}}^{0.803} & W_{\text{sprint}} &= W_{\text{out}} \end{aligned} \tag{8}$$

Wing structural model:

$$\begin{aligned} 2q &\geq 1 + p & 8 &\geq \frac{N_{\text{lift}} \bar{M}_r A q^2 \tau}{S \bar{I}_{\text{cap}} \sigma_{\text{max}}} \\ p &\geq 1.9 & 12 &\geq \frac{A \tilde{W} N_{\text{lift}} q^2}{\tau S \bar{t}_{\text{web}} \sigma_{\text{max,shear}}} \\ \tau &\leq 0.15 & \nu^{3.94} &\geq 0.86 p^{-2.38} + 0.14 p^{0.56} \\ \bar{M}_r &\geq \frac{\tilde{W} A p}{24} & W_{\text{cap}} &\geq \frac{8 \rho_{\text{cap}} g \tilde{w} \bar{t}_{\text{cap}} S^{3/2} \nu}{3 A^{1/2}} \\ 0.92 \bar{w} \tau \bar{t}_{\text{cap}}^2 + \bar{I}_{\text{cap}} &\leq \frac{0.92^2}{2} \bar{w} \tau^2 \bar{t}_{\text{cap}} & W_{\text{web}} &\geq \frac{8 \rho_{\text{web}} g r_h \tau \bar{t}_{\text{web}} S^{3/2} \nu}{3 A^{1/2}} \end{aligned} \tag{9}$$

Constant parameters are shown in Table 1. As formulated, this is a geometric program which is convex and therefore rapidly converges to a globally optimal design. However, the implicit constraint for $C_{D_p} = f(C_L, Re, \tau)$, shown in Eq. 5, is a fit to approximately 2500 data points generated using XFOIL [13] and presents one of the largest sources of uncertainty in aircraft aerodynamic performance. Over the course of the following sections, this paper will describe a *Phasing* process to: 1) Increase the fidelity of this model by using the `CAPS` AIM for MSES [14], and 2) Reduce the required fuel consumption by introducing higher fidelity geometry representations with more degrees of freedom in the optimization.

Table 1 Fixed constant parameters for the design problem

Quantity	Value	Description
N_{lift}	6.0	Wing loading multiplier
σ_{max}	250×10^6 Pa	Allowable stress, 6061-T6
$\sigma_{\text{max,shear}}$	167×10^6 Pa	Allowable shear stress
g	9.81 m/s^2	Gravitational constant
$\bar{w}$	0.5	Wing-box width/chord
f_{wadd}	2.0	Wing added weight fraction
W_{fixed}	14.700 N	Fixed weight
$C_{L,\text{max}}$	1.5	Maximum C_L, flaps down
ρ	0.91 kg/m^3	Air density, 3000 m
ρ_{sl}	1.23 kg/m^3	Air density, sea level
μ	1.69×10^{-3} kg/m/s	Dynamic viscosity, 3000 m
e	0.95	Wing spanwise efficiency
A_{prop}	0.785 m^2	Propeller disk area
h_{fuel}	46×10^6 J/kg	Fuel heating value

IV. Phasing Workflow and Outline

A *Phase* in ESP is essentially defined by two input files: a CSM file that captures the parametric geometry variables, and a pyCAPS [10] script that notes the intent of the current *Phase*, constructs and solves the optimization problem, and updates the geometry with the optimal values. As the analysis fidelity increases across *Phases*, the geometric complexity should also increase to accurately capture and model a larger design space. In this design example, the first phase begins with variable wing area, aspect ratio, taper, and airfoil thickness to chord ratio. The airfoil thickness is confined to the four digit NACA 24XX family of airfoils and is taken as a constant section across the entire wing. Later phases evolve to include additional variables that represent a larger design space of airfoils, such as camber and Kulfan [15] airfoil parameters. ESP allows users to define their own geometric primitives (UDPs), which could define anything from a specific 2D shape to a fully parametric fuselage [2]. Many UDPs are predefined in ESP, including primitives for NACA and Kulfan airfoils, which simplifies the airfoil geometry definition. The complete CSM scripts used across the *Phases* are shown in Appendices A-B.

The optimization in this example is defined and solved using Corsair*, which facilitates run-time calls to black-box analysis tools while taking advantage of underlying convexity in the formulation. Each design phase has an associated python file which defines the optimization formulation – variables, constants, and constraints – with Corsair and uses pyCAPS to enable phasing capability, interaction with the ESP geometry, and connection to analysis tools. These complete pyCAPS scripts are shown in Appendices C-I

Figure 1 outlines the design phases that are present in Section V of this paper. The blue boxes represent the successfully optimized phases, and the red boxes represent phases with unrealistic or infeasible results which are eventually pruned. The phases are listed and summarized here.

1. **GPSize**: Solves the original Geometric Program formulation
2. **MSES**: Introduces a runtime call to MSES to calculate profile drag within the optimization loop

3A. **Camber**: Introduces maximum camber and camber location in geometry; results in unrealistic structures

3B. **CMConstraint**: Adds constraints on airfoil moment coefficient

4A. **Kulfan2**: Defines airfoil with two upper and two lower Kulfan coefficients; results in unrealistic aerodynamics

4B. **FlowTrip**: Adds a specified flow trip location within MSES to force transition

5. **Kulfan4**: Increases Kulfan order from two to four coefficients.

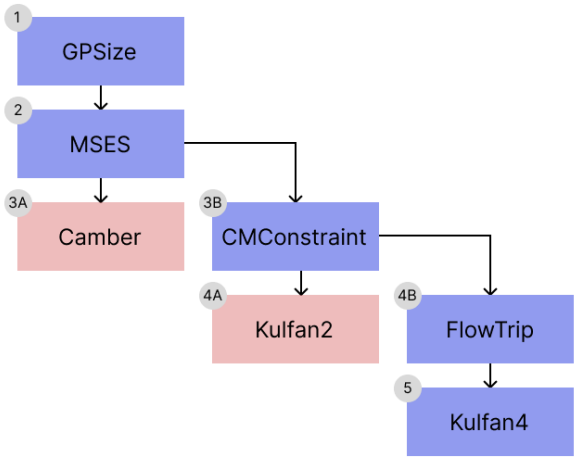


Fig. 1 Flowchart of the Design Phases

*Corsair is an internal optimization tool with methods described in [16–18]

V. Design Phases

A. Phase 1. GPSize: Solving the Original Geometric Program

1. Phase Intent and Formulation

The demonstration begins with solving the original problem described in Section III. In the original Geometric Program formulation, the model for drag coefficient assumes a NACA 24XX airfoil with thickness to chord ratio, τ, as the sole airfoil design parameter. The wing is assumed to operate at a Mach number $M = 0$, and compressibility effects are not modeled. The wing structural model drives τ higher, seeking to improve structural efficiency, whereas the drag model should drive τ lower, as thick airfoils tend to have an increased drag. In this specific GP-compatible formulation however, τ must be constrained as $\tau \leq 0.15$ for two reasons: first, the XFOIL fitted aerodynamic model does not sufficiently capture the aerodynamics of thicker airfoils, and second, the wing structural model is conservative in sizing for bending loads such that τ is strongly driven upward.

Because Geometric Programs are convex, the optimization converges to a global optimum without iteration, and therefore, no initial guess is required for Phase 1. The guaranteed global optimality comes at the cost of decreased modeling flexibility and fidelity, because all constraints must fit into GP-compatible formats. The associated NACA CSM file is shown in Appendix A and the python script for Phase 1 is shown in Appendix C.

2. Examining the Phase Result

Though the problem has more than 40 variables, the most relevant have been summarized in Table 2. The optimal design drives τ up to the maximum of 0.15, resulting in a NACA 2415 airfoil, which is consistent with the original formulation results of Hoburg and Abbeel [12]. This result will serve as the baseline for the cases that follow. Figure 2 shows the NACA 2415 airfoil C_p plot from MSES at the optimized outbound cruise point of $C_L = 0.5499$. The MSES results are at the optimized cruise speed of $M = 0.2$ which accounts for the 4% increase in C_{D_p} from the value in Table 2. The discrepancy arises from the GP compatible fitted C_{D_p} model ignoring compressibility. Because small changes in aircraft drag have large implications for overall performance, reducing the viscous drag uncertainty will be a priority for future *Phases*.

Table 2 Phase 1 optimal variables summary

Variable	Value	Units
W_{fuel}	6320.52	N
AR	18.10	-
$C_{D_p,\text{out}}$.	0.005403	-
$C_{D_i,\text{out}}$.	0.005596	-
$C_{L,\text{out}}$	0.5499	-
S	28.38	m^2
W_{MTO}	37.64	kN
τ	0.150	-

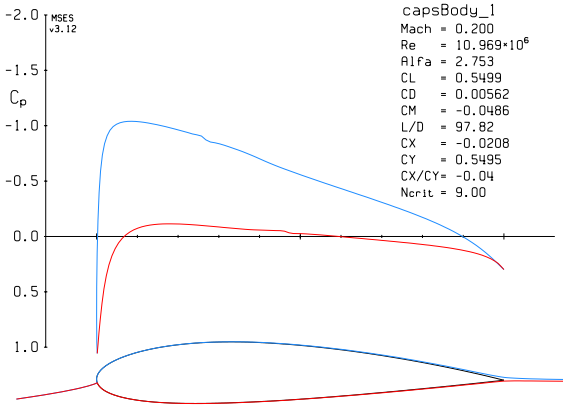


Fig. 2 Phase 1 MSES analysis, outbound conditions

3. ESP Viewer

Although the *Phasing* workflow can be completed exclusively by running the `pyCAPS` scripts that interface between Corsair and `ESP`, the optimized geometry can be viewed using the `ESP` browser user interface which is shown in Fig. 3. An alternative to running the `pyCAPS` scripts in the shell is to open up the `ESP` viewer, which displays the geometry build as well as all the design parameters and optimization values. Users also have the ability to load `pyCAPS` scripts and run them in this user interface as well as start, modify, and end phases.

Fig. 3 ESP viewer at the end of Phase 1

B. Phase 2. MSES: Increasing Aerodynamic Model Fidelity with MSES

1. Phase Intent and Formulation

The surrogate model for XFOIL [13] used in the GP formulation is undesirable for a number of reasons:

1) The least-squares fitting process introduces some error since all points in the fitting set are not guaranteed to lie exactly on the fit.
2) The surrogate model introduces interpolation error for cases not contained in the fitting set.
3) The GP compatible fitting process is unable to capture some of the higher order features of the set.
4) Thickness had to be arbitrarily constrained to ensure the validity of the surrogate model.
5) XFOIL uses the Prandtl-Glauert model for increasing Mach number, but the surrogate was fit at $M = 0$ and therefore under-predicts drag.

It is therefore desirable to remove the surrogate model altogether. As was discussed above, the surrogate model captures the relationship $C_{D_p} = f(C_L, Re, \tau)$ in a GP compatible way. Utilizing recent work developing optimization algorithms [16–18], non GP compatible models of $C_{D_p} = f(C_L, Re, \tau)$ can be optimized while still taking advantage of underlying convexity in C_{D_p}. Though XFOIL is a valid method of representing $C_{D_p} = f(C_L, Re, \tau)$, it is not capable of returning analytical derivatives with respect to functional inputs and geometry variables, which is required for efficient gradient-based optimization methods. Thus, MSES, which is a higher fidelity tool for airfoil analysis capable of returning derivatives, is used instead.

