UNIVERSITY OF
GEORGIA
College of Engineering

# Lockheed Martin
# Aircraft Wing Design Optimization
# Final Report

May 4th, 2026

---


**Design Team:**

| | | |
|---|---|---|
| | Griffin Cantrell | gtc21014@uga.edu |
| | Edwin Harper | jeh71407@uga.edu |
| | Garrett Kuhn | gmk55337@uga.edu |
| | Cooper Mendlinger | cam03006@uga.edu |
| | Randal Richards | rjr23574@uga.edu |
| | Martin Vassilev | msv37717@uga.edu |
| **Project Supervisor** | Dr. Eliza Banu | eab43235@uga.edu |
| **Sponsor/Client** | Mr. Matthew Seay | matthew.d.seay@lmco.com |
| **Sponsor/Client** | Mr. Thomas Wells | thomas.h.wells@lmco.com |

# | Disclaimer |

The assumptions, findings, calculations, and conclusions expressed and described in this report and its exhibits were developed by undergraduate engineering students who are not licensed professional engineers. This report was prepared as an academic exercise as partial fulfillment of the College of Engineering Senior Design 4910/4911 course. No part of this report should be used for planning, budgeting, construction, or fiscal related decisions without a complete review and written endorsement from an independent, qualified, and licensed engineer who is willing and able to become the engineer of record for all aspects of the study, calculations, findings, recommendations, and the project.

A complete copy of this report was provided to the client without any financial reimbursement to its authors or the University of Georgia. The client may keep one copy of the report and is hereby given permission to copy and share the report as their needs dictate; however, a copy of this disclaimer shall accompany all copies made. By the acceptance of and/or use of this report and the exhibits hereto, the client and all reviewer of the content included herein shall indemnify and hold harmless the University of Georgia, College of Engineering, University employees, and the authors of this report from any and all liability, of whatsoever nature, that may result from such review, acceptance, or use.

# Table of Contents

# Executive Summary

This project develops a multi-fidelity aerodynamic wing optimization workflow that reflects the modeling and analysis practices used throughout contemporary aerospace design. Accurate aerodynamic prediction during the conceptual design phase is essential because performance shortfalls discovered later in development can lead to costly redesigns, increased fuel burn, and reduced mission capability. Fuel consumption represents a significant portion of long-term aircraft operating expense, and even small aerodynamic deficiencies can accumulate into substantial life-cycle penalties for military and commercial fleets [1]. Currently, many conceptual design processes can require extensive computing cost, time, and iteration, leading to inefficiencies in early aerodynamic evaluation and optimization. To address these concerns, this work establishes a structured and repeatable design process capable of supporting early decision-making for unmanned aircraft wing geometries.

The primary objective of this effort is to build an integrated optimization framework that combines parametric CAD modeling, low-fidelity aerodynamic prediction tools, structural estimation methods, and high-fidelity computational fluid dynamics, while applying strict geometric and structural constraints to promote real-world feasibility. The workflow incorporates OpenVSP for geometry generation, AVL for vortex-lattice aerodynamic evaluation, MATLAB and its Optimization Toolbox for executing optimization routines, and Ansys Fluent for validating performance using high-fidelity CFD simulations. This multi-tool approach enables rapid exploration of aerodynamic, geometric, and structural trade spaces while maintaining accessibility for student engineering teams. The methodology is grounded in modern optimization and multidisciplinary design strategies described in the aerospace literature [2–4], allowing the workflow to scale in fidelity and complexity as the design matures.

Throughout the project timeline, the team will continue refining model interactions, improving geometry parameterization, and strengthening consistency between low- and high-fidelity predictions. Classical aircraft design principles from Raymer and Nikolai provide additional structure for evaluating sizing relationships, aerodynamic trends, and geometric feasibility, ensuring that computational results remain aligned with accepted design theory [10, 11]. These improvements support the long-term goal of

creating a workflow that is adaptable to multiple aircraft configurations, mission scenarios, and performance criteria, thereby establishing a foundation that future design teams can expand upon.

The resulting optimization process is expected to generate wing geometries with improved lift-to-drag ratio, reduced drag, and enhanced overall aerodynamic performance. These improvements have direct implications for extending mission range, lowering fuel consumption, and improving endurance, all of which contribute to operational effectiveness and long-term sustainability [1]. The project also provides meaningful educational value by giving students hands-on experience with computational tools, optimization algorithms, and multidisciplinary analysis frameworks used by aerospace organizations such as Lockheed Martin. By integrating modern computational practices with established conceptual-design theory, the workflow equips students with the skills needed to contribute to innovative and efficient aircraft systems in future engineering roles. Our final deliverables for this project are delivering a repeatable optimization workflow to our client along with a comprehensive report and client presentation to demonstrate that our multi-fidelity wing design processes were developed by engineering methodologies, rigorous validation, and data-driven solutions.

# Design Description – Project Understanding

## 1. Project Background

This project develops a structured multi-fidelity aerodynamic wing optimization workflow that reflects the modeling methodologies employed in contemporary aerospace vehicle design. Early-stage aerodynamic assessment requires a deliberate balance between computational efficiency and predictive accuracy, and modern conceptual design workflows routinely integrate low-fidelity and high-fidelity analysis to support rapid trade-space exploration [10, 11]. Following these practices, the present effort combines parametric CAD modeling, vortex-lattice aerodynamic prediction, structural estimation, and high-fidelity computational fluid dynamics to generate and refine wing geometries for unmanned aircraft applications. This integrated methodology provides a systems-level understanding of wing design by linking geometry, aerodynamics, and optimization theory within a repeatable engineering framework [2, 4].

The broader aerospace industry continues to evolve toward highly digital, simulation-driven design processes. Advances in computational capability have reduced the need for exclusive reliance on wind-tunnel prototyping and have enabled rapid exploration of aerodynamic concepts during preliminary design. Wing optimization remains a central focus of these developments, as variations in camber, chord length, twist, winglets, and structural tailoring play critical roles in improving performance, fuel efficiency, stability, and mission capability. Multi-fidelity workflows—now used extensively by organizations such as Lockheed Martin, Boeing, and Airbus—provide the ability to iterate quickly while reserving expensive high-fidelity simulations for validation. Additional background on these industry trends is provided in Appendix A.

A benchmarking study was conducted to evaluate commercial and open-source optimization environments and identify tools most appropriate for conceptual aerodynamic design. Commercial design and MDO platforms provide advanced workflow automation and solver coupling but require substantial licensing fees and offer limited transparency into algorithmic behavior. In contrast, open-source and

academically accessible tools—including OpenVSP, AVL, MATLAB, and Ansys Fluent—offer the flexibility, control, and accessibility needed for iterative student-driven development [5–9]. The comparative analysis and supporting evidence from this benchmarking effort are summarized in Appendix A.

An intellectual property review was also performed to ensure that the project's parameterization strategies and optimization procedures remain consistent with existing aerospace design patents. The aerospace sector is characterized by extensive historical precedent, making many design concepts difficult to protect. As noted by Mr. William "Bob" Clark of Lockheed Martin, most ideas in aerospace have been explored in some form, underscoring the need to understand prior work, identify limitations, and develop improved combinations of established methods. Accordingly, the workflow developed in this project relies exclusively on public-domain aerodynamic theory, open-source modeling tools, and parameterizations created specifically for this research. Additional IP context and compliance documentation are provided in Appendix A.

## 2. Project Needs

This project involves several core stakeholders whose needs shape the requirements of the multi-fidelity aerodynamic wing optimization workflow. The primary customer is the Lockheed Martin conceptual design team, represented by project sponsors and senior engineers. Their expectations, identified through interview questions and responses documented in Appendix A, include accurate and traceable aerodynamic modeling, clearly defined assumptions, and a workflow capable of supporting rapid conceptual-design trades. They also require consistent milestone reporting, professional documentation, and validated outputs that can be compared directly with existing Lockheed methodologies. Their role and influence are illustrated in the stakeholder map located in Appendix B.

Secondary stakeholders include the UGA Wind Tunnel Team and the faculty supervisor. Their needs, derived from the interview responses summarized in Appendix B, emphasize reliable geometry handoff, consistent data formatting, and alignment between computational predictions and physical testing procedures. These stakeholders depend on well-organized code, reproducible analysis, and clear communication regarding assumptions and expected aerodynamic behavior. Their relationships and relative influence are also shown in the stakeholder map in Appendix B.

Tertiary stakeholders, such as commercial pilots providing operational insights and previous Lockheed-aligned capstone teams, contribute additional context to performance requirements and validation needs. Their expectations were gathered through targeted interview questions and discussions documented in Appendix A, focusing on flight-relevant metrics, standardized data structures, and compatibility with earlier project baselines. These groups rely on transparent methodology, user-friendly output, and clearly interpreted results to support comparison or operational understanding. Their placement within the broader ecosystem of the project is captured in the stakeholder map in Appendix B. Collectively, the stakeholder interviews and mapping efforts highlight the need for a workflow that is technically rigorous, transparent, and adaptable. The consolidated customer needs include:

- A repeatable multi-fidelity optimization process with verifiable assumptions
- Rapid but reliable conceptual-level performance estimates
- High-fidelity validation for accuracy and credibility
- Standardized documentation, file structures, and communication channels

• Outputs that are meaningful to engineers, supervisors, pilots, and testing teams
• Professional quality aligned with Lockheed Martin expectations

## 3. Design Objective

The objective of this project is to develop a repeatable, CAD-centric wing design optimization pipeline capable of reducing drag and improving aerodynamic efficiency while maintaining realistic geometric and structural constraints. Current conceptual design processes require significant time, manual iteration, and the use of disconnected simulation environments, leading to inefficiencies in early aerodynamic evaluation. This project addresses the lack of a unified, transparent, multi-fidelity workflow that can generate, analyze, and refine aircraft wing geometries with both speed and accuracy.

The goal of this project is to create and document an end-to-end wing optimization process that integrates CAD modeling, low- and high- fidelity aerodynamic analysis, structural considerations, and optimization methods into a reproducible digital workflow. This workflow is intended to enable rapid translation of conceptual wing designs into validated aerodynamic performance metrics and optimized geometric outputs for unmanned aircraft systems. In collaboration with Lockheed Martin, this project also aims to provide students with hands-on experience in real-world aerodynamic optimization, emphasizing the balance between accuracy, efficiency, and design trade-offs using industry-relevant tools and workflows.

The final deliverables will include a comprehensive report and presentation summarizing the developed optimization workflow, along with the team's optimized wing design and the rationale behind key decisions. This documentation will also highlight trade-offs, assumptions, and lessons learned throughout the process.

To achieve these goals, the project will focus on the following objectives: developing a CAD-driven parametric wing model suitable for automized optimization, integrating low-fidelity aerodynamic methods for rapid concept iteration, automating the optimization process using MATLAB, Python, and compactible CAD/CFD tools, producing clear comparisons between baseline and optimized wing configurations, including lift-to-drag ratio, drag reduction, and mission-relevant performance details, and documenting the complete workflow, including modeling decisions, solver settings, assumptions, and parameter sensitivities.

## 4. Engineering Requirements, Specifications and Codes, Standards, and Regulations

To ensure the Aircraft Wing Optimization Workflow meets the rigorous demands of multi-fidelity aerodynamic analysis, the team has established a set of quantifiable engineering specifications derived directly from stakeholder needs and the project charter. These requirements govern the pipeline's performance across three critical dimensions: computational efficiency, aerodynamic accuracy, and software interoperability. By defining clear acceptance criteria for each stage of fidelity, from the rapid conceptual design in OpenVSP to the high-fidelity validation in Ansys, the team established a concrete framework for verification. The resulting specifications were used to guide design decisions, evaluate workflow performance, and confirm that the final process aligned with customer expectations and project objectives.

The engineering requirements were organized around stakeholder priorities such as data integrity, quality control, cost efficiency, workflow functionality, and repeatability. These needs were translated into measurable design requirements so that the workflow could be evaluated against objective criteria rather than subjective judgement. In practice, this meant the workflow had to preserve data between software environments, produce aerodynamic results that were traceable and credible, remain computationally efficient enough for iterative use, and support reuse across multiple wing configurations. Table 1 summarized the major stakeholder needs, corresponding design requirements, acceptance criteria, and the standards or regulations adopted for this project.

**Table 1. Engineering Specifications Description and Comparison**

| Category | Stakeholder Needs | Design Requirements | Acceptance Criteria | Qualifications | Regulatory Requirements |
|---|---|---|---|---|---|
| Safety | Data Integrity | Ensure workflow with secure computational environment | Workflow runs safely without risk of corrupted simulations or data loss | Requirement | STD-8719.13C (Software Safety) |
| Quality Control | Accurate aerodynamic results | Workflow produces traceable and precise results | CFD results and low-fidelity models validated against known standards | Requirement | AIAA R-101-2013 (CFD Verification & Validation) |
| Cost | Affordable and efficient use of resources | Efficient resource usage | Optimized computational workflow | Requirement | N/A |
| Functionality | Maintainability | Multi-fidelity integration | Successfully links CAD to low and high-fidelity tools without data loss | Requirement | ISO 10303 (STEP format interoperability standard) |
| Repeatability | Must be reusable for given wing configurations | Modular Workflow with input parameters | Re-run workflow on various wing configurations | Requirement | N/A |

The specifications in Table 1 served as the basis for verification throughout the project. Rather than treating the workflow as a single software product, the team evaluated each major stage independently and as part of an integrated pipeline. This approach helped ensure that the workflow was not only capable of generating optimized wing geometriess, but also capable of doing so in a repeatable, traceable, and technically defensible manner.

## QFD Discussion

A Quality Function Deployment (QFD) matrix in the Appendix was developed earlier in the project to translate stakeholder needs into technical characteristics of the optimization workflow. The QFD connected customer-focused goals such as maximizing cruise range, reducing drag, ensuring structural integrity, maintaining manufacturability, and supporting a repeatable digital workflow to technical design drivers including planform geometry, CST airfoil coefficients, structural weight, flap sizing, and CFD/aerodynamic solver fidelity. This allowed the team to systematically identify which technical areas had the largest impact on customer value and therefore deserved the greatest design emphasis.

The QFD results showed that CFD and aerodynamic solver fidelity carried the highest technical importance, followed by planform geometry and CST airfoil definition. This result aligned with the project objective of building a workflow that could rapidly generate concepts while still producing trustworthy aerodynamic predictions. Structural weight remained important because it affects mission performance and design realism, while flap sizing ranked lower because it had less influence on the primary cruise-focused optimization objective. These rankings helped the team prioritize refinement effort and justify where modeling time should be invested.

The competitive evaluation portion of the QFD compared the team's workflow against three reference baselines: a manual or legacy design process, basic disconnected analysis tools, and comparable academic workflows. In that comparison, the proposed workflow scored highest in cruise-range improvement, drag reduction, and repeatable digital integration because it combined parametric geometry generation, automated analysis, and multi-fidelity evaluation in a single process. The benchmarking also showed that although manufacturability and structural realism remained important, the largest competitive advantage came from reducing manual iteration and improving the consistency of aerodynamic decision-making.

Technical benchmarking was then used to convert these QFD findings into design priorities. CFD/aero fidelity was ranked as the top improvement priority because it was the primary means of validating whether the optimized geometry produced credible aerodynamic gains. Planform geometry and CST airfoil coefficients followed closely because they directly controlled drag, lift-to-drag ratio, and range performance. Structural weight was treated as a supporting but necessary benchmark for realism, while flap sizing was retained as a lower-priority variable within the broader optimization framework. The complete QFD, including relationship scores, competitive evaluation, and technical benchmarking, is provided in the appendices.

## Codes, Standards, and Regulations

Several standards and technical references were adopted to make the workflow more defensible and aligned with accepted aerospace engineering practice. NASA-STD-8719.13C was used as a reference for software safety concepts, particularly in the context of managing risks associated with software-driven workflows, preserving data integrity, and ensuring that computational processes were planned and documented in a controlled way. Although the project did not involve flight-critical embedded software, this standard still provided useful guidance for identifying and reducing software-related process risk.

AIAA R-101-2013 was adopted to support verification and validation practices for CFD. This was especially relevant during the higher-fidelity analysis stage, where the credibility of simulation results depended on appropriate solver setup, consistency checks, and comparison to accepted aerodynamic behavior or reference cases. Using this guidance helped the team justify that CFD was not being treated as a black box, but rather as an engineering tool whose results required verification before being used for design decisions.

ISO 10303, commonly implemented through STEP file exchange, was adopted to support CAD interoperability between tools in the workflow. Because the project depended on transforming geometry between modeling and analysis environments, maintaining a vendor-neutral exchange standard reduced the risk of geometry corruption, incompatible file formats, and loss of design intent during handoff between software packages. This standard was particularly important for preserving a CAD-centric workflow and supporting the repeatability of the project.

Together, these standards improved the rigor of the design process by addressing three separate concerns: software/process safety, CFD credibility, and CAD interoperability. Their use strengthened the engineering basis of the workflow and ensured the final design process was not only functional, but also better aligned with accepted professional practice in aerospace design.

# 5. Prototype / Product Development

## Design Description

The notebook implements a Multi-Disciplinary Design Optimization (MDO) framework with staged fidelity for an aircraft wing. The central design goal is to maximize cruise range (via the Breguet range equation) by simultaneously optimizing 39 design variables: planform geometry (span, taper, sweep, angle of attack), airfoil shape at root and tip using Class-Shape Transformation (CST) coefficients (32 variables), wing structural weight (as an implicit Raymer variable), and flap sizing (chord ratio and span fraction). The design supports five aircraft classes, GA, Transport, Fighter, Business Jet, and UCAV , each with its own physics-calibrated presets. The flowchart of the MDO, shown in Figure X. below, showcases the process of building the 3D wing geometry.

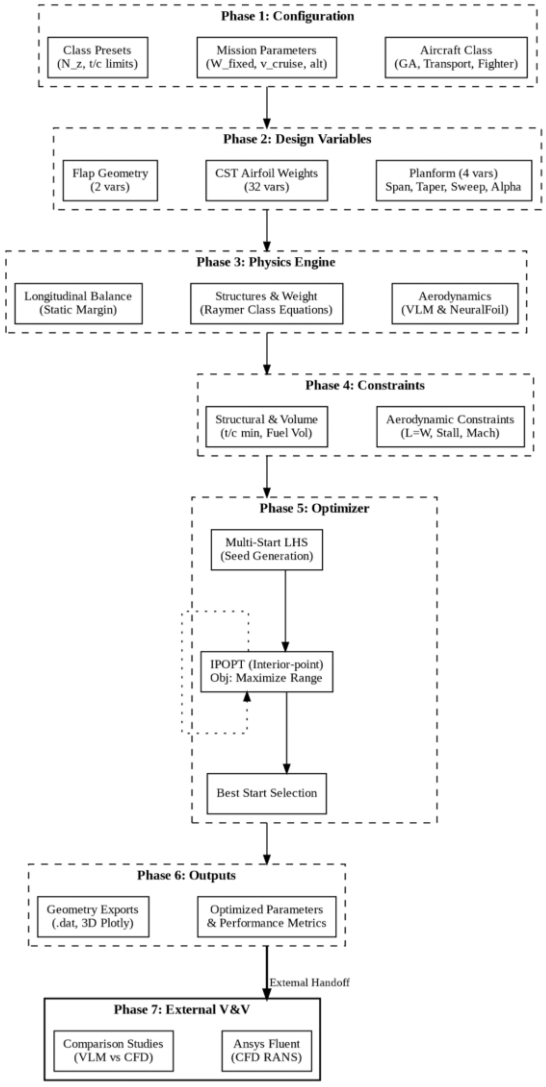



**Fig 1.** Aircraft Wing MDO Flowchart

## Component Analysis

**1. Aerodynamic Analysis (NeuralFoil + VLM)** The aerodynamics are computed using AeroSandbox's AeroBuildup, which internally calls **NeuralFoil, a neural network trained on XFOIL data, to evaluate 2D viscous drag and lift at each spanwise station. The wing is discretized into 6 spanwise cross-sections** (stations at $\eta = 0, 0.2, 0.4, 0.6, 0.8, 1.0$) to accurately capture the nonlinear drag-versus-t/c relationship across the span. A **Vortex Lattice Method (VLM)** with 24 spanwise × 12 chordwise panels is run in post-processing to compute spanwise lift distribution and compare it against the elliptical ideal.

**2. Wave Drag Correction (Korn + Lock)** Because NeuralFoil doesn't apply swept-wing Mach correction, there is an explicit correction in the **Korn drag-divergence equation** to compute compressive flow penalties as a function of sweep, t/c, and CL. It then uses a smooth transition to preserve gradient continuity for the IPOPT solver. This gives the optimizer a physically correct incentive to use aft sweep at transonic speeds.

**3. Structural Analysis (Raymer Empirical Weight)** Wing structural weight is computed using **Raymer's empirical equations** (Table 15.2 for GA, 15.46 for Transport/Business Jet, 15.25 for Fighter/UCAV), all enforced as NLP equality constraints. This creates a self-consistent algebraic weight loop (wing weight → gross weight → dynamic pressure → wing weight) that IPOPT resolves simultaneously with all aerodynamic constraints. This makes it a true MDO process.

**4. Airfoil Shape (CST Parameterization)** Airfoil geometry is parameterized with an 8-coefficient Bernstein polynomial CST basis at both root and tip (16 upper + 16 lower weights = 32 variables). Thickness is evaluated analytically at $x/c = 1/3$ (the CST maximum-thickness point, near the forward spar), and cross-sectional area is computed exactly using Beta function integrals for the fuel volume constraint.

**5. Constraint System** The optimizer enforces: lift = weight at cruise; minimum t/c at root ($\geq 0.15$ for GA) and tip; spar slenderness (semi-span / spar depth $\leq 20$); fuel volume $\geq$ required volume; approach stall speed ($1.2 \cdot V_{stall} \leq V_{app\_max}$); and static margin within ±10% MAC. The static margin constraint uses an absolute fuselage coordinate model that accounts for wing position fraction, CG of fixed weight (payload/structure), and horizontal tail NP contribution.

**6. Fuselage Geometry** A proportional 5-section body of revolution is generated from class-specific fractions of the MAC, with nose fineness, tail exit radius, and length all scaling continuously with the optimizer's taper variable. This allows AeroBuildup to compute fuselage viscous and pressure drag (using Hoerner's form factor) within the optimization loop.

## Simulation Integration

The optimization uses **IPOPT** (Interior Point OPTimizer) via CasADi's NLP interface. To avoid local optima, a **35-start Latin Hypercube Sampling (LHS)** multi-start strategy uniformly samples all 39

variable bounds, with the physics-consistent warm start (class-appropriate span midpoint, Raymer-estimated wing weight) always assigned to start 0. The best feasible solution by range is selected.

## Design Evaluation & Visualization

Post-processing generates a complete evaluation suite:

- **Airfoil profiles** and camber/thickness distributions at root and tip
- **NeuralFoil aerodynamic polars** (CL vs. CD and CL vs. α) at cruise Mach and Reynolds number
- **Spanwise lift distribution** (VLM) vs. elliptical reference
- **Top-view planform** with flap/aileron regions, MAC indicator, and fuselage silhouette
- **3D wing surface** rendered with Plotly
- **Pressure distribution (ΔCp)** on upper and lower surfaces with chordwise and spanwise profile slices
- **Convergence history** (range, L/D, span, sweep, alpha, taper vs. IPOPT iteration) and design-space trajectory plots
- **Weight breakdown** and **drag breakdown** (wing, fuselage, wave, miscellaneous)
- **Multi-start scatter** and parallel coordinates plots across all feasible starts

## Increasing Fidelity: Computational Fluid Dynamics VandV

The final phase of the design process involves an external handoff to high-fidelity verification and validation tools, serving as the critical bridge between the MDO optimizer's conceptual results and real-world aerodynamic confidence. In this phase, the optimized wing geometry produced by the IPOPT solver is exported and cross-examined through two parallel workflows. First, a direct comparison study is conducted between the Vortex Lattice Method (VLM) results generated internally by AeroSandbox and a full Reynolds-Averaged Navier-Stokes (RANS) CFD solution, exposing the limitations of inviscid panel theory, such as the absence of flow separation, viscous boundary layer effects, and shock-induced separation at transonic conditions, against a physics-complete simulation. Second, the validated geometry is imported into Ansys Fluent, where a RANS CFD analysis is performed on the optimized wing surface using an industry-standard turbulence model (such as the k-ω SST) to resolve the full viscous flow field, surface pressure distribution, and skin friction contributions. The agreement or discrepancy between VLM predictions and RANS results quantifies the fidelity gap inherent in the optimization framework, and any significant deviation in lift, drag, or pressure distribution informs either a design revision or an update to the drag correction models used within the optimizer. This phase ultimately validates that the aerodynamic performance metrics projected during optimization: cruise L/D, spanwise load distribution, and wave drag, are physically realizable in a production-fidelity analysis environment.

# 6. Prototype Refining & Evaluation

The wing design was prototyped through a multi-stage digital workflow centered on a CAD-based, multi-disciplinary design optimization (MDO) framework. The initial prototype consisted of a parametric wing model built from planform variables, airfoil geometry defined using Class-Shape Transformation (CST) coefficients, structural weight estimates, and flap sizing parameters. This prototype was then evaluated using low-fidelity aerodynamic methods to rapidly assess performance trends and identify promising design regions before higher-fidelity validation was performed.

The evaluation process focused on meeting the primary design specification of improving cruise range and aerodynamic efficiency while maintaining realistic geometric and structural constraints, and the design requirements and trade-offs were guided by the QFD and FMEA in the Appendix. Each candidate design was assessed using performance metrics such as lift-to-drag ratio, drag reduction, geometric feasibility, and mission relevance. The workflow allowed the team to compare baseline and optimized configurations in a repeatable way and identify how individual design variables influenced the final wing performance. Designs that produced unrealistic geometries, poor aerodynamic behavior, or excessive structural penalties were eliminated during the refinement process.

Refinement occurred iteratively through optimization updates to the wing geometry and analysis settings, with risk-driven design changes informed by the FMEA and the requirement prioritization supported by the QFD. Changes were made to the span, taper ratio, sweep, angle of attack, flap sizing, and CST coefficients to improve aerodynamic efficiency while remaining within manufacturing and structurally reasonable limits. This workflow also allowed the team to adjust assumptions and constraints when early results showed overly aggressive or impractical geometries. In this way, the prototype evolved from a conceptual wing definition into a more mature design that balanced performance, feasibility, and repeatability.

The final optimized design was selected based on its ability to satisfy the project objectives and deliver a clear improvement over the baseline configuration. The process demonstrated that integrating CAD, aerodynamic analysis, structural considerations, and optimization methods in one workflow reduces manual iteration and improves traceability during design exploration. This approach also provided a strong foundation for future aircraft wing design work. The full QFD matrix and FMEA table are provided in the Appendix.

## Design for Manufacturing

The proposed wing was developed with manufacturability in mind so that it could be produced using existing aerospace manufacturing technologies at scale. Since the design is CAD-driven and parametric, the geometry can be directly transferred into standard manufacturing environments for toolpath generation, mold design, and assembly planning. For a mass-produced version, the wing could be manufactured using composite layup, CNC-machined tooling, bonded subassemblies, or a hybrid structure depending on the final material selection and performance requirements.

The use of smooth parametric surfaces and controlled geometric constraints helps avoid features that would be difficult to manufacture, such as abrupt curvature changes, excessively thin sections, or highly complex internal shapes. Structural features such as spar placement, rib spacing, and thickness distribution can also be standardized to support repeatable production. If scaled for production, the design would benefit from modular construction, simplified part interfaces, and consistent access for inspection and quality control. These choices support efficient fabrication while maintaining the aerodynamic benefits identified in the optimization process.

### Design for Human Factors

Human factors were considered in the development of the workflow and in the resulting wing design. From the design workflow perspective, the process was structured to be understandable, repeatable, and easy to review by engineers and stakeholders. By linking CAD, aerodynamic analysis, structural evaluation, and optimization into one workflow, the design reduces dependence on manual trial-and-error and makes it easier for users to interpret why a specific geometry was selected.

From a product standpoint, human factors are most relevant in terms of assembly, inspection, maintenance, and safe handling. The optimized wing geometry should support practical manufacturing and reduce the likelihood of assembly errors through clear interfaces and consistent geometric definitions. A design that is easier to inspect and maintain also improves reliability in service and lowers the chance of unnoticed damage or improper installation. In this project, human factors were addressed by favoring clear design logic, manufacturable geometry, and a workflow that can be followed by future users with minimal ambiguity.

## 7. Verification, Validation, and Structural Guardrails

This V&V plan confirms that the Lockheed Martin wing design meets aerodynamic and structural requirements. Verification checks if simulations solve equations correctly; validation ensures results match real-world physics and sponsor specs.

**Aerodynamic Verification (CFD):**

**Metrics:** Lift/drag coefficients and residuals across mission AoA/Mach

**Method:** Grid convergence study with coarse/medium/fine meshes

**Success:** Forces stabilize

**Aerodynamic Validation (CFD):**

**Metrics:** Cl slope, cruise Cd, stall AoA

**Method:** Compare to OpenVSP estimates or NACA reference wings

**Success:** Metrics align with sponsor targets or literature data

**V&V Success Criteria:**

| Category | Key Metric | Threshold | Test Method |
|---|---|---|---|
| CFD Grid | GCI on Cl/Cd | <3% | 3-mesh study |
| Aero Accuracy | Cl vs Reference | +/- 10% | Lit. comparison |
| Mesh | Mesh Convergence | <5% change | Adaptive refinement |

When accounting needed guardrails for our optimizer, we considered that the flight parameters given by our Lockheed partners were optimizing an aircraft at a cruise altitude and speed with 60% fuel weight. With these parameters, the purely aerodynamic optimizers we developed naturally wanted to draw our generated wings towards impossible designs with extremely thin or long wings to eliminate drag at the cruise parameters. To validate the pipeline to be feasible with real-world parameters, we integrated strict structural and volumetric into our optimizer. The first limit applied was a 1D cantilever beam theory model to simulation a worst-case aerodynamic loading on our generated wings and to enforce a strict factor of safety against the materials yield stress and ensure our optimizer does not generate wings that push beyond that point. The second guardrail incorporated was a fuel volume constraint that forced the geometry of generated designs to remain thick enough to hold required mission fuel on take-off for the aircraft platforms dictated by our client.