MSES uses the same Integral Boundary Layer (IBL) method as XFOIL near the surface of the airfoil, but the Full Potential solution for the far-field in XFOIL has been replaced in MSES by a 2D Euler method, making MSES superior for Mach numbers where compressibility is non-negligible. MSES is only capable of returning derivatives when run for a fixed angle of attack, and not a fixed C_L, and so the problem formulation must be adjusted slightly. Rather than directly imposing the constraint,

$$C_{D_p} \geq f(C_L, Re, \tau), \tag{10}$$

angle of attack, α, is introduced along with $C_{L_{\text{MSES}}}$, and the following constraints are imposed:

$$\begin{aligned} C_{D_p} &\geq f(\alpha, Re, \tau) \\ C_{L_{\text{MSES}}} &= f(\alpha, Re, \tau) \\ C_L &= C_{L_{\text{MSES}}} \end{aligned} \tag{11}$$

Valid values of α here may be zero or negative under certain conditions, but the log-transformation optimization methods from [16–18] cannot handle negative variable values. Therefore, in practice the alpha variable is transformed using an additive shift to ensure it remains positive under all reasonable cases.

2. Setting Up the Problem in a Phase and Obtaining an Optimal Solution

The set up of Phase 2 is similar to Phase 1 with the addition of a few new constraints. Corsair models the use of MSES in constraint definition as a 'RuntimeConstraint' object that calls MSES in a just-in-time fashion. For full mathematical details of the optimization process, see [16–18]. Unlike the previous geometric programming phase, the optimizer now requires an initial guess for the more complex problem formulation. With ESP *Phasing*, the optimized result of the previous phase can be maintained and retrieved to serve as the initial guess of the new phase. Unlike the GP compatible formulation of Phase 1, which solves in a few seconds, running with MSES in the loop increases the solve time to several minutes. The python script for Phase 2 is shown in Appendix D.

3. Examining the Phase Result

The relevant design variables are reported in Table 3. Note that the fuel burn has actually *increased* slightly due to the fact that the previous GP compatible constraint was not modeling the modest compressibility effect of cruising at $M \approx 0.2$. For the rest of the paper, Phase 2 will serve as a baseline, since it accurately models the compressibility drag penalties. The resulting airfoil is again a NACA 2415. The thickness ratio, τ, is still driven to the imposed maximum of $\tau \leq 0.15$ because of the conservative bending structural models. Fig. 4 shows the result from MSES during the outbound cruise leg which now matches the C_{D_p} output of the optimzation.

Table 3 Phase 2 optimal variables summary

Variable	Value	Units
W_{fuel}	6425.68	N
AR	18.13	-
$C_{D_p,\text{out}}$	0.005634	-
$C_{D_i,\text{out}}$	0.005484	-
$C_{L,\text{out}}$	0.5447	-
S	28.34	m^2
W_{MTO}	37.60	kN
τ	0.150	-

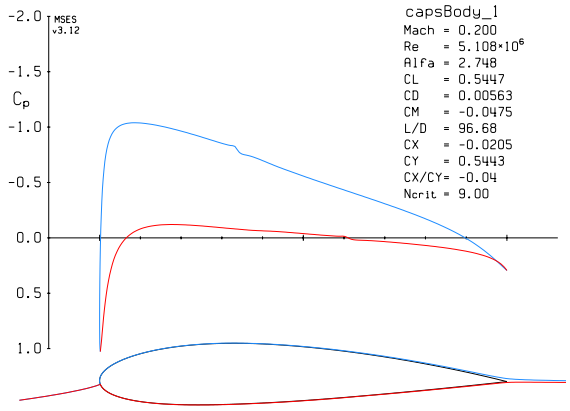


Fig. 4 Phase 2 MSES analysis, outbound conditions

C. Phase 3A. Camber: Adding New Geometry Variables

1. Phase Intent and Formulation

In this design phase, the aerodynamic fidelity could be further evolved with a 2D CFD tool such as SU2 [19]. However, given the lower modeling fidelity of the other disciplines such as structures and geometry, further increasing only the aerodynamic model fidelity would produce diminishing returns. So instead, the remaining phases will focus on refining the airfoil geometry. By increasing the number of optimization variables that define the airfoil, the optimizer will be capable of driving to a higher performing design. The simplest first step is to expand the design space from NACA 24XX to the full NACA 4 series definition of airfoil geometry. This introduces two new variables to the optimization problem: maximum camber per unit chord c_{max}, and the location of the maximum camber as a fraction of chord length $d_{c_{\text{max}}}$. Note that the optimization is not limited to the classic discrete values for camber, maximum location, and thickness as implied by the NACA XXXX notation, but rather, optimizes over continuous values. These new design variables change the relevant optimization constraints as follows:

$$\begin{aligned} C_{D_p} &\geq f(\alpha, Re, c_{\text{max}}, d_{c_{\text{max}}}, \tau) \\ C_{L_{\text{MSES}}} &= f(\alpha, Re, c_{\text{max}}, d_{c_{\text{max}}}, \tau) \\ C_L &= C_{L_{\text{MSES}}} \end{aligned} \tag{12}$$

The python script for Phase 3A is shown in Appendix E.

2. Examining the Phase Result

The results from MSES (Fig. 5) and the report from the optimizer (Table 4) give cause for concern. First, the optimized solution has a cruise segment with a wing angle of attack of $-2.55°$ and a sprint segment with $\alpha = -5.87°$. Note also that the lift coefficient is higher during the return segment, $C_{L,\text{ret}}$, when the aircraft should be lighter. Second, examining Fig. 5 shows a significant amount of lift being generated on the aft section of the airfoil, where the thin geometry would struggle to take these loads for any reasonable structural design. Third, transition is occurring at around 85% on the top surface, which is unlikely to be realized a real flow.

Table 4 Phase 3A optimal variables summary

Variable	Value	Units
W_{fuel}	5386.69	N
AR	18.75	-
$C_{D_p,\text{out}}$	0.003967	-
$C_{D_i,\text{out}}$	0.004400	-
$C_{L,\text{out}}$	0.4956	-
$C_{L,\text{ret}}$	0.5242	-
α_{out}	-2.547	deg
α_{sprint}	-5.875	deg
S	27.64	m^2
W_{MTO}	36.67	kN
$c_{\max}$	0.0399	-
$d_{c_{\max}}$	0.7981	-
τ	0.148	-

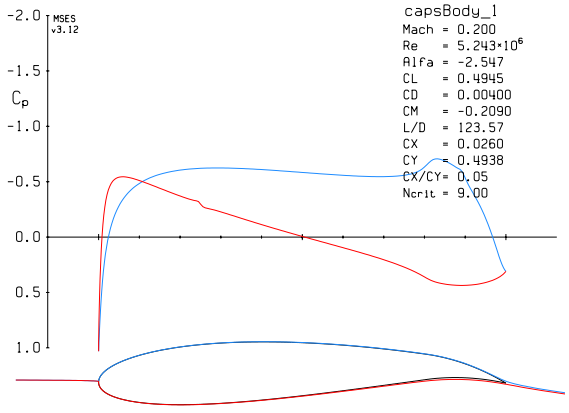
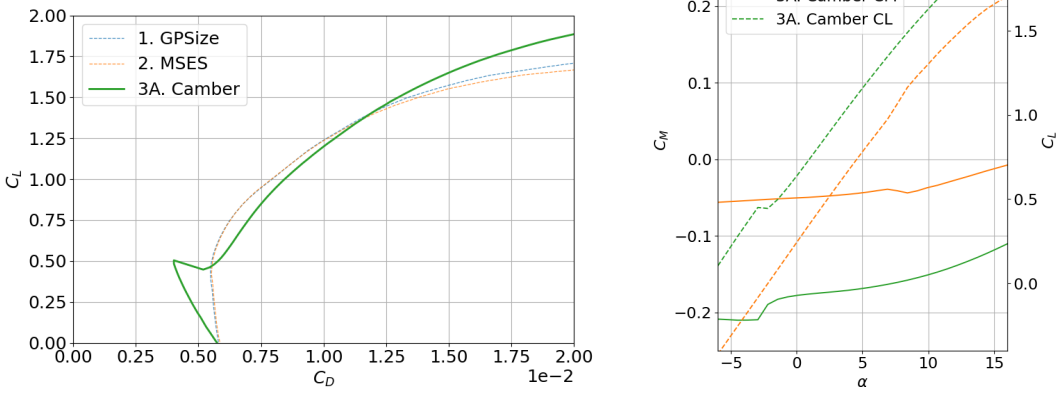


Fig. 5 Phase 3A MSES analysis, outbound conditions

Plotting the drag polar confirms many of these worries (Fig. 6a), with a clear corner occurring at the design C_L of 0.5. For the current Phase, MSES reports a large moment coefficient, $C_M \approx -0.2$ for the resulting angle of attacks of the optimization, whereas the previous Phase reports a cruise at $C_M \approx -0.05$ (Fig. 6b). Clearly, the optimization has exploited a poor design formulation. More specifically, the lack of a thickness constraint on the trailing edge of the airfoil has allowed the optimization to ignore real structural physics. Thus, the significant fuel savings reported by the optimization in this phase are not physically realizable.



(a) Lift and Drag Coefficients **(b) Lift and Moment Coefficients**

Fig. 6 Polars for the first three design phases

D. Phase 3B. CMConstraint: Imposing a Constraint on Moment Coefficient

1. Phase Intent and Formulation

In the process of refining the airfoil geometry, Phase 3A showed the need to include a structural constraint that limits the aft camber available for the optimization. Rather than write a detailed model analyzing the stress in the airfoil section, a bound on the moment coefficient of the airfoil is imposed for each flight segment. The relevant constraints are:

$$\begin{aligned} C_{D_p} &\geq f(\alpha, Re, c_{\max}, d_{c_{\max}}, \tau) \\ C_{L_{\text{MSES}}} &= f(\alpha, Re, c_{\max}, d_{c_{\max}}, \tau) \\ C_M &= f(\alpha, Re, c_{\max}, d_{c_{\max}}, \tau) \\ C_L &= C_{L_{\text{MSES}}} \\ C_M &\geq -0.06 \end{aligned} \tag{13}$$

Given the issues with Phase 3A, this phase once again takes the result of Phase 2 as the initial guess. The python script for Phase 3B is shown in Appendix F.

2. Examining the Phase Result

Although the objective, W_{fuel}, in Table 5 has increased from the previous Phase, the optimized result is now much more realistic. The angle of attacks, $\alpha_{\text{out}}, \alpha_{\text{sprint}}$, now resemble realistic α for a efficient cruise and sprint segment. The MSES result (Fig. 7) and the drag polar (Fig. 8) confirm that the airfoil has similar characteristics to the baseline NACA 2415 with a slight improvement in performance at the design $C_L = 0.55$. The corner problems highlighted in the result of Phase 3A are no longer present. The total fuel burn has also decreased by 1% from the baseline of Phase 2.