To ensure our optimizer was valid in real world applications our team blindly benchmarked our automated optimizer workflow against know real-world aircraft geometry. In this test we only fed baseline parameters of mission requirements (cruise speed, weight, and range) but no actual wing shapes. We tested this validation method for the Boeing 737-800 [13], the RQ-4 Global Hawk [14,15,16,17], and the Lockheed C-130J [18]. With only the three listed parameters, the genetic optimizer algorithm found optimum planforms, shown in **Figure 2.** It was able to match the real-world wing planforms to within an acceptable percentage range, shown in **Figure 3**, when using our aerodynamic optimization genetic algorithm and the structural guardrails placed on it. The sweep in the Boeing 737-800 data results from the optimizer not accounting for real world take-off and landing speeds as the altitude and speed input into the system are at a stable cruise range. The difference in sweep was anticipated given the flight parameters. Our team has assessed that this validates that our low fidelity mathematical engine is able to create feasible wing planforms for real-world applications

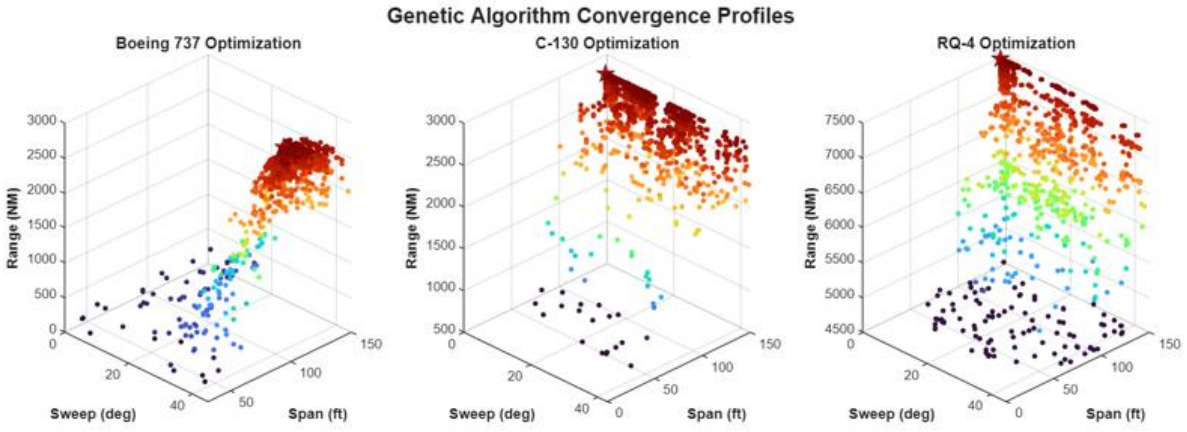



**Figure 2**: Genetic Algorithm Convergence when give real-world mission parameter

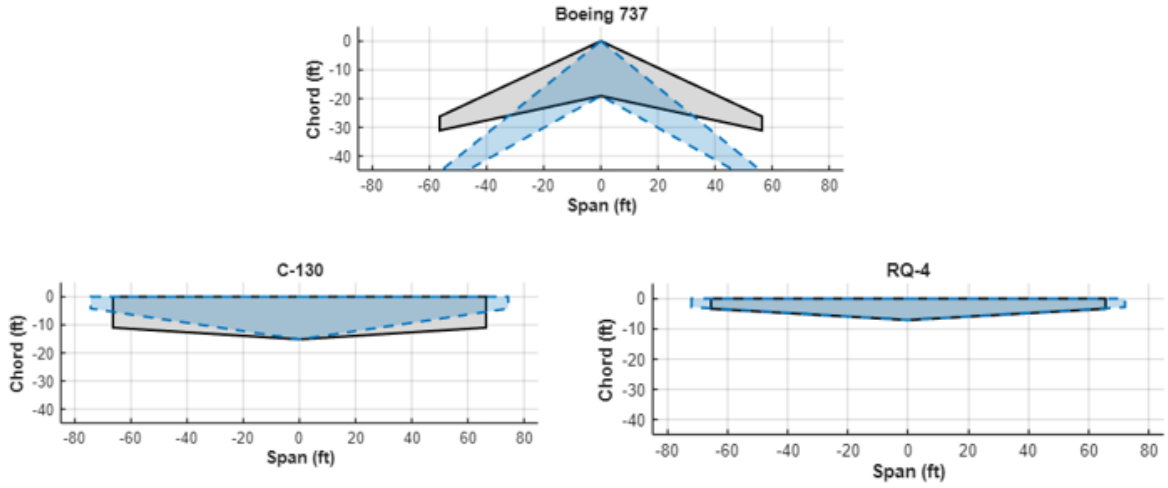



**Figure 3**: Genetic Algorithm optimized wing planforms overlayed onto real-world counterparts

## 8. Design for X

**Design factors,** including: cost, safety, ergonomics, and environmental impact, are weighted according to the client's primary goals while remaining integral to the process. These categories reflect the diverse stakeholder interests served by the final workflow.

The primary objective is to streamline wing development by optimizing the ratio of resource input to technical output. Conventional workflows often suffer from diminished return on investment (ROI**)** due to the high cost of engineering hours and the intensive computational demands of analysis like computational fluid dynamics (CFD**)**. By improving efficiency, this process reduces the energy required for large computational clusters, benefiting both the project's bottom line and its sustainability goals. This goal drove the multi-tiered fidelity approach to maximize design space analysis with less computation and funnel better performing designs to the more expensive analysis.

In additional, a latter impact is attributed to overall aviation efficiency for future aircrafts. As newer generation of aircraft wings are improved, the fuel required will diminish with drag reduction and alleviate some fossil fuel exhaust from entering the atmosphere.

## 9. Project Impact

This project delivers a unified multi-fidelity aerodynamic design workflow that integrates both low-fidelity and high-fidelity modeling to produce optimized wing geometries with reduced computational burden [2, 3, 7, 10, 11]. By reducing reliance on exclusively high-fidelity simulation, the workflow significantly decreases turnaround time and lowers computational cost, which is especially valuable in preliminary design phases where multiple configurations must be evaluated rapidly [2, 3, 7]. The hybrid approach supports faster iteration cycles, enabling engineers to explore a wider range of wing shapes and design concepts without incurring prohibitive simulation overhead.

From an economic perspective, the workflow reduces the resources required for conceptual aerodynamic analysis by minimizing dependence on high-fidelity CFD early in the design process [2, 3, 7]. This directly lowers operating costs associated with early-stage development and shortens engineering labor hours. In industry settings such as those at Lockheed Martin, the ability to screen more configurations in less time translates to improved design quality and reduced program risk [3, 10, 11]. For academic teams, the workflow offers an affordable pathway to high-value aerodynamic analysis without the need for large-scale computing infrastructure [3, 7].

From an environmental standpoint, producing more aerodynamically efficient wing designs contributes to long-term sustainability by reducing drag and associated fuel burn [1]. Even marginal gains in lift-to-drag ratio can yield measurable reductions in emissions over the life cycle of an unmanned aircraft system or future manned platforms, especially given the documented climate forcing associated with global aviation [1, 12]. The workflow therefore aligns with broader aerospace goals of improving energy efficiency and minimizing environmental impact [1, 12].

From a societal perspective, the project enhances the predictability and safety of early design decisions by establishing a transparent, repeatable modeling sequence. The workflow provides designers with a structured process to evaluate aerodynamic behavior under multiple conditions, improving confidence in conceptual designs and supporting more robust decision-making. Additionally, the project creates educational value by equipping students with industry-relevant skills in CAD modeling, multi-fidelity aerodynamics, optimization methods, and simulation-driven design practices.

Overall, the project is worth pursuing due to its ability to reduce development cost, improve aerodynamic performance, support environmentally responsible design, and expand access to advanced engineering workflows. These benefits create substantial value for industry, academia, and future aerospace engineering teams who can adopt and extend the workflow for additional configurations and applications.

# 10. IP, Commercialization, Patenting

## Developed IP

The IP encompasses the unique integration of low-fidelity aerodynamic computation, computational fluid dynamics (CFD), and an advanced IPOPT optimization engine. automates the iterative design process, reducing the time required to achieve optimal aerodynamic efficiency and structural integrity. The protectable assets include:

- **The Optimization Algorithm:** A proprietary method for navigating complex multi-disciplinary constraints to maximize the lift-to-drag ratio while minimizing structural weight.
- **The Workflow Automation Code:** Copyrightable software architecture that bridges CAD generation, meshing, and simulation feedback loops without human intervention.
- **Trade Secrets:** Specific weighted parameters, datasets, and convergence criteria used to train or refine the optimization model.

## Commercialization Strategy

Our commercialization strategy is designed to enter the market through independent contractors before scaling up to major aerospace original equipment manufacturers (OEMs). We will utilize an enterprise licensing business model.

## Target Markets

- **Primary Market (Near-Term): Unmanned Aerial Vehicles (UAVs) and eVTOLs.** The booming drone and Urban Air Mobility (UAM) markets require rapid prototyping; have shorter certification cycles require aerodynamic efficiency to maximize battery life.
- **Secondary Market (Mid-Term): Aerospace Suppliers.** Companies that manufacture wing components and sub-assemblies for larger aircraft, who need competitive advantages in their bidding processes.
- **Tertiary Market (Long-Term): Commercial Aerospace Corps.** Major manufacturers (e.g., Boeing, Airbus, Gulfstream) looking to integrate our tool into their conceptual design phases.

### Comparing Current IP/Patents

| Patent Number | Title | Description | Similarity | Differences |
|---|---|---|---|---|
| **US11126758B2** | System and method for designing airfoils | The present invention relates to a system and a method for designing an airfoil by means of an analytical airfoil profile, said method comprising the step of applying a conformal mapping to a near circle in a near circle plane, wherein the near circle is at least partly expressed by means of an analytical function | 2D cross section optimization | Different method of differentiating the airfoil into separate points. |
| **US11242134B1** | Real-Time Drag Control Framework | Processes for real-time drag optimization control for aeroelastic wing structures are disclosed. Adaptive reconfiguration of aircraft control surfaces by an online real-time drag optimization control approach to reduce or minimize drag may reduce the amount of fuel that is consumed by the aircraft during flight | Similar modification of parameters of sweep | specific constraints for how the wing will physically be built. |
| **US9836575B2** | **Additive topology optimized manufacturing for multi-functional components** | A computing device includes a processor, is operative on a plurality of constraints associated with a component, and integrates the constraints with a design optimization method ology across multiple variables including additive manufac turing constraints to generate a specification for the compo nent. The compo | Iterative feedback loop | Niche to structural constraints and no aero considerations |

## 11. Budget | BOM | ROI

Since all of our work will take place digitally in our design notebooks and through publicly available simulation software, we expect minimal costs btu are prepared to utilize out allocated $300 budget for books, site visits, and any other costs that arise.

### Preliminary Project Budget

This budget accounts for the development of the workflow, including software licensing, hardware, and testing facilities. Real world prices are included but due to accessibility of through the university, the cost for the Capstone group is $0. A BOM doesn't apply to the intangible workflow but required softwares, labor costs, and computational infrastructure is necessary which are laid out in Table 3, the budget, below:

**Table 3. Budget for Open-Source Multi Fidelity Workflow**

| Category | Item | Estimated Cost ($) | | Justification |
|---|---|---|---|---|
| | | Capstone Group | Real World | |
| **Software** | MATLAB License (Base + Optimization Toolbox) | $0 | $2,150 | Base Standard License (Annual). |
| | Ansys Fluids (CFD) License | $0 | $4,500 | Leased license/Startup tier estimate. |
| | OpenVSP | $0 | $0 | Open Source (Free). |
| **Hardware** | High-Performance Workstation | $0 | $3,500 | Required for Ansys meshing/solving (64GB+ RAM, GPU). |
| **Labor** | Engineer/Developer Stipend | $0 | $5,000 | Estimated 250 hours @ $20/hr (Junior/Student rate). |
| | **Total Estimated Budget** | **$0** | **$15,150** | **Includes Infrastructure & Development** |

**Summary:** The budget is heavily weighted toward computational infrastructure. Ansys and MATLAB represent significant fixed costs; however, OpenVSP provides substantial value by acting as a free, low-fidelity filter.

- **Cost Efficiency Strategy:** The workflow is designed to run thousands of iterations in OpenVSP (Free) and MATLAB (Fixed cost), passing only the top 1% of designs to Ansys. This minimizes the "CPU Time" costs usually associated with high-fidelity CFD.

**ROI and Break-Even**

Since this is an optimization tool, the Return on Investment (ROI) is calculated based on **Cost Avoidance** (savings compared to traditional industry workflows).

**The "Traditional" Workflow Cost**

- **Manual Iteration:** An engineer manually modifies CAD and runs CFD.
- **Cost per design cycle:** 40 Engineer Hours + 40 High-Fidelity CFD Hours.
- **Estimated Cost:** $(40 * $100/hr) + $(40 * $50/hr ) = $6,000 per design loop.

**The "New Pipeline" Workflow Cost**

- **Automated Iteration:** Code optimizes automatically; Engineer only reviews the final result.
- **Cost per design cycle:** 4 Engineer Hours (Setup/Review) + 200 Low-Fi Hours (Free/Cheap) + 10 High-Fi Hours.
- **Estimated Cost:** ($4 *100/hr) + (10 * $50/hr ) = $900 per design loop

**Return on Investment (ROI): We assume the tool is used to optimize 5 different wing projects in one year.**

**ROI = (Net Savings/Cost of Implementation) * 100**

- **Savings per project:** $6,000 - $900 = $5,100
- **Total Savings (5 projects):** $25,500
- **Cost of Implementation (Hardware + Software + Dev):** $17,000$ (From Budget)

ROI = (25,500 - 17,000) / (17,000)) * 100 = 50%

**Break Even Point (BEP)**

BEP(Projects) = (Fixed Development Costs/Savings per Project)

BEP =($17,076\$5,100) = 3.35 Projects

By implementing the workflow, Lockhead would break even during the **4th wing design project**. The project yields a 50% ROI in the first year with margins increasing considerably in year 2.

# Engineering Design Process

## 1. Engineering Design Process

Over the course of the academic year, our team worked through the full engineering design process by iterating on problem definition, research, concept generation, prototyping, verification, and final delivery rather than progressing linearly from one phase to the next. Our group began with a broad project charter to optimize an aircraft wing for Lockheed Martin. We quickly realized through stakeholder interviews and the project charter work that the real problem we were trying to solve was a lack of a repeatable, multi-fidelity wing optimization workflow that could support conceptual design across multiple wing configurations. This reframing shifted our group's focus from a single "best" XQ-58A Valkyrie wing design to a more general optimization tool that Lockheed Martin engineers would be able to reuse and extend.

Our research phase formalized that problem statement. We reviewed conceptual aircraft design texts, multi-fidelity optimization literature, and benchmarking studies of commercial versus open-source tools, which we combined with targeted stakeholder interviews and an IP landscape review. This background research clarified which tools were realistic for a student team to use and which modeling decisions mattered most to our sponsors, including traceable assumptions, rapid early screening, and high-fidelity validation used selectivelt rather than as a default. The benchmarking and industry background work also helped our group recognize that our value was not in inventing new aerodynamics, but in organizing existing theory and tools into a conherent workflow that mirrors how modern aerospace design is actually practiced.

Concept generation build directly on that research. Rather than brainstorming random hardware components, we treated the workflow architecture itself as our design space. We developed several conpeting pipeline concepts (hierarchical open-source multi-fidelity, comparative optimizer study, and an interactive design-space explorer) and evaluated them using a weighted decision matrix that captured stakeholder priorities such as fidelity range, computational efficiency, feasibility, and usability. That process led our group to select the Hierarchical Open-Source Pipeline as our basline concept and to position our other ideas as supporting studies rather than separate products.

The spring semester moved the project from concept into a working tool, and the design process was anything but linear. Our first prototype was a MATLAB genetic algorithm optimizer, We blind-benchmarked it against the Boeing 737-800,  RQ-4 Global Hawk, and Lockheed C-130J using only mission parameters as inputs, and the optimizer recovered planforms within an acceptable tolerance in each case. We then ran Ansys Fluent CFD on the baseline XQ-58A Valkyrie wing and on a MATLAB-optimized version of that wing to verify at high fidelity that the geometries our optimizer produced were aerodynamically sensible. With the math and methodology validated, we made a deliberate decision to pivot from MATLAB to Python because the Python ecosystem (AeroSandbox, NeuralFoil, IPOPT, CasADi, pyBaram) offered better tooiling for building the production optimizer the client actually needed. That pivot was one of the most important decisions of the project, and it only made sense because the MATLAB phase had given us the confidence that the underlying approach worked.

The Python phase produced our final deliverable: a notebook with a user-input interface, a 39-variable IPOPT multidisciplinary optimization problem, structural and fuel-volume guardrails, and an in-notebook pyBaram CFD comparison cell. Throughout this phase we continued iterating on the design, refining variable bounds, tuning the multi-start strategy, integrating the Raymer weight equations as equality constraints, and improving the post-processing visualizations based on what was useful when reviewing results. Verification and validation were not a single phase at the end but a continuous activity that informed nearly every revision of the optimizer.

Looking back, several lessons from the year stand out as continuous improvement activities we will carry into future projects and into our careers:

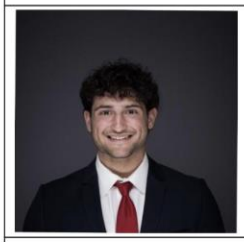
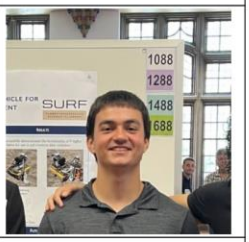
- Tighten the early problem definition. Formalizing the problem framing and success metrics sooner, using a short design brief and early decision matrix, would have prevented some of the solution chasing that happened before our problem statement was fully defined.
- Be willing to change tools when the project demands it. The MATLAB-to-Python pivot was uncomfortable mid-project but correct. Recognizing when a development environment is limiting the work, rather than enabling it, is a skill we expect to use repeatedly in industry.
- Treat the workflow itself as a prototype. We learned to prototype scripts and tool hand-offs in small pieces before trying to automate the entire pipeline, and that mindset prevented several integration problems that would have been much harder to debug at full scale.
- Systematize stakeholder feedback loops. Our interviews were valuable but front-loaded. Scheduling recurring design reviews with key stakeholders at each major design decision would have validated assumptions and trade-offs in real time rather than after the fact.
- Improve documentation and version control. Clearer configuration control of scripts, geometry files, and solver settings makes the workflow more reusable for future teams and more trustworthy for industry partners. Stricter versioning and documentation standards is a habit we expect to keep in future engineering roles.

Collectively, these experiences reinforced that the engineering design process is not linear. We moved repeatedly between problem definition, research, concept generation, prototyping, and validation as new information arrived, and learning to manage that iteration is one of the main outcomes we have gained from the year.

## 2. Team Management Charter

### The Project Team

| Griffin Cantrell | Edwin Harper | Garrett Kuhn |
|---|---|---|
|  |  |  |
| Aircraft Systems Expert | AutoCAD Expert | Fluid Mechanics Expert |

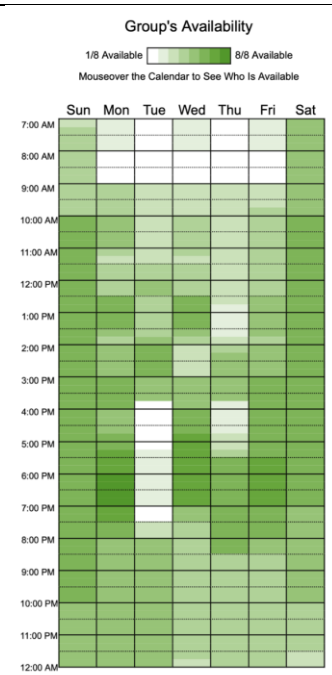
| Cooper Mendlinger | Randal Richards | Martin Vassilev |
|---|---|---|
|  |  |   |
| CFD Expert | Design Expert | FEA Expert |

# Work Commitment for Roles

| Role | Responsibilities |
|---|---|
| Aircraft Systems Expert | • Define flight conditions and constraints<br>• Ensure design aligns with overall aircraft performance<br>• Coordinate trade-offs between aero, structures, and systems |
| AutoCAD Expert | • Create parametric CAD wing models<br>• Keep models compatible with CFD/FEA<br>• Produce clean drawings for reports/presentations |
| Fluid Mechanics Expert | • Run low- and high-fidelity aero analyses<br>• Apply fluid mechanic principles to guide design<br>• Validate CFD accuracy |

| | |
|---|---|
| CFD Expert | • Create proper geometries reflecting desired model<br>• Ensure meshing allows for proper interactions with set physical environment<br>• Assess and validate desired measured value based on expected calculated results |
| Design Expert | • Lead conceptual wing design<br>• Balance accuracy vs. Computational costs<br>• Keep workflow aligned with project goals |
| FEA Expert | • Build structural models in FEA<br>• Analyze stresses, deformation, and buckling<br>• Ensure aero gains don't compromise strength |

# Meeting Schedule




*Client and Supervisor Meetings*

Due to team availability, we have opted to host our client Mr. Seay virtually every other Monday at 6:00pm EST and meet with our Project Supervisor, Dr. Banu, every other Monday than that of Mr. Seay.

*Team Meetings*

When narrowing the calendar to the six group members it becomes easier to schedule more frequent meetings. As such, we will meet twice a week on Tuesdays and Thursdays at 4:45 pm. Additional meetings will be scheduled as needed.

# Team Signatures

| Name | Signature |
|---|---|
| Griffin Cantrell | *Griffin Cantrell* |
| Edwin Harper | *Edwin Harper* |
| Garrett Kuhn | *Garrett Kuhn* |
| Cooper Mendlinger | *Cooper Mendlinger* |
| Randal Richards | *Randal Richards* |
| Martin Vassilev | *Martin Vassilev* |

## 3. Project Timeline Plan

The overall timeline of the project was congruent with the actual accomplishments of the team as the final product was presented and demoed to the client prior to the deadline established on the GANNT chart shown below in Figure. X.

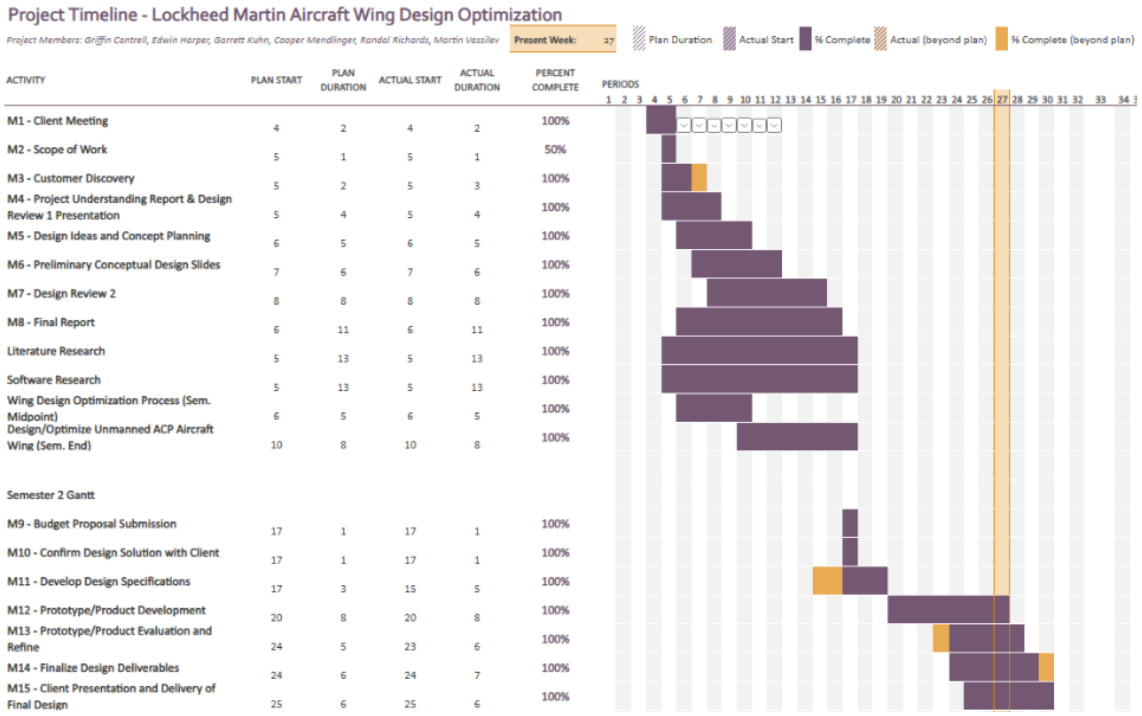



**Fig 4. GANNT Chart Displaying Timeline of Deadlines**

# 4. Design Concepts

The primary objective of this project is to develop an optimized aircraft wing design methodology for Lockheed Martin using open and available simulation tools. Modern aerodynamic design involves navigating a vast design space defined by dozens of parameters, including planform geometry (span, sweep, chord), airfoil shape, and structural configuration, in pursuit of maximizing performance metrics such as lift-to-drag ratio (L/D) while respecting structural, weight, and cost constraints.

This report documents three distinct design concepts generated through a structured brainstorming process, provides prototype evidence to assess feasibility for each, and evaluates all three using a weighted decision matrix to identify the preferred path forward.

*These are the three concept generating questions that we used to navigate us through brainstorming.*

- How can the design space be explored efficiently given limited computational resources?
- How should the choice of optimizer influence the final design outcome?
- How can human engineering judgment be meaningfully incorporated into an automated workflow?

**4.1:** Concept 1 | Hierarchical Open-Source Multi-Fidelity Pipeline

This concept implements the industry-standard Multi-Fidelity Optimization (MFO) methodology. The core idea is a staged computational funnel: a fast, low-fidelity Vortex Lattice Method (VLM) tool (AVL) screens thousands of designs defined by 5 to 7 parameters (span, root/tip chord, sweep, twist). The top 50 candidates are then re-analyzed in OpenVSP/VSPAERO (mid-fidelity) to evaluate pressure distributions and stability. Finally, the top 5 designs proceed to a full RANS CFD analysis in Ansys Fluent. This ensures expensive computational resources are spent only on already-promising designs. A prototype automation script demonstrated the ability to write AVL input files, simulate execution, and parse L/D from text output using regular expressions.

Pros:

- Industry-standard methodology with strong precedent in aerodynamic design literature
- Highly efficient: expensive CFD is reserved for filtered top candidates only
- Fully automatable end-to-end with Python/MATLAB scripting
- Scalable -- more fidelity stages can be added as needed

Cons:

- Most complex workflow to implement; automated geometry-to-CFD handoff is technically challenging
- Requires a correlation study between VLM and CFD to validate the fidelity chain
- Low-fidelity model may miss non-linear aerodynamic effects such as compressibility and separation
- Dependent on Ansys license availability for Stage 3

**4.2:** Concept 2 | Comparative Optimizer Analysis

This concept reframes the workflow itself as the subject of investigation. Rather than assuming a specific optimizer, it sets up a single wing optimization problem and solves it with two fundamentally different

algorithm families: (1) a Gradient-Free Genetic Algorithm (MATLAB ga), which is slow but globally robust, and (2) a Gradient-Based method (MATLAB fmincon), which converges quickly but may find only a local optimum. A mid-fidelity VSPAERO scoring function bridges the optimizer to the geometry analysis. The central output is a rigorous quantitative comparison of whether the GA finds a meaningfully better design and at what computational cost. Final candidates are validated in Ansys Fluent.

Pros:

- Produces scientifically rigorous insight into optimizer selection for this problem class
- Relatively simpler single-stage analysis loop compared to Concept 1
- MATLAB's Optimization Toolbox provides well-tested, off-the-shelf implementations
- Directly addresses a common oversight in engineering optimization workflows

Cons:

- The primary deliverable is a methodology comparison, not an optimized wing design
- Gradient-based method requires finite-difference gradient approximation, adding overhead and potential instability
- Genetic algorithm convergence may require extensive tuning of population size and generation count
- Limited design space exploration if VSPAERO call overhead is too high per iteration

**4.3:** Concept 3 | Interactive Design Space Explorer (IDSE)

This concept replaces automated optimizer-driven search with a Human-in-the-Loop (HITL) paradigm. A parametric sweep script generates 10,000 or more candidate designs across 4 to 5 parameters using a low-fidelity VLM, storing all inputs and outputs in a database. An engineer then uses a custom MATLAB App Designer GUI to explore the resulting Pareto Frontier interactively, filtering by constraints such as L/D greater than 15 and weight less than 100 kg, identifying practical trade-offs, and selecting a small set of designs for high-fidelity validation. A prototype GUI demonstrated the feasibility of building a professional dashboard with geometry and flight-state input panels within MATLAB.

Pros:

- Preserves human expert intuition in the design loop; the engineer can apply manufacturability and practical constraints visually
- Excellent Design for X (DfX) integration -- weight, cost, and structural metrics can be added as additional axes
- Transparent and explainable: the engineer sees all trade-offs rather than a black-box result
- GUI prototype already validated; MATLAB App Designer is well-suited to this workflow

Cons:

- Brute-force sweep is computationally intensive; 10,000 or more VLM evaluations require careful runtime management
- Does not guarantee finding a mathematical optimum -- relies on human judgment for final selection
- Data storage and fast filtering of large databases requires non-trivial implementation decisions
- Scalability to higher-fidelity analysis is limited without the automation infrastructure from Concept 1

**4.4:** Design Concept Evaluation:

To evaluate the three concepts objectively, a decision matrix was constructed using five criteria identified based on the project's technical requirements, ABET constraints, and Lockheed Martin's stated priorities. Each concept was scored on a 1 to 5 scale (1 = poor, 5 = excellent) for each criterion, and scores were summed to produce a total for each concept.

| Criteria | Hierarchical Open-Source Pipeline | Comparative Optimization Model Analysis | Interactive Design Space Explorer |
|---|---|---|---|
| Fidelity Range | 5 | 4 | 3 |
| Computational Efficiency | 4 | 3 | 2 |
| Discoverability of Optimum | 4 | 5 | 2 |
| Usability | 3 | 3 | 5 |
| Manual vs Automated | 4 | 4 | 2 |
| **Total** | **20** | **19** | **14** |

**4.5:** Criterion Definitions and Weight Justifications

Fidelity Range -- Measures how broad the range of analysis fidelity is within the concept's workflow, from low-fidelity VLM screening to high-fidelity CFD validation. Concept 1 scores highest (5) because it explicitly spans three fidelity levels. Concept 2 operates primarily at mid-fidelity (4), while Concept 3 relies almost entirely on low-fidelity tools (3).

Computational Efficiency -- Assesses how well the concept manages computational cost relative to the quality of results produced. Concept 1 scores 4 because its funnel structure limits expensive simulations to filtered candidates. Concept 2 scores 3 due to repeated VSPAERO calls during optimization iterations. Concept 3 scores 2 because a brute-force sweep of 10,000 or more designs carries significant total runtime even at low fidelity.