Table 5 Phase 3B optimal variables summary

Variable	Value	Units
W_{fuel}	6373.58	N
AR	18.02	-
$C_{D_p,\text{out}}$.	0.005518	-
$C_{D_i,\text{out}}$.	0.005576	-
$C_{L,\text{out}}$	0.5476	-
α_{out}	2.478	deg
α_{sprint}	-1.319	deg
S	28.29	m^2
W_{MTO}	37.53	kN
$c_{\max}$	0.0232	-
$d_{c_{\max}}$	0.3656	-
τ	0.150	-

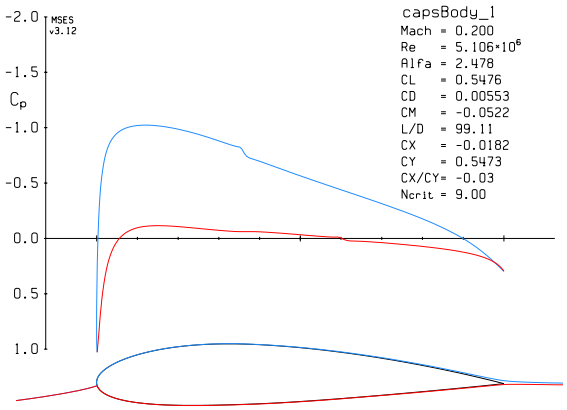


Fig. 7 Phase 3B MSES analysis, outbound conditions

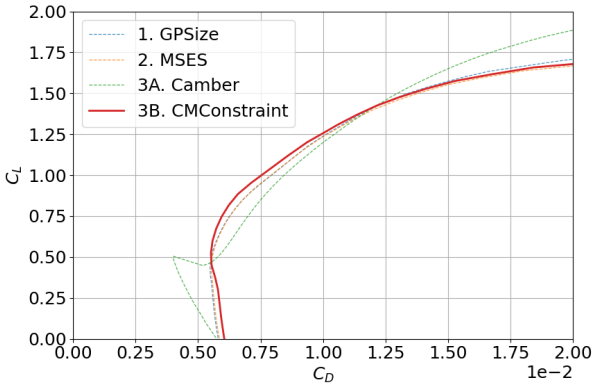


Fig. 8 Drag polars for the first four design phases

The NACA 4 series of airfoils are easy to work with and have an intuitive geometry definition, but are generally considered to be poorly performing airfoils, hence the need for the NACA 5 series and NACA 6 series. Thus, we seek to develop a geometry representation that can extend beyond the NACA 4 design space.

E. Phase 4A: Using a Kulfan CST2 Representation of the Airfoil Geometry

1. Phase Intent and Formulation

The Kulfan representation [15], which defines geometries given a Class function and Shape function Transformation (CST), is widely utilized for higher fidelity geometry representations. The Kulfan shape function spans the design space of any round nose, sharp aft-end airfoil and can be modified with n coefficients for the upper surface and n for the lower surface. For this phase, we consider only 2 Kulfan coefficients on each surface, for a total of 4 variables that define the airfoil. The design space therefore only increases by a single dimension, but takes on a significantly different underlying shape. Beyond seeking to keep the dimensionality down, there are significant issues with using large numbers of geometry variables to define airfoils in optimization [20].

Given the introduction of the Kulfan coefficients to the new design formulation, previous variables $c_{\max}$, $d_{c_{\max}}$, and τ are no longer necessary. However, the wing structural model still depends on τ, so an additional constraint that computes τ given Kulfan coefficients must be included. Packing the relevant Kulfan coefficients into vector $\mathbf{A}$, the new constraint set is as follows:

$$\begin{aligned} C_{D_p} &\geq f(\alpha, Re, \mathbf{A}) \\ C_{L_{\text{MSES}}} &= f(\alpha, Re, \mathbf{A}) \\ C_M &= f(\alpha, Re, \mathbf{A}) \\ C_L &= C_{L_{\text{MSES}}} \\ C_M &\geq -0.06 \\ \tau &= f(\mathbf{A}) \end{aligned} \tag{14}$$

This phase takes its initial guess from the successful Phase 3B. Because the Kulfan airfoil geometry is parameterized differently from NACA airfoils, a new CSM geometry script with Kulfan coefficient design parameters is required and is shown in Appendix B. The python script for Phase 4A is shown in Appendix G.

2. Examining the Phase Result

A first glance at the optimization output in Table 6 suggests a significant decrease in the objective, W_{fuel}, relative to Phase 3B from 6374 N to 4949 N. The vehicle weight, W_{MTO} also decreased significantly from 37.53 kN to 36.71 kN. However, the MSES output in Fig. 9 shows an airfoil that unrealistically maintains laminar flow as long as possible on both the top and bottom surfaces. The transition location, visible by the sharp change in the C_P, occurs on the top surface occurs around $x = 0.7c$ and on the bottom surface around $x = 0.85c$. In reality, maintaining laminar flow so far aft will be difficult.

Table 6 Phase 4A optimal variables summary

Variable	Value	Units
W_{fuel}	4949.48	N
AR	18.33	-
$C_{D_p,\text{out}}$	0.003076	-
$C_{D_i,\text{out}}$	0.004018	-
$C_{L,\text{out}}$	0.4673	-
S	27.74	m^2
W_{MTO}	36.71	kN
τ	0.149	-
A_{u_1}	0.181	-
A_{u_2}	0.433	-
A_{l_1}	-0.045	-
A_{l_2}	-0.188	-

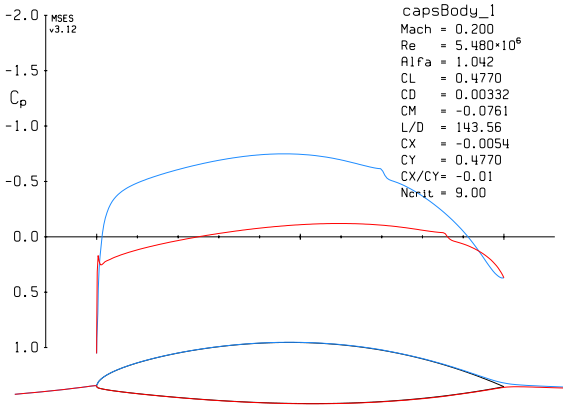
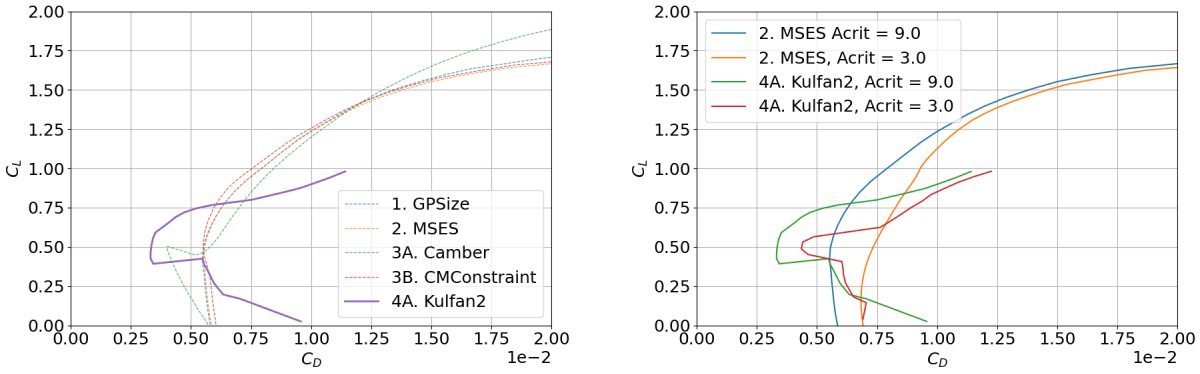


Fig. 9 Phase 4A MSES analysis, outbound conditions

This optimized airfoil has an incredibly narrow operating range between C_L values of approximately 0.4 and 0.7, but the benefit is a significant reduction in drag at cruise (Fig. 10a). Clearly this airfoil is not practical for many reasons, e.g. high takeoff and landing speeds due to low $C_{L_{\text{max}}}$ and poor stall performance due to the sharp leading edge. Even though clever operational procedures can mitigate these challenges, the airfoil's lack of robustness to increased turbulence poses a problem that cannot be easily mitigated. Dropping the N_{crit} value in MSES from 9.0 to 3.0 (representing significantly more turbulence in the incoming flow) produces the results shown in Fig. 10b. In the case of the Phase 2 airfoil, increased turbulence for the most part simply shifts the curve to the right – a drag penalty for sure, but not a fundamental change in the airfoil performance. In contrast, the increased turbulence takes the Phase 4 airfoil's already small operating range and shrinks it by more than half. Furthermore, the rightward shift of the Phase 4 airfoil polar represents a much higher proportional change, which will have ramifications throughout the entire optimal design. This airfoil is not practical or robust and therefore must be redesigned while accounting for transition location.



(a) First five design phases

(b) Comparing decreased values of N_{crit}

Fig. 10 Drag polars for the first five design phases

F. Phase 4B. FlowTrip: Tripping the Flow with Kulfan CST2

1. Phase Intent and Formulation

The primary issue with the design from Phase 4A was that free transition was allowed to take place well beyond where it would be reasonably expected in a real flow. Thus, for this phase, an MSES setting is adjusted to force transition to occur before or at 35% of the chord length along the upper surface of the airfoil. No changes are necessary to the

optimization formulation from Phase 4A, as this is an internal MSES setting, but practically it is an addition of one more constraint. This phase again takes the result of Phase 3B as the initial guess. The python script for Phase 4B is shown in Appendix H.

2. Examining the Phase Result

The optimization output in Table 7 now shows a modest decrease in the objective, W_{fuel}, relative to Phase 3B from 6374 N to 6223 N. The vehicle weight W_{MTO} is also essentially identical to the Phase 3B result. However, the problems with lack of robustness presented in the Phase 4A airfoil are no longer present in Fig. 11. Transition occurs at the imposed $x = .35c$ location for the top surface, and the drag polar in Fig. 12 shows that the corner has been eliminated.

Table 7 Phase 4B optimal variables summary

Variable	Value	Units
W_{fuel}	6222.56	N
AR	17.62	-
$C_{D_p,\text{out}}$	0.004882	-
$C_{D_i,\text{out}}$	0.005040	-
$C_{L,\text{out}}$	0.5147	-
S	28.28	m^2
W_{MTO}	37.52	kN
τ	0.150	-
A_{u_1}	0.262	-
A_{u_2}	0.296	-
A_{l_1}	-0.113	-
A_{l_2}	-0.127	-

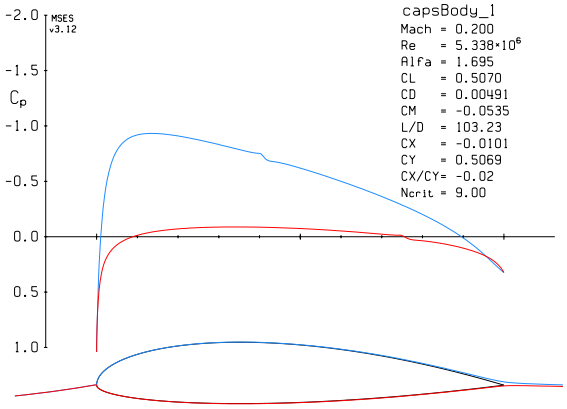


Fig. 11 Phase 4B MSES analysis, outbound conditions

In hindsight, this case should also have utilized a trip on the lower surface, which likely would have eliminated the slight reflex seen at the trailing edge in the pressure distribution. Further evidence that this airfoil (though optimal) is not a great design is that the minimum drag on the polar occurs at a much lower $C_L = 0.35$ than the design target at $C_L = 0.5$ (Fig. 12). A reasonable next phase would be to expand the design space to converge on a better performing airfoil at the optimum.

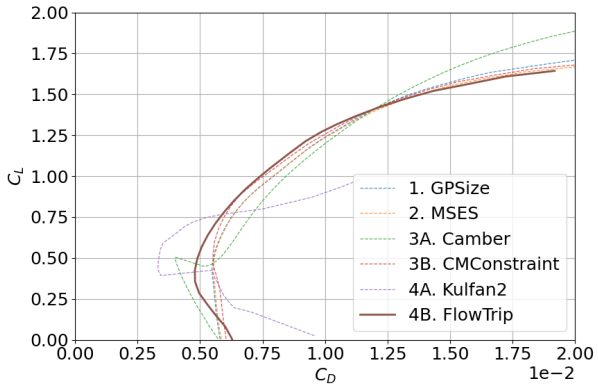


Fig. 12 Drag polars for the first six design phases

G. Phase 5. Kulfan4: Increasing Geometry Fidelity to Kulfan CST4

1. Phase Intent and Formulation

Although fuel burn decreased by expanding the design space to CST2, the optimal design can still be improved by increasing the geometry to 4 Kulfan coefficients on both top and bottom surfaces, for a total of 8 variables that define the airfoil. The optimization formulation once again does not change, though the length of vector **A** is now 8. Some airfoil shaping constraints are added to the Kulfan coefficients to ensure leading and trailing edges have reasonable minimum thicknesses, which can be seen in Appendix I, the python script for Phase 5. As before, the flow is tripped at 35% chord on the top surface, but allowed to transition freely on the bottom surface. This phase takes the result of Phase 4B as the initial guess.

2. Examining the Phase Result

The fuel burn W_{fuel} reported in Table 8 once again shows a notable decrease relative to the previous Phase 4B from 6222 N to 5948 N. The vehicle weight also has decreased by almost 1kN. The optimal airfoil shown in Fig. 13 is a different albeit reasonable geometry. The lower side transition remains unforced and occurs at $x = 0.65c$. The airfoil at this phase is again rather thick, indicating that perhaps the conservative structural model causes the optimization to converge on a corner of the design space that should be considered carefully.

Table 8 Phase 5 optimal variables summary

Variable	Value	Units	Variable	Value	Units
W_{fuel}	5947.8	N	A_{u_1}	0.192	-
AR	17.61	-	A_{u_2}	0.313	-
$C_{D_p,\text{out}}$	0.00488	-	A_{u_3}	0.179	-
$C_{D_i,\text{out}}$	0.00401	-	A_{u_4}	0.193	-
$C_{L,\text{out}}$	0.5053	-	A_{l_1}	-0.210	-
S	27.57	m^2	A_{l_2}	-0.088	-
W_{MTO}	36.58	kN	A_{l_3}	-0.191	-
τ	0.150	-	A_{l_4}	-0.058	-

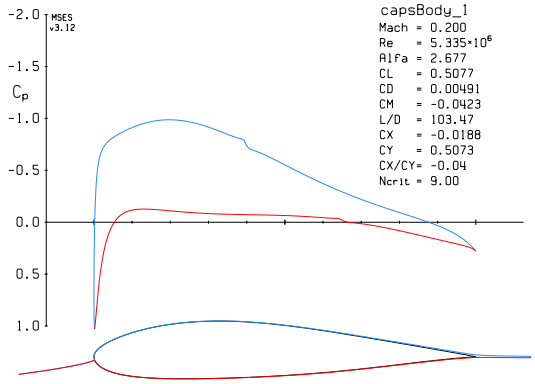


Fig. 13 Phase 5 outbound MSES airfoil analysis

The drag polar (Fig. 14) reveals that the optimizer is already starting to push for a single point optimum on at the operating point of $C_L = 0.5$, a phenomenon noted by Drela [20]. But overall performance is reasonable.

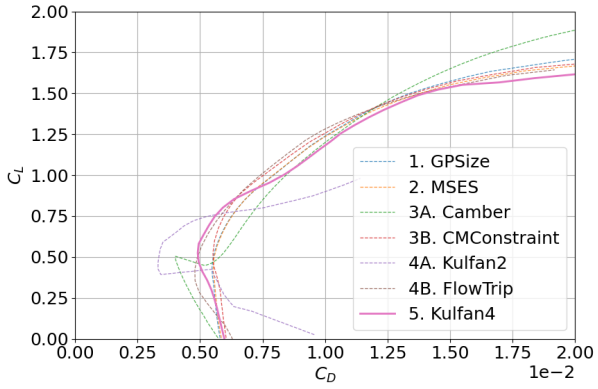


Fig. 14 Drag polars for the all 7 design phases

VI. Post Design Discussion

Phasing enabled a workflow to improve the fuel burn objective by incrementally refining the aerodynamic model and geometry fidelity. Phase 1 of the design optimized the aircraft using Geometric Programming with a drag model fitted to data points generated using XFOIL. Phase 2 represents a realistic baseline design, swapping out the fitted drag model out with runtime queries to MSES for better drag estimates that accounted compressible drag penalties. The two unsuccessful phases, 3A and 4A, provided valuable insight on additional constraints necessary to keep the optimization well-behaved in a higher dimensional design space, and at the completion of Phase 5, a total savings of 7.5% has been achieved over the baseline design in Phase 2. The history of the fuel burn in each phase shown in Fig. 15 with the unsuccessful phases shown in red and successful phases in blue.

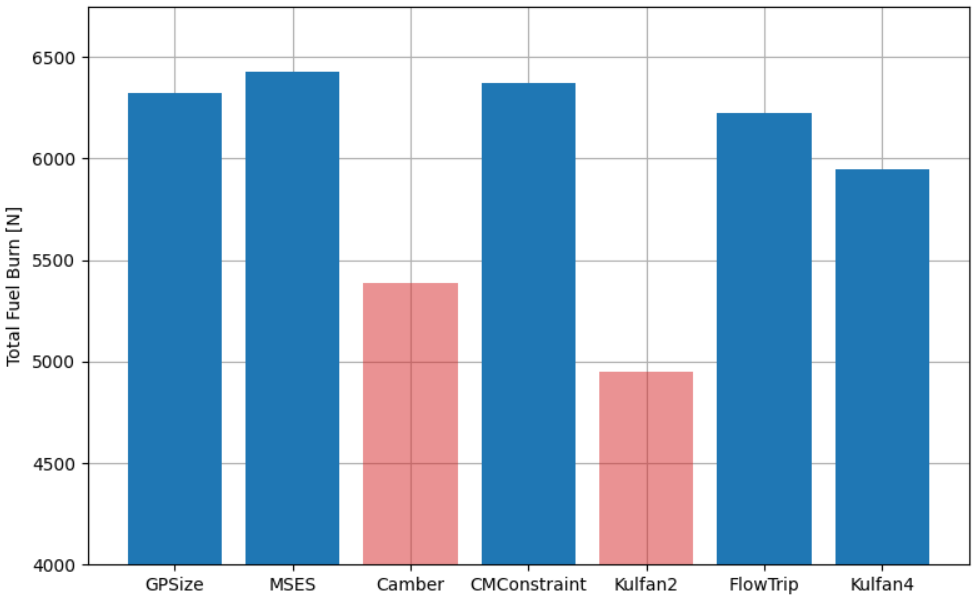


Fig. 15 Total fuel burn in each phase

VII. Conclusions

This paper walks the reader through a simple example that demonstrates the construction of a complete design system using ESP and its *Phasing* capability. The example starts out with a conceptual-level formulation with low modeling and geometric fidelity, then evolves to higher fidelity typically seen in preliminary design. ESP provides the ability to directly interface with both the geometry and the optimization, which was used to compute analytic derivatives from MSES with respect to airfoil geometry. The notion of ESP's *Phasing* makes it possible to decompose a design workflow and blur the line between conceptual and preliminary design. Although not included in this design example, one could easily further evolve the analysis models to higher fidelity by incorporating 3D CFD or FEA. Because ESP maintains the underlying parametric geometry and can automatically transfer data between analysis models, increasing the fidelity becomes as simple as replacing a few lines of the pyCAPS script (e.g. replacing the call to MSES for C_L, C_{D_p} with a call to 3D CFD to compute the full wing lift and drag). Beyond increasing analysis fidelity, future *Phases* could address and analyze performance in off-design conditions, study robustness of the design to uncertainties in the models and assumptions, or even study entirely new configurations for the same mission.

ESP's *Phasing* encompasses a single repository for the design and its associated analysis data. Because the repository tree is not static, any *Phase* can be taken as a starting point at any given time to answer "what if" questions on earlier portions of the design without rerunning the whole process. For example, new information during a later design phase could lead the designer to examine a new configuration and retrace the existing *Phases* to understand the implications of the changes. The repository also contains an archive of the tree and any notes or annotations as to the design decisions made. This may be useful within the *Digital Thread* arena. With access to the design decision tree, the reason for a particular configuration, shape or artifact can be tracked down to the pivotal point in the tree. This can provide crucial

understanding towards the actual lifetime/performance of the design, which in turn, can enlighten the designer/design engineer and discipline experts.

Appendix A: Phase 1-3B NACA Airfoil CSM

```
attribute capsIntent $LINEARAERO
attribute capsAIM    $msesAIM

cfgpmtr   view:MSES 0

dimension params 1 3 0
set params 0.10 0.1 6

despmtr    camber    0.02
despmtr    maxloc    0.40
despmtr    thickness 0.12
despmtr    area      30.0
despmtr    aspect    20.0
despmtr    taper     1.0

set     span    sqrt(area*aspect)
set     croot   area/span*2/(taper+1)
set     ctip    croot*taper
set     xtip    (croot-ctip)/2
outpmtr span

ifthen view:MSES NE 1
    MARK
        udparg naca thickness  thickness
        udparg naca camber     camber
        udparg naca maxloc     maxloc
        udparg naca sharpte    1
        udprim naca
        rotatex   90
        scale     ctip
        translate xtip -span/2 0
        store     wing -1 1

        udparg naca thickness  thickness
        udparg naca camber     camber
        udparg naca maxloc     maxloc
        udparg naca sharpte    1
        udprim naca
        rotatex   90
        scale     croot
        store     wing -1 1

        udparg naca thickness  thickness
        udparg naca camber     camber
        udparg naca maxloc     maxloc
        udparg naca sharpte    1
        udprim naca
        rotatex   90
        scale     ctip
        translate xtip span/2 0
        store     wing -1 1
    RULE

    store tessBody
    restore tessBody
    attribute .tParams params
else
    udparg naca thickness  thickness
    udparg naca camber     camber
    udparg naca maxloc     maxloc
    udparg naca sharpte    1
    udprim naca
endif

end
```