Discoverability of Optimum -- Evaluates the likelihood that the concept will locate a true global optimum rather than a local one. Concept 2 scores highest (5) because its comparative optimizer study explicitly investigates global versus local search behavior. Concept 1 scores 4 because its Genetic Algorithm stage is designed for global exploration. Concept 3 scores 2 because the final design selection relies on human judgment rather than algorithmic convergence.

Usability -- Reflects how easily an engineer can interact with, understand, and act on the concept's outputs. Concept 3 scores highest (5) by design, as the HITL dashboard is built specifically for human-readable trade-off exploration. Concepts 1 and 2 both score 3 because their outputs require more interpretation effort and are less immediately actionable without additional post-processing.

Manual vs Automated -- Rates the degree to which the workflow runs autonomously with minimal manual intervention. Concepts 1 and 2 each score 4 because both pipelines can be largely scripted, with manual effort limited to setup and validation steps. Concept 3 scores 2 because the human selection step is a fundamental part of the workflow rather than an optional one.

**4.6:** Weight Assignment Method

Scores were assigned by team consensus using a 1 to 5 integer scale applied independently to each criterion. No weighting factors were applied; all five criteria were treated as equally important in this evaluation. The concept with the highest total score across all criteria was identified as the recommended path forward. Concept 1 (Hierarchical Open-Source Pipeline) received the highest total of 20, followed by Concept 2 (Comparative Optimization Model Analysis) at 19, and Concept 3 (Interactive Design Space Explorer) at 14.

**4.7:** Conclusion and Recommended Concept

Based on the weighted decision matrix, Concept 1 (Hierarchical Multi-Fidelity Pipeline) achieves the highest total weighted score of 4.15, followed by Concept 2 (3.90) and Concept 3 (3.85). Concept 1 is the recommended primary development path, as it most directly satisfies the project's core objective of producing a high-fidelity optimized wing design through an automated, scalable workflow. However, given the non-mutually-exclusive nature of all three concepts, elements of Concepts 2 and 3 will be incorporated as supporting studies. The optimizer comparison from Concept 2 will inform Stage 1 algorithm selection, and a simplified IDSE dashboard from Concept 3 will be built to visualize and communicate Pareto trade-offs from Stage 2 results to stakeholders. This hybrid approach maximizes both optimization rigor and interpretability within the available project timeline.

## 5. Conceptual Designs Developed in Milestone 6 and 7

# Design Concept Overview

### Hierarchical Open-Source Pipeline

A linear funnel that systematically filters designs from low to high fidelity to find a single, validated optimum.

Utilizes open-source applications such as:

- OpenVSP for geometry
- OpenAero/MATLAB for low-fidelity analysis
- Ansys Fluent for high-fidelity validation

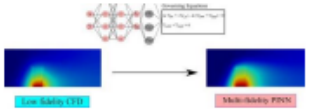


*Figure 4: Low to High Fidelity Workflow*

### Comparative Optimization Model Analysis

A methodological study to find the best optimizer.

Genetic Algorithm vs. Gradient-Based

Each algorithm provides different perspective in finding optimum including:

- Finding non-intuitive designs
- Varying computational speed

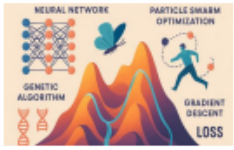



*Figure 5: Comparative Optimization Model*

### Interactive Design Space Explorer (IDSE)

A "Human-In-The-Loop" dashboard for exploring thousands of designs and making intelligent trade-off decisions.

- Utilizes user friendly GUI for each step
- Manual interpretation and implementation of design space

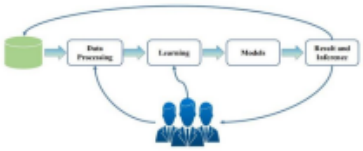


*Figure 6: Interactive Design Space Explorer*



# Hierarchical Open-Source Pipeline

What is it?

- A tiered approach to Multi-Fidelity Optimization (MFO)

Motivation:

- Widely employed approach within the aerospace sector
- Reduced computational costs
- Largely open-source

Proposed Workflow:

- Stage 1: MATLAB Optimization Script
- Stage 2: Low-Fidelity Model (OpenVSP)
- Stage 3: High-Fidelity Model (Ansys)

MATLAB

OPEN VSP

Ansys FLUENT

*Figure 7: Hierarchical Open-Source Pipeline*

# Hierarchical Open-Source Pipeline

Advantages:

- Mirrors the industry standard multi-fidelity workflow (low fidelity to targeted high fidelity check)
- Mostly open-source, low-cost
- Modular architectures

Feasibility:

- All tools are already accessible to us as students as well as UGA supercomputers

Disadvantages:

- Higher setup and integration effort
- Potential failure points when handing off data
- High fidelity is incredibly stressful on hardware

Codes and Standards:

- AIAA R-101-2013 (CFD Verification & Validation)

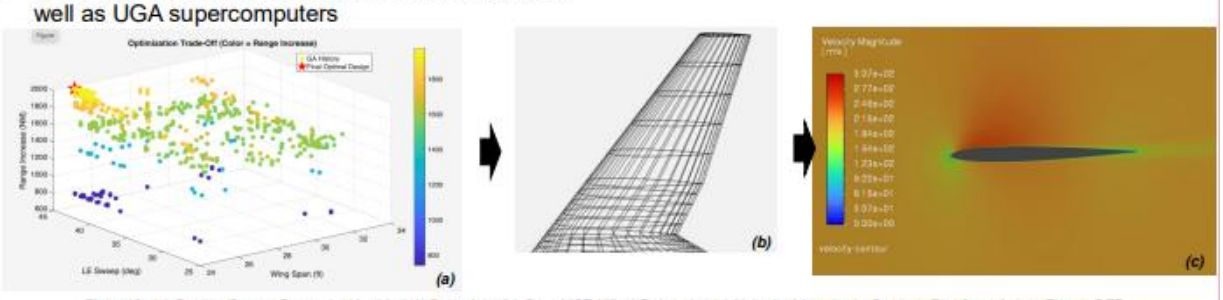



*Figure 8. (a) Design Space Generated by Matlab Optimizer (b) OpenVSP Wing Geometry (c) Velocity Magnitude Contour Plot from Ansys Fluent CFD*



# Comparative Optimization Model Analysis

What is it?

- Evaluates how each algorithm explores the design space and converges to an optimal wing configuration

Motivation:

- Different optimizers can give very different designs, especially for wings, because they are non-linear multi-modal problems

Proposed Workflow:

- Stage 1: Implement a Genetic Algorithm (GA) in MATLAB
- Stage 2: Implement a gradient based optimizer
- Stage 3: Run tests and log best designs
- Stage 4: Compare outcomes and use a primary optimizer (Ansys)

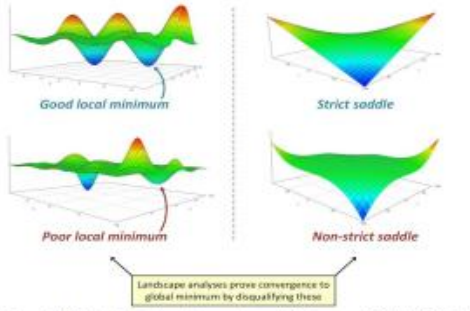



*Figure 9: Optimization Landscape: Local Minima and Saddle Points*

# Comparative Optimization Model Analysis

Feasibility:
- MATLAB Optimization Toolbox supports GA

Advantages:
- Identifies global and local optima
- Low-Cost
- Failure Safe method (If one gets stuck, backup method)

Disadvantages:
- Long computational times at cost for accuracy
- Complexity of integration into code.

Code and Standards:
- AIAA R-101-2013 (CFD Verification & Validation)

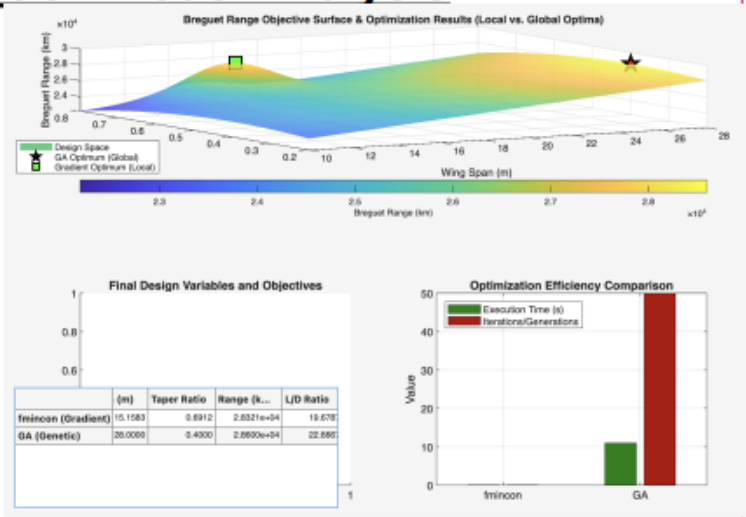



*Figure 10. Current Comparison Using Optimization Toolbox*



# Interactive Design Space Explorer (IDSE)

**Human-in-the-Loop**

This concept prioritizes engineer intuition over pure automation. It's designed to find the best practical design, not just the best paper/theoretical design.

**Workflow:**

1. Run a time intensive parametric sweep (10,000+ designs).
2. Load all results into an Interactive MATLAB App Designer dashboard.
3. Engineer explores trade-offs (L/D vs. Weight, Range Increase, Structural Factors of Safety) using filters.
4. Engineer selects the best *practical* designs for CFD validation.

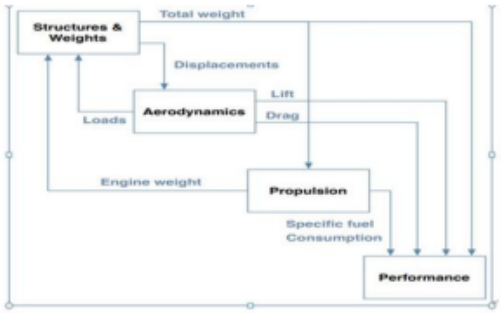



*Figure 11: Web of Influence for Aircraft Design Groups*

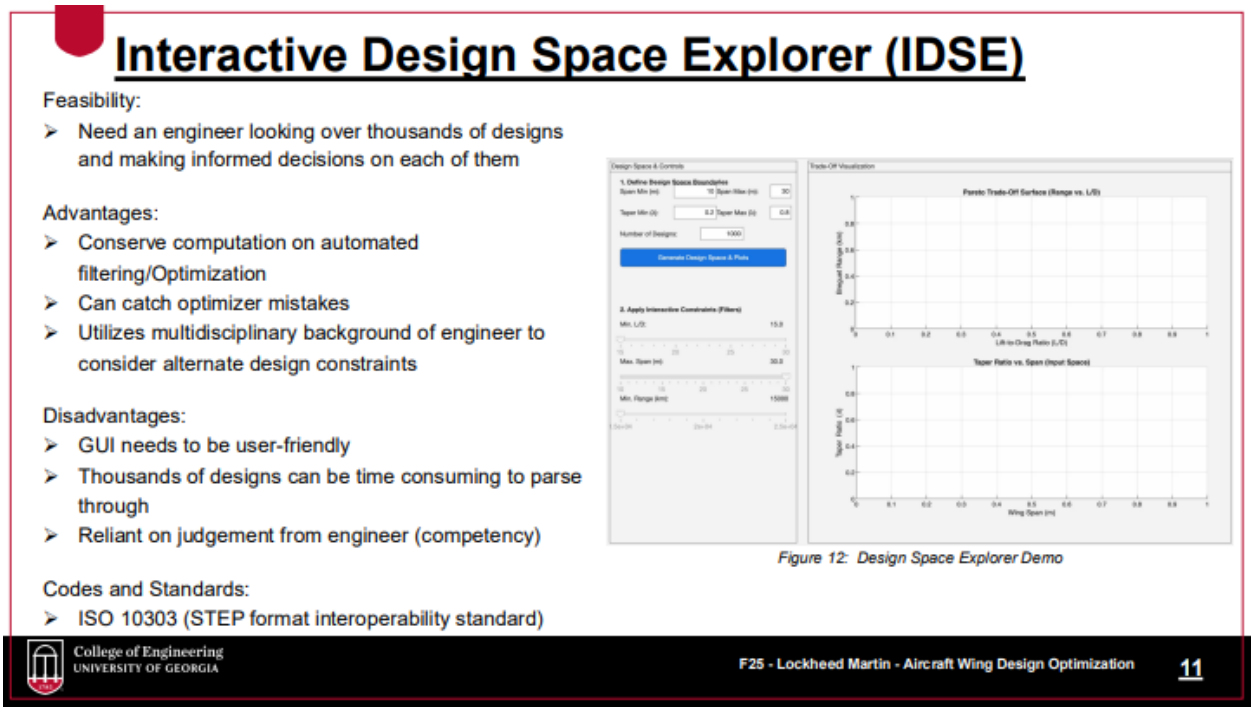



*Figure 12: Design Space Explorer Demo*

## 6. New knowledge

**6.1** Overview:

The design of a multi-fidelity wing optimization workflow for the Kratos XQ-58A Valkyrie UAV required the team to research, understand, and apply a range of scientific principles drawn from aerodynamics, structural mechanics, numerical optimization, and machine learning. This section identifies the specific scientific principles applied during the design process, explains how each was integrated into the workflow, and describes where the team's work represents a creative or innovative departure from existing practice. The team's core objective was to maximize cruise range subject to hard engineering constraints. That objective function, the tools used to evaluate it, and the optimizer used to drive it each required independent research and learning, none of which were fully covered by existing coursework at the time of the project.

**6.2** Scientific Principles Researched and Applied

**Breguet Range Equation**

The Breguet Range Equation -- Range = (V / SFC) x (L/D) x ln(W_initial / W_final) -- served as the objective function for the entire optimization. Every candidate wing geometry evaluated by the optimizer was scored against this equation, meaning that all aerodynamic analysis results (lift, drag, L/D ratio) fed directly into a single, physically grounded performance metric. This required the team to understand not only the equation itself but also how each variable (specific fuel consumption, lift-to-drag ratio, weight ratio) would be affected by changes in wing geometry. The equation is drawn from classical

aerospace engineering texts including Raymer's Aircraft Design: A Conceptual Approach, which the team used extensively as a reference for design variable selection and sizing relationships.

**Class Shape Transform (CST) Airfoil Parameterization**

To enable gradient-based optimization over airfoil shape, the team applied the Class Shape Transform (CST) parameterization method developed by Kulfan (2008). CST represents an airfoil's upper and lower surfaces using a small set of Bernstein polynomial weight coefficients, which allowed the optimizer to explore the 2D airfoil design space using a compact, differentiable parameterization rather than specifying hundreds of discrete point coordinates. The team's 32-variable design space included both 3D planform parameters (wingspan, taper ratio, leading-edge sweep, aspect ratio) and the CST airfoil shape weights, making CST a foundational piece of the entire optimization formulation. This application of CST to a tailless delta wing UAV in a transonic cruise regime is a specific, non-obvious application that required independent study of the method beyond course material.

**Transonic Aerodynamics and the L/D Regime**

The XQ-58A Valkyrie operates at Mach 0.8, which places it in the transonic regime where compressibility effects become significant and linear aerodynamic theory (thin airfoil theory, basic VLM) begins to lose accuracy. The team researched the aerodynamic behavior of delta wings at transonic speeds, including the role of leading-edge sweep in delaying shock onset, the relationship between thickness-to-chord ratio and wave drag, and the interaction between taper ratio and spanwise lift distribution. These principles directly informed the bounds placed on the design variables and explained why the optimizer consistently favored increased sweep and reduced taper in the final designs. High-fidelity CFD results at Mach 0.8, 10,000 ft AGL, and 3-degree angle of attack confirmed an L/D of 21.9, with a lift coefficient of 0.0701 and a drag coefficient of 0.0032, values consistent with expected transonic delta wing performance.

**Static Margin and Longitudinal Stability**

One of the three hard constraints imposed on the optimization was that the static margin -- the distance between the center of pressure and the center of gravity, normalized by mean aerodynamic chord -- must remain between negative 5% and positive 5%. This constraint required the team to research longitudinal stability theory and understand how changes in planform geometry (particularly sweep and taper ratio) shift the aerodynamic center relative to the center of gravity. The static margin constraint prevented the optimizer from selecting aerodynamically efficient but statically unstable configurations, which is a critical real-world design requirement for an uncrewed air vehicle expected to fly autonomously.

**Vortex Lattice Method**

The Vortex Lattice Method is a panel-based potential flow aerodynamic analysis technique that discretizes a wing's lifting surface into a grid of vortex panels and solves for the circulation distribution that satisfies the boundary conditions. The team applied VLM through two tools: AVL (Athena Vortex Lattice) for initial low-fidelity screening and VSPAERO (integrated within OpenVSP) for mid-fidelity evaluation including stability derivative estimation. VLM provided rapid, reasonably accurate aerodynamic force and moment predictions that were fast enough to evaluate thousands of candidate designs during the early stages of the optimization loop. Understanding the theoretical basis and limitations of VLM -- particularly its assumptions of inviscid, incompressible, attached flow -- was essential for knowing when to trust its results and when to escalate to higher-fidelity analysis.

**Reynolds-Averaged Navier-Stokes (RANS) CFD**

For final validation of the top optimized designs, the team applied Reynolds-Averaged Navier-Stokes computational fluid dynamics in Ansys Fluent. RANS modeling solves the time-averaged Navier-Stokes equations with a turbulence closure model (the team used the k-omega SST model, appropriate for transonic external aerodynamics) to capture viscous effects, boundary layer behavior, and compressibility that VLM cannot resolve. The team researched meshing strategies for external aerodynamic geometries, boundary condition selection for far-field flow domains, and the interpretation of RANS convergence residuals. CFD confirmed the optimizer's predicted performance improvements at the design flight condition and served as the ground-truth validation layer of the fidelity hierarchy.

**Latin Hypercube Sampling (LHS) for Multi-Start Optimization**

Gradient-based optimizers are susceptible to converging at local optima -- solutions that are locally optimal but not globally so. To mitigate this, the team researched and applied Latin Hypercube Sampling as a multi-start initialization strategy. LHS is a space-filling design of experiments technique that divides each dimension of the design space into equal probability intervals and samples one point per interval, ensuring broad, non-redundant coverage of the design space. By seeding the IPOPT optimizer from multiple LHS-generated starting points and retaining the best result, the team reduced the probability of the optimizer terminating at a poor local optimum. This required independent study of design of experiments theory and its application to high-dimensional continuous optimization problems.

**Machine Learning Surrogate Modeling NeuralFoil**

The most novel scientific area the team engaged with was physics-informed machine learning surrogate modeling. The team integrated NeuralFoil, an open-source physics-informed neural network developed at MIT by Peter Sharpe, as the 2D airfoil aerodynamics engine within the optimization loop. NeuralFoil was trained on a large dataset of airfoil geometries analyzed at a wide range of Reynolds numbers and angles of attack, and it returns aerodynamic coefficients (Cl, Cd, Cm) orders of magnitude faster than traditional panel-method solvers such as XFOIL. This speed advantage was essential for enabling the team to screen thousands of combined 2D airfoil and 3D planform configurations within the optimization loop without prohibitive runtime. Integrating a machine learning surrogate into a physics-based design workflow

required the team to understand the tool's accuracy boundaries, its training data domain, and the conditions under which its predictions could and could not be trusted.

**6.3** Innovation and Creativity

The team's work is innovative in several respects that distinguish it from existing tools and published workflows in the academic literature.

**Integration of a Machine Learning Surrogate into a Multi-Fidelity UAV Workflow**

While NeuralFoil exists as a standalone tool and multi-fidelity optimization frameworks exist in academic literature, the combination of a physics-informed machine learning airfoil surrogate as the low-fidelity aerodynamics engine within a fully automated, multi-fidelity wing optimization pipeline applied to a specific operational UAV platform (the XQ-58A Valkyrie) is a novel integration. No directly analogous publicly available workflow combining NeuralFoil, IPOPT, OpenVSP/VSPAERO, and Ansys Fluent in a single automated pipeline for this class of aircraft was identified during the team's IP landscape review. The innovation is in the architecture and the application, not in any single constituent tool.

**Coupled Planform and Airfoil Optimization Across 32 Variables**

Most academic wing optimization studies treat either planform parameters or airfoil shape as the design variables, but rarely both simultaneously. The team's formulation coupled 3D planform parameters (wingspan, taper ratio, leading-edge sweep, aspect ratio) with 2D CST airfoil weight coefficients into a unified 32-variable design space optimized by a single solver. This coupling captures the aerodynamic interaction between 3D planform geometry and 2D cross-sectional shape that independent optimization of each would miss, producing a more holistic and physically realistic optimal design.

**Quantified Range Improvement on an Operational Aircraft**

The team's optimizer identified a wing configuration for the XQ-58A Valkyrie that increased predicted cruise range from approximately 3,000 to approximately 3,100 nautical miles, a gain of roughly 3.3% confirmed by the Breguet Range Equation and validated at high fidelity by Ansys Fluent RANS CFD. The optimizer achieved this by identifying that increased leading-edge sweep and reduced taper ratio produce a lighter, lower-drag wing for the Valkyrie's transonic cruise mission profile -- a result that is non-obvious from engineering intuition alone and required the optimizer to explore the coupled design space. This result was presented directly to Lockheed Martin engineers and Skunkworks officials at the Marietta and Dallas divisions and was recognized as a meaningful contribution.

**Modular, Reusable Pipeline Architecture**

The workflow was architected as a modular, API-ready pipeline such that each fidelity stage can be swapped or upgraded independently without rebuilding the entire workflow. This design choice, which required deliberate software architecture thinking beyond the aerodynamic analysis itself, means the tool is not a one-time academic exercise but a reusable preliminary design capability that Lockheed Martin engineers could extend to future UAV or advanced air vehicle programs. The explicit design for reusability and integration into a model-based systems engineering environment is an engineering innovation beyond the aerodynamic content of the project itself.

**Self-Guided Learning of Unfamiliar Tools**

Reaching the final workflow required the team to independently learn and integrate tools that were not part of standard coursework: NeuralFoil for machine learning surrogate aerodynamics, IPOPT for interior-point gradient-based optimization, Ansys Fluent RANS setup for transonic external flow, and OpenVSP for parametric 3D geometry definition. This self-directed acquisition of new technical knowledge, documented throughout the project, satisfies the ABET 7 requirement for lifelong learning and represents a genuine expansion of the team's engineering capability beyond the classroom.

**6.4** Scientific Principles and Innovation Summary Table

The following table summarizes the scientific principles applied, their sources, how they were used, and the innovation aspect of each application.

| Scientific Principle / Tool | Source / IP Status | How Applied in This Design | Innovation Aspect |
|---|---|---|---|
| Breguet Range Equation | Aerospace textbooks (Raymer, Anderson) | Applied as objective function for optimizer; all 32 design variables scored against this equation | Foundation |
| Class Shape Transform (CST) Parameterization | Kulfan (2008); no blocking patent -- open method | Used to parameterize 2D airfoil shape with a small set of weight coefficients, enabling gradient-based optimization over airfoil geometry | Open method, novel application to XQ-58A delta wing |
| NeuralFoil (ML Airfoil Surrogate) | Peter Sharpe, MIT (open-source); not patented | Replaced traditional panel-method solver for 2D airfoil aerodynamics; runs orders of magnitude faster and enables | Novel integration into multi-fidelity UAV workflow |

| | | | |
|---|---|---|---|
| | | screening of thousands of airfoil shapes | |
| Vortex Lattice Method (VLM) | Standard aerodynamics theory; no patent | Used as Stage 1 low-fidelity screening tool via AVL and VSPAERO; validated against Ansys Fluent RANS results | Standard tool, novel use in automated pipeline |
| IPOPT Gradient-Based Optimizer | Waechter & Biegler (2006); open-source | Applied as the core optimization solver; fed by multi-start Latin hypercube seeding to mitigate local optima | Novel seeding strategy combined with gradient solver |
| Latin Hypercube Sampling (LHS) | McKay et al. (1979); no patent | Used as multi-start seeding strategy to initialize IPOPT from diverse starting points across the 32-variable design space | Application to wing opt. design space is novel |
| OpenVSP + VSPAERO | NASA (open-source); not patented | Used for 3D wing geometry definition and mid-fidelity aerodynamic analysis in Stage 2 of the fidelity pipeline | Integration into automated multi-fidelity chain |
| Ansys Fluent RANS CFD | Ansys, Inc. (commercial license) | Used for high-fidelity validation of final optimized designs at Mach 0.8, 10,000 ft AGL, 3-degree angle of attack | Validation tool; results: L/D = 21.9, Cd = 0.0032, Cl = 0.0701 |

**6.5** Conclusion

The new knowledge developed through this project spans classical aerodynamic theory, numerical optimization methods, and machine learning surrogate modeling -- a combination that reflects the direction modern aerospace design practice is moving. The team's primary innovation is not the invention of a new scientific principle but the novel integration of existing principles and open-source tools into a coherent, validated, reusable workflow applied to a specific operational aircraft for a real industry partner. The 3.3% range improvement on the XQ-58A Valkyrie is a traceable, physics-grounded outcome of that integrated design process.

# Nomenclature Summary

## Aircraft & Wing Parameters

| Symbol | Description | Formula / Context |
|---|---|---|
| S | Reference planform area (usually wing) | L=CL qS, D=CD qS |
| b | Wing span | Used in aspect ratio and rolling moment |
| c | Wing or airfoil chord | Used in coefficients (CM ,Cd ), c¯ is the mean aerodynamic chord |
| λ | Wing taper ratio | λ=ctip /croot |
| Λ | Wing sweep angle | Used in high-speed aerodynamics |
| Γ | Dihedral angle | Influences lateral stability |
| h | Altitude | Affects air density and speed of sound |

**Flow & Force Coefficients**

These coefficients normalize forces and moments by dynamic pressure (q=(½)ρV^2) and a reference area (S) or chord (c)

| Symbol | Description (3D / Finite Wing) | 2D / Airfoil Equivalent |
|---|---|---|
| CL | Lift coefficient | Cl (airfoil lift coefficient) |
| CD | Total drag coefficient | Cd (airfoil drag coefficient) |
| CM | Pitching moment coefficient | Cm (airfoil pitching moment coefficient) |

**Drag Breakdown:**

- C_Do : Zero-lift drag coefficient (primarily from skin friction and pressure drag at zero lift).
- C_Di : Induced drag coefficient (drag due to lift, caused by wingtip vortices).
- CD =C_Do +C_Di

| Symbol | Description | Context |
|---|---|---|
| T | Thrust | Tavail (available), Treq (required, Treq =D) |
| P | Power | Pavail (available), Preq (required, Preq =DV) |
| c | Specific fuel consumption (reciprocating engines) | Used for range/endurance calculations |
| ct | Thrust-specific fuel consumption (turbine engines) | Used for range/endurance calculations |
| R | Range | Total distance an aircraft can fly |

| | | |
|---|---|---|
| n | Load factor | Ratio of lift to weight (L/W) |
| Ps | Specific excess power | V(T−D)/W; a measure of climb performance |

| Symbol | Description | Context |
|---|---|---|
| α | Angle of attack | Angle between chord line and freestream velocity V∞ |
| αL=0 | Zero-lift angle of attack | Angle where L=0 |
| ϵ | Downwash angle | Angle by which the flow is deflected downward behind the wing, caused by vortices |
| V | Velocity (magnitude) | V∞ is the freestream velocity |
| M | Mach number | Ratio of velocity to speed of sound (V/a) |
| Re | Reynolds number | ρVx/μ; relates inertial to viscous forces, crucial for boundary layer behavior |
| ρ | Air density | Varies with altitude and temperature |

## Key Concepts in Wing Aerodynamics

- Finite Wing vs. Airfoil (2D vs. 3D)
- Airfoil (2D): A cross-section of a wing, theoretically in a flow that extends infinitely in the spanwise direction. It generates l (lift) and d (drag) per unit span.
- Finite Wing (3D): An actual wing of finite length (span, b). The key distinction is the presence of trailing vortices at the wing tips.
- These vortices cause the air behind the wing to be deflected downward (downwash, ϵ).
- Downwash reduces the effective angle of attack and, therefore, the lift, especially near the wing tips.
- Downwash also creates induced drag (Di ) as the local lift vector is tilted backward.
- Wing Twist and Taper
- These features are used to control the spanwise distribution of lift and manage the effect of downwash.

| Characteristic | Description | Purpose / Effect |
|---|---|---|
| Geometric Twist (Washout) | The wing is physically twisted so the tip airfoil has a lower angle of attack than the root airfoil. | Helps ensure the wing root stalls before the tip, providing aileron control after initial stall. Reduces induced drag. |
| Aerodynamic Twist | The airfoil shape (camber) changes along the span. | Effectively acts like geometric twist to control lift distribution. |
| Taper (λ<1) | The wing tip chord (ctip ) is smaller than the root chord (croot ). | Improves structural efficiency and reduces weight. Can be combined with twist to approximate an elliptical lift distribution. |

- Elliptical Lift Distribution (Ideal)
- A wing that is tapered and/or twisted to achieve an elliptical spanwise distribution of lift will have a constant downwash ($\epsilon$) at every spanwise position.
- This uniform downwash results in the minimum possible induced drag ($C_{Di}$ ) for a given span and lift.
- The span efficiency factor (e or eo ) is used to quantify how close an actual wing's induced drag is to this ideal minimum.

## Different Flight Regimes

The notes mention three broad flight regimes, which primarily relate to the Mach number (M):

| Regime | Mach Number Range | Key Aerodynamic Features |
|---|---|---|
| Subsonic | M<0.75 approx. | Air flow is incompressible (or nearly so); standard lift/drag theory applies. |
| Transonic | 0.75<M<1.2 approx. | Highly complex; localized flow can be supersonic, leading to shock waves and a rapid increase in wave drag (Dwave ). |
| Supersonic | M>1.2 approx. | Flow everywhere is faster than the speed of sound; lift and drag are generated by distinct shock and expansion waves. Wing sweep ($\Lambda$) becomes critical. |
| Hypersonic (Implied) | M>5 approx. | Extreme heat and chemical effects dominate, often requiring specialized, blunt-body shapes. |

## 15.2. Bill of Materials

| Item | Category | Item / Part | Description / Purpose | Qty | Unit | Unit Cost (USD) | Ext. Cost (USD) | Supplier / Source | Status | Lead Time (days) | Notes / Assumptions |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Software | MATLAB + Optimization Toolbox | Optimization scripting, GA/gradient methods | 1 | license | $0.00 | $0.00 | MathWorks / UGA | Planned | 0 | Assume covered by university; set price if not. |
| 2 | Software | Ansys Fluent (Academic) | High-fidelity CFD validation runs | 1 | license | $0.00 | $0.00 | Ansys / UGA | Planned | 0 | Often available via academic license. |
| 3 | Software | OpenVSP | Parametric geometry generation for low-fidelity stage | 1 | package | $0.00 | $0.00 | NASA / OpenVSP | On-hand | 0 | Free/open-source. |
| 4 | Software | AVL | Vortex-lattice aerodynamic evaluation | 1 | package | $0.00 | $0.00 | MIT / AVL | On-hand | 0 | Free (research/education). |
| 5 | Software | Python | Automation/glue code between tools | 1 | package | $0.00 | $0.00 | Python.org | On-hand | 0 | Free/open-source. |
| 6 | Software | CAD (AutoCAD/SolidWorks) | CAD edits, STEP export | 1 | license | $0.00 | $0.00 | Autodesk/Dassault / UGA | Planned | 0 | Use whichever is available. |
| 7 | Computing | HPC compute allocation | CFD batch runs; queue time and cores | 2000 | core-hr | $0.00 | $0.00 | UGA HPC | Planned | 0 | Track usage; add internal rate if needed. |
| 8 | Computing | Local workstation | Pre/post-processing, CAD, scripting | 1 | unit | $0.00 | $0.00 | Team / Lab | On-hand | 0 | If purchasing, replace $0 with actual. |
| 9 | Computing | External SSD (2 TB) | Store CFD cases/results and backups | 1 | ea | $0.00 | $0.00 | UGA lab | Needed | 3 | Optional but recommended. |
| 10 | Computing | Cloud backup (1 year) | Offsite backup for key files | 1 | yr | $0.00 | $0.00 | Cloud provider | Optional | 0 | Can be $0 if using university storage. |
| 15 | Documentation | Poster printing | End-of-year showcase | 1 | ea | $0.00 | $0.00 | UGA print services | Planned | 2 | Capstone will pay for it |
| 16 | Documentation | Report printing/binding | Hardcopy for client/faculty | 3 | ea | $0.00 | $0.00 | UGA print services | Planned | 2 | Capstone will pay for it |
| 17 | Misc | Transportation | Meetings, campus travel | 1 | lot | $300.00 | $300.00 | Personal / UGA | Planned | 0 | Budget item. |
| 18 | Misc | Textbook/reference materials | Optional references | 1 | lot | $0.00 | $0.00 | AIAA/Google | Optional | 0 | Budget item. |
| | | | | | TOTAL | | $300.00 | | | | |

## 16. Group Member Contributions

| Team Member | Project Contributions | Final Report Contributions | ABET 5 Contributions | Signature |
|---|---|---|---|---|
| **Griffin Cantrell** | Led project communication, sponsor coordination, and team update organization. Supported development of the multi-fidelity wing optimization workflow through OpenVSP modeling, aircraft systems research, project planning, and integration of technical requirements into the overall design process. Contributed to research on aircraft performance, workflow structure, stakeholder needs, and industry-relevant design practices. | Developed and maintained major project documentation, including the Team Charter, Project Charter, meeting agendas, client meeting records, customer discovery materials, tables, figures, and report organization. Contributed to full-document editing, formatting, technical consistency, and alignment of the final report with project objectives and sponsor expectations. | Supported effective team coordination by organizing communication, documenting meetings, tracking project expectations, and helping align team efforts with milestone requirements. Facilitated collaboration between technical subteams by translating sponsor feedback, customer needs, and project goals into actionable tasks. Contributed to project planning, stakeholder communication, and professional documentation throughout the design process. | *Griffin Cantrell* |
| **Edwin Harper** | Structural Guardrails & Aerodynamic Loading Limits, Fuel Volumetric Constraints, Genetic Algorithm, Validation Testing, Gantt Chart & Generating Schedule | Full Document Editing & Formatting, Verification & Validation, References & Appendices, Executive Summary, Gantt Chart scheduling | Managed Project Gantt Chart Schedule, Facilitated Teamwork, Established validation goals, ensured milestone deadlines met | *Edwin Harper* |
| **Garrett Kuhn** | Contributed to development of the MATLAB genetic algorithm optimizer. Focused primarily on atomization between tools used throughout the project. Focused on OpenVSP | Full Document Editing & Formatting, Project Summary and Recommendations, Contributed to drafting and revision across all major section of the final report, Body Content, Engineering | Collaborated across all phases of the engineering design process; coordinated with team members on the transition from original MATLAB to Python workflow, supported team | *Garrett Kuhn* |

| | | | | |
|---|---|---|---|---|
| | validation through VSPAero and structural analysis of wing designs within the open-source model, focused on developing and validating how weight was utilized and changed through the optimizastion workflow. | Design Process, ABET criteria checks. | decision-making on tool selection and scope changes. | |
| **Cooper Mendlinger** | In depth research into theoretical aerodynamic equations for estimates of wing performance. Python MDO with 2D and 3D optimization with structural, aero, and stability constraints. | Full Document Editing & Formatting, Project Impact, Design for X, IP Commercialization & Patenting, Prototype / Product Development. Appendix C Design Documentation, Disclaimer. | Facilitated Teamwork<br><br>Preparing new content for meetings with client | *Cooper Mendlinger* |
| **Randal Richards** | In depth research of MATLAB principles and optimization pathways. Contributed to construction of MATLAB framework for parameters, equations, and operational capacity. OpenVSP work and construction of integration between programs. | Full Document Editing & Formatting, Decision Matrix, Patent Table, FMEA, New Knowledge Development, Design Concepts, Conceptual Designs, Figures, Appendices, Engineering Design Process, ABET Criteria checks | Collaborated across CFD, Python, MATLAB, OpenVSP, and roles to develop Optimization workflow. Created hierarchies for documents, completed and submit team content, formatting and presentation help. Helped with all aspects of project.<br><br>Facilitated Teamwork | *Randal Richards* |
| **Martin Vassilev** | Complete computational fluid analysis (CFD) of all candidate wing designs. From geometry to mesh to results validation throughout the entire project. OpenVSP | Full Document Editing & Formatting, Cover Page, Design Objectives, Prototype Refining / Evaluation, FMEA, QFD, Engineering Requirements, Specifications, and | Facilitated teamwork. Ensuring coordination between group members to get milestones done on time. Preparing content for meetings. | *Martin Vassilev* |

| | | | | |
|---|---|---|---|---|
| | work modeling aircraft. | Codes, Standards, and Regulations | | |

# 17. Project Summary and Recommendations

The project successfully developed and documented a repeatable, CAD-centric wing design optimization pipeline that addresses the inefficiencies of disconnected conceptual design processes. By integrating parametric CAD modeling, multi-fidelity aerodynamic analysis, and structural considerations into a unified workflow, the team created a toolset capable of rapidly translating conceptual wing ideas into validated performance metrics for aircraft systems. The framework allows for the exploration of a large design space which spans 39 variables including planform geometry, CST airfoil coefficients, structural weight, and flap sizing while maintaining a focus on cruise range and aerodynamic efficiency.

The integration of low-fidelity screening and high-fidelity CFD validation successfully demonstrated a clear improvement over traditional manual iteration. The workflow now provides a transparent, automated, and industry-relevant process that balances accuracy, design trade-offs, and computational efficiency. By aligning the project with established standards, the team ensured that the resulting workflow is technically defensible, scalable, and adaptable for future design requirements.

**Recommendations for Future Capstone Teams**

Future teams continuing this project should focus on the following technical extensions:

- Expand the design variable set to include control surface sizing, dihedral, and multi-section airfoil stacks rather than the current root-and-tip CST parameterization
- Replace the 1D cantilever beam structural model with a higher-fidelity FEA-based weight estimate, ideally coupled inside the optimization loop rather than a post-check
- Strengthen the in-notebook pyBaram CFD comparison by adding a 3-mesh grid convergence study and automated GCI calculation, completing the V&V plan laid out earlier in our report
- Add a mission analysis layer that scores design across multiple flight conditions (takeoff, climb, cruise, loiter, descent) rather than optimizing solely at a single cruise point
- Validate the optimizer against additional aircraft beyond the three benchmark cases to broaden each class preset's calibration data
- Incorporate the UGA Wing Tunnel Team's experimental data into the validation loop once their test platform is operational

**Recommendations for Client Implementation**

For Lockheed Martin to adopt this workflow into conceptual design practice:

- Run the validation suite (737, RQ-4, C-130J) against a current internal sizing tool to benchmark agreement and build confidence in the pipeline before applying it to active programs
- Use the Python notebook as a front-end screening tool that feeds candidate designs into existing Lockheed in-house aerodynamic and structural codes for final analysis
- Maintain the open-source dependency stack (AeroSandbox, NeuralFoil, IPOPT, CasADi, pyBaram) under internal version control to ensure long-term reproducibility, since these tools update independently

- Treat the 39-variable formulation and Raymer weight presets as starting points and tailor variable bounds to the specific aircraft class or program being studied
- Consider extending the user interface to allow saved configuration files so engineers can reproduce or share specific runs with documented inputs

**Limitations**

The team acknowledges several limitations in the current state of the workflow. The cruise-only optimization does not capture takeoff, landing, or maneuvering loads, which explains the sweep mismatch observed in the 737-800 benchmark case. The pyBaram CFD cell included in the notebook provides a documented path to high-fidelity validation but has not yet been run as a converged grid independence study against the optimized geometries. The Raymer empirical weight equations used for structural sizing are calibrated to historical aircraft and may not extrapolate well to newer UCAV configurations. NeuralFoil aerodynamic predictions are validated up to approximately Mach 0.90, which constrains the tool's applicability for supersonic designs. Future work should address each of these before the workflow is used for design decisions outside the conceptual phase.'

# References


[1] Air Transport Action Group, Aviation Benefits Beyond Borders, Geneva, 2020.

[2] Marler, R. T., and Arora, J. S., "Survey of Multi-Objective Optimization Methods for Engineering," Structural and Multidisciplinary Optimization, Vol. 26, No. 6, 2004, pp. 1845–1875.

[3] Martins, J. R. R. A., and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," AIAA Journal, Vol. 51, No. 9, 2013, pp. 1–32. doi:10.2514/1.J051901.

[4] Rao, S. S., Engineering Optimization: Theory and Practice, 4th ed., John Wiley and Sons, Hoboken, NJ, 2009.

[5] Gloudemans, J., Saiki, T., and Edwards, J., "OpenVSP: A Parametric Geometry Tool for Conceptual Aircraft Design," AIAA Scitech 2019 Forum, American Institute of Aeronautics and Astronautics, 2019. doi:10.2514/6.2019-0OpenVSP.

[6] Drela, M., and Youngren, H., AVL: A Vortex Lattice Program for Aerodynamic and Flight Dynamic Analysis, MIT Aero and Astro, Cambridge, MA, 2014.

[7] Ansys Fluent, Ansys, Inc., Canonsburg, PA, 2025. Software.

[8] MATLAB, Version R2025a, The MathWorks, Inc., Natick, MA, 2025. Software.

[9] Optimization Toolbox, , Inc., Natick, MA, 2025. Software Documentation.

[10] Raymer, D. P., Aircraft Design: A Conceptual Approach, 6th ed., American Institute of Aeronautics and Astronautics, Reston, VA, 2018.

[11] Nikolai, L. M., Fundamentals of Aircraft and Airship Design: Volume I – Aircraft Design, American Institute of Aeronautics and Astronautics, Reston, VA, 2010.

[12] Lee, D. S., Fahey, D. W., Skowron, A., Allen, M. R., Burkhardt, U., Chen, Q., Doherty, S. J., Freeman, S., Forster, P. M., Fuglestvedt, J., Gettelman, A., De León, R. R., Lim, L. L., Lund, M. T., Millar, R. J., Owen, B., Penner, J. E., Pitari, G., Prather, M. J., Sausen, R., and Wilcox, L. J., "The Contribution of Global Aviation to Anthropogenic Climate Forcing for 2000 to 2018," Atmospheric Environment, Vol. 244, 2021, Paper 117834, doi:10.1016/j.atmosenv.2020.117834.

[13] Boeing Commercial Airplanes. (2013). *737 Airplane Characteristics for Airport Planning* (Document D6-58325-6). The Boeing Company. Retrieved from https://www.boeing.com/commercial/airports/plan_manuals.page

[14] Deagel. (n.d.). *RQ-4 Global Hawk*. Retrieved March 5, 2026, from https://www.deagel.com/aerospace%20forces/rq-4%20global%20hawk/a000556

[15] United States Air Force. (2014, October 27). *RQ-4 Global Hawk*. https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104516/rq-4-global-hawk/

[16] Air & Space Forces Magazine. (n.d.). *RQ-4*. Retrieved March 5, 2026, from https://www.airandspaceforces.com/weapons/rq-4/

[17] Kass, H. (2025, November 20). How the RQ-4 Global Hawk changed American intelligence collection forever. *The National Interest*. https://nationalinterest.org/blog/buzz/how-rq-4-global-hawk-changed-american-intelligence-collection-forever-hk-112025

[18] United States Air Force. (n.d.). C-130 Hercules. Retrieved from https://www.af.mil/About-Us/Fact-Sheets/Display/Article/1555054/c-130-hercules/

# Appendix A

## A-1. Benchmarking Summary

A benchmarking evaluation was conducted to compare the three proposed design concepts against two industry-representative competitor approaches (Pure CFD and Linear Solutions). The assessment focused on five key stakeholder needs: fidelity range, computational efficiency, discoverability of the optimum, usability, and scalability. Each attribute was scored on a scale from 1 (weakest) to 5 (strongest), reflecting expected performance in a conceptual aerodynamic design environment.

Concept 1, the Hierarchical Open-Source Pipeline, achieved the highest overall score. It demonstrated strong performance across fidelity range, computational efficiency, discoverability of the optimum, and scalability. This reflects its ability to blend low-fidelity and high-fidelity tools in a structured sequence, providing both speed and depth when needed. Concept 2, the Comparative Optimization Model Analysis, followed closely behind, excelling in the discoverability of the optimum due to its ability to incorporate multiple optimization strategies but offering moderate computational efficiency. Concept 3, the Interactive Design Space

Explorer, performed well in usability but scored lower in computational efficiency and scalability, indicating that its strength lies primarily in user interaction rather than robust optimization or rapid iteration.

The competitor approaches showed clear limitations. Pure CFD scored modestly, providing reasonable discoverability and usability but exhibiting poor computational efficiency due to the high cost of CFD-only workflows. Linear Solutions scored lowest overall, reflecting significant limitations in fidelity, computational power, and optimization capability. These results underscore the inherent weaknesses of single-method approaches in conceptual aerodynamic design, where rapid iteration and multi-fidelity transitions are essential.

Overall, benchmarking results indicate that Concept 1 best satisfies stakeholder requirements, aligning closely with project goals related to efficiency, model fidelity, scalability, and optimization transparency. The full detailed scoring chart is included below for reference.

**Customer Benchmarking (5 = Strongest, 1 = Weakest)**

| Stakeholder Needs | Concept 1: Hierarchical Open-Source Pipeline | Concept 2: Comparative Optimization Model Analysis | Concept 3: Interactive Design Space Explorer | Competitor 1: Pure CFD | Competitor 2: Linear Solutions |
|---|---|---|---|---|---|
| Fidelity Range | 5 | 4 | 3 | 3 | 1 |
| Computational Efficiency | 4 | 3 | 2 | 1 | 1 |
| Discoverability of Optimum | 4 | 5 | 2 | 4 | 2 |
| Usability | 3 | 3 | 5 | 2 | 4 |
| Scalability | 4 | 4 | 2 | 3 | 2 |
| **Total** | **20** | **19** | **14** | **13** | **10** |

LOCKHEED MARTIN

## A-2. Solver and Workflow Capability Review

This subsection provides comparison data on aerodynamic modeling tools, solver fidelity, mesh generation capability, and multidisciplinary integration. Low-fidelity tools support rapid iteration during optimization, while high-fidelity CFD platforms enable final aerodynamic validation.

## A-3. Intellectual Property Landscape Overview

The aerospace sector is saturated with innovative configurations, analysis methods, and design concepts, but unlike many other industries, protecting design ideas is uniquely difficult. As stated by Mr. William "Bob" Clark of Lockheed Martin, "In aerospace, everything has been tried at least once; if you have an innovative idea, it is likely not too innovative. Research different concepts and learn from those before you." This perspective highlights a central challenge of aerospace design: most concepts have historical precedent, and genuine novelty is rare. Effective innovation therefore relies on studying prior work, identifying its limitations, and developing improved combinations of existing ideas rather than pursuing entirely new concepts.

## A-4. Industry Background

The aerospace industry, spanning both commercial and defense sectors, is driven by continuous innovation toward higher efficiency, improved performance, and reduced operational cost. Increasing computational capability has transformed aircraft development by enabling multi-fidelity simulation workflows that reduce dependence on full-scale prototyping and wind-tunnel testing. These digital design environments support rapid iteration of aerodynamic concepts while lowering development time and cost.

Wing optimization plays a central role in these advancements. Small changes in wing shape—such as adjustments to camber, chord distribution, twist, and winglets—can produce measurable improvements in fuel efficiency, stability, and mission performance. Low-fidelity aerodynamic tools support rapid early evaluation, while high-fidelity CFD provides more detailed analysis where needed. This combination of methods has become standard practice at major aerospace firms including Lockheed Martin, Boeing, and Airbus.

Academic institutions also contribute significantly to the advancement of multidisciplinary design optimization (MDO), though challenges remain. Low-fidelity tools may oversimplify complex flow physics, whereas high-fidelity simulations require substantial computational resources. Balancing these trade-offs remains a key focus of the industry. The multi-fidelity optimization workflow developed in this project directly addresses these challenges by enabling fast early-stage analysis supported by targeted high-fidelity validation.

## A-5. Industry Fundamentals

The aerospace industry in both military and civilian fields has remained one of the most competitive and innovative sectors globally. Aircraft design must meet rigorous safety,

performance, efficiency, and cost requirements, making optimization one of the key foundations in aerospace research and design. Rapidly progressing computational power and ability to process input data through software simulations has revolutionized the aerodynamic optimizations.

This has allowed the industry to move away from physical and wind tunnel prototyping and in turn shortened and lowered cost for the design process. This leap forward has allowed for innovative optimization and concepts to be proposed and tested at much higher rates. Software that is able to compute fluid dynamics and finite element analysis and pushed forward the ability to explore variables and inputs at a continuously increasing rate.

Lockheed Martin has been one of the legacy aerospace innovators since the industry began and continues to be among the largest aerospace and defense company in the world. One of Lockheed Martin's driving principles in recent years has been investing in research and development for innovative workflows and computational software that allows them to pursue optimization and experimental concepts to accelerate development times and quality. Considering this, optimization remains a cornerstone that allows the company to continue in its dominant role in the aerospace industry.

## A-6. Stakeholders' Needs

**Stakeholder Summary**

This project engages multiple stakeholders across Lockheed Martin, the University of Georgia, and the broader aviation community. Each stakeholder provides unique insight into conceptual design practices, workflow expectations, and performance requirements. Their needs, gathered through interviews and milestone documentation, help shape the structure and priorities of the multi-fidelity wing optimization workflow.

Mr. Matthew Seay serves as the project sponsor and primary stakeholder. He provides strategic direction, milestone oversight, and ensures alignment with Lockheed Martin engineering standards. He requires a credible and traceable multi-fidelity workflow, clear biweekly progress updates, well-defined milestones, reproducible code and data, strong documentation, and concise deliverables including validated case studies, selected tools, and a complete final workflow summary.

Mr. Bob Clark is a conceptual design engineer and aircraft sizing expert at Lockheed Martin. He offers guidance on parameter sensitivities and early-stage design trade studies. He needs rapid and dependable sizing outputs, intuitive control over geometric variables such as span, sweep, camber, and twist, assumptions that allow quick hand-check verification, clean plots and tables for concept studies, and documentation that integrates seamlessly with conceptual design activities.

Mr. Paul McClure, a Lockheed Martin Fellow, contributes extensive experience in aerodynamics and performance optimization. He needs decision-ready results supported by clearly stated inputs and assumptions, concise summaries with effective before-and-after comparisons, regular updates, documented risk tracking, and a reusable workflow with traceable logic that supports both low- and high-fidelity modeling.

The Wind Tunnel Team at the University of Georgia acts as a secondary stakeholder. Their work involves designing an aerodynamic testing platform that must align with our simulation outputs. They require consistent geometry handoff, agreed test conditions, standardized data formats, documented assumptions, coordinated schedules for model delivery and tunnel access, and clear communication throughout the design and testing process.

Dr. Eliza Banu serves as the faculty supervisor, providing academic and technical oversight. She needs defined milestones and deliverables, reproducible analysis, clean and traceable documentation, clear justification for modeling choices, and advance notice of required software, hardware, or laboratory resources.

Madison Warner, a certified commercial pilot and instrument instructor, provides operational insight into flight performance and handling. She needs design outputs that relate directly to pilot-relevant metrics such as lift-to-drag behavior, stall margins, takeoff and climb performance, and turbulence response. She benefits from clear visualizations, brief summaries, early access to simulation results, and opportunities to provide feedback that can be incorporated into design updates.

The previous Lockheed capstone group serves as a tertiary stakeholder contributing high-fidelity CFD datasets and expertise related to transonic wing performance and deep learning workflows. They require clean data exchange, consistent test conditions, comparable performance metrics, shared baseline geometries, version-controlled cases, and a simple, coordinated schedule for joint validation tasks.


| Stakeholder | Primary Focus | Key Needs |
|---|---|---|
| Lockheed Client | Project oversight | Feasibility, alignment, communication |
| Lockheed Martin (LM) | Usable engineering tool | Validated, scalable, professional workflow |
| LM Technical Fellows | Technical depth | Accurate models, innovation, efficiency |
| Supervising Professor (UGA) | Student learning & rigor | Sound methodology, documentation, progress |
| University of Georgia (UGA) | Academic success & partnership | Educational impact, successful collaboration |
| Wind Tunnel Capstone Group | Experimental collaboration | Interface coordination, validation plan |
| Public Pilot | Real-world applicability | Safety, performance, practicality |

| Priority Level | Rationale |
|---|---|
| High Priority | High influence and direct technical or academic oversight. Their needs shape core project direction and deliverable acceptance. Consistant communication required |
| Medium Priority | Important collaborators and institutional stakeholders; impact project integration and academic success. |
| Low Priority | Provides valuable operational insight but lower influence on technical decisions. Used primarily for validation and context. |

# A-7. Problem and Need Analysis

### 1.1 Perceived Problems/Needs

Modern aerospace conceptual design faces increasing pressure to evaluate a growing number of wing configurations within shorter timelines while maintaining accuracy across multiple flight regimes. The prevalence of computational bottlenecks in high-fidelity simulations, combined with the incidence of errors produced by relying solely on low-fidelity models, contributes to inefficient early-stage decision-making. Across industry and academia, teams frequently encounter challenges where early design iterations become slow, inconsistent, or disconnected from downstream validation tools. The need for a unified, repeatable multi-fidelity workflow arises from this gap, as designers require both speed and accuracy to remain competitive.

### 1.2 Operating Environment

These problems occur within preliminary aircraft design environments involving CAD modeling platforms, vortex-lattice solvers, high-fidelity CFD tools, and light-to-moderate computing resources typical of university or conceptual-design settings. Personnel include aerospace engineers, aerodynamic analysts, conceptual designers, and student researcher teams. Inputs consist of geometric parameters, mission constraints, initial aerodynamic estimates, and performance targets. Outputs include lift-to-drag predictions, aerodynamic load distributions, CAD geometries, and validated CFD performance metrics. The workflow spans both digital simulation environments and coordinated exchanges with external teams such as wind-tunnel groups.

### 1.3 Problem/Need Factors

The absence of a structured multi-fidelity workflow affects multiple levels of the design process. Low-fidelity models may oversimplify flow physics, causing inaccurate predictions during optimization, while exclusive use of high-fidelity CFD increases cost, slows iteration, and restricts the design space explored. These issues create risks such as overfitting to incomplete data, design stagnation, unverified

assumptions, and misalignment between early sizing estimates and final performance. Timelines are particularly sensitive, as delays in initial sizing and iteration propagate through the full design cycle and complicate downstream testing and integration.

### 1.4 Problem/Need Presentation

Symptoms of these problems include slow iteration speeds, mismatched predictions between low- and high-fidelity tools, inconsistent data formatting between teams, difficulty validating early designs, and unexpected deviations in CFD performance. Other indicators include avoidable redesigns late in the process, ambiguous parameter sensitivities, and inefficient communication between conceptual design and analysis groups.

### 1.5 Problem/Need Outcomes

If left unaddressed, these issues lead to increased development cost, reduced design quality, and greater program risk. Poor early-stage predictions may propagate to flight-critical phases, requiring extensive redesign later or reducing operational efficiency. For organizations, these failures can result in wasted engineering hours, missed deadlines, and diminished competitiveness. For student teams, they lead to fragmented workflows, unreliable results, and limited educational value. In high-stakes environments such as defense aerospace, persistent inefficiencies can compromise mission capability and long-term fleet performance.

### 1.6 Economic Impact

The economic cost of inefficient preliminary wing design includes unnecessary CFD compute hours, extended labor expenditure, slow concept evaluation, and reduced ability to explore the design space. In industry, these costs accumulate rapidly, as delays in early design stages affect multi-million-dollar development schedules. High-fidelity simulation loads can incur significant cloud-compute or HPC usage fees, while inaccurate early predictions lead to expensive redesign cycles. For academic teams, the economic impact is reflected in limited access to computing hardware, reduced productivity, and constrained project scope. Over time, these inefficiencies increase the total cost of developing competitive unmanned aircraft systems.

# Appendix B

## B-1. Client Meeting / Site visits

## Generic Client Meeting Agenda

The following is an example of a meeting agenda provided to our client.

---

UNIVERSITY OF GEORGIA
College of Engineering

LOCKHEED MARTIN

**Agenda**
**Introductory Meeting**
Aircraft Wing Design Optimization
September 15, 2025

**Attendees:**

**UGA:**

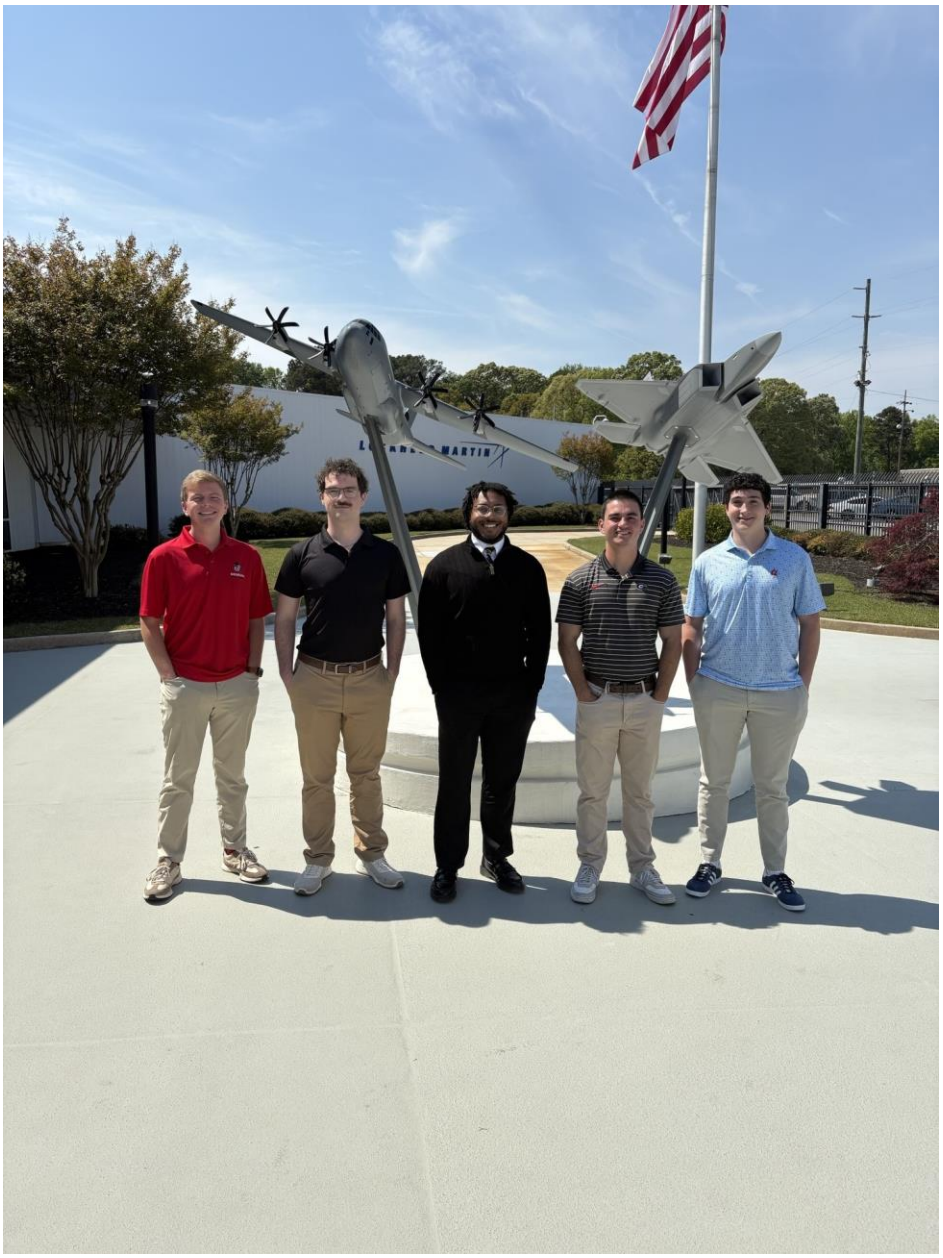
Project Supervisor: Dr. Eliza Banu

Students: Griffen Cantrell, Edwin Harper, Garrett Kuhn, Randal Richards, Martin Vassilev, Cooper Mendlinger

**Lockheed Martin:** Mr. Matthew Seay

| Topics | Lead |
|---|---|
| 1. Introductions | UGA/LM |
| 2. Defining Scope & Goals | LM |
| 3. Outcomes/Expectations | LM |
| 4. Layout Timeline | LM |
| 5. Q&A/Open Discussion | UGA/LM |
| 6. Schedule Reoccurring Meeting | UGA/LM |
| 7. Closing Remarks | UGA/LM |

## Site Visit



On April 8, 2026, The Lockheed Team had the opportunity to visit Lockheed's Marietta assembly line of the C-130.