Appendix B: Phase 4A-5 Kulfan Airfoil CSM

```
attribute capsIntent $LINEARAERO
attribute capsAIM    $xfoilAIM;tsfoilAIM;msesAIM

cfgpmtr   view:MSES 0

dimension params 1 3 0
set params 0.10 0.1 6

despmtr   nparams 2
dimension class 1 2
dimension ztail 1 2
dimension aupper 1 nparams
dimension alower 1 nparams

despmtr    class    "0.5; 1.0;"
despmtr    ztail    "0.00; 0.00;"
despmtr    aupper   "0.2; 0.2;"
despmtr    alower   "-0.2; -0.2;"
despmtr    area     30.0
despmtr    aspect   20.0
despmtr    taper    1.0

set   span    sqrt(area*aspect)
set   croot   area/span*2/(taper+1)
set   ctip    croot*taper
set   xtip    (croot-ctip)/2
outpmtr span

ifthen view:MSES NE 1
    MARK
        udparg kulfan class class
        udparg kulfan ztail ztail
        udparg kulfan aupper aupper
        udparg kulfan alower alower
        udprim kulfan
        scale     ctip
        rotatex   90
        translate xtip -span/2 0
        store     wing -1 1

        udparg kulfan class class
        udparg kulfan ztail ztail
        udparg kulfan aupper aupper
        udparg kulfan alower alower
        udprim kulfan
        scale     croot
        rotatex   90
        store     wing -1 1

        udparg kulfan class class
        udparg kulfan ztail ztail
        udparg kulfan aupper aupper
        udparg kulfan alower alower
        udprim kulfan
        scale     ctip
        rotatex   90
        translate xtip span/2 0
        store     wing -1 1
    RULE

    store tessBody
    restore tessBody
    attribute .tParams params
else
    udparg kulfan class class
    udparg kulfan ztail ztail
    udparg kulfan aupper aupper
    udparg kulfan alower alower
    udprim kulfan
    extract 0
endif

end
```

Appendix C: Phase 1. GPSize

```
import os
import pyCAPS
import numpy as np
pi = np.pi
from corsairlite import units
from corsairlite.optimization import Formulation, Constant, Variable
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite.optimization.solve import solve

import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "naca.csm"),
                           phaseName="GPSize",
                           phaseContinuation=False,
                           outLevel=args.outLevel)

myProblem.intentPhrase(["Initial Geometric Programming sizing of wing"])

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
```

```
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

objective = W_fuel_out + W_fuel_ret

constraints = []

# ===================================
# SLF
# ===================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ===================================
# Landing
# ===================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ===================================
# Sprint
# ===================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ===================================
# Drag Model
# ===================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ===================================
# Propulsive Efficiency
# ===================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ===================================
# Range
# ===================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ===================================
# Weight
# ===================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ===================================
# Wing Structure
# ===================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ===================================
# GP compatible XFOIL fit drag model
# ===================================
for i in range(0,N_segments):
    constraints += [ 1 >= (2.56    * C_L[i]**( 5.88) * tau**(-3.32) * Re[i]**(-1.54) * C_Dp[i]**(-2.26) +
                           3.80e-9 * C_L[i]**(-0.92) * tau**( 6.23) * Re[i]**(-1.38) * C_Dp[i]**(-9.57) +
                           2.20e-3 * C_L[i]**(-0.01) * tau**( 0.03) * Re[i]**( 0.14) * C_Dp[i]**(-0.73) +
                           1.19e4  * C_L[i]**( 9.78) * tau**( 1.76) * Re[i]**(-1.00) * C_Dp[i]**(-0.91) +
                           6.14e-6 * C_L[i]**( 6.53) * tau**(-0.52) * Re[i]**(-0.99) * C_Dp[i]**(-5.19) ) ]

formulation = Formulation(objective, constraints)

# Limit thickness
bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Set to GP and solve
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'gp'
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
```

```
    ut = '{:C}'.format(rs.variables[vname].units)
    myProblem.parameter.create(vname, vl * pyCAPS.Unit(ut))

print(rs.result(10))

# Geometry quantities from optimization result
maxCamber = 0.02
maxLoc    = 0.4
thickness = rs.variables['tau'].to('').magnitude
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelTot   = rs.objective.magnitude
fuelTot  *= pyCAPS.caps.Unit(str(rs.objective.units))
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["camber"   ].value = maxCamber
myProblem.geometry.despmtr["maxloc"   ].value = maxLoc
myProblem.geometry.despmtr["thickness"].value = thickness
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper

myProblem.closePhase()
```

Appendix D: Phase 2. MSES

```
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from fullModelNACA import fullModel

import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm","naca.csm"),
                           phaseName="MSES",
                           phaseStart="GPSize",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Run MSES for better drag model"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
```

```
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add MSES relevant variables
Alpha_p20_MSES = []
C_L_MSES = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))


objective = W_fuel_out + W_fuel_ret

constraints = []

# ====================================
# SLF
# ====================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ====================================
# Landing
# ====================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ====================================
# Sprint
# ====================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ====================================
# Drag Model
# ====================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ====================================
# Propulsive Efficiency
# ====================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ====================================
# Range
# ====================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ====================================
# Weight
# ====================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ====================================
# Wing Structure
# ====================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ====================================
# MSES Runtime Drag Model
# ====================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.inputMode = 6
    fm.outputMode = 1
    fm.maxCamber = 0.02
    fm.camberDistance = 0.40
    fm.Mach = 0.2
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
    fm.problemObj = myProblem
```

```
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i]],['>=', '=='],[Alpha_p20_MSES[i],Re[i],tau],fm)]
constraints += [ C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new variables
newX0['Alpha_p20_MSES_0'] = 22 * units.deg
newX0['Alpha_p20_MSES_1'] = 22 * units.deg
newX0['Alpha_p20_MSES_2'] = 19 * units.deg
newX0['C_L_MSES_0'] = 0.5 * units.dimensionless
newX0['C_L_MSES_1'] = 0.5 * units.dimensionless
newX0['C_L_MSES_2'] = 0.2 * units.dimensionless

formulation.solverOptions.x0 = newX0
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,100,100)**0.6).tolist()
formulation.solverOptions.progressFilename = None
rs = solve(formulation)

print(rs.result(10))

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

# Geometric quantities from optimization result
maxCamber = 0.02
maxLoc    = 0.4
thickness = rs.variables['tau'].to('').magnitude
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["camber"   ].value = maxCamber
myProblem.geometry.despmtr["maxloc"   ].value = maxLoc
myProblem.geometry.despmtr["thickness"].value = thickness
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

myProblem.closePhase()
```

Appendix E: Phase 3A. Camber

```
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from fullModelNACA import fullModel

import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "naca.csm"),
                           phaseName="Camber",
                           phaseStart="MSES",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Vary airfoil max camber and camber location"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
```

```
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add MSES relevant variables and camber variables
maxCamber      = Variable(name = "maxCamber",        guess = 0.02,  units = "-",   description = "maximum airfoil camber scaled by chord")
camberLocation = Variable(name = "camberLocation",   guess = 0.5,   units = "-",   description = "location of maximum airfoil camber scaled by chord")
Alpha_p20_MSES = []
C_L_MSES = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))


objective = W_fuel_out + W_fuel_ret

constraints = []

# ====================================
# SLF
# ====================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ====================================
# Landing
# ====================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ====================================
# Sprint
# ====================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ====================================
# Drag Model
# ====================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ====================================
# Propulsive Efficiency
# ====================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ====================================
# Range
# ====================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ====================================
# Weight
# ====================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ====================================
# Wing Structure
# ====================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ====================================
# MSES Runtime Drag Model
# ====================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.inputMode = 5
    fm.outputMode = 1
    fm.Mach = 0.2
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
    fm.problemObj = myProblem
```

```
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i]],['>=', '=='],[Alpha_p20_MSES[i],Re[i],maxCamber,camberLocation,tau],fm)]
constraints += [ C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new variables
newX0['maxCamber']      = 0.03 * units.dimensionless
newX0['camberLocation'] = 0.6  * units.dimensionless

formulation.solverOptions.x0 = newX0
formulation.solverOptions.tau = 0.3
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,100,100)**0.6).tolist()
formulation.solverOptions.progressFilename = None
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

print(rs.result(10))

# Geometric quantities from optimization result
maxCamber = rs.variables['maxCamber'].to('').magnitude
maxLoc    = rs.variables['camberLocation'].to('').magnitude
thickness = rs.variables['tau'].to('').magnitude
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["camber"   ].value = maxCamber
myProblem.geometry.despmtr["maxloc"   ].value = maxLoc
myProblem.geometry.despmtr["thickness"].value = thickness
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

# myProblem.parameter["W_fuel"].value = fuelTot
# myProblem.parameter["CD"].value = CD_avg

myProblem.closePhase()
```

Appendix F: Phase 3B. CMConstraint

```
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from fullModelNACA import fullModel

import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "naca.csm"),
                           phaseName="CMConstraint",
                           phaseStart="MSES",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Add CM constraint to reduce aft camber"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
```

```
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add MSES and camber relevant variables
maxCamber      = Variable(name = "maxCamber",        guess = 0.02,  units = "-",   description = "maximum airfoil camber scaled by chord")
camberLocation = Variable(name = "camberLocation",   guess = 0.5,   units = "-",   description = "location of maximum airfoil camber scaled by chord")
Alpha_p20_MSES = []
C_L_MSES = []
negCMp1 = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))
    negCMp1.append(        Variable(name="negCMp1_%d"%(i), guess=1.05, units = "-", description="Negative CM about 0.25c + 1, segment %d"%(i)))

objective = W_fuel_out + W_fuel_ret

constraints = []

# ===================================
# SLF
# ===================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ===================================
# Landing
# ===================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ===================================
# Sprint
# ===================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ===================================
# Drag Model
# ===================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ===================================
# Propulsive Efficiency
# ===================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ===================================
# Range
# ===================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ===================================
# Weight
# ===================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ===================================
# Wing Structure
# ===================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ===================================
# MSES Runtime Drag Model
# ===================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.inputMode = 5
    fm.outputMode = 3
    fm.Mach = 0.2
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
```

```
    fm.problemObj = myProblem
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i], negCMp1[i]],['>=', '==', '>='],[Alpha_p20_MSES[i],Re[i],maxCamber,camberLocation,tau],fm)]
constraints += [ C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]
# Impose moment constraint
constraints += [negCMp1[i] <= 1.06 for i in range(0,N_segments)]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

names = [vr.name for vr in formulation.variables_only]
AR = formulation.variables_only[names.index('AR')]
S = formulation.variables_only[names.index('S')]

phantomConstraints = [
    AR >= 17,
    S <= 40 * units.m**2
]
formulation.constraints.extend(phantomConstraints)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new variables
newX0['maxCamber']      = 0.02 * units.dimensionless
newX0['camberLocation'] = 0.4 * units.dimensionless
newX0['negCMp1_0']      = 1.05 * units.dimensionless
newX0['negCMp1_1']      = 1.05 * units.dimensionless
newX0['negCMp1_2']      = 1.05 * units.dimensionless

formulation.solverOptions.x0 = newX0
formulation.solverOptions.tau = 0.3
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,100,100)**0.6).tolist()
formulation.solverOptions.progressFilename = None
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

print(rs.result(10))

# Geometric quantities from optimization result
maxCamber = rs.variables['maxCamber'].to('').magnitude
maxLoc    = rs.variables['camberLocation'].to('').magnitude
thickness = rs.variables['tau'].to('').magnitude
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["camber"   ].value = maxCamber
myProblem.geometry.despmtr["maxloc"   ].value = maxLoc
myProblem.geometry.despmtr["thickness"].value = thickness
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

myProblem.closePhase()
```

Appendix G: Phase 4A. Kulfan2

```
import dill
import copy
import sys
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from corsairlite.analysis.models.geometry.airfoilThickness.kulfan import computeThickness
from corsairlite.core.dataTypes.kulfan import Kulfan
from fullModelKulfan import fullModel
import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "kulfan.csm"),
                           phaseName="Kulfan2",
                           phaseStart="CMConstraint",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Increase geometry DOF with 2 parameter Kulfan airfoil"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1
myProblem.geometry.despmtr["nparams"].value = 2

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
```

```
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add Kulfan variables
Aupper1 = Variable(name = "Aupper1", guess = 1.2,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 1")
Aupper2 = Variable(name = "Aupper2", guess = 1.3,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 2 ")
Alower1 = Variable(name = "Alower1", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 1")
Alower2 = Variable(name = "Alower2", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 2")

Alpha_p20_MSES = []
C_L_MSES = []
negCMp1 = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))
    negCMp1.append(        Variable(name="negCMp1_%d"%(i), guess=1.05, units = "-", description="Negative CM about 0.25c + 1, segment %d"%(i)))

objective = W_fuel_out + W_fuel_ret

constraints = []

# ===================================
# SLF
# ===================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ===================================
# Landing
# ===================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ===================================
# Sprint
# ===================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ===================================
# Drag Model
# ===================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ===================================
# Propulsive Efficiency
# ===================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ===================================
# Range
# ===================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ===================================
# Weight
# ===================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ===================================
# Wing Structure
# ===================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ===================================
```

```
# MSES Runtime Drag Model
# ===================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.N = 2
    fm.inputMode = 4
    fm.outputMode = 3
    fm.Mach = 0.2
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
    fm.problemObj = myProblem
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i], negCMp1[i]],['>=', '==', '>='],[Alpha_p20_MSES[i],Re[i],Aupper1, Aupper2, Alower1, Alower2],fm)]
constraints += [C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]

variableStack = [Aupper1, Aupper2, Alower1, Alower2 ]
ct = computeThickness()
ct.N = 2
ct.inputMode = 2
constraints += [RuntimeConstraint([tau],['=='],variableStack,ct)]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new variables
afl = Kulfan()
afl.naca4_like(newX0['maxCamber'].to('').magnitude*100,
               newX0['camberLocation'].to('').magnitude*10,
               newX0['tau'].to('').magnitude*100 )
afl.changeOrder(2)
newX0['Aupper1'] = (   afl.upperCoefficients[0] + 1.0) * units.dimensionless
newX0['Aupper2'] = (   afl.upperCoefficients[1] + 1.0) * units.dimensionless
newX0['Alower1'] = (-1*afl.lowerCoefficients[0] + 1.0) * units.dimensionless
newX0['Alower2'] = (-1*afl.lowerCoefficients[1] + 1.0) * units.dimensionless
newX0['Alpha_p20_MSES_2'] = 22.0 * units.deg

formulation.solverOptions.x0 = newX0
formulation.solverOptions.tau = 0.3
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,100,100)**0.6).tolist()
formulation.solverOptions.progressFilename = None
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

print(rs.result(10))

# Geometric quantities from optimization result
Aupper1       = rs.variables['Aupper1'].to('').magnitude - 1
Aupper2       = rs.variables['Aupper2'].to('').magnitude - 1
Alower1       = -rs.variables['Alower1'].to('').magnitude - 1
Alower2       = -rs.variables['Alower2'].to('').magnitude - 1
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["aupper"   ].value = [Aupper1, Aupper2]
myProblem.geometry.despmtr["alower"   ].value = [Alower1, Alower2]
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

myProblem.closePhase()
```

Appendix H: Phase 4B. FlowTrip

```
import dill
import copy
import sys
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from corsairlite.analysis.models.geometry.airfoilThickness.kulfan import computeThickness
from corsairlite.core.dataTypes.kulfan import Kulfan
from fullModelKulfan import fullModel

import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "kulfan.csm"),
                           phaseName="FlowTrip",
                           phaseStart="CMConstraint",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Impose a turbulent flow trip at .35c in MSES"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1
myProblem.geometry.despmtr["nparams"].value = 2

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",          description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",          description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",            description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",            description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",             description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",             description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",             description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",        description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",         description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",       description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",            description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",            description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",       description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",       description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",       description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",       description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",          description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",          description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",            description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",            description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
```

```
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add Kulfan variables
Aupper1 = Variable(name = "Aupper1", guess = 1.2,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 1")
Aupper2 = Variable(name = "Aupper2", guess = 1.3,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 2 ")
Alower1 = Variable(name = "Alower1", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 1")
Alower2 = Variable(name = "Alower2", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 2")

Alpha_p20_MSES = []
C_L_MSES = []
negCMp1 = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))
    negCMp1.append(        Variable(name="negCMp1_%d"%(i), guess=1.05, units = "-", description="Negative CM about 0.25c + 1, segment %d"%(i)))

objective = W_fuel_out + W_fuel_ret

constraints = []

# ====================================
# SLF
# ====================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ====================================
# Landing
# ====================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ====================================
# Sprint
# ====================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ====================================
# Drag Model
# ====================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ====================================
# Propulsive Efficiency
# ====================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ====================================
# Range
# ====================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ====================================
# Weight
# ====================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ====================================
# Wing Structure
# ====================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
    tau <= 0.15,
    q <= 2
]
# ====================================
```

```
# MSES Runtime Drag Model
# ===================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.N = 2
    fm.inputMode = 4
    fm.outputMode = 3
    fm.Mach = 0.2
    fm.xTransition_Upper =  0.40
    fm.xTransition_Lower =  0.65
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
    fm.problemObj = myProblem
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i], negCMp1[i]],['>=', '==', '>='],[Alpha_p20_MSES[i],Re[i],Aupper1, Aupper2, Alower1, Alower2],fm)]

constraints += [C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]

constraints += [Alpha_p20_MSES[0] >= Alpha_p20_MSES[1]]
constraints += [Alpha_p20_MSES[1] >= Alpha_p20_MSES[2]]

constraints += [Alpha_p20_MSES[0] <= 25*units.deg ,
                    Alpha_p20_MSES[1] <= 25*units.deg ,
                    Alpha_p20_MSES[2] <= 20*units.deg ]
constraints += [Alpha_p20_MSES[0] >= 20*units.deg ,
                    Alpha_p20_MSES[1] >= 20*units.deg
                  ]

variableStack = [Aupper1, Aupper2, Alower1, Alower2 ]
ct = computeThickness()
ct.N = 2
ct.inputMode = 2
constraints += [RuntimeConstraint([tau],['=='],variableStack,ct)]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new variables
afl = Kulfan()
afl.naca4_like(newX0['maxCamber'].to('').magnitude*100,
               newX0['camberLocation'].to('').magnitude*10,
               newX0['tau'].to('').magnitude*100 )
afl.changeOrder(2)
newX0['Aupper1'] = (   afl.upperCoefficients[0] + 1.0) * units.dimensionless
newX0['Aupper2'] = (   afl.upperCoefficients[1] + 1.0) * units.dimensionless
newX0['Alower1'] = (-1*afl.lowerCoefficients[0] + 1.0) * units.dimensionless
newX0['Alower2'] = (-1*afl.lowerCoefficients[1] + 1.0) * units.dimensionless

formulation.solverOptions.x0 = newX0
formulation.solverOptions.tau = 0.3
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,300,300)**0.6).tolist()
formulation.solverOptions.progressFilename = None
formulation.solverOptions.debugOutput = True
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

print(rs.result(10))

# Geometric quantities from optimization result
Aupper1       = rs.variables['Aupper1'].to('').magnitude - 1
Aupper2       = rs.variables['Aupper2'].to('').magnitude - 1
Alower1       = -rs.variables['Alower1'].to('').magnitude - 1
Alower2       = -rs.variables['Alower2'].to('').magnitude - 1
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["aupper"   ].value = [Aupper1, Aupper2]
myProblem.geometry.despmtr["alower"   ].value = [Alower1, Alower2]
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

myProblem.closePhase()
```