## Attendance (Client Meetings + Site Visit)

| Date | Student | | | | | | Instructor | Sponsor | |
|---|---|---|---|---|---|---|---|---|---|
| 2025 | Griffin Cantrell | Edwin Harper | Garrett Kuhn | Cooper Mendlinger | Randal Richards | Martin Vassilev | Dr. Eliza Banu | Matthew Seay | Thomas Wells |
| Oct/06 | | | | | | | | | |
| Oct/20 | | | | | | | | | |
| Nov/03 | | | | | | | | | |
| Nov/17 | | | | | | | | | |
| Dec/01 | | | | | | | | | |
| Feb/09 | | | | | | | | | |
| Feb/16 | | | | | | | | | |
| Mar/02 | | | | | | | | | |
| Mar/23 | | | | | | | | | |
| Apr/06 | | | | | | | | | |
| Apr/08 | | | | | | | | | |
| Apr/28 | | | | | | | | | |

Legend: Present | Absent | N/A

# Agenda
## Introductory Meeting

Aircraft Wing Design Optimization

September 15, 2025

**Attendees:**

**UGA:**

Project Supervisor: Dr. Eliza Banu

Students: Griffen Cantrell, Edwin Harper, Garrett Kuhn, Randal Richards, Martin Vassilev, Cooper Mendlinger

**Lockheed Martin:** Mr. Matthew Seay

| Topics | Lead |
|---|---|
| 1. Introductions | UGA/LM |
| 2. Defining Scope & Goals | LM |
| 3. Outcomes/Expectations | LM |
| 4. Layout Timeline | LM |
| 5. Q&A/Open Discussion | UGA/LM |
| 6. Schedule Reoccurring Meeting | UGA/LM |
| 7. Closing Remarks | UGA/LM |

# B-2. Project Charter

UNIVERSITY OF GEORGIA

LOCKHEED MARTIN

## Project Charter

**Project Information**

| | | | |
|---|---|---|---|
| **Project Title:** | Aircraft Wing Design Optimization | | |
| **Project Start Date:** | 9/2/2025 | **Status Date:** | 9/18/2025 |
| **Area of Business:** | Aerospace Engineering | | |
| **UGA Team Leader:** | Griffin Cantrell | **UGA Project Supervisor:** | Eliza Banu |
| **email address:** | Griffin.cantrell@uga.edu | **email address:** | Eliza.banu@uga.edu |
| | | | |
| **Capstone Partners):** | | **Capstone Partner Contact 1:** | Matthew Seay |
| | | **email address:** | matthew.d.seay@lmco.com |
| | | **Capstone PartnerContact 2:** | |
| | | **email address:** | |

**Project Purpose / Problem Definition: What is the problem? In what process?**

Lockheed Martin seeks a repeatable, CAD-centric wing design optimization pipeline to cut drag and other inefficiencies using realistic geometric and structural constraints. Engineers are going to use independent research on current aircraft wings and design processes and translate it to a digital workflow that accelerates efficiency for aircraft models. Deliverables will include: initial-to-optimized wing comparison, low & high-fidelity CFD results, documented start-to-finish process (programs, configurations, metrics), and final CAD/parametric models with performance deltas versus the baseline

**Project Impact: What is the potential impact to the business, its customers, and society?**

The optimized wing and efficient workflow will translate directly to lower fuel burn, greater range/endurance, and a maximized flight speed at 1-g CL low flight passes. The independent research gained for aerodynamics and structures gives a traceable, repeatable process as an outcome that will shorten design cycles. Manufacturing and certification see clearer constraints early, improving productivity and reliability. This reusable pipeline will scale to future applications, compounding ROI (Return on Investment) across the company.

**Goal Statement - What is the Project Goal? What does the Capstone Partner want to achieve? What is the deliverable?**

The goal of our project is to develop and document a complete aerodynamic wing optimization workflow. This workflow will integrate CAD, aerodynamic analysis (both low- and high-fidelity), structural analysis, and optimization methods for various aircraft configurations. Our capstone partner, Lockheed Martin hopes to provide students with a hands-on experience in real-world aerodynamic optimization. They hope to teach the students how to balance accuracy, efficiency, and design trade-offs using industry relevant tool and workflows. Our deliverable will be a final report and presentation summarizing the optimization workflow. This final report and presentation will also depict the students optimized wing design as well as their rationale behind key decisions, accompanied by documentation of trade-offs and lessons learned.

**What are the Project Objectives, i.e, what steps must be completed to meet your project goal(s) and customer requirements?**

| |
|---|
| We will confirm mission restraints and requirements from the client. Afterwards, we will confirm the CAD application to use and their counterparts that Lockheed Martin will have cleared for us. Using the CAD to CFD optimization toolchain we will create a workflow and document the project deliverables. Using low and high fidelity tests, we will find an optimized wing design to then compare against the baseline model and deliver the CAD for an unmanned drone, agreed-upon metrics, and a reproducible pipeline for client sign-off. |

**Projected Budget and Project Cost: What is the expected budget for the project? How much does the Capstone Partner expect a fully-realized solution to cost?**

| | | | | | |
|---|---|---|---|---|---|
| Funds Furnished by Client: | 0$ | In kind value from Client: | 0$ | Est. Project Cost: | 300$ |

**Project Timeline: What are the date ranges for each phase?**

| Problem Definition | Research | Planning | Prototyping / Refining | Delivery |
|---|---|---|---|---|
| 9/2-9/19 | 9/19-10/30 | 10/31-12/10 | 1/12-3/31 | 4/1-5/1 |

**Project Scope:**

| In Scope: | Out of Scope: |
|---|---|
| Structural considerations for wing, Parameterizing 3D models for wings, CFD validation and iterativeness, Integrated workflow to optimize wing shape through iterative analysis and modification. | Control surface impacts, Varying weather conditions, internal components of wing, full body considerations (fuselages) |

**Project Participants**

| UGA Team | | Capstone Partner | |
|---|---|---|---|
| **Name** | **Signature** | **Name** | **Signature** |
| Griffin Cantrell | *Griffin Cantrell* | Matthew Seay | |
| Edwin Harper | *Edwin Harper* | | |
| Garrett Kuhn | *Garrett Kuhn* | | |
| Cooper Mendlinger | *Cooper Mendlinger* | | |
| Randal Richards | *Randal Richards* | | |
| Martin Vassilev | *Martin Vassilev* | | |
| Dr. Eliza Banu | | | |
| | | | |

## B-3. Stakeholders Interviews

**Stakeholder: Mr. Matthew Seay**

**Stakeholder Description**

Mr. Seay serves as the primary stakeholder and funding source for the Capstone Design project. He provides financial support, sets strategic objectives, and ensures that the project aligns with Lockheed Martin's mission of advancing aerospace design and optimization. He acts as the liaison between the capstone team and Lockheed Martin engineers, reviewing deliverables at key milestones and offering feedback to maintain consistency with industry standards. With high influence and high interest in the project, his guidance and support are essential both for the successful completion of the work and for fostering future collaboration between Lockheed Martin and the University of Georgia.

**Interview**

Q) When beginning an aerodynamic wing optimization project, is it better to jump directly into CFD or start with simpler methods?
A) It is best to start with linear theory, since it allows for faster optimization of multiple variables in the early stages of the project. CFD should then be used later to check the accuracy of those results and refine the design.

Q) What are the main advantages of using linear theory early in the design process?
A) Linear theory provides a quick, efficient way to explore multiple designs without the heavy computational cost of CFD. It also gives the team an intuitive understanding of aerodynamic trends before moving into more advanced tools.

Q) At what point should we rely on CFD analysis instead of linear methods?
A) CFD should come after initial designs are developed. It serves as a way to validate whether the optimizer and linear predictions are accurate. This ensures time is spent efficiently while still building toward high-fidelity results.

Q) Were there any challenges with introducing CFD early in past projects?
A) Initial CFD codes can be complicated and time consuming. For this reason, Lockheed Martin engineers prefer to use CFD later as a checkpoint rather than as a starting point.

Q) How can our team balance efficiency with accuracy in our workflow?
A) The team should begin with linear theory for broad exploration and then use CFD strategically for validating findings and refining the optimization strategy. Results from different methods should be compared to ensure consistency.

Q) How should we handle discrepancies between linear predictions and CFD results?
A) Discrepancies should be treated as learning opportunities. These differences can identify where assumptions in linear theory break down and indicate how the design should be refined.

## Interpretation

From our discussions with Mr. Seay, we learned the importance of starting with linear theory to guide early design iterations before moving into more complex CFD analysis. This approach allows faster optimization of multiple variables and creates a foundation for checking the accuracy of higher-fidelity models later in the workflow. An expected takeaway was his emphasis on balancing efficiency and accuracy, which aligns with our project goals. An unexpected insight was how much value Lockheed Martin places on simpler models not only as a starting point but also as a validation tool for ensuring that CFD predictions are reliable. We will use this information to structure our workflow by prioritizing linear theory for fast exploration and then applying CFD strategically for refinement and validation, saving time while maintaining technical credibility.

**Stakeholder: Mr. Bob Clark**

## Stakeholder description

Mr. Bob Clark is a Conceptual Design Engineer at Lockheed Martin and an aircraft sizing expert. With his experience in conceptual design and sizing, we hope to understand how our research and workflow could support his role and how his perspectives on early-stage design can inform our optimization strategy.

## Interview

Q) In conceptual design, which wing sizing parameters, such as aspect ratio, wing loading, or taper ratio, tend to have the greatest leverage on aircraft performance during early iterations?
A) Wing area dictates wing loading, and aspect ratio has a major effect on induced drag. Designers sometimes chase induced drag too aggressively, which can negatively affect thickness-to-chord ratio. Mach number largely dictates wing thickness as well as sweep. Sweep, both at the leading and trailing edge, is very important for military missions. Application and mission determine which parameter is most critical to optimize. Design of Experiments (DOE) is used during optimization to create a matrix of cases based on the equations and variables selected.

Q) From a conceptual sizing perspective, what methods or rules of thumb do you find most reliable for estimating wing area and thickness before higher-fidelity analysis?
A) Historical data can be used to estimate parameters such as sweep for a given cruise Mach number. From there, charts provide ballpark values for key sizing quantities. Lockheed Martin uses an in-house tool that links lift, drag, and related equations to yield initial aerodynamic and weight estimates. Once these first-order estimates are available, CFD can be used to test and refine the results. Early in the process, simple data and tests are often the most valuable.

Q) How have you approached multi-role aircraft design where the platform must operate across multiple flight regimes?
A) Response not yet recorded.

Q) How do you and your team typically communicate or coordinate with aerodynamic analysts? Do you share models, summarized results, or collaborate iteratively?
A) Collaboration is challenging. Engineers generally prefer face-to-face discussion, so current communication methods can be difficult. The team works to identify each person's strengths and then leverages those strengths. Initial meetings are serious and focused; over time, humor and informal interaction can help build trust and make collaboration smoother.

**Interpretation**

Interpretation and lessons learned for this interview are still in progress and will be added once the remaining responses are collected and synthesized.

**Stakeholder: Mr. Paul McClure**

**Stakeholder description**

Paul McClure is a Lockheed Martin Fellow with over 37 years of experience at the company. He spent three decades as a Senior Staff Engineer before being named a Lockheed Martin Fellow in 2018, one of the company's highest technical honors. He holds a master's degree in Aerospace, Aeronautical, and Astronautical Engineering from the University of Illinois (1987) and is an expert in aerodynamics and project management. His experience in aircraft performance and aerodynamic optimization provides valuable guidance for our wing optimization workflow.

**Interview**

Q) At the conceptual design stage, which performance metrics do you look at first for a transonic wing, and why those over others?
A) The metrics depend on the mission of the aircraft. Range is normally the primary metric. Sustained load factor, takeoff and landing performance, loiter capability, and flying qualities are also important. For a fighter aircraft, flying qualities may be most critical, but range remains essential. The ranking of metrics changes based on mission priorities.

Q) When you are moving from low-fidelity to high-fidelity analyses, what simple "gate checks" tell you it is worth spending CFD time on a candidate wing?
A) CFD is used when the team believes it has thoroughly explored the available design space and identified a promising region. After examining ranges of sweep, wing area, and initial camber distributions, the team checks whether the wing appears appropriately sized and whether cruise and loiter conditions are satisfied. CFD is then used with simple twist distributions and camber magnitudes to determine whether the candidate design warrants further refinement.

**Interpretation**

Interpretation and lessons learned for this interview will be completed once the remaining responses are finalized.

**Stakeholder: Wing Optimization Research Group**

## Stakeholder description

Arjun Surve is an undergraduate researcher at the University of Georgia focusing on aerospace optimization of wing designs. His research investigates how deep learning can be applied to transonic wing design. His group generates high-fidelity simulations of transonic wings using Ansys Fluent and employs parametric approaches to modify wing geometries. The goal is to create a deep learning model that can produce unique wing geometries and predict their performance efficiently.

## Interview

Q) How do you define success for your project?
A) Success is defined by completing a deep learning model that can reliably generate new wing geometries while predicting their performance with high fidelity and efficiency. The approach should reduce the time and computational resources required for optimization compared with traditional CFD workflows. Metrics include accurate prediction of aerodynamic coefficients close to high-fidelity Ansys simulations, acceleration of the design process, and development of methodologies that are translatable to industry.

Q) What aircraft configurations do you prioritize?
A) The long-term goal is to account for all key wing design parameters, but current work focuses on camber. Camber significantly influences lift and transonic shock strength and location. Modifying camber can lead to increases or decreases in lift and drag.

Q) Are there specific aerodynamic parameters that are most relevant to your objectives?
A) The lift-to-drag ratio is a primary metric, since it provides a good measure of aerodynamic efficiency; higher values generally indicate greater efficiency.

Q) Do you already have access to existing transonic wing data sets for training, or do you generate this data from scratch?
A) The data set is being generated from scratch and is the main focus of the current phase of the research.

Q) Can you briefly walk through your workflow from initial CAD modeling to high-fidelity CFD simulations? What platforms, automation, and scripting did you find essential?
A) The team uses well-known wing geometries such as the ONERA M6 as a baseline. Using OpenVSP, they alter the camber of the airfoil to create related geometries and evaluate the resulting aerodynamic changes. The Adjoint Solver in Ansys is being explored to streamline the process of generating alternative airfoils from a single geometry, but this method is not yet finalized. The researcher's primary role is data acquisition to support the deep learning model; other team members focus on how to use the data for training.

Q) What were the biggest challenges or bottlenecks in generating and managing CFD data for deep learning, and how did you address them?
A) The largest challenge is the sheer amount of data required to train a reliable model. Tools such as the Adjoint Solver are being investigated to reduce the effort required to produce this dataset. A finalized solution has not yet been established.

Q) What criteria did you use to select or engineer input features for your deep learning models?
A) Features are selected based on physical significance, sensitivity to performance outputs, diversity of the data, and model interpretability.

**Interpretation**

We learned that the ONERA M6 wing is a suitable baseline for testing our optimization workflow and that Ansys Fluent and OpenVSP are valuable for high-fidelity and parametric geometry tasks. While this research focuses mainly on camber, our project will consider a wider range of parameters. Insights from this work help validate our choice of tools and highlight the importance of efficient data generation for advanced modeling approaches.

**Stakeholder: Dr. Eliza Banu**

**Stakeholder description**

Dr. Eliza Banu is a professor of Mechanical Engineering and a specialist in fluid mechanics and computational fluid dynamics. She serves as the project supervisor and is well connected within the engineering research community. Through her experience and expertise, we seek guidance on advanced CFD practices, access to computational resources, and advice on effective methods for approaching optimization problems in an academic environment.

**Interview**

Q) What background do you have with computational fluid dynamics or CFD?
A) She describes CFD as an area one continues to learn throughout a career. Approaches evolve over time, and there is an "art-like" aspect to choosing combinations of models and numerical methods. She continues to learn and refine her understanding with each project.

Q) For this project, would you see CFD as more of a validation tool or an optimization tool?
A) CFD is primarily a validation tool and secondarily an optimization tool. It provides detailed information about where potential issues originate, such as pressure zones and drag distributions. Once this information is available, CFD can then be used to refine and optimize aspects of the design.

Q) How important is the fidelity of the geometry for a project in the conceptual design phase when conducting CFD?
A) It is important to start with the simplest geometry that can capture the relevant behavior. Starting with a very complex model makes it difficult to isolate the source of problems. Complexity should be added gradually once the effect of the basic geometry is understood.

Q) Which performance parameters (lift-to-drag ratio, pressure distribution, flow separation, etc.) should we prioritize in evaluating optimized wings?
A) Lift-to-drag ratio is the first priority, but the focus depends on flight conditions such as altitude, velocity, and the phase of flight (takeoff, climb, cruise, or landing).

Q) For modeling external flow, what are some frequent mistakes students or professionals encounter in setting up the physical environment, boundary conditions, and initial conditions?

A) Common issues include enclosure shape and size, inappropriate meshing, and incorrect turbulence model selection. Rectangular enclosures can introduce artificial effects, so more rounded or streamlined domains are preferred. Improperly defined inlets or offsets relative to the leading edge can also cause problems.

Q) What results of this project would be most interesting or helpful to an educator of CFD and fluid dynamics?
A) She is most interested in the process and how different software tools are combined to achieve the final result. The academic value lies in understanding the workflow, the choices made, and how the project differs from other CFD efforts, rather than in a single specific scientific outcome.

**Interpretation**

We learned that CFD should be treated as both a validation and refinement tool rather than the starting point for the design process. Dr. Banu's emphasis on beginning with simple geometries and gradually increasing complexity reinforces the multi-fidelity approach recommended by industry stakeholders. Her focus on process and methodology also highlights the importance of documenting the workflow and explaining tool choices in a way that is useful for both education and future projects.

## B-4. Quality Function Deployment (QFD)

### Quality Function Deployment

| | |
|---|---|
| Project title: | Aircraft Wing Design Optimization |
| Project Supervisor | Dr. Eliza Banu |
| Date: | May 4th, 2026 |

Correlation:

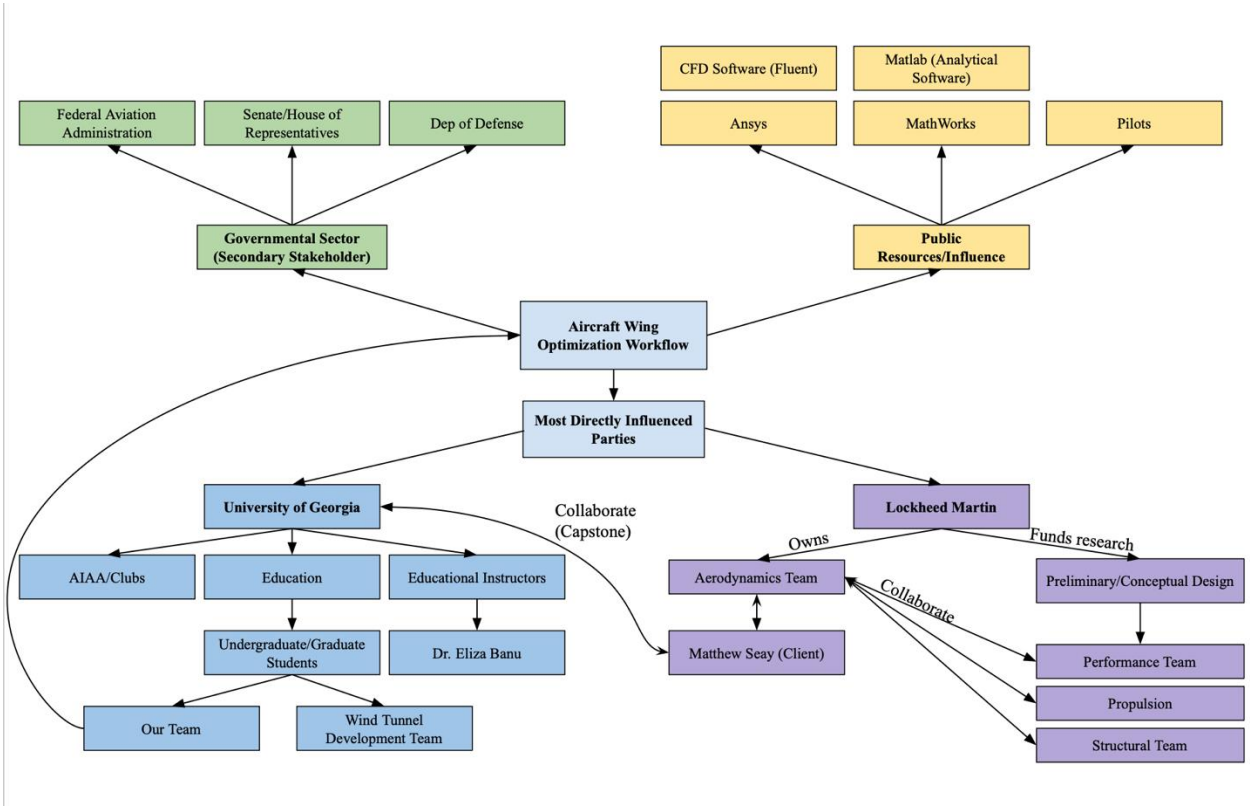
| + | . | - |
|---|---|---|
| Positive | No correlation | Negative |

Relationships:

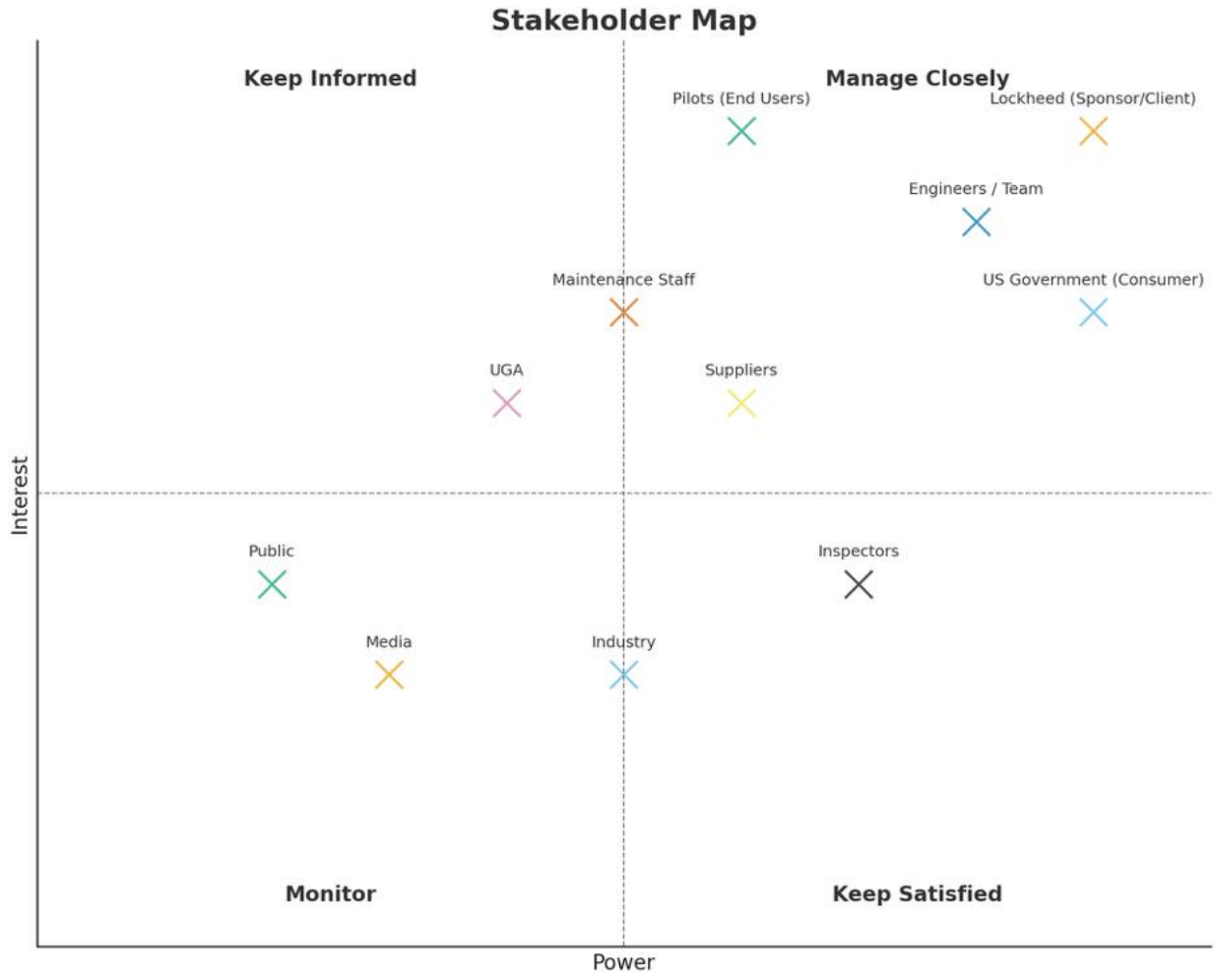
| 9 | 3 | 1 | |
|---|---|---|---|
| Strong | Moderate | Weak | None |

| | | Desired direction of improvement (↑,0,↓) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1: low, 5: high | Functional Requirements (How's) → | CFD/Aero Fidelity | Planform Geometry | CST Airfoil Coefficients | Structural Weight | Flap Sizing | Competitive evaluation (1: low, 5: high) | | | | |
| | Customer importance rating | Customer Requirements - (What's) ↓ | | | | | | Weighted Score | Satisfaction rating | Competitor rating 1 | Competitor rating 2 | Competitor rating 3 |
| 1 | | Maximize cruise range | 9 | 9 | 9 | 3 | 3 | 0 | 5 | 2 | 3 | 4 |
| 2 | | Reduce total drag | 9 | 9 | 9 | 1 | 3 | 0 | 5 | 5 | 2 | 3 |
| 3 | | Ensure structural integrity | 3 | 3 | 3 | 9 | 1 | 0 | 4 | 3 | 3 | 3 |
| 4 | | Ease of manufacturing | 1 | 1 | 1 | 9 | 3 | 0 | 4 | 4 | 3 | 3 |
| 5 | | Repeatable Digital Workflow | 9 | 3 | 3 | 3 | 3 | 0 | 5 | 2 | 3 | 4 |
| 6 | | | | | | | | 0 | | | | |
| 7 | | | | | | | | 0 | | | | |
| 8 | | | | | | | | 0 | | | | |
| 9 | | | | | | | | 0 | | | | |
| | | Technical importance score | 141 | 117 | 117 | 95 | 55 | 0 | | | | |
| | | Importance % | | | | | | 0% | | | | |
| | | Priorities rank | 1 | 2 | 2 | 4 | 5 | | | | | |
| | | Current performance | 4 | 3 | 3 | 3 | 2 | | | | | |
| | | Target | 5 | 5 | 5 | 4 | 3 | | | | | |
| | | Benchmark | 4 | 4 | 4 | 4 | 3 | | | | | |
| | | Difficulty | 5 | 3 | 4 | 4 | 2 | 1: very easy, 5: very difficult | | | | |
| | | Cost and time | 5 | 3 | 4 | 4 | 2 | 1: low, 5: high | | | | |
| | | **Priority to improve** | **1** | **2** | **2** | **4** | **5** | | | | | |

## B-5. Stakeholder Map



## B-6. Power-Interest Matrices



## B-7. FMEA

### FMEA Matrix

Failure Mode and Effects Analysis (FMEA)

| System / Item / Process Step | Potential Failure Mode | Potential Failure Effects | Severity | Causes | Occurrence | Current Controls | Detection | RPN | Recommended Actions |
|---|---|---|---|---|---|---|---|---|---|
| Name, ID number, etc. | How could the system / item / process potentially fail? | Consequential impact on other systems, departments, etc. | | All contributing factors | | Prevention / Detection | | Risk Priority Number | Steps required to reduce severity, occurrence, and detection |
| MATLAB – optimizer configuration | Poor optimizer settings (step size, tolerance, max iterations) | Premature convergence or non-convergence; sub-optimal wing passed forward as "best" design | 6 | Default settings used blindly, insufficient iteration budget, non-convex design landscape | 5 | Visual inspection of convergence history; limited sensitivity study | 5 | 150 | Document default vs tuned settings, perform structured sensitivity study on key parameters, encode recommended settings in a config file |
| OpenVSP geometry generation | OpenVSP creates geometries with self-intersections or gaps | CFD meshing fails or produces highly skewed elements, results unreliable or simulations crash | 7 | Extreme parameter combinations, inadequate surface tolerances, poor trimming | 5 | Occasional manual geometry cleanup | 4 | 140 | Add geometry quality script (check intersections, sharp angles), define safe parameter limits, reject designs that violate quality metrics |
| CFD Case setup | Mis-specified boundary conditions (freestream, symmetry, far-field, wall or reference area) | Mis-predicted loads and performance metrics; direct impact on mission fuel burn/range estimation | 9 | Copy-paste of case files; inconsistent reference area vs optimizer, wrong Mach/Re input | 3 | Template case files; spot check against handbook coefficients | 6 | 162 | Create a setup checklist, script all boundary condition and reference values from a single source of truth |
| Post-processing | Incorrect integration or reference normalization | CL, CD, CM reported incorrectly; optimizer optimizes incorrect objective | 9 | Wrong reference area/span, mis-aligned force axes; scripting error | 4 | One-time check against CFD GUI output | 5 | 180 | Build automated cross-check, compute coefficients via two independent methods, compare to a "golden" case, assert acceptable tolerance |

| | |
|---|---|
| Severity | Rates the impact of a failure on a scale from 1 to 10. Higher scores indicate more serious effects on the system or process. |
| Occurrence | Measures how frequently a failure happens on a scale from 1 to 10. Higher scores reflect more common failures. |
| Detection | Assesses the likelihood that existing controls will catch the failure on a scale from 1 to 10. Higher scores indicate less likely detection. |
| RPN | Calculate the overall risk by multiplying severity, occurrence, and detection scores. Higher RPNs signal more urgent issues that require attention. |

| Failure Mode | Effect | Cause | Current Controls | S | O | D | RPN | Recommended Action |
|---|---|---|---|---|---|---|---|---|
| Wing geometry becomes aerodynamically inefficient | Reduced lift-to-drag ratio and angle | Over-optimization of sweep, taper, or CST shape | Low-fidelity screening, CFD validation | 8 | 4 | 4 | 128 | Tighten design bounds and compare against baseline more frequently |
| Wing geometry is not structurally feasible | Excessive deflection or failure under load | Thin airfoil sections or inadequate spar depth | Structural weight constraint, geometry limits | 9 | 3 | 5 | 135 | Add structural constraint checks earlier in the optimization loop |
| Optimized wing is difficult to manufacture | Increased cost and production complexity | Excessive curvature, thin sections, or complex transitions | CAD review, parameter bounds | 7 | 4 | 5 | 140 | Simplify geometry and add manufacturability constraints |
| CFD results do not match lower-fidelity trends | Poor confidence in selected design | Mesh sensitivity, solver settings, or modeling assumptions | High-fidelity validation step | 8 | 3 | 6 | 144 | Perform mesh independence checks and calibrate against known cases |
| Optimization converges to local optimum | Suboptimal final wing design | Limited search space or poor initial guess | Multi-variable optimization setup | 7 | 5 | 4 | 140 | Use multiple starting points and compare candidate solutions |
| Parameter mapping errors in automation | Incorrect geometry or invalid outputs | Script bugs or variable misalignment | Manual checks and workflow review | 8 | 3 | 4 | 96 | Add automated validation and input-output checks |