Appendix I: Phase 5. Kulfan4

```
import dill
import copy
import sys
import os
import numpy as np
pi = np.pi
import pyCAPS
from corsairlite.optimization.solve import solve
from corsairlite.optimization import Variable, RuntimeConstraint, Constant, Formulation
from corsairlite.core.data.standardAtmosphere import atm
from corsairlite import units
from corsairlite.analysis.models.geometry.airfoilThickness.kulfan import computeThickness
from corsairlite.core.dataTypes.kulfan import Kulfan
from fullModelKulfan import fullModel
import argparse

# Setup and read command line options
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)

# Set up available commandline options
parser.add_argument("-outLevel", default=0, type=int, choices=[0, 1, 2], help="Set output verbosity")
args = parser.parse_args()

# Get the pyCAPS Problem object
myProblem = pyCAPS.Problem("hoburg",
                           capsFile=os.path.join("csm", "kulfan.csm"),
                           phaseName="Kulfan4",
                           phaseStart="FlowTrip",
                           phaseContinuation=True,
                           outLevel=args.outLevel)
myProblem.intentPhrase(["Increase geometry DOF with 4 parameter Kulfan airfoil"])

myProblem.geometry.cfgpmtr["view:MSES"].value = 1
myProblem.geometry.despmtr["nparams"].value = 4

# Define optimization formulation
N_segments = 3
# Constants
A_prop          = Constant(name = "A_prop",          value = 0.785,                 units = "m^2",         description = "Disk area of propeller")
CDA0            = Constant(name = "CDA0",            value = 0.05,                  units = "m^2",         description = "fuselage drag area")
C_Lmax          = Constant(name = "C_Lmax",          value = 1.5,                   units = "-",           description = "max CL with flaps down")
e               = Constant(name = "e",               value = 0.95,                  units = "-",           description = "Oswald efficiency factor")
eta_eng         = Constant(name = "eta_eng",         value = 0.35,                  units = "",            description = "Engine Efficiency")
eta_v           = Constant(name = "eta_v",           value = 0.85,                  units = "",            description = "Propeller Viscous Efficiency")
f_wadd          = Constant(name = "f_wadd",          value = 2.0,                   units = "",            description = "Added Weight Fraction")
g               = Constant(name = "g",               value = 9.81,                  units = "m/s^2",       description = "Gravitational constant")
h_fuel          = Constant(name = "h_fuel",          value = 46e6,                  units = "J/kg",        description = "Fuel Specific energy density")
k_ew            = Constant(name = "k_ew",            value = 0.0372,                units = "N/W^(0.803)", description = "Constant for engine weight")
mu              = Constant(name = "mu",              value = atm.mu(3000*units.m),  units = "kg/m/s",      description = "viscosity of air")
N_lift          = Constant(name = "N_lift",          value = 6.0,                   units = "-",           description = "Ultimate Load Factor" )
r_h             = Constant(name = "r_h",             value = 0.75,                  units = "-",           description = "Ratio of height at rear spar to maximum wing height")
rho             = Constant(name = "rho",             value = atm.rho(3000*units.m), units = "kg/m^3",      description = "density of air")
rho_cap         = Constant(name = "rho_cap",         value = 2700,                  units = "kg/m^3",      description = "Density of wing cap material (aluminum)")
rho_SL          = Constant(name = "rho_SL",          value = atm.rho(0*units.m),    units = "kg/m^3",      description = "density of air, sea level")
rho_web         = Constant(name = "rho_web",         value = 2700,                  units = "kg/m^3",      description = "Density of wing web material (aluminum)")
sigma_max       = Constant(name = "sigma_max",       value = 250,                   units = "MPa",         description = "Allowable tensile stress of aluminum" )
sigma_max_shear = Constant(name = "sigma_max_shear", value = 167,                   units = "MPa",         description = "Allowable shear stress of aluminum" )
w_bar           = Constant(name = "w_bar",           value = .5,                    units = "-",           description = "Ratio of spar box width to chord length" )
W_fixed         = Constant(name = "W_fixed",         value = 14700,                 units = "N",           description = "fixed weight")

# Vector Variables
V        = []
C_L      = []
C_D      = []
C_Dfuse  = []
C_Dp     = []
C_Di     = []
C_f      = []
T        = []
W        = []
Re       = []
eta_i    = []
eta_prop = []
eta_0    = []
z_bre    = []
for i in range(0,N_segments):
    V.append(        Variable(name="V_%d"%(i),        guess=1.0, units = "m/s", description="Velocity, segment %d"%(i)))
    C_L.append(      Variable(name="C_L_%d"%(i),      guess=1.0, units = "",    description="Lift Coefficient, segment %d"%(i)))
    C_D.append(      Variable(name="C_D_%d"%(i),      guess=1.0, units = "",    description="Drag Coefficient, segment %d"%(i)))
    C_Dfuse.append(  Variable(name="C_Dfuse_%d"%(i),  guess=1.0, units = "",    description="Fuselage Drag Coefficient, segment %d"%(i)))
    C_Dp.append(     Variable(name="C_Dp_%d"%(i),     guess=1.0, units = "",    description="Wing Profile Drag Coefficient, segment %d"%(i)))
    C_Di.append(     Variable(name="C_Di_%d"%(i),     guess=1.0, units = "",    description="Induced Drag Coefficient, segment %d"%(i)))
    C_f.append(      Variable(name="C_f_%d"%(i),      guess=1.0, units = "",    description="Friction Coefficient, segment %d"%(i)))
    T.append(        Variable(name="T_%d"%(i),        guess=1.0, units = "N",   description="Thrust, segment %d"%(i)))
    W.append(        Variable(name="W_%d"%(i),        guess=1.0, units = "N",   description="Weight, segment %d"%(i)))
    Re.append(       Variable(name="Re_%d"%(i),       guess=1e7, units = "",    description="Reynolds Number, segment %d"%(i)))
    eta_i.append(    Variable(name="eta_i_%d"%(i),    guess=1.0, units = "",    description="Inviscid Propeller Efficiency, segment %d"%(i)))
    eta_prop.append( Variable(name="eta_prop_%d"%(i), guess=1.0, units = "",    description="Propeller Efficiency, segment %d"%(i)))
    eta_0.append(    Variable(name="eta_0_%d"%(i),    guess=1.0, units = "",    description="Overall Efficiency, segment %d"%(i)))
    z_bre.append(    Variable(name="z_bre_%d"%(i),    guess=1.0, units = "",    description="Breguet Range Factor, segment %d"%(i)))

# Free Variables
AR         = Variable(name = "AR",         guess = 10.0,   units = "-",   description = "aspect ratio")
I_cap_bar  = Variable(name = "I_cap_bar",  guess = 1,      units = "-",   description = "Area moment of inertia of cap on 2D cross section, normalized by chord^4" )
M_r_bar    = Variable(name = "M_r_bar",    guess = 1000,   units = "N",   description = "Root Bending unit per unit chord" )
nu         = Variable(name = "nu",         guess = 0.5,    units = "-",   description = "Placeholder, (1+lam_w+lam_w**2)/(1+lam_w)**2" )
p          = Variable(name = "p",          guess = 3,      units = "-",   description = "Dummy Variable (1+2*lam_w)" )
P_max      = Variable(name = "P_max",      guess = 1000.0, units = "W",   description = "Maximum Engine Power")
q          = Variable(name = "q",          guess = 2,      units = "-",   description = "Dummy Variable (1+lam_w)" )
R          = Variable(name = "R",          guess = 5000,   units = "km",  description = "Single Segment range")
S          = Variable(name = "S",          guess = 10.0,   units = "m^2", description = "total wing area")
t_cap_bar  = Variable(name = "t_cap_bar",  guess = .05,    units = "-",   description = "Spar cap thickness per unit chord" )
t_web_bar  = Variable(name = "t_web_bar",  guess = .05,    units = "-",   description = "Spar web thickness per unit chord" )
```

```
tau        = Variable(name = "tau",        guess = 0.15,   units = "-",   description = "airfoil thickness to chord ratio")
V_stall    = Variable(name = "V_stall",    guess = 30.0,   units = "m/s", description = "stall speed")
W_cap      = Variable(name = "W_cap",      guess = 600,    units = "N",   description = "Weight of Wing Spar Cap" )
W_eng      = Variable(name = "W_eng",      guess = 5000,   units = "N",   description = "Engine Weight")
W_fuel_out = Variable(name = "W_fuel_out", guess = 2500,   units = "N",   description = "Weight of fuel, outbound")
W_fuel_ret = Variable(name = "W_fuel_ret", guess = 2500,   units = "N",   description = "Weight of fuel, return")
W_MTO      = Variable(name = "W_MTO",      guess = 2500,   units = "N",   description = "Maximum Takeoff Weight")
W_pay      = Variable(name = "W_pay",      guess = 5000,   units = "N",   description = "Payload Weight")
W_tilde    = Variable(name = "W_tilde",    guess = 5000,   units = "N",   description = "Dry Weight, no wing")
W_web      = Variable(name = "W_web",      guess = 400,    units = "N",   description = "Weight of Wing Spar Shear Web" )
W_wing     = Variable(name = "W_wing",     guess = 2500,   units = "N",   description = "wing weight")
W_zfw      = Variable(name = "W_zfw",      guess = 5000,   units = "N",   description = "Zero Fuel Weight")

# Add Kulfan variables
Aupper1    = Variable(name = "Aupper1", guess = 1.2,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 1")
Aupper2    = Variable(name = "Aupper2", guess = 1.3,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 2")
Aupper3    = Variable(name = "Aupper3", guess = 1.2,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 3")
Aupper4    = Variable(name = "Aupper4", guess = 1.3,  units = "-",   description = "Kulfan coefficient + 1, upper surface mode 4")
Alower1    = Variable(name = "Alower1", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 1")
Alower2    = Variable(name = "Alower2", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 2")
Alower3    = Variable(name = "Alower3", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 3")
Alower4    = Variable(name = "Alower4", guess = 1.2,  units = "-",   description = "-Kulfan coefficient + 1, lower surface mode 4")

Alpha_p20_MSES = []
C_L_MSES = []
negCMp1 = []
for i in range(0,N_segments):
    C_L_MSES.append(       Variable(name="C_L_MSES_%d"%(i),        guess=1.0,   units = "-",     description="Lift Coefficient from MSES, segment %d"%(i)))
    Alpha_p20_MSES.append( Variable(name="Alpha_p20_MSES_%d"%(i),  guess=22.0,  units = "deg",   description="Angle of Attack + 20deg from MSES, segment %d"%(i)))
    negCMp1.append(        Variable(name="negCMp1_%d"%(i), guess=1.05, units = "-", description="Negative CM about 0.25c + 1, segment %d"%(i)))

objective = W_fuel_out + W_fuel_ret

constraints = []

# ====================================
# SLF
# ====================================
for i in range(0,N_segments):
    constraints += [ W[i] == 0.5 * rho * V[i]**2 * C_L[i] * S ]
    constraints += [ T[i] >= 0.5 * rho * V[i]**2 * C_D[i] * S ]
    constraints += [ Re[i] == rho * V[i] * S**0.5 / (AR**0.5 * mu)]
# ====================================
# Landing
# ====================================
constraints += [
    W_MTO == 0.5 * rho_SL * V_stall**2 * C_Lmax * S,
    V_stall <= 38*units.m/units.s
]
# ====================================
# Sprint
# ====================================
constraints += [
    P_max >= T[2] * V[2] / eta_0[2],
    V[2] >= 150*units.m/units.s
]
# ====================================
# Drag Model
# ====================================
for i in range(0,N_segments):
    constraints += [ C_Dfuse[i] == CDA0/S ]
    constraints += [ C_Di[i] == C_L[i]**2/(np.pi * e * AR) ]
    constraints += [ C_D[i] >= C_Dfuse[i] + C_Dp[i] + C_Di[i] ]
# ====================================
# Propulsive Efficiency
# ====================================
for i in range(0,N_segments):
    constraints += [ eta_0[i] == eta_eng * eta_prop[i]]
    constraints += [ eta_prop[i] == eta_i[i] * eta_v]
    constraints += [ 4*eta_i[i] + T[i]*eta_i[i]**2 / (0.5 * rho * V[i]**2 * A_prop) <= 4]
# ====================================
# Range
# ====================================
constraints += [ R >= 5000 * units.km ]
for i in range(0,N_segments-1):
    constraints += [ z_bre[i] == g * R * T[i] / (h_fuel * eta_0[i] * W[i])]
constraints += [ W_fuel_out/W[0] >= z_bre[0] + z_bre[0]**2/2 + z_bre[0]**3/6 + z_bre[0]**4/24]
constraints += [ W_fuel_ret/W[1] >= z_bre[1] + z_bre[1]**2/2 + z_bre[1]**3/6 + z_bre[1]**4/24]
# ====================================
# Weight
# ====================================
constraints += [
    W_pay >= 500*units.kg * g,
    W_tilde >= W_fixed + W_pay + W_eng,
    W_zfw >= W_tilde + W_wing,
    W_eng >= k_ew * P_max**0.803,
    W_wing / f_wadd >= W_web + W_cap,
    W[0] >= W_zfw + W_fuel_ret,
    W_MTO >= W[0] + W_fuel_out,
    W[1] >= W_zfw,
    W[2] == W[0]
]
# ====================================
# Wing Structure
# ====================================
constraints += [
    2*q >= 1 + p,
    p >= 1.9,
    M_r_bar == W_tilde * AR * p / 24,
    0.92 * w_bar * tau * t_cap_bar**2 + I_cap_bar <= 0.92**2 / 2 * w_bar * tau**2 * t_cap_bar ,
    8 == N_lift * M_r_bar * AR * q**2 * tau / (S * I_cap_bar * sigma_max),
    12 == AR * W_tilde * N_lift * q**2 / (tau * S * t_web_bar * sigma_max_shear),
    nu**3.94 >= 0.86 * p**(-2.38) + 0.14*p**(0.56),
    W_cap >= 8 * rho_cap * g * w_bar * t_cap_bar * S**1.5 * nu / (3*AR**0.5),
    W_web >= 8 * rho_web * g * r_h * tau * t_web_bar * S**1.5 * nu / (3 * AR**0.5),
```

```
    tau <= 0.15,
    q <= 2
]
# ====================================
# MSES Runtime Drag Model
# ====================================
for i in range(0,N_segments):
    fm = fullModel()
    fm.N = 4
    fm.inputMode = 4
    fm.outputMode = 3
    fm.Mach = 0.2
    fm.xTransition_Upper =  0.40
    fm.xTransition_Lower =  0.65
    fm.Coarse_Iteration  =  50
    fm.Fine_Iteration    =  50
    fm.problemObj = myProblem
    constraints += [RuntimeConstraint([C_Dp[i], C_L_MSES[i], negCMp1[i]],['>=', '==', '=='],[Alpha_p20_MSES[i],Re[i],Aupper1, Aupper2, Aupper3, Aupper4,
                                                                                              Alower1, Alower2, Alower3, Alower4],fm)]

constraints += [C_L_MSES[i] == C_L[i] for i in range(0,N_segments)]

constraints += [Alpha_p20_MSES[0] >= Alpha_p20_MSES[1]]
constraints += [Alpha_p20_MSES[1] >= Alpha_p20_MSES[2]]
constraints += [Alpha_p20_MSES[0] <= 26*units.deg ,
                    Alpha_p20_MSES[1] <= 26*units.deg ,
                    Alpha_p20_MSES[2] <= 20*units.deg ]
constraints += [Alpha_p20_MSES[0] >= 20*units.deg ,
                    Alpha_p20_MSES[1] >= 20*units.deg]

variableStack = [Aupper1, Aupper2, Aupper3, Aupper4, Alower1, Alower2, Alower3, Alower4 ]
ct = computeThickness()
ct.N = 4
ct.inputMode = 2
constraints += [RuntimeConstraint([tau],['=='],variableStack,ct)]

# constrain kulfan coefficients to reasonable values
for kulfanCoeff in variableStack:
    constraints += [kulfanCoeff >= 1.0]
# constraints += [Aupper1 == Alower1]
# constraints += [Alower1 >= 1.18]
constraints += [
    Aupper3 >= 1.15,
    Alower1 >= 1.2,
    Aupper4 + Alower4 >= 2.25,
    Aupper1 + Alower1 >= 2.4
]

formulation = Formulation(objective, constraints)

bdc = [vr >= 1e-12 * vr.units for vr in formulation.variables_only]
formulation.constraints.extend(bdc)

# Get initial guess from previous phase parameters
newX0 = {}
for vname in myProblem.parameter.keys():
    vl = myProblem.parameter[vname].value
    if isinstance(vl, pyCAPS.Quantity):
        vl = vl.value() * getattr(units, str(vl.unit()))
    else:
        vl = vl * units.dimensionless
    newX0[vname] = vl

# Set initial guesses for new Kulfan4 variables
afl = Kulfan()
afl.upperCoefficients = np.array([   newX0['Aupper1'].magnitude-1,    newX0['Aupper2'].magnitude-1]) * units.dimensionless
afl.lowerCoefficients = np.array([-1*newX0['Alower1'].magnitude+1, -1*newX0['Alower2'].magnitude+1]) * units.dimensionless
afl.changeOrder(4)
newX0['Aupper1'] = (   afl.upperCoefficients[0] + 1.0) * units.dimensionless
newX0['Aupper2'] = (   afl.upperCoefficients[1] + 1.0) * units.dimensionless
newX0['Aupper3'] = (   afl.upperCoefficients[2] + 1.0) * units.dimensionless
newX0['Aupper4'] = (   afl.upperCoefficients[3] + 1.0) * units.dimensionless
newX0['Alower1'] = (-1*afl.lowerCoefficients[0] + 1.0) * units.dimensionless
newX0['Alower2'] = (-1*afl.lowerCoefficients[1] + 1.0) * units.dimensionless
newX0['Alower3'] = (-1*afl.lowerCoefficients[2] + 1.0) * units.dimensionless
newX0['Alower4'] = (-1*afl.lowerCoefficients[3] + 1.0) * units.dimensionless

formulation.solverOptions.x0 = newX0
formulation.solverOptions.tau = 0.3
formulation.solverOptions.solver = 'cvxopt'
formulation.solverOptions.solveType = 'slcp'
formulation.solverOptions.relativeTolerance = 1e-5
formulation.solverOptions.baseStepSchedule = [1.0] * 15 + (1/np.linspace(1,100,100)**0.6).tolist()
formulation.solverOptions.progressFilename = None
formulation.solverOptions.debugOutput = True
rs = solve(formulation)

# Save results as CAPS parameters
for vname in rs.variables.keys():
    vl = rs.variables[vname].magnitude
    ut = '{:C}'.format(rs.variables[vname].units)
    capsVar = vl * pyCAPS.Unit(ut)
    if vname in myProblem.parameter:
        myProblem.parameter[vname].value = capsVar
    else:
        myProblem.parameter.create(vname, capsVar)

print(rs.result(10))

# Geometric quantities from optimization result
Aupper1   =  rs.variables['Aupper1'].to('').magnitude - 1
Aupper2   =  rs.variables['Aupper2'].to('').magnitude - 1
Aupper3   =  rs.variables['Aupper3'].to('').magnitude - 1
Aupper4   =  rs.variables['Aupper4'].to('').magnitude - 1
Alower1   = -rs.variables['Alower1'].to('').magnitude - 1
```

```
Alower2   = -rs.variables['Alower2'].to('').magnitude - 1
Alower3   = -rs.variables['Alower3'].to('').magnitude - 1
Alower4   = -rs.variables['Alower4'].to('').magnitude - 1
area      = rs.variables['S'].to('m^2').magnitude
aspect    = rs.variables['AR'].to('').magnitude
taper     = rs.variables['q'].to('').magnitude - 1
# Fuel weight and drag from optimization result
fuelOut   = rs.variables['W_fuel_out'].to('N').magnitude
fuelRet   = rs.variables['W_fuel_ret'].to('N').magnitude
fuelTot   = fuelOut + fuelRet
CD0       = rs.variables['C_D_0'].to('').magnitude
CD1       = rs.variables['C_D_1'].to('').magnitude
CD_avg    = (CD0 + CD1) / 2

# Update geometry in ESP
myProblem.geometry.despmtr["aupper"   ].value = [Aupper1, Aupper2, Aupper3, Aupper4]
myProblem.geometry.despmtr["alower"   ].value = [Alower1, Alower2, Alower3, Alower4]
myProblem.geometry.despmtr["area"     ].value = area
myProblem.geometry.despmtr["aspect"   ].value = aspect
myProblem.geometry.despmtr["taper"    ].value = taper
myProblem.geometry.cfgpmtr["view:MSES"].value = 0

myProblem.closePhase()
```