## B-8. Decision Matrix

| Criteria | Hierarchical Open-Source Pipeline | Comparative Optimization Model Analysis | Interactive Design Space Explorer |
|---|---|---|---|
| Fidelity Range | 5 | 4 | 3 |
| Computational Efficiency | 4 | 3 | 2 |
| Discoverability of Optimum | 4 | 5 | 2 |
| Usability | 3 | 3 | 5 |
| Manual vs Automated | 4 | 4 | 2 |
| Total | 20 | 19 | 14 |

## B-9. Group Approval

| Name | Signature | Date |
|---|---|---|
| Griffin Cantrell | *Griffin Cantrell* | 4-May-26 |
| Edwin Harper | *Edwin Harper* | 4-May-26 |
| Garrett Kuhn | *Garrett Kuhn* | 4-May-26 |
| Cooper Mendlinger | *Cooper Mendlinger* | 4-May-26 |
| Randal Richards | *Randal Richards* | 4-May-26 |
| Martin Vassilev | *Martin Vassilev* | 4-May-26 |

## B-10. Engineering Specifications Table

| Category | Stakeholder Needs | Design Requirements | Acceptance Criteria | Qualification | Regulatory Requirements |
|---|---|---|---|---|---|
| Safety | Data Integrity | Ensure workflow with secure computational environment | Workflow runs safely without risk of corrupted simulations or data loss | Requirement | STD-8719.13C (Software Safety) |
| Quality Control | Accurate aerodynamic results | Workflow produces traceable and precise results | CFD results and low-fidelity models validated against known standards | Requirement | AIAA R-101-2013 (CFD Verification & Validation) |
| Cost | Affordable and efficient use of resources | Efficient resource usage | Optimized computational workflow | Requirement | N/A |
| Functionality | Maintainability | Multi-fidelity integration | Successfully links CAD to low and high-fidelity tools without data loss | Requirement | ISO 10303 (STEP format interoperability standard) |
| Repeatability | Must be reusable for given wing configurations | Modular Workflow with input parameters | Re-run workflow on various wing configurations | Requirement | N/A |

## B-11. Bill of Materials

# Bill of Materials (BOM) — Aircraft Wing Design Optimization Workflow

**Prepared: 2026-01-28** | **Milestone 11**

Project: Multi-fidelity aerodynamic wing optimization workflow (OpenVSP / AVL / MATLAB / Ansys Fluent)

**Assumptions**

**Currenc** USD

**Prices** Estimates; adjust to match actual quotes

**License** Many tools may be provided by UGA / sponsor; set Unit Cost to $0 if covered

**HPC** If compute time is allocated by UGA, set cost to $0 or internal rate

| Ite | Category | Item / Part | Description / Purpose | C | Un | Unit Cost (US | Ext. Cost (US | Supplier / Source | Status | Lead Tim (days) | Notes / Assumptions |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Software | MATLAB + Optimization Toolbox | Optimization scripting, GA/gradient methods | 1 | license | $0.00 | $0.00 | MathWorks / UGA | Planned | 0 | Assume covered by university; set price if not. |
| 2 | Software | Ansys Fluent (Academic) | High-fidelity CFD validation runs | 1 | license | $0.00 | $0.00 | Ansys / UGA | Planned | 0 | Often available via academic license. |
| 3 | Software | OpenVSP | Parametric geometry generation for low-fidelity stage | 1 | package | $0.00 | $0.00 | NASA / OpenVSP | On-hand | 0 | Free/open-source. |
| 4 | Software | AVL | Vortex-lattice aerodynamic evaluation | 1 | package | $0.00 | $0.00 | MIT / AVL | On-hand | 0 | Free (research/education). |
| 5 | Software | Python | Automation/glue code between tools | 1 | package | $0.00 | $0.00 | Python.org | On-hand | 0 | Free/open-source. |
| 6 | Software | CAD (AutoCAD/SolidWorks) | CAD edits, STEP export | 1 | license | $0.00 | $0.00 | Autodesk/Dassault / UGA | Planned | 0 | Use whichever is available. |
| 7 | Computing | HPC compute allocation | CFD batch runs; queue time and cores | 2000 | core-hr | $0.00 | $0.00 | UGA HPC | Planned | 0 | Track usage; add internal rate if needed. |
| 8 | Computing | Local workstation | Pre/post-processing, CAD, scripting | 1 | unit | $0.00 | $0.00 | Team / Lab | On-hand | 0 | If purchasing, replace $0 with actual. |
| 9 | Computing | External SSD (2 TB) | Store CFD cases/results and backups | 1 | ea | $0.00 | $0.00 | UGA lab | Needed | 3 | Optional but recommended. |
| 10 | Computing | Cloud backup (1 year) | Offsite backup for key files | 1 | yr | $0.00 | $0.00 | Cloud provider | Optional | 0 | Can be $0 if using university storage. |
| 15 | Documentation | Poster printing | End-of-year showcase | 1 | ea | $0.00 | $0.00 | UGA print services | Planned | 2 | Capstone will pay for it |
| 16 | Documentation | Report printing/binding | Hardcopy for client/faculty | 3 | ea | $0.00 | $0.00 | UGA print services | Planned | 2 | Capstone will pay for it |
| 17 | Misc | Transportation | Meetings, campus travel | 1 | lot | $300.00 | $300.00 | Personal / UGA | Planned | 0 | Budget item. |
| 18 | Misc | Textbook/reference materials | Optional references | 1 | lot | $0.00 | $0.00 | AIAA/Google | Optional | 0 | Budget item. |
| | | | | | **TOTAL** | | **$300.00** | | | | |