Acknowledgements

The authors would like to thank Justin Gray for his input on the optimization formulation and Mark Drela for his help with MSES.

This work was funded by the **EnCAPS** project, AFRL Contract FA8650-20-2-2002 – Enhanced Computational Aircraft Prototype Syntheses, with Dr. Ryan Durscher as the Technical Monitor.

References

[1] Haimes, R., and Dannenhoffer, J. F., "The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry," 2013. https://doi.org/10.2514/6.2013-3073.

[2] Dannenhoffer, J. F., "OpenCSM: An open-source constructive solid modeler for MDAO," 2013. https://doi.org/10.2514/6.2013-701.

[3] Haimes, R., and Drela, M., "On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design," American Institute of Aeronautics and Astronautics, 2012. https://doi.org/10.2514/6.2012-683, URL https://arc.aiaa.org/doi/10.2514/6.2012-683.

[4] Galbraith, M. C., and Haimes, R., "A Parametric G1-continuous Rounded Wing Tip Treatment for Preliminary Aircraft Design," American Institute of Aeronautics and Astronautics, 2022. https://doi.org/10.2514/6.2022-1734, URL https://arc.aiaa.org/doi/10.2514/6.2022-1734.

[5] Galbraith, M. C., Allmaras, S. R., and Darmofal, D. L., "A verification driven process for rapid development of CFD software," American Institute of Aeronautics and Astronautics Inc, AIAA, 2015. https://doi.org/10.2514/6.2015-0818.

[6] Bhagat, N., and Alyanak, E., "Computational geometry for multifidelity and multidisciplinary analysis and optimization," American Institute of Aeronautics and Astronautics Inc., 2014. https://doi.org/10.2514/6.2014-0188.

[7] Dannenhoffer, J., and Haimes, R., "Conservative Fitting for Multi-Disciplinary Analysis," American Institute of Aeronautics and Astronautics, 2014. https://doi.org/10.2514/6.2014-0294, URL https://arc.aiaa.org/doi/10.2514/6.2014-0294.

[8] Dannenhoffer, J., and Haimes, R., "Generation of Multi-fidelity, Multi-discipline Air Vehicle Models with the Engineering Sketch Pad," American Institute of Aeronautics and Astronautics, 2016. https://doi.org/10.2514/6.2016-1925, URL https://arc.aiaa.org/doi/10.2514/6.2016-1925.

[9] Bryson, D. E., Haimes, R., and Dannenhoffer, J. F., "Toward the realization of a highly integrated, multidisciplinary, multifidelity design environment," American Institute of Aeronautics and Astronautics Inc, AIAA, 2019. https://doi.org/10.2514/6.2019-2225.

[10] Durscher, R., and Reedy, D., "Pycaps: A python interface to the computational aircraft prototype syntheses," American Institute of Aeronautics and Astronautics Inc, AIAA, 2019. https://doi.org/10.2514/6.2019-2226.

[11] 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.

[12] Hoburg, W., and Abbeel, P., "Geometric Programming for Aircraft Design Optimization," *AIAA Journal*, Vol. 52, No. 11, 2014, pp. 2414–2426. https://doi.org/10.2514/1.J052732.

[13] Drela, M., "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils," *Low Reynolds Number Aerodynamics*, edited by T. J. Mueller, Springer Berlin Heidelberg, Berlin, Heidelberg, 1989, pp. 1–12.

[14] Drela, M., "A User's Guide to MSES 3.05," , Jul 2017. URL https://web.mit.edu/drela/Public/web/mses/mses.pdf.

[15] Kulfan, B. M., "Universal Parametric Geometry Representation Method," *Journal of Aircraft*, Vol. 45, No. 1, 2008, pp. 142–158. https://doi.org/10.2514/1.29958.

[16] Karcher, C. J., "Logspace Sequential Quadratic Programming for Design Optimization," *AIAA Journal*, Vol. 60, No. 3, 2022, pp. 1471–1481. https://doi.org/10.2514/1.J060950.

[17] Karcher, C., and Haimes, R., "A Method of Sequential Log-Convex Programming for Engineering Design," *Optimization and Engineering*, 2022. https://doi.org/10.1007/s11081-022-09750-3.

[18] Karcher, C., "An Optimization Centered Approach to Multifidelity Aircraft Design," Ph.D. thesis, Massachusetts Institute of Technology, 2022.

[19] Palacios, F., Colonno, M. R., Aranake, A. C., Campos, A., Copeland, S. R., Economon, T. D., Lonkar, A. K., Lukaczyk, T. W., Taylor, T. W., and Alonso, J. J., "Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design," 2013. https://doi.org/10.2514/6.2013-287.

[20] Drela, M., "Pros & Cons of Airfoil Optimization," *Frontiers of Computational Fluid Dynamics 1998*, World Scientific, 1998, pp. 363–381. https://doi.org/10.1142/9789812815774_0019.