# Appendix C

## Design Documentation

### C-1 Finalized MDO Python Code

```
import aerosandbox as asb
import aerosandbox.numpy as np
import numpy as _np                    # plain numpy for pre-solve Python calculations
                from math import comb as math_comb
                from scipy.special import beta as beta_fn# ============================================
# 1. Modular Aircraft Configuration
# ============================================#
# When running in Google Colab with the widget UI (optimizer_ui.py),
# the `config` dict is injected into the namespace by the UI cell and this
# block is skipped entirely — the widget valus take priority.
## When running standalone (no widget cell), the defaults below are used.
## Set aircraft_class to one of:#   "GA"          — general aviation piston/turboprop (current defaults)
#   "transport"   — commercial jetliner (737 through 777 class)
                #   "fighter"     — air superiority / multirole (F-16 / F-18 / F-35 class)
                #   "business_jet" — FAR Part 25 bizjet (Citation through Gulfstream class)
                #   "ucav"        — unmanned combat aerial vehicle (X-47B / nEUROn class)
                #
============================================if "config" not in dir():   # widget UI sets this; only build default if absent
                    config = {
                    # ── Aircraft class
    ──────────────────────────────────────────────────────────────────────────
                    "aircraft_class":      "GA",         # "GA" | "transport" | "fighter"
    # ── Mission parameters (always set explicitly per design)
    ──────────────────────
                    "W_fixed_N":           1800 * 4.448,  # fixed structure + payload weight (N)
                    "W_fuel_N":            240 * 4.448,  # fuel weight (N)
                    "propulsion_factor":    1.2e6,        # Breguet η·V/SFC (m)
                                            #   GA piston:       ~1.2e6
                                            #   Commercial turbofan: ~7.0e6
                                            #   Military dry:      ~2.5e6
                                            #   Military afterburner: ~0.8e6
                    "v_cruise_mps":         62,          # cruise true airspeed (m/s)
                    "altitude_m":           0,           # cruise altitude (m)
                    #
============================================================
                    # 18b. Wave Drag Correction (Korn + Lock)
                    #
                    # AeroBuildup calls NeuralFoil at the freestream Mach number for each 2D
                    # section; it does NOT apply the sweep-cosine Mach correction
                    #   M_perp = M∞ · cos(Λ_{c/4})
                    # that gives swept wings their transonic advantage.  Without this correction
                    # the optimizer sees no aerodynamic incentive for aft sweep, causing it to
                    # drift to the lower bound.
                    #
                    # This block supplies the missing physics explicitly:
                    #
                    #   Step 1 — Effective section Mach:
                    #     The wing section "sees" the velocity component perpendicular to the
                    #     quarter-chord line.  NeuralFoil should (but doesn't) evaluate at this
                    #     reduced Mach:   M_perp = M∞ · cos(Λ_{c/4})
                    #
                    #   Step 2 — Drag-divergence Mach via the Korn equation:
                    #     Mdd = κ / cos(Λ) − (t/c) / cos²(Λ) − CL / (10 · cos³(Λ))
                    #     κ = 0.87  (technology factor for conventional supercritical sections;
                    #                use 0.95 for advanced supercritical airfoils)
                    #     This is evaluated at the mean aerodynamic chord t/c, taken here as the
                    #     root t/c (conservative — root is thicker and typically critical).
                    #
                    #   Step 3 — Wave drag via the Lock quartic:
                    #     ΔCD_wave = 20 · max(0, M∞ − Mdd)⁴
                    #     The hard max() is non-smooth; a soft-ramp
                    #       softmax(x, ε) = 0.5 · (x + √(x² + ε²))
                    ), row=1, col=2)
                    # Spar location reference
                    fig_ct.add_vline(x=1.0/3.0, row=1, col=2,

    line=dict(color="rgba(180,0,0,0.5)", width=1.2, dash="dot"),

    annotation_text="x/c=1/3 (spar)", annotation_font_size=9,

    annotation_position="top right")

    fig_ct.update_xaxes(title_text="x/c", showgrid=True, gridcolor="#e8e8e8")

    fig_ct.update_yaxes(title_text="y/c", row=1, col=1, showgrid=True, gridcolor="#e8e8e8",
                                                zeroline=True, zerolinecolor="#aaa")

    fig_ct.update_yaxes(title_text="t/c", row=1, col=2, showgrid=True, gridcolor="#e8e8e8")
                    fig_ct.update_layout(
                        title=dict(text="Optimized Airfoil Camber & Thickness", font=dict(size=13)),
                        height=380, template="plotly_white",
                        font=dict(family="Arial", size=11),
                        legend=dict(x=0.02, y=0.98, bgcolor="rgba(255,255,255,0.85)",

    bordercolor="#cccccc", borderwidth=1),
                        )
                    fig_ct.show()

    fig_ct.write_html("airfoil_camber_thickness.html")
                    print("Saved: airfoil_camber_thickness.html")
                except Exception as _ct_err:
                    print(f" ⚠ Camber/thickness chart skipped: {_ct_err}")


                # ── NeuralFoil Aerodynamic Polars (CL vs CD & CL vs alpha)
    ──────────────────
```

```
                "root_chord_m":          1.63,       # root chord (m) — set per aircraft
                "twist_tip_deg":         -3.0,       # tip washout (deg)
                "usable_wing_fraction":  0.50,       # fraction of wing volume usable for fuel
                # ── Solver ──────────────────────────────────────────────────────────
                "mach_limit":            0.88,       # NeuralFoil accuracy ceiling
                "n_starts":              35,         # multi-start LHS count
            # ── Class-specific overrides (leave absent to use preset defaults) ──
            # Uncomment and edit to override any preset value, e.g.:
            #   "N_z":              9.0,          # custom ultimate load factor
            #   "CLmax_approach":   3.2,          # custom high-lift CLmax
            #   "V_approach_max_mps": 72.0,       # custom approach speed
            #   "fuel_density_kg_m3": 800,        # Jet-A instead of avgas
            #   "root_tc_min":      0.10,         # custom spar thickness limit
            #   "tip_tc_min":       0.07,
        }  # end of default config
    }
# ============================================================
# 1b. Aircraft-Class Presets
# ============================================================
# Each preset fills all class-specific parameters that differ across GA,
# transport, and fighter classes.  After the merge, every subsequent section
# reads from config["key"] without needing to know whether the value came from
# a preset or an explicit user override.
#
# Raymer wing weight equations (Table 15.2 / 15.25 / 15.46, 5th ed.):
#   "GA"       — Eq 15.2:  W_w = 0.036·Sw^0.758·Wfw^0.0035·(AR/cos²Λ)^0.6
#                          ·q^0.006·λ^0.04·(100t/c/cosΛ)^-0.3·(Nz·W0)^0.49
#   "transport" — Eq 15.46: W_w = 0.0051·(Wdg·Nz)^0.557·Sw^0.649·AR^0.5
#                          ·(t/c)^-0.4·(1+λ)^0.1·cosΛ^-1·Scsw^0.1
#   "fighter"   — Eq 15.25: W_w = 0.0103·(W0·Nz)^0.5·Sw^0.622·AR^0.785
#                          ·(t/c)^-0.4·(1+λ)^0.05·cosΛ^-1·Scsw^0.04
#
# ============================================================
PRESETS = {
    "GA": {
        # General aviation: FAR Part 23, piston/turboprop, ~1,000–6,000 lb MTOW
        "N_z":              5.25,   # 3.5g limit × 1.5 safety factor
        "CLmax_approach":   1.5,    # clean wing, no significant high-lift
        "V_approach_max_mps": 36.0,   # 70 kt — FAR 23 light aircraft
        "fuel_density_kg_m3": 720,    # avgas 100LL
        "alpha_bounds":     (-5.0,  15.0),
        "span_bounds":      (5.0,   30.0),   # m
        "taper_bounds":     (0.3,   1.0),
        "sweep_bounds":     (0.0,   35.0),   # deg
        "W_wing_bounds_N":  (100.0, 20000.0),
        "root_tc_min":      0.15,
        "tip_tc_min":       0.09,
        "slenderness_max":  20.0,
        "raymer_eq":        "GA",
        # CD_misc: Raymer Table 12.6 — fixed gear (0.015), cowling (0.003), junction+excrescence (0.004)
        "CD_misc":          0.022,
        # Flap sizing (slotted plain flap ~40°)
        "CLmax_clean":      1.35,
        "K_flap":           2.4,
        "flap_chord_bounds": (0.15, 0.35),
        "flap_span_bounds":  (0.30, 0.70),
        # Fuselage proportionality (derived from Cessna 172S)
        "fus_length_frac":  6.1,    # L_fus / MAC
        #   is used instead so the gradient is well-defined at x = 0 for IPOPT.
        #   ε = 0.002 (0.002 Mach units ≈ one-quarter of typical solver tolerance).
        #
        # Step 4 — Convert ΔCD_wave to force and subtract from net drag:
        #   ΔD_wave = ΔCD_wave · q · S_w
        #   Then  D_total = D_aero + ΔD_wave
        #         L/D     = L / D_total
        #
        # Physical check: at Ma 0.8 with Λ_{c/4} = 25°, root t/c = 0.18, CL ≈ 0.35:
        #   M_perp = 0.8 · cos(25°) = 0.725
        #   Mdd    = 0.87/0.906 − 0.18/0.821 − 0.35/0.743 = 0.960 − 0.219 − 0.471 = 0.270
        #   Wait — that's clearly wrong.  Let me redo with the SWEPT formula
        #   (arguments are evaluated at M_perp, not M∞):
        #   Mdd = κ/cos(Λ) − (t/c)/cos²(Λ) − CL_perp/(10·cos³(Λ))
        #   With Λ=25°, cos=0.906:
        #   Mdd = 0.87/0.906 − 0.18/0.821 − 0.35/7.43 = 0.960 − 0.219 − 0.047 = 0.694
        #   M_perp = 0.725 > Mdd = 0.694 → small wave drag penalty at this condition.
        #   With Λ=0 (unswept): Mdd = 0.87 − 0.18 − 0.035 = 0.655
        #   M_perp = M∞ = 0.800 >> Mdd → large wave drag penalty.
        #   → Optimizer will strongly prefer ~25° aft sweep at Ma 0.8.  ✓
        #
        # ============================================================
        "kappa":            0.87,       # Korn technology factor (conventional airfoils)
        "eps_soft":         0.002,      # softmax smoothing width (Mach units)
        # CL at cruise (from the lift = weight constraint already enforced above)
        "CL_cruise":        W_initial_N / (0.5 * rho_cruise * config["v_cruise_mps"]**2 * S_w),
        # Korn drag-divergence Mach (evaluated with sweep and root section thickness)
        "cos_qc":           np.cos(sweep_qc_rad),
        "Mdd":              (kappa / cos_qc
                             - root_t_tc / cos_qc**2
                             - CL_cruise / (10.0 * cos_qc**3)),
        # Soft-ramp:  softmax(x, ε) ≈ max(0, x)  but C¹ everywhere
        "excess":           mach_cruise - Mdd,   # scalar (M∞ - Mdd); CasADi expression
        "soft_exc":         0.5 * (excess + np.sqrt(excess**2 + eps_soft**2)),
        # Wave drag coefficient and force increment
        "CD_wave":          20.0 * soft_exc**4,
        "q_cruise":         0.5 * rho_cruise * config["v_cruise_mps"]**2,
        "D_wave":           CD_wave * q_cruise * S_w  # N
        try:
            print("--- Aerodynamic Polars (NeuralFoil sweep) ---")
            alpha_sw_deg = np.linspace(-4.0, 18.0, 45)
            Re_cr = float(rho_cruise * config["v_cruise_mps"] * config["root_chord_m"]
                          / atm_cruise.kinematic_viscosity())

            def foil_polar(foil):
                res = foil.get_aero_from_neuralfoil(
                    alpha=alpha_sw_deg, Re=Re_cr, mach=mach_cruise)
                return np.array(res["CL"]), np.array(res["CD"])

            CL_root_pol, CD_root_pol = foil_polar(final_root)
            CL_tip_pol,  CD_tip_pol  = foil_polar(final_tip)

            # Cruise 2-D point for root section
            res_cr = final_root.get_aero_from_neuralfoil(
                alpha=alpha, Re=Re_cr, mach=mach_cruise)
            CL_cr_2d = float(np.asarray(res_cr["CL"]).flat[0])
            CD_cr_2d = float(np.asarray(res_cr["CD"]).flat[0])

            fig_polar = make_subplots(rows=1, cols=2,
                subplot_titles=["Drag Polar  CL vs CD", "Lift Curve  CL vs α"],
                horizontal_spacing=0.12)

            for (cl_arr, cd_arr, lbl, col, dsh) in [
                (CL_root_pol, CD_root_pol, "Root airfoil", "#2e86c1", "solid"),
                (CL_tip_pol,  CD_tip_pol,  "Tip airfoil",  "#e67e22", "dot"),
            ]:
                fig_polar.add_trace(go.Scatter(
                    x=cd_arr, y=cl_arr, mode="lines",
                    line=dict(color=col, width=2.2, dash=dsh), name=lbl,
                    hovertemplate="CD=%{x:.5f}<br>CL=%{y:.4f}<extra>" + lbl + "</extra>",
                ), row=1, col=1)
                fig_polar.add_trace(go.Scatter(
                    x=alpha_sw_deg, y=cl_arr, mode="lines",
                    line=dict(color=col, width=2.2, dash=dsh), showlegend=False,
                    hovertemplate="α=%{x:.1f}°<br>CL=%{y:.4f}<extra>" + lbl + "</extra>",
                ), row=1, col=2)

            # Cruise point
            for (row_n, xv, yv, xt) in [
```

```
                "fus_radius_frac":    0.375,  # r_max /
root_chord_m
                "fus_nose_frac":      0.12,   # blunt cabin nose;
early shoulder
                "fus_tail_r_frac":    0.15,   # moderate tail taper
                # Longitudinal balance — static margin model
                "wing_pos_frac":      0.12,   # wing LE at 12% of
fus length from nose (engine fwd)
                "cg_fixed_frac":      0.179,  # payload+struct CG
at 17.9% fus length from nose
                "np_tail_frac":       0.12,   # horizontal tail NP shift
(frac of MAC)
                "sm_limit":           0.10,   # ±10% SM bound
(accounts for model uncertainty)
            },
            "transport": {
                # Commercial jet transport: FAR Part 25,
turbofan, 100,000–900,000 lb MTOW
                # Covers narrow-body (737/A320) through wide-
body (777/A350) classes.
                "N_z":                3.75,   # 2.5g limit × 1.5
                "CLmax_approach":     2.8,    # full flaps +
leading-edge slats
                "V_approach_max_mps": 77.2,   # 150 kt Vref —
FAR 25 transport
                "fuel_density_kg_m3": 800,    # Jet-A
                "alpha_bounds":       (-5.0,  15.0),
                "span_bounds":        (20.0,  70.0),   # m  (737:
35m, 777: 65m)
                "taper_bounds":       (0.2,   0.5),    # transports
have moderate taper
                "sweep_bounds":       (0.0,   40.0),   # deg
                "W_wing_bounds_N":    (5000.0, 600000.0),
                "root_tc_min":        0.12,
                "tip_tc_min":         0.09,
                "slenderness_max":    20.0,
                "raymer_eq":          "transport",
                # CD_misc: Raymer Table 12.6 — 2x
nacelle+pylon (0.008), junction (0.002), excrescence (0.002)
                "CD_misc":            0.012,
                # Flap sizing (double slotted Fowler + LE slats)
                "CLmax_clean":        1.50,
                "K_flap":             3.0,
                "flap_chord_bounds":  (0.20, 0.40),
                "flap_span_bounds":   (0.40, 0.80),
                # Fuselage proportionality (derived from 737-
800)
                "fus_length_frac":    10.2,   # long narrow-body
tube
                "fus_radius_frac":    0.342,  # r_max /
root_chord_m
                "fus_nose_frac":      0.07,   # streamlined radome
                "fus_tail_r_frac":    0.10,   # sharp APU tail cone
                # Longitudinal balance — static margin model
                "wing_pos_frac":      0.40,   # wing LE at 40% of
long fuselage from nose
                "cg_fixed_frac":      0.52,   # payload+struct CG
at 52% fus length from nose
                "np_tail_frac":       0.20,   # large horizontal
stabilizer NP shift
                "sm_limit":           0.10,   # ±10% SM bound
            },
            "fighter": {
                # Military fighter/attack: MIL-SPEC,
turbofan/turbojet, 20,000–70,000 lb MTOW
                # Covers F-16 (26,000 lb) through F-15E (81,000
lb) class.
                "N_z":                13.5,   # 9g limit × 1.5 (F-16
structural design)
                "CLmax_approach":     1.8,    # limited high-lift
(no large flaps)
                "V_approach_max_mps": 82.3,   # 160 kt —
typical military approach
                "fuel_density_kg_m3": 800,    # JP-8 / Jet-A
                "alpha_bounds":       (-5.0,  20.0),   # fighters
operate at higher AoA
                "span_bounds":        (4.0,   16.0),   # m  (F-16:
10m, F-15: 13m)
```

```
                # Miscellaneous parasite drag
(Raymer Table 12.6)
                # Accounts for all components not
modelled by AeroBuildup or wave drag:
                #   landing gear, engine
nacelles/pylons/intakes, wing-body interference,
                #   and excrescence (doors,
antennas, vents, seals).
                # CD_misc is referenced to S_w
and is class-specific (see presets).
                # This is a constant drag offset — it
does not vary with alpha or CL,
                # which is the standard Raymer
treatment for "miscellaneous" drag.
                _D_misc = config["CD_misc"] *
_q_cruise * S_w  # N


                # Total drag and corrected L/D
                _D_total   = aero_results["D"] +
_D_wave + _D_misc
                L_over_D   = aero_results["L"] /
_D_total
                range_m    =
config["propulsion_factor"] * L_over_D *
np.log(W_initial_N / W_final_N)


                opti.minimize(-range_m)


                #
=====================================
=====
                # 19. Multi-Start IPOPT
                #
                # Strategy: Latin Hypercube
Sampling (LHS) across all 37 variable bounds
                # generates n_starts diverse
starting points that cover the design space
                # uniformly, avoiding the clustering
that pure random sampling produces.
                # LHS guarantees each variable's
range is sampled evenly by dividing it into
                # n_starts equal-width strata and
drawing exactly one point per stratum.
                #
                # Each start:
                #   1. Sets initial values via
opti.set_initial() — CasADi retains the full
                #      NLP structure (variables,
constraints, objective) between solves; only
                #      the starting point changes.  No
rebuilding of the symbolic graph.
                #   2. Registers a fresh per-start
callback closure so convergence history is
                #      captured independently per
start (opti.callback() replaces the hook).
                #   3. Calls opti.solve().  Failures
are caught and logged; the NLP is not
                #      modified so subsequent starts
are unaffected.
                #
                # Start 0 always uses the physics-
consistent warm start (_alpha_init,
                # _W_wing_init) so the original
single-start result is always reproducible.
                # Starts 1..n_starts-1 use LHS;
seed=42 makes the draw deterministic.
                #
                # After the loop the feasible solution
with the highest range is selected
                # as `sol`; its per-start history feeds
the convergence plots in section 22.
                #
                # Timing budget:
                #   n_starts = 35, ~7 s/start nominal
→ ~245 s ≈ 4 min
```

```
                    (1, _CD_cr_2d,
_CL_cr_2d, "CD"), (2, _alpha,    _CL_cr_2d,
"α")]:

fig_polar.add_trace(go.Scatter(
                        x=[xv], y=[yv],
mode="markers",

marker=dict(symbol="star", size=16,
color="red",

line=dict(color="darkred", width=1.5)),
                        name=f"Cruise
(α={_alpha:.1f}°)" if row_n == 1 else "",
                        showlegend=(row_n
== 1),

hovertemplate=f"{xt}=%{{x:.4f}}<br>CL={_CL_
cr_2d:.4f}<extra>Cruise</extra>",
                    ), row=1, col=row_n)



fig_polar.update_xaxes(title_text="CD",
row=1, col=1, showgrid=True,
gridcolor="#e8e8e8")

fig_polar.update_yaxes(title_text="CL",
row=1, col=1, showgrid=True,
gridcolor="#e8e8e8")

fig_polar.update_xaxes(title_text="α
(deg)",row=1, col=2, showgrid=True,
gridcolor="#e8e8e8")

fig_polar.update_yaxes(title_text="CL",
row=1, col=2, showgrid=True,
gridcolor="#e8e8e8")
                fig_polar.update_layout(
                    title=dict(
                        text=(f"NeuralFoil
Section Polars  ·  Ma={mach_cruise:.3f}  "

f"Re={_Re_cr:.2e}<br>"
                            f"<sup>2D section
aerodynamics — root (blue) vs tip
(orange)</sup>"),
                        font=dict(size=13)),
                    height=420,
template="plotly_white",
                    font=dict(family="Arial",
size=11),
                    legend=dict(x=0.02,
y=0.98, bgcolor="rgba(255,255,255,0.85)",

bordercolor="#cccccc", borderwidth=1),
                    )
                    fig_polar.show()

fig_polar.write_html("aero_polars.html")
                    print("Saved:
aero_polars.html")
                except Exception as _pol_err:
                    print(f"  ⚠  Polar sweep
skipped: {_pol_err}")




                # ── Top-view planform with
flaps
──────────────────────────────────────
──────────
                # Shows: wing outline,
quarter-chord line, flap region, aileron region,
                # MAC indicator, fuselage
silhouette, and key dimension annotations.
                _sw_r = _np.radians(_sweep)
```

```
        "taper_bounds":      (0.2,   0.6),    # delta / low-AR wings
        "sweep_bounds":      (0.0,   65.0),   # deg  (delta up to 60°+)
        "W_wing_bounds_N":   (500.0, 100000.0),
        "root_tc_min":       0.04,   # thin wings essential for transonic/supersonic
        "tip_tc_min":        0.03,
        "slenderness_max":   30.0,   # fighter spars are structurally slender
        "raymer_eq":         "fighter",
        # CD_misc: Raymer Table 12.6 — intake+nozzle (0.006), canopy (0.002), excrescence (0.002)
        "CD_misc":           0.010,
        # Flap sizing (simple trailing-edge flap)
        "CLmax_clean":       1.20,
        "K_flap":            2.7,
        "flap_chord_bounds": (0.10, 0.25),
        "flap_span_bounds":  (0.10, 0.40),
        # Fuselage proportionality (derived from F-16C Block 50)
        "fus_length_frac":   4.8,    # compact slender body
        "fus_radius_frac":   0.111,  # slender; engine-dominated cross-section
        "fus_nose_frac":     0.20,   # pointed radar nose cone
        "fus_tail_r_frac":   0.22,   # large engine nozzle exit
        # Longitudinal balance — static margin model
        "wing_pos_frac":     0.45,   # wing LE at 45% of fus length (mid-body)
        "cg_fixed_frac":     0.60,   # payload+struct CG at 60% fus length from nose
        "np_tail_frac":      0.05,   # small tail / FBW — modest NP shift
        "sm_limit":          0.10,   # ±10% SM bound (fighters near neutral)
    },
    "business_jet": {
        # Business jet: FAR Part 25, turbofan, 10,000–100,000 lb MTOW.
        # Covers light jets (Citation CJ4) through ultra-long-range
        # (Gulfstream G700, Bombardier Global 7500).
        #
        # Uses the transport Raymer equation (Eq 15.46): FAR Part 25 certified,
        # same bending-dominated structural regime as commercial transports, but
        # span and weight are dramatically smaller.
        #
        # Key differences from "transport":
        #   - Span 15–35 m (G650: 31 m; Citation CJ4: 15 m)
        #   - Higher cruise altitude: FL450–510 → lower Re, matters for NeuralFoil
        #   - Approach speed ~130 kt (67 m/s) — between GA (70 kt) and transport (150 kt)
        #   - Moderate high-lift: slats + single-slotted flaps → CLmax ≈ 2.2
        #   - Nz same as transport (FAR Part 25: 2.5g limit → 3.75 ultimate)
        "N_z":               3.75,   # 2.5g limit × 1.5  (FAR Part 25)
        "CLmax_approach":    2.2,    # slats + single-slotted flaps
        "V_approach_max_mps": 67.0,  # 130 kt Vref — typical bizjet
        "fuel_density_kg_m3": 800,   # Jet-A
        "alpha_bounds":      (-5.0,  15.0),
        "span_bounds":       (10.0,  38.0),   # m
        "taper_bounds":      (0.2,   0.5),    # similar to transports
        "sweep_bounds":      (10.0,  40.0),   # bizjets always have meaningful sweep
        "W_wing_bounds_N":   (2000.0, 120000.0),
        # Upper estimate: ~10 s/start (slow starts near constraint boundaries)
        #                 → 350 s ≈ 6 min  (just over the 5 min target)
        # Reduce n_starts to 25 if you consistently hit 6+ min.
        #
    }
}
# ============================================================


import math  as _math
import time  as _time
from scipy.stats import qmc as _qmc


n_starts = config["n_starts"]
# ── LHS variable layout (37-dimensional) ──────────────────────────────────
#  [0]    alpha          deg     from config["alpha_bounds"]
#  [1]    span           m       from config["span_bounds"]
#  [2]    taper                  from config["taper_bounds"]
#  [3]    sweep_deg      deg     from config["sweep_bounds"]
#  [4]    W_wing_N       N       from config["W_wing_bounds_N"]
#  [5:13]  root_w_up     [0.001, 0.5]  × 8
#  [13:21] root_w_low    [-0.5,-0.001]  × 8
#  [21:29] tip_w_up      [0.001, 0.5]  × 8
#  [29:37] tip_w_low     [-0.5,-0.001]  × 8
#
# Bounds are derived from the same config values as the opti variables so
# LHS samples always fall inside the feasible variable box — no hardcoding.
_n_vars = 39   # 37 original + 2 flap variables
_lb_lhs = _np.array([
    config["alpha_bounds"][0],
    config["span_bounds"][0],
    config["taper_bounds"][0],
    config["sweep_bounds"][0],
    config["W_wing_bounds_N"][0],
    *[0.001]*8, *[-0.500]*8,
    *[0.001]*8, *[-0.500]*8,
    config["flap_chord_bounds"][0],
    config["flap_span_bounds"][0],
])
_ub_lhs = _np.array([
    config["alpha_bounds"][1],
    config["span_bounds"][1],
    config["taper_bounds"][1],
    config["sweep_bounds"][1],
    config["W_wing_bounds_N"][1],
    *[0.500]*8, *[-0.001]*8,
    *[0.500]*8, *[-0.001]*8,
    config["flap_chord_bounds"][1],
    config["flap_span_bounds"][1],
])


_lhs_unit = _qmc.LatinHypercube(d=_n_vars, seed=42).random(n=n_starts)
_lhs_scaled = _qmc.scale(_lhs_unit, _lb_lhs, _ub_lhs)
        _b2   = _span / 2.0
        _cr   = config["root_chord_m"]
        _fus_r_tv = _cr * config["fus_radius_frac"]
        _fus_mac_tv = _cr*(2/3)*(1+_taper+_taper**2)/(1+_taper)
        _fus_len_tv = _fus_mac_tv * config["fus_length_frac"]


        def _planform_half(sign, n=80):
            e   = _np.linspace(0, 1, n)
            y   = sign * e * _b2
            xle = e * _b2 * _np.tan(_sw_r)
            xte = xle + _cr*(1-(1-_taper)*e)
            return (_np.concatenate([xle, xte[::-1], [xle[0]]]),
                    _np.concatenate([y, y[::-1],  [y[0]]]))


        fig_plan = go.Figure()


        # Wing fill (both halves)
        for s in (1, -1):
            xs, ys = _planform_half(s)
            fig_plan.add_trace(go.Scatter(
                x=ys, y=xs, mode="lines",
                fill="toself",
                fillcolor="rgba(46,134,193,0.13)",
                line=dict(color="#1a5276", width=2.0),
                showlegend=False,
                hoverinfo="skip"))


        # Fuselage top-view silhouette
        _th = _np.linspace(0, 2*_np.pi, 80)
        _wp = config["wing_pos_frac"] * _fus_len_tv   # wing LE x in fus coords
        # Map fuselage to wing coord frame: fus nose at x = -_wp
        _fus_cx = _fus_len_tv/2 - _wp  # fus centre x in wing frame
        fig_plan.add_trace(go.Scatter(
            x=_fus_r_tv*_np.sin(_th),
            y=_fus_cx + (_fus_len_tv/2)*_np.cos(_th),
            mode="lines",
            line=dict(color="rgba(120,120,120,0.7)", width=1.2, dash="dot"),
            fill="toself",
            fillcolor="rgba(200,200,200,0.20)",
            showlegend=False,
            hoverinfo="skip"))


        # Quarter-chord line (full span)
        _e_qc = _np.linspace(0, 1, 40)
        _y_qc = _np.concatenate([-_e_qc[::-1], _e_qc]) * _b2
        _x_qc = (_np.concatenate([_e_qc[::-1], _e_qc]) * _b2 * _np.tan(_sw_r)
                 + 0.25*_cr*(1-(1-_taper)*_np.concatenate([_e_qc[::-1], _e_qc])))
        fig_plan.add_trace(go.Scatter(
            x=_y_qc, y=_x_qc,
            mode="lines",
```

```
                "root_tc_min":       0.10,   # thinner than transport; supercritical sections
                "tip_tc_min":        0.08,
                "slenderness_max":   22.0,   # slightly more slender than transport
                "raymer_eq":         "transport",   # FAR 25 structural regime
                # CD_misc: Raymer Table 12.6 — rear nacelle+pylon (0.005), junction (0.001), excrescence (0.001)
                "CD_misc":           0.007,
                # Flap sizing (single/double slotted Fowler)
                "CLmax_clean":       1.40,
                "K_flap":            3.0,
                "flap_chord_bounds": (0.20, 0.38),
                "flap_span_bounds":  (0.40, 0.75),
                # Fuselage proportionality (derived from Gulfstream G650)
                "fus_length_frac":   8.0,    # long-range cabin
                "fus_radius_frac":   0.253,  # narrower than transport per chord
                "fus_nose_frac":     0.10,   # tapered radome
                "fus_tail_r_frac":   0.12,   # tight tail cone
                # Longitudinal balance — static margin model
                "wing_pos_frac":     0.38,   # wing LE at 38% of fus length from nose
                "cg_fixed_frac":     0.58,   # payload+struct CG at 58% fus length from nose
                "np_tail_frac":      0.18,   # moderate horizontal tail NP shift
                "sm_limit":          0.10,   # ±10% SM bound
            },
            "ucav": {
                # Unmanned Combat Aerial Vehicle: MIL-SPEC, turbofan,
                # 10,000–50,000 lb MTOW.
                # Covers X-47B (44,000 lb), nEUROn (14,300 lb), Dassault Neuron,
                # RQ-170 Sentinel, and next-generation loyal-wingman class UCAVs.
                #
                # Uses the fighter Raymer equation (Eq 15.25): MIL-SPEC structural
                # regime, but three key differences from a manned fighter drive
                # substantially different parameter choices:
                #
                #   1. No pilot → no ejection seat, no cockpit pressurisation, no
                #      life-support mass.  This allows a higher structural mass fraction
                #      to go to the airframe, but in practice designers use the savings
                #      to reduce MTOW rather than increase load factor.
                #
                #   2. Sustained-loiter / ISR persistence is often co-optimised with
                #      the strike mission, favouring moderate-to-high AR and lower wing
                #      loading than a pure fighter — but the platform must still survive
                #      in a contested environment, so sweep and t/c cannot be relaxed as
                #      far as for a transport.
                #
                #   3. No approach speed constraint from a human pilot's tolerance.
                #      UCAVs can land at any speed the airframe and undercarriage support.
                #      The constraint here (180 kt / 92.6 m/s) reflects carrier-capable
                #      designs (X-47B trap speed) as a conservative bound; pure land-based
                #      UCAVs can be loosened further.
                #
                # Key differences from "fighter":
                #   - Nz = 9.0 (6g limit × 1.5): lower than manned fighter; no sustained
```

```
# Start 0: physics-consistent warm start using class-aware span/taper init.
_taper_init_0 = float(_np.clip(0.45, config["taper_bounds"][0], config["taper_bounds"][1]))
_sweep_init_0 = float(_np.clip(5.0, config["sweep_bounds"][0], config["sweep_bounds"][1]))
_fcr_init_0   = float(0.5 * (config["flap_chord_bounds"][0] + config["flap_chord_bounds"][1]))
_fsf_init_0   = float(0.5 * (config["flap_span_bounds"][0]  + config["flap_span_bounds"][1]))
_lhs_scaled[0] = [_alpha_init, _span_init, _taper_init_0, _sweep_init_0, _W_wing_init,
                  *init_upper, *init_lower, *init_upper, *init_lower,
                  _fcr_init_0, _fsf_init_0]


def _set_start(row):
    """Load one LHS row into the opti initial-guess slots (39 variables)."""
    opti.set_initial(alpha, float(row[0]))
    opti.set_initial(span, float(row[1]))
    opti.set_initial(taper, float(row[2]))
    opti.set_initial(sweep_deg, float(row[3]))
    opti.set_initial(W_wing_N, float(row[4]))
    opti.set_initial(root_w_up, row[5:13])
    opti.set_initial(root_w_low, row[13:21])
    opti.set_initial(tip_w_up, row[21:29])
    opti.set_initial(tip_w_low, row[29:37])
    opti.set_initial(flap_chord_ratio, float(row[37]))
    opti.set_initial(flap_span_frac, float(row[38]))


# ── Solver options (set once; reused for every start) ──────────────────────
# ipopt.print_level = 0 suppresses the per-iteration console table that
# would otherwise produce thousands of lines across 35 starts.
opti.solver("ipopt", {
    "ipopt.mu_strategy": "adaptive",
    "ipopt.nlp_scaling_method": "gradient-based",
    "ipopt.tol":                1e-6,
    "ipopt.acceptable_tol":     1e-4,
    "ipopt.max_iter":           3000,
    "ipopt.print_level":        0,
})


# ── Multi-start loop ──────────────────────────────────────────────────────
_results = []   # (range_val, sweep_val, span_val, sol_obj, hist_dict)
_t0_wall = _time.time()


print(f"\nMulti-Start IPOPT — {n_starts} starts")
```

```
line=dict(color="rgba(200,0,0,0.55)", width=1.4, dash="dash"),
                name="c/4 line", showlegend=True,
                hovertemplate="y=%{x:.2f} m  x_c/4=%{y:.2f} m<extra>c/4</extra>"))


            # MAC indicator (centred at y_mac, width=MAC, at 25% chord)
            _y_mac_tv = ((_span/6)*(1+2*_taper)/(1+_taper))
            _x_mac_le_tv = _y_mac_tv * _np.tan(_sw_r)
            for s in (1,-1):

fig_plan.add_trace(go.Scatter(
                x=[s*_y_mac_tv, s*_y_mac_tv],
                y=[_x_mac_le_tv, _x_mac_le_tv + _fus_mac_tv],
                mode="lines",

line=dict(color="rgba(34,139,34,0.8)", width=3),
                showlegend=(s==1), name=f"MAC = {_fus_mac_tv:.2f} m",
                hovertemplate=f"MAC = {_fus_mac_tv:.3f} m<extra>MAC</extra>"))


            # Flap regions (inboard, both sides) — filled orange
            _flap_col_fill = "rgba(230,126,34,0.28)"
            _flap_col_line = "darkorange"
            for s in (1,-1):
                _ef   = _np.linspace(0, _flap_sf, 50)
                _y_f  = s * _ef * _b2
                _xle_f = _ef * _b2 * _np.tan(_sw_r)
                _cf   = _cr*(1-(1-_taper)*_ef)
                # Flap hinge line at (1 - flap_chord_ratio) * chord from LE
                _x_hinge = _xle_f + (1.0 - _flap_cr)*_cf
                _x_te_f  = _xle_f + _cf
                _xs_f = _np.concatenate([_x_hinge, _x_te_f[::-1], [_x_hinge[0]]])
                _ys_f = _np.concatenate([_y_f,     _y_f[::-1], [_y_f[0]]])

fig_plan.add_trace(go.Scatter(
                    x=_ys_f, y=_xs_f, mode="lines",
                    fill="toself", fillcolor=_flap_col_fill,

line=dict(color=_flap_col_line, width=1.5),
                    showlegend=(s==1),
                    name=(f"Flap c_f/c={_flap_cr:.2f}  "
                          f"b_f/b={_flap_sf:.2f}  ΔCLmax={_dCL_flap:.3f}"),
                    hovertemplate=(f"Flap region<br>"
                                   f"c_f/c = {_flap_cr:.3f}<br>"
                                   f"ΔCLmax = {_dCL_flap:.3f}<extra>Flap</extra>")))


                # Hinge line (dashed)
```

```
                #    9g requirement since there is no pilot to lose consciousness.
                #    Some designs spec 6g (X-47B), others up to 7.5g for agile wingmen.
                #   - Thicker root: 5–6% vs fighter's 4%.  Flying-wing / cranked-arrow
                #    planforms used by most UCAVs need slightly more internal volume for
                #    fuel and weapons bays while keeping radar cross-section low.
                #   - Wider span: 10–20 m covers loyal-wingman (MQ-28: 7.5 m) through
                #    large strike UCAV (X-47B: 19 m).
                #   - Moderate sweep: 35–55° cranked-arrow or tailless delta planforms
                #    are the dominant UCAV geometry; lower bound raised to 20° to
                #    exclude low-speed designs that are not aerodynamically UCAV-like.
                #   - Taper down to 0.05: near-triangular cranked planforms (X-47B,
                #    nEUROn) have very low effective taper at the tip panel.
                "N_z":              9.0,    # 6g limit × 1.5 (no pilot g-tolerance)
                "CLmax_approach":   1.6,    # no large high-lift; RCS-constrained planform
                "V_approach_max_mps": 92.6,   # 180 kt — carrier-capable trap speed bound
                "fuel_density_kg_m3": 800,    # JP-8 / Jet-A
                "alpha_bounds":     (-5.0,  20.0),   # high-AoA capability for agile UCAVs
                "span_bounds":      (5.0,   22.0),   # m  (MQ-28: 7.5 m, X-47B: 19 m)
                "taper_bounds":     (0.05,  0.5),    # near-triangular through moderate taper
                "sweep_bounds":     (20.0,  65.0),   # deg (cranked-arrow / delta dominant)
                "W_wing_bounds_N":   (500.0, 80000.0),
                "root_tc_min":      0.05,   # slightly thicker than fighter for weapons bays
                "tip_tc_min":       0.03,   # thin tip panels retained for low RCS
                "slenderness_max":  28.0,   # between fighter (30) and transport (20)
                "raymer_eq":        "fighter",   # MIL-SPEC structural regime
                # CD_misc: Raymer Table 12.6 — buried intake (0.003), bay doors (0.002), excrescence (0.001)
                "CD_misc":          0.005,
                # Flap sizing (elevon — combined lift/control surface)
                "CLmax_clean":      1.00,
                "K_flap":           0.65,
                "flap_chord_bounds":  (0.20, 0.40),
                "flap_span_bounds":   (0.40, 0.80),
                # Fuselage proportionality (derived from X-47B centerbody)
                "fus_length_frac":  2.9,    # short integrated centerbody
                "fus_radius_frac":  0.125,  # thin; blended wing-body
                "fus_nose_frac":    0.15,   # moderate nose taper
                "fus_tail_r_frac":  0.20,   # engine/bay exit; blunt tail
                # Longitudinal balance — static margin model
                "wing_pos_frac":    0.05,   # blended centerbody; wing starts near nose
                "cg_fixed_frac":    0.42,   # payload+struct CG at 42% fus length from nose
                "np_tail_frac":     0.00,   # tailless — no tail NP contribution
                "sm_limit":         0.10,   # ±10% SM bound (FBW managed)
            },
        }
```

```
        print(f"Estimated time: ~{n_starts*7//60} – {n_starts*10//60} min")
        print(f"\n{'Start':>7}  {'Status':8}  {'Range km':>9}  "
              f"{'Sweep °':>8}  {'Span m':>7}  {'L/D':>6}  {'Sec':>5}")
        print("─" * 62)


        for _i in range(n_starts):
            _t0i = _time.time()
            _set_start(_lhs_scaled[_i])


            # Per-start history dict — captured by the closure below
            _h = {"iter":[], "range_km":[], "LoD":[], "span_m":[],
                  "sweep_v":[], "alpha_v":[], "taper_v":[]}


            def _cb(iteration, _h=_h):
                try:
                    r  = float(opti.debug.value(range_m))
                    ld = float(opti.debug.value(L_over_D))
                    sp = float(opti.debug.value(span))
                    sw = float(opti.debug.value(sweep_deg))
                    al = float(opti.debug.value(alpha))
                    tp = float(opti.debug.value(taper))
                    if all(_math.isfinite(v) for v in (r, ld, sp, sw, al, tp)):
                        _h["iter"].append(iteration)
                        _h["range_km"].append(r / 1000.0)
                        _h["LoD"].append(ld)
                        _h["span_m"].append(sp)
                        _h["sweep_v"].append(sw)
                        _h["alpha_v"].append(al)
                        _h["taper_v"].append(tp)
                except Exception:
                    pass


            opti.callback(_cb)


            try:
                _sol = opti.solve(verbose=False)
                _r   = float(_sol.value(range_m))
                _sw  = float(_sol.value(sweep_deg))
                _sp  = float(_sol.value(span))
                _ld  = float(_sol.value(L_over_D))
                _dt  = _time.time() - _t0i
                _results.append((_r, _sw, _sp, _ld, float(_sol.value(taper)),
                                 float(_sol.value(alpha)), _sol, _h))
                print(f"  {_i+1:4d}/{n_starts}  {'☑':8}  {_r/1000:>9.1f}  "
                      f"{_sw:>8.1f}  {_sp:>7.2f}  {_ld:>6.2f}  {_dt:>5.1f}")
            except Exception as _ex:
                _dt = _time.time() - _t0i
                print(f"  {_i+1:4d}/{n_starts}  {'✗':8}  {'—':>9}  "
```

```
fig_plan.add_trace(go.Scatter(
                    x=_y_f, y=_x_hinge, mode="lines",
                    line=dict(color=_flap_col_line, width=1.2, dash="dot"),
                    showlegend=False, hoverinfo="skip"))


            # Aileron region (outboard, flap_sf → ~0.90 span)
            _ail_out = min(_flap_sf + 0.35, 0.92)
            _ail_col_fill = "rgba(127,60,141,0.18)"
            for s in (1,-1):
                _ea = _np.linspace(_flap_sf, _ail_out, 40)
                _y_a  = s * _ea * _b2
                _xle_a = _ea * _b2 * _np.tan(_sw_r)
                _cf_a  = _cr*(1-(1-_taper)*_ea)
                _x_ail_h = _xle_a + 0.75*_cf_a   # aileron at 75% chord
                _x_te_a  = _xle_a + _cf_a
                _xs_a = _np.concatenate([_x_ail_h, _x_te_a[::-1], [_x_ail_h[0]]])
                _ys_a = _np.concatenate([_y_a,     _y_a[::-1], [_y_a[0]]])

                fig_plan.add_trace(go.Scatter(
                    x=_ys_a, y=_xs_a, mode="lines",
                    fill="toself", fillcolor=_ail_col_fill,
                    line=dict(color="mediumpurple", width=1.2),
                    showlegend=(s==1), name="Aileron region",
                    hovertemplate="Aileron region<extra></extra>"))


            # Span arrow annotation (below the planform)
            _x_arr_y = _cr * 1.35
            fig_plan.add_annotation(
                x=-_b2, y=_x_arr_y, ax=_b2, ay=_x_arr_y,
                xref="x", yref="y", axref="x", ayref="y",
                arrowhead=2, arrowcolor="#1a5276", arrowwidth=2,
                text=f"b = {_span:.2f} m", showarrow=True,
                font=dict(size=11, color="#1a5276"),
                xanchor="center", yanchor="bottom")


            # Root chord label
            fig_plan.add_annotation(
                x=0, y=_cr/2, text=f"c_root = {_cr:.2f} m",
                showarrow=False, font=dict(size=10, color="#1a5276"),
                xanchor="left", bgcolor="rgba(255,255,255,0.7)")


            fig_plan.update_layout(
                title=dict(
```

```
_cls = config.get("aircraft_class", "GA")
if _cls not in _PRESETS:
    raise ValueError(
        f"aircraft_class = {_cls!r} is not recognised.  "
        f"Choose one of: {list(_PRESETS.keys())}"
    )
_preset = _PRESETS[_cls]


# Merge: preset values fill in any key NOT already set in config.
# Keys the user set explicitly in config always take priority.
for _k, _v in _preset.items():
    if _k not in config:
        config[_k] = _v


print(f"Aircraft class: {_cls}  "
      f"(Nz={config['N_z']:.2f}, Raymer={config['raymer_eq']}, "
      f"sweep ≤ {config['sweep_bounds'][1]:.0f}°, "
      f"span [{config['span_bounds'][0]:.0f}–{config['span_bounds'][1]:.0f}] m)")


fuel_mass_kg           = config["W_fuel_N"] / 9.81
required_fuel_volume_m3 = fuel_mass_kg / config["fuel_density_kg_m3"]


#
# ============================================
# 2. Atmosphere Objects  (#6)
#
# Two separate atmosphere instances are kept intentionally:
#   atm_cruise  — cruise altitude; used for Raymer q, OperatingPoint, and Re.
#               Changing altitude_m in config automatically propagates to
#               all three consumers in one place.
#   atm_sl      — sea level; used only for the approach/stall constraint.
#               Landing is evaluated at sea level regardless of cruise altitude,
#               which is the conservative (higher density → lower Vstall) direction.
#
# ============================================


atm_cruise = asb.Atmosphere(altitude=config["altitude_m"])
atm_sl     = asb.Atmosphere(altitude=0)


rho_cruise = atm_cruise.density()         # kg/m³  — cruise altitude air density
a_cruise   = atm_cruise.speed_of_sound()  # m/s  — speed of sound at cruise altitude
rho_sl     = atm_sl.density()             # kg/m³  — sea-level density (stall constraint)


#
# ============================================
# 3. Mach Guard  (pre-solve, Python-level)
#
# Checked before the NLP is constructed so the user gets a clear diagnostic
# instead of a cryptic IPOPT failure or silent bad NeuralFoil results.
# NeuralFoil's documented accuracy begins to degrade above ~Ma 0.85–0.90;
# mach_limit = 0.88 provides a 2% buffer below that.
                                f"{'—':>8}  {'—':>7}  {'—':>6}  {_dt:>5.1f}"
                                f"  [{str(_ex)[:35]}]")


        _t_wall = _time.time() - _t0_wall
        print("─" * 62)
        print(f"  {len(_results)}/{n_starts} successful  |  "
              f"{_t_wall:.0f} s total ({_t_wall/60:.1f} min)")


        if not _results:
            raise RuntimeError(
                "All starts failed — check variable bounds, Mach limit, or constraints.")


        # ── Select best feasible solution ──────────────────────────────────────
        _results.sort(key=lambda x: x[0], reverse=True)
        _best_range, _best_sweep, _best_span, _best_LoD_ms, _best_taper_ms, _best_alpha_ms, sol, _hist = _results[0]


        print(f"\n★  Best:  Range = {_best_range/1000:.1f} km  "
              f"|  Sweep = {_best_sweep:.1f}°  |  Span = {_best_span:.2f} m")


        # ── Multi-start summary figure ─────────────────────────────────────────
        # Scatter of every feasible start: x = LE sweep, y = range, colour = span.
        # This reveals the sweep vs range trade-off across the full population and
        # confirms the best result is not an isolated outlier.
        import plotly.graph_objects as _pgo


        _r_all  = [x[0]/1000 for x in _results]
        _sw_all = [x[1]      for x in _results]
        _sp_all = [x[2]      for x in _results]
        _ld_all = [x[3]      for x in _results]
        _tp_all = [x[4]      for x in _results]


        _fig_ms = _pgo.Figure()
        _fig_ms.add_trace(_pgo.Scatter(
            x=_sw_all, y=_r_all,
            mode="markers",
            marker=dict(
                size=11,
                color=_sp_all,
                colorscale="Viridis",
                colorbar=dict(title="Span (m)", thickness=14, len=0.75),
                line=dict(color="rgba(40,40,40,0.4)", width=0.8),
            ),
            text=[f"Start {i+1}<br>Range: {r:.1f} km<br>"
                  f"Sweep: {s:.1f}°<br>Span: {sp:.2f} m<br>"
                  f"L/D: {ld:.2f}  λ={tp:.3f}"
                  for i, (r, s, sp, ld, tp, *_) in enumerate(_results)],
            hovertemplate="%{text}<extra></extra>",
            name="Feasible starts",
        ))
        _fig_ms.add_trace(_pgo.Scatter(
                        text=(f"Wing Planform with High-Lift Devices  ·  "
                              f"b = {_span:.2f} m  AR = {_AR:.2f}  "
                              f"Λ_LE = {_sweep:.1f}°  λ = {_taper:.3f}"),
                        font=dict(size=13, color="#1a3a5c")),
                    height=520,
                    template="plotly_white",
                    font=dict(family="Arial", size=11),
                    xaxis=dict(title="Span y (m)", showgrid=True, gridcolor="#e0e8f0",
                               zeroline=True, zerolinecolor="#aaa"),
                    yaxis=dict(title="Chord x (m, aft →)", showgrid=True,
                               gridcolor="#e0e8f0", autorange="reversed",
                               scaleanchor="x", scaleratio=1),
                    legend=dict(x=0.01, y=0.01, bgcolor="rgba(255,255,255,0.88)",
                                bordercolor="#cccccc", borderwidth=1,
                                font=dict(size=10)),
                )
                fig_plan.show()

fig_plan.write_html("planform_with_flaps.html")
                print("Saved: planform_with_flaps.html")
                # ── Spanwise Lift Distribution (VLM) ──────────────────────────────────
                try:
                    print("--- Spanwise Lift & Load Distribution (VLM) ---")
                    _N_SP_ld, _N_CH_ld = 30, 8
                    _fus_r_ld  = config["root_chord_m"] * config["fus_radius_frac"]
                    _b2_ld     = _span / 2.0
                    _sw_ld     = _np.radians(_sweep)
                    _eexp_ld   = _fus_r_ld / _b2_ld


                    _xsecs_ld = []
                    for _eta_ld in [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]:
                        _eov_ld  = _eexp_ld + _eta_ld * (1.0 - _eexp_ld)
                        _yp_ld   = _fus_r_ld + _eta_ld * (_b2_ld - _fus_r_ld)
                        _wu_ld   = ((1-_eov_ld) * sol.value(root_w_up) + _eov_ld * sol.value(tip_w_up))
                        _wl_ld   = ((1-_eov_ld) * sol.value(root_w_low) + _eov_ld * sol.value(tip_w_low))


                        _xsecs_ld.append(asb.WingXSec(
                            xyz_le=[_yp_ld * _np.tan(_sw_ld), _yp_ld, 0],
                            chord=config["root_chord_m"] * (1.0 - (1.0 - _taper) * _eov_ld),
                            twist=config["twist_tip_deg"] * _eov_ld,
                            airfoil=asb.KulfanAirfoil(
```

```
        # To explore higher speeds, raise mach_limit with awareness of that caveat.
        #
# ============================================


        mach_cruise = config["v_cruise_mps"] / a_cruise
        if mach_cruise > config["mach_limit"]:
            raise ValueError(
                f"\n  v_cruise = {config['v_cruise_mps']:.1f} m/s → Ma = {mach_cruise:.3f}  "
                f"at altitude {config['altitude_m']} m\n"
                f"  This exceeds mach_limit = {config['mach_limit']:.2f}.\n"
                f"  Either reduce v_cruise_mps, increase altitude_m (higher altitude raises Ma "
                f"for the same TAS), or raise mach_limit if you accept reduced NeuralFoil accuracy."
            )


        print(f"Cruise Mach = {mach_cruise:.3f}  (limit {config['mach_limit']:.2f})  ✓")


        #
# ============================================
        # 4. Precomputed CST Basis Coefficients  (#3)
        #    Plain Python/numpy constants — evaluated once before the optimizer runs.
        #
# ============================================


        N_cst = 7  # CST polynomial order (8 weights → order-7 Bernstein basis)


        # ── Spar / max-thickness evaluation point ──────────────────────────────
        # The CST class function  C(x) = x^0.5·(1-x)  is maximised at x = 1/3
        # (dC/dx = 0).  Evaluating t/c here gives the best single-point proxy for
        # the true maximum thickness and coincides with a typical forward spar.
        x_spar  = 1.0 / 3.0
        C_spar  = x_spar**0.5 * (1.0 - x_spar)   # ≈ 0.3849


        B_spar_list = [
            math_comb(N_cst, i) * x_spar**i * (1.0 - x_spar)**(N_cst - i)
            for i in range(N_cst + 1)
        ]


        # ── Exact cross-sectional area integral coefficients ──────────────────
        # A/c² = Σᵢ (w_upᵢ − w_lowᵢ) · C(N,i) · B(i+1.5, N-i+2)
        # Second Beta argument is N-i+2 (not N-i+1); the wrong index omits the
        # (1-x) factor from the class function, underestimating area by ~30%.
        area_coeff_list = [
            math_comb(N_cst, i) * beta_fn(i + 1.5, N_cst - i + 2)
            for i in range(N_cst + 1)
        ]


        #
# ============================================
        # 5. Dynamic Initial Guesses  (Mach-aware)
        #
        # IPOPT convergence is sensitive to initial guess quality.  For a fixed
        # geometry configuration the defaults below are fine, but at high Mach the
```

```
                x=[_best_sweep],
y=[_best_range/1000],
                mode="markers",
                marker=dict(symbol="star",
size=22, color="red",
                            line=dict(color="darkred",
width=1.5)),
                name=f"Best
({_best_range/1000:.1f} km)",
                hovertemplate=(f"Best
start<br>Range: {_best_range/1000:.1f} km<br>"
                               f"Sweep:
{_best_sweep:.1f}°<br>"
                               f"Span:
{_best_span:.2f} m<extra></extra>"),
                ))
            _fig_ms.update_layout(
                title=dict(
                    text=(f"Multi-Start Population:
Range vs LE Sweep<br>"
                          f"<sup>{len(_results)}
feasible / {n_starts} total  |  "
                          f"Ma = {mach_cruise:.3f}  |
colour = span (m)</sup>"),
                    font=dict(size=13),
                ),
                xaxis=dict(title="LE Sweep
(deg)", showgrid=True, gridcolor="#e8e8e8"),
                yaxis=dict(title="Range (km)",
showgrid=True, gridcolor="#e8e8e8"),
                legend=dict(x=0.02, y=0.02,
bgcolor="rgba(255,255,255,0.85)",
                            bordercolor="#cccccc",
borderwidth=1),
                height=490,
                template="plotly_white",
                font=dict(family="Arial", size=12),
            )
            _fig_ms.show()
            _fig_ms.write_html("multistart_summary.html")
            print("Saved:
multistart_summary.html")
        # ── Multi-start parallel coordinates ──────────────────────────────────
        # Each line is one feasible start. Drag the axis range sliders to isolate
        # design families; line colour = range (green = high, purple = low).
        import pandas as _pd
        _fig_pc =
_pgo.Figure(data=_pgo.Parcoords(
            line=dict(
                color=_r_all,
                colorscale="Viridis",
                showscale=True,

colorbar=dict(title=dict(text="Range<br>(km)",
side="right"),
                              thickness=14,
len=0.70, tickformat=".0f"),
            ),
            dimensions=[
                dict(range=[min(_r_all),
max(_r_all)],  label="Range (km)",
values=_r_all),
                dict(range=[min(_sw_all),
max(_sw_all)],  label="LE Sweep (°)",
values=_sw_all),
                dict(range=[min(_sp_all),
max(_sp_all)],  label="Span (m)",
values=_sp_all),
                dict(range=[min(_ld_all),
max(_ld_all)],  label="L / D",        values=_ld_all),
                dict(range=[min(_tp_all),
max(_tp_all)],  label="Taper λ",
values=_tp_all),
```

```
name=f"ld{int(_eta_ld*100)}",

upper_weights=_wu_ld,
lower_weights=_wl_ld),
                        ))
        _wing_ld  =
asb.Wing(name="W_ld", symmetric=True,
xsecs=_xsecs_ld)
        _plane_ld =
asb.Airplane(name="P_ld", wings=[_wing_ld])
        _op_ld    =
asb.OperatingPoint(

velocity=config["v_cruise_mps"],
alpha=_alpha, atmosphere=atm_cruise)
        _vlm_ld   =
asb.VortexLatticeMethod(
                    airplane=_plane_ld,
op_point=_op_ld,

spanwise_resolution=_N_SP_ld,
chordwise_resolution=_N_CH_ld)
        _vlm_ld.run()


        # Spanwise stations
        _sp_e_ld = _np.linspace(0,
1, _N_SP_ld + 1)
        _sp_c_ld = 0.5 *
(_sp_e_ld[:-1] + _sp_e_ld[1:])
        _y_ld    = _fus_r_ld +
_sp_c_ld * (_b2_ld - _fus_r_ld)
        _y_norm  = _y_ld / _b2_ld
# y / (b/2)


        # Chord at each station
(linear taper)
        _eov_sp  = _eexp_ld +
_sp_c_ld * (1.0 - _eexp_ld)
        _c_sp    =
config["root_chord_m"] * (1.0 - (1.0 - _taper) *
_eov_sp)


        # Sum gammas chordwise
per spanwise station → bound circulation Γ(y)
        _gammas_ld =
_np.array(_vlm_ld.vortex_strengths).flatten()
        _n_half_ld = _N_SP_ld *
_N_CH_ld
        _gam_sb    =
_gammas_ld[:_n_half_ld].reshape(_N_SP_ld,
_N_CH_ld)
        _Gamma_y   =
_gam_sb.sum(axis=1)   # total circulation per strip (m²/s)


        # Kutta–Joukowski: dL/dy =
rho * V * Gamma(y)
        _dL_dy   = rho_cruise *
config["v_cruise_mps"] * _Gamma_y
        _q_ld    = 0.5 * rho_cruise *
config["v_cruise_mps"]**2
        # Local section lift
coefficient: cl(y) = dL/dy / (q * c(y))
        _cl_y    = _dL_dy / (_q_ld *
_c_sp)
        # Normalised loading = cl *
c / (CL_3D * S / b)
        _CL_3d   =
float(sol.value(aero_results["L"]) / (_q_ld *
float(sol.value(S_w))))
        _ref_val = _CL_3d *
float(sol.value(S_w)) / _span   # normalisation
```

```
        # required cruise CL and alpha are dramatically lower than at low speed, and
        # the Raymer wing weight is higher (larger q). Computing physics-consistent
        # init guesses here means the same script converges reliably from Ma 0.1 to
        # Ma 0.88 without manual tuning.
        #
# ============================================


        # — alpha init: reverse-engineer the CL at the class-appropriate nominal planform.
        #   Use the midpoint of span_bounds (not a hardcoded 11 m) so the estimate
        #   scales correctly from GA (span ~11 m) through transport (span ~45 m).
        _q_guess      = 0.5 * rho_cruise * config["v_cruise_mps"]**2
        _span_mid     = 0.5 * (config["span_bounds"][0] + config["span_bounds"][1])
        _S_guess      = _span_mid * config["root_chord_m"] * (1.0 + 0.45) / 2.0
        _W0_guess     = config["W_fixed_N"] + config["W_fuel_N"] + 2000.0    # + rough wing weight
        _CL_guess     = _W0_guess / (_q_guess * _S_guess)
        _alpha_init   = float(min(max(_CL_guess / 0.1, 0.5), 10.0))         # clamp [0.5, 10] deg


        # — W_wing init: single-pass Raymer evaluation at nominal conditions so the
        #   weight-loop algebraic constraint starts close to its solution.
        _S_w_ft2_0  = _S_guess * 10.7639
        _q_psf_0    = _q_guess * 0.020885
        _W_fw_lb_0  = config["W_fuel_N"] / 4.4482
        _AR_0       = 11.0**2 / _S_guess
        _W0_lb_0    = _W0_guess / 4.4482
        _raymer_0   = (
            0.036
            * _S_w_ft2_0**0.758
            * _W_fw_lb_0**0.0035
            * _AR_0**0.6
            * _q_psf_0**0.006
            * 0.7**0.04
            * (100.0 * 0.18)**(-0.3)          # nominal t/c ≈ 0.18
            * (config["N_z"] * _W0_lb_0)**0.49
        )
        _W_wing_init = max(_raymer_0 * 4.4482, 200.0)   # N, clamped to lower bound


        #
# ============================================
        # 6. Optimizer
        #
# ============================================


        opti = asb.Opti()


        #
# ============================================
        # 7. Planform Variables  (4 design variables)
        #
# ============================================


        # Bounds are read from the presets-merged config so every class gets the
        # correct search space.  The lower bound on sweep is always 0 (no forward
        # sweep — physically inadmissible without an aeroelastic divergence model).
        _ab = config["alpha_bounds"]
        _sb = config["span_bounds"]
        _tb = config["taper_bounds"]
            ],
        ))
        _fig_pc.update_layout(
            title=dict(
                text=(f"Multi-Start Parallel Coordinates — {len(_results)} feasible / "
                      f"{n_starts} total<br>"
                      f"<sup>Each line = one feasible start · colour = range · "
                      f"drag axes to filter design families</sup>"),
                font=dict(size=13)),
            height=400,
            template="plotly_white", font=dict(family="Arial", size=11),
            margin=dict(l=80, r=80, t=80, b=40),
        )
        _fig_pc.show()
        _fig_pc.write_html("multistart_parallel_coords.html")
        print("Saved: multistart_parallel_coords.html")




        #
# ============================================
        # 20. Extract Best-Start Scalars & Print Summary
        #
# ============================================


        try:
            _span    = sol.value(span)
            _taper   = sol.value(taper)
            _alpha   = sol.value(alpha)
            _sweep   = sol.value(sweep_deg)
            _S_w     = sol.value(S_w)
            _AR      = sol.value(AR)
            _tip_x   = sol.value(tip_le_x)
            _W_wing  = sol.value(W_wing_N)
            _W_init  = sol.value(W_initial_N)
            _W_final = sol.value(W_final_N)
            _root_tc = sol.value(root_t_tc)
            _tip_tc  = sol.value(tip_t_tc)
            _LoD     = sol.value(L_over_D)
            _range   = sol.value(range_m)
            _Vstall  = sol.value(V_stall_sq)**0.5
            _V_app   = 1.2 * _Vstall
            _slend   = (_span / 2.0) / (_root_tc * config["root_chord_m"])
            _Mdd_val = sol.value(_Mdd)
            _CDw_val = sol.value(_CD_wave)
            _Dw_val  = sol.value(_D_wave)
            _LoD_inv = sol.value(aero_results["L"] / aero_results["D"])


            print("\n☑ Optimization Successful!\n")


            print("--- Optimized Mission & Planform ---")
            print(f"Max Range: {_range / 1000:.1f} km")
            print(f"Cruise L/D (net): {_LoD:.2f}  (aero-only: {_LoD_inv:.2f})")
            _load_y  = _cl_y * _c_sp / _ref_val


            # Elliptical reference loading: sqrt(1 - (2y/b)²)
            _y_ell   = _np.linspace(0, 1, 200)
            _ell_ref = _np.sqrt(_np.maximum(1.0 - _y_ell**2, 0.0))


            fig_ld = go.Figure()
            # Starboard side
            fig_ld.add_trace(go.Scatter(
                x=_y_norm, y=_load_y, mode="lines+markers",

line=dict(color="#2e86c1", width=2.5),
                marker=dict(size=5, color="#2e86c1"),
                name="Optimized loading", fill="tozeroy",

fillcolor="rgba(46,134,193,0.12)",

hovertemplate="y/(b/2)=%{x:.3f}<br>cl·c/c̄=%{y:.3f}<extra>Optimized</extra>",
            ))
            # Port side (mirror)
            fig_ld.add_trace(go.Scatter(
                x=-_y_norm[::-1], y=_load_y[::-1], mode="lines+markers",

line=dict(color="#2e86c1", width=2.5),
                marker=dict(size=5, color="#2e86c1"),
                showlegend=False, fill="tozeroy",

fillcolor="rgba(46,134,193,0.12)",

hovertemplate="y/(b/2)=%{x:.3f}<br>cl·c/c̄=%{y:.3f}<extra>Optimized</extra>",
            ))
            # Elliptical reference (both sides)
            for _sign, _legend in [(1, True), (-1, False)]:

fig_ld.add_trace(go.Scatter(
                    x=_sign * _y_ell, y=_ell_ref, mode="lines",

line=dict(color="rgba(200,0,0,0.55)", width=1.8, dash="dash"),
                    name="Elliptical reference", showlegend=_legend,

hovertemplate="y/(b/2)=%{x:.3f}<br>elliptical=%{y:.3f}<extra></extra>",
                ))
            fig_ld.add_vline(x=0, line=dict(color="#aaa", width=1, dash="dot"))
            fig_ld.update_layout(
                title=dict(
                    text=(f"Spanwise Lift Distribution (VLM) · "
                          f"b={_span:.2f} m  AR={_AR:.2f}  α={_alpha:.1f}°<br>"
                          f"<sup>Normalised loading cl·c/c̄ vs η=y/(b/2) — "
                          f"dashed=elliptical ideal</sup>"),
                    font=dict(size=13)),
                xaxis=dict(title="η = y / (b/2)", showgrid=True, gridcolor="#e0e8f0",
```

```
        _wb = config["sweep_bounds"]
        _span_init = float(0.5 * (_sb[0] + _sb[1]))   # class-
appropriate span midpoint


        alpha     = opti.variable(init_guess=_alpha_init,
lower_bound=_ab[0], upper_bound=_ab[1])
        span      = opti.variable(init_guess=_span_init,
lower_bound=_sb[0], upper_bound=_sb[1])
        taper     = opti.variable(init_guess=0.45,
lower_bound=_tb[0], upper_bound=_tb[1])
        sweep_deg = opti.variable(init_guess=max(_wb[0],
5.0),
                                  lower_bound=_wb[0],
upper_bound=_wb[1])
        sweep_rad = np.radians(sweep_deg)


        #
==========================================
        # 8. Wing Weight Variable  (1 implicit variable —
Raymer)
        #
==========================================


        W_wing_N = opti.variable(init_guess=_W_wing_init,

lower_bound=config["W_wing_bounds_N"][0],

upper_bound=config["W_wing_bounds_N"][1])


        #
==========================================
        # 8b. Flap Design Variables  (2 design variables)
        #
        # flap_chord_ratio : c_f/c — flap chord as fraction of
local wing chord.
        # flap_span_frac   : fraction of exposed semi-span
covered by flap (inboard).
        #
        # Both bounded by class-specific presets.  The
optimizer trades smaller
        # required wing area (lower induced drag, wave drag,
Raymer weight) against
        # the flap geometry needed to satisfy the approach
speed stall constraint.
        #
        # Flap weight note: Raymer wing weight equations
(Eq 15.2, 15.46, 15.25)
        # are empirical fits to complete wing assemblies
including flap structure.
        # A separate flap weight term would double-count; the
Raymer equality
        # already captures the weight penalty indirectly
through S_w and MTOW.
        #
==========================================


        _fcb = config["flap_chord_bounds"]
        _fsb = config["flap_span_bounds"]


        flap_chord_ratio = opti.variable(
            init_guess=float(0.5 * (_fcb[0] + _fcb[1])),
            lower_bound=_fcb[0], upper_bound=_fcb[1],
        )
        flap_span_frac = opti.variable(
            init_guess=float(0.5 * (_fsb[0] + _fsb[1])),
            lower_bound=_fsb[0], upper_bound=_fsb[1],
        )


        #
==========================================
        # 9. Derived Planform Geometry
```

```
        print(f"Cruise Altitude:
{config['altitude_m']:.0f} m  "
              f"({config['altitude_m'] *
3.281:.0f} ft)  |  Ma = {mach_cruise:.3f}")
        print(f"VLM stations / panels: 6
spanwise stations  →  "
              f"5 panels/half-wing  (4
NeuralFoil midspan stations active)")
        print(f"Optimal Span:
{_span:.2f} m")
        print(f"Optimal Taper:
{_taper:.2f}")
        print(f"Optimal Alpha:
{_alpha:.2f} deg")
        print(f"Optimal LE Sweep:
{_sweep:.2f} deg")
        print(f"Tip LE x-offset:
{_tip_x:.3f} m")
        print(f"Wing Area:
{_S_w:.2f} m²")
        print(f"Aspect Ratio:
{_AR:.2f}")
        print()


        _flap_cr  =
sol.value(flap_chord_ratio)
        _flap_sf  =
sol.value(flap_span_frac)
        _dCL_flap =
sol.value(dCL_max_flap)
        _CLmax_t  =
sol.value(CLmax_total)


        print("--- Flap Sizing ---")
        print(f"Flap chord ratio (c_f/c):
{_flap_cr:.3f}  "
              f"(bounds
[{config['flap_chord_bounds'][0]:.2f} –
{config['flap_chord_bounds'][1]:.2f}])")
        print(f"Flap span fraction:
{_flap_sf:.3f}  "
              f"(bounds
[{config['flap_span_bounds'][0]:.2f} –
{config['flap_span_bounds'][1]:.2f}])")
        print(f"Delta CLmax from flap:
{_dCL_flap:.3f}  "
              f"(K_flap={config['K_flap']:.2f},
class={config['aircraft_class']})")
        print(f"CLmax_clean (no flap):
{config['CLmax_clean']:.2f}")
        print(f"CLmax_total (with flap):
{_CLmax_t:.3f}  "
              f"(ceiling:
{config['CLmax_approach']:.2f})")
        print(f"Vstall (landing, flapped):
{_Vstall:.1f} m/s  ({_Vstall*1.944:.1f} kt)")
        print()


        # ── Solved airfoil objects (used
by drag split, dat export, and plots) ──
        final_root = asb.KulfanAirfoil(
            name="Optimized_Root",

upper_weights=sol.value(root_w_up),

lower_weights=sol.value(root_w_low),
        )
        final_tip = asb.KulfanAirfoil(
            name="Optimized_Tip",

upper_weights=sol.value(tip_w_up),

lower_weights=sol.value(tip_w_low),
        )
```

```
                zeroline=True,
zerolinecolor="#aaa"),
                yaxis=dict(title="cl · c / c̄
(normalised loading)", showgrid=True,

gridcolor="#e0e8f0", range=[0, None]),
                height=420,
template="plotly_white",
                font=dict(family="Arial",
size=11),
                legend=dict(x=0.70,
y=0.97, bgcolor="rgba(255,255,255,0.85)",

bordercolor="#cccccc", borderwidth=1),
            )
            fig_ld.show()

fig_ld.write_html("spanwise_load_distribution.
html")
            print("Saved:
spanwise_load_distribution.html")
        except Exception as _ld_err:
            print(f"  ⚠  Spanwise
distribution skipped: {_ld_err}")



        # ── 3D wing surface
─────────────────────────────────────
──────────────────────────────────

        def _hover(X_arr, Y_arr,
Z_arr):
            return [[f"x: {X_arr[i,j]:.3f}
m<br>y: {Y_arr[i,j]:.3f} m<br>z: {Z_arr[i,j]:.4f}
m"
                     for j in
range(X_arr.shape[1])]
                    for i in
range(X_arr.shape[0])]


        fig_3d = go.Figure()
        for X_h, Y_h, Z_h, side in [(Xr,
Yr, Zr, "Starboard"), (Xl, Yl, Zl, "Port")]:

fig_3d.add_trace(go.Surface(
            x=X_h, y=Y_h, z=Z_h,
            colorscale=[[0,
"#aed6f1"], [0.5, "#2e86c1"], [1, "#1a5276"]],
            cmin=-0.05, cmax=0.05,
surfacecolor=Z_h,
            showscale=False,
opacity=0.92, name=side,
            text=_hover(X_h, Y_h,
Z_h),

hovertemplate="%{text}<extra>" + side +
"</extra>",

lighting=dict(ambient=0.6, diffuse=0.8,
specular=0.3,
                          roughness=0.5,
fresnel=0.2),
            lightposition=dict(x=1000,
y=500, z=2000),
        ))


        for eta, foil, label in [(0.0,
final_root, "Root"), (1.0, final_tip, "Tip")]:
            c       =
_np.array(foil.coordinates)
            chord_e =
config["root_chord_m"] * (1.0 - (1.0 - _taper) *
eta)
            x_le_e  = eta * (_span / 2.0)
* _np.tan(_np.radians(_sweep))
```

```
        #
========================================


        tip_le_x    = (span / 2.0) * np.tan(sweep_rad)
        S_w         = span * config["root_chord_m"] * (1.0 +
taper) / 2.0
        AR          = span**2 / S_w
        tan_qc      = np.tan(sweep_rad) - (1.0 / AR) * (1.0 -
taper) / (1.0 + taper)
        sweep_qc_rad = np.arctan(tan_qc)


        # Trailing-edge sweep (used as conservative hinge-
line sweep for flap model)
        # tan(Λ_TE) = tan(Λ_LE) - (2/AR)·(1-λ)/(1+λ)
        tan_te      = np.tan(sweep_rad) - (2.0 / AR) * (1.0 -
taper) / (1.0 + taper)
        sweep_te_rad = np.arctan(tan_te)


        # CLmax augmentation from flap (Raymer Eq. 12.21
form):
        #   ΔCL_max = K_flap × span_frac × (c_f/c)^0.5 ×
cos(Λ_TE)
        # The (c_f/c)^0.5 captures the nonlinear 2D ΔCl vs
chord ratio relationship
        # from Raymer Fig. 12.22 across the 0.10–0.40 chord
ratio range.
        # K_flap is class-specific and rolls up: 3D correction
(0.9), flap type
        # effectiveness (ΔCl_2D), and any LE device
contribution.
        dCL_max_flap = (
            config["K_flap"]
            * flap_span_frac
            * flap_chord_ratio**0.5
            * np.cos(sweep_te_rad)
        )
        CLmax_total = config["CLmax_clean"] +
dCL_max_flap


        # Mean aerodynamic chord and fuselage length —
defined here (section 9) so
        # they are available to both section 16b (static margin)
and section 17b
        # (fuselage geometry builder).  Both depend only on
taper and config
        # constants, which are fully defined by this point.
        mac         = config["root_chord_m"] * (2.0/3.0) * (1.0 +
taper + taper**2) / (1.0 + taper)
        fus_length = mac * config["fus_length_frac"]


        #
========================================
        # 10. Airfoil Shape Variables  (32 design variables)
        #
========================================


        init_upper = np.array([0.15, 0.15, 0.20, 0.20, 0.20,
0.18, 0.15, 0.12])
        init_lower = np.array([-0.10, -0.10, -0.12, -0.12, -0.10,
-0.08, -0.05, -0.02])


        root_w_up  = opti.variable(init_guess=init_upper,
lower_bound=0.001, upper_bound=0.5)
        root_w_low = opti.variable(init_guess=init_lower,
lower_bound=-0.5,  upper_bound=-0.001)
        tip_w_up   = opti.variable(init_guess=init_upper,
lower_bound=0.001, upper_bound=0.5)
        tip_w_low  = opti.variable(init_guess=init_lower,
lower_bound=-0.5,  upper_bound=-0.001)


        # ── Post-solve fuselage drag
split
─────────────────────────────────────────
─────

        # AeroBuildup already included
fuselage drag in aero_results["D"].
        # To isolate it, run a wing-only
AeroBuildup at the solved state and
        # subtract.  This is post-
processing only — no NLP cost.
        # Rebuild exposed multi-station
wing from solved scalars for drag split.
        # Root xsec at y =
fus_max_radius (fuselage side skin), consistent
with
        # the exposed planform used
during optimization.
        _fus_r_post   =
config["root_chord_m"] * config["fus_radius_frac"]
        _eta_exp_post = _fus_r_post /
(_span / 2.0)


        # Exposed root chord and
position
        _c_root_exp_post =
config["root_chord_m"] * (1.0 - (1.0 - _taper) *
_eta_exp_post)
        _x_root_exp_post = _fus_r_post
* _np.tan(_np.radians(_sweep))
        _w_up_root_exp   = ((1.0 -
_eta_exp_post) * sol.value(root_w_up)
                            + _eta_exp_post
* sol.value(tip_w_up))
        _w_low_root_exp  = ((1.0 -
_eta_exp_post) * sol.value(root_w_low)
                            + _eta_exp_post
* sol.value(tip_w_low))


        _mid_xsecs_post = []
        for _eta_exp in [0.2, 0.4, 0.6,
0.8]:
            _eta_ov    = _eta_exp_post +
_eta_exp * (1.0 - _eta_exp_post)
            _y_post    = _fus_r_post +
_eta_exp * (_span / 2.0 - _fus_r_post)
            _w_up_eta  = (1.0 - _eta_ov) *
sol.value(root_w_up)  + _eta_ov *
sol.value(tip_w_up)
            _w_low_eta = (1.0 - _eta_ov) *
sol.value(root_w_low) + _eta_ov *
sol.value(tip_w_low)
            _chord_eta =
config["root_chord_m"] * (1.0 - (1.0 - _taper) *
_eta_ov)

_mid_xsecs_post.append(asb.WingXSec(
                xyz_le=[_y_post *
_np.tan(_np.radians(_sweep)), _y_post, 0],
                chord=_chord_eta,
                twist=config["twist_tip_deg"]
* _eta_ov,
                airfoil=asb.KulfanAirfoil(

name=f"Post_etaexp{int(_eta_exp*100):02d}",

upper_weights=_w_up_eta,

lower_weights=_w_low_eta,
                ),
            ))


        _final_wing_only = asb.Airplane(
            name="Wing only",
            twist_e =
_np.radians(config["twist_tip_deg"] * eta)
            xc_r    = 0.25 + (c[:,0]-
0.25)*_np.cos(twist_e) - c[:,1]*_np.sin(twist_e)
            zc_r    =       (c[:,0]-
0.25)*_np.sin(twist_e) + c[:,1]*_np.cos(twist_e)
            for y_sign in (1.0, -1.0):

fig_3d.add_trace(go.Scatter3d(
                    x=x_le_e + xc_r *
chord_e,
                    y=y_sign * eta *
(_span / 2.0) * _np.ones(len(c)),
                    z=zc_r * chord_e,
                    mode="lines",
line=dict(color="#1a3a5c", width=3),
                    showlegend=False,
hoverinfo="skip",
                ))


        for xc_frac, label in [(0.0,
"LE"), (1.0, "TE")]:
            etas_le = _np.linspace(0, 1,
40)
            x_le_s  = (etas_le *
(_span/2) * _np.tan(_np.radians(_sweep))
                       + xc_frac *
config["root_chord_m"] * (1-(1-
_taper)*etas_le))
            y_le_s  = etas_le * (_span /
2.0)
            x_full  =
_np.concatenate([ x_le_s[::-1], x_le_s])
            y_full  = _np.concatenate([-
y_le_s[::-1], y_le_s])
            z_full  =
_np.zeros_like(x_full)

fig_3d.add_trace(go.Scatter3d(
                x=x_full, y=y_full,
z=z_full, mode="lines",

line=dict(color="#1a3a5c", width=2,
dash="dot"),
                name=label,
hoverinfo="skip",
            ))


        fig_3d.update_layout(
            title=dict(
                text=(f"3D Wing Surface
|  b = {_span:.2f} m, AR = {_AR:.2f}, "
                      f"Λ_LE =
{_sweep:.1f}°, λ = {_taper:.2f}<br>"
                      f"<sup>Range =
{_range/1000:.0f} km, L/D = {_LoD:.1f}, "
                      f"Ma =
{mach_cruise:.3f}</sup>"),
                font=dict(size=13)),
            scene=dict(
                xaxis=dict(title="X —
chord (m)", showgrid=True,

gridcolor="#d5d8dc",
backgroundcolor="#f8f9fa"),
                yaxis=dict(title="Y —
span (m)",  showgrid=True,

gridcolor="#d5d8dc",
backgroundcolor="#f0f3f4"),
                zaxis=dict(title="Z —
normal (m)", showgrid=True,

gridcolor="#d5d8dc",
backgroundcolor="#fdfefe"),
```

```
        #
============================================
        # 11. CST Physical Quantities
        #
============================================


        root_t_tc = C_spar * sum(
            B_spar_list[i] * (root_w_up[i] - root_w_low[i]) for i in
range(N_cst + 1)
        )
        tip_t_tc = C_spar * sum(
            B_spar_list[i] * (tip_w_up[i] - tip_w_low[i]) for i in
range(N_cst + 1)
        )
        root_area_tc = sum(
            area_coeff_list[i] * (root_w_up[i] - root_w_low[i]) for
i in range(N_cst + 1)
        )


        total_wing_volume = (
            root_area_tc
            * config["root_chord_m"]**2
            * span
            * (1.0 + taper + taper**2) / 3.0
        )
        usable_fuel_volume = total_wing_volume *
config["usable_wing_fraction"]


        #
============================================
        # 12. Weight Buildup
        #
============================================


        W_final_N   = config["W_fixed_N"] + W_wing_N
        W_initial_N = W_final_N + config["W_fuel_N"]


        #
============================================
        # 13. Structural & Surface-Validity Constraints
        #
============================================


        opti.subject_to(root_w_up > root_w_low)
        opti.subject_to(tip_w_up  > tip_w_low)


        opti.subject_to(root_t_tc >= config["root_tc_min"])
        opti.subject_to(tip_t_tc  >= config["tip_tc_min"])


        root_spar_depth_m = root_t_tc *
config["root_chord_m"]
        opti.subject_to(span / 2.0 <=
config["slenderness_max"] * root_spar_depth_m)
        #
============================================
        # 14. Fuel Volume Constraint
        #
============================================


        opti.subject_to(usable_fuel_volume >=
required_fuel_volume_m3)


        #
============================================
        # 15. Raymer Wing Weight Equality Constraint
(class-aware)
        #
        # Three separate empirical equations cover the three
aircraft classes.
```

```
                    wings=[asb.Wing(
                        name="Optimized Wing
(exposed)", symmetric=True,
                        xsecs=[
                            asb.WingXSec(

xyz_le=[_x_root_exp_post, _fus_r_post, 0],

chord=_c_root_exp_post, twist=0,

airfoil=asb.KulfanAirfoil(

name="Post_root_exp",

upper_weights=_w_up_root_exp,

lower_weights=_w_low_root_exp,
                                ),
                            ),
                            *_mid_xsecs_post,

asb.WingXSec(xyz_le=[_tip_x, _span/2, 0],

chord=config["root_chord_m"]*_taper,

twist=config["twist_tip_deg"], airfoil=final_tip),
                            ],
                        )],
                    )
                    _op_final  =
asb.OperatingPoint(velocity=config["v_cruise_mp
s"],

alpha=_alpha, atmosphere=atm_cruise)
                    _aero_wing =
asb.AeroBuildup(airplane=_final_wing_only,

op_point=_op_final).run()
                    # AeroBuildup returns numpy
arrays; use .item() to extract the scalar.
                    # sol.value() already returns a
numpy scalar so float() works there.
                    _D_wing_N     =
float(_np.asarray(_aero_wing["D"]).item())
                    _D_aero_total =
float(sol.value(aero_results["D"]))
                    _D_fus_N      =
max(_D_aero_total - _D_wing_N, 0.0)  # guard
numerical noise


                    _mac_val   =
config["root_chord_m"] * (2/3) * (1 + _taper +
_taper**2) / (1 + _taper)
                    _fus_L     = _mac_val *
config["fus_length_frac"]
                    _fus_R     =
config["root_chord_m"] * config["fus_radius_frac"]
                    _fin_ratio = _fus_L / (2.0 *
_fus_R)


                    _D_misc_val =
float(sol.value(_D_misc))


                    print("--- Drag Breakdown ---")
                    print(f"Drag-divergence Mach:
{_Mdd_val:.4f}  (M∞ = {mach_cruise:.3f})")
                    print(f"ΔCD_wave:
{_CDw_val:.6f}")
                    _D_grand_pre = _D_aero_total +
_Dw_val + _D_misc_val
                    print(f"D_wing+fus (AeroBuildup):
{_D_aero_total:.1f} N  "

f"({100*_D_aero_total/_D_grand_pre:.1f}%)")
```

```
                        camera=dict(eye=dict(x=-
1.6, y=-1.8, z=0.9)),
                        aspectmode="data",
                    ),
                    height=680,
template="plotly_white",
                    font=dict(family="Arial",
size=11),
                    legend=dict(x=0.01, y=0.99,
bgcolor="rgba(255,255,255,0.8)",

bordercolor="#cccccc", borderwidth=1),
                    margin=dict(l=0, r=0, t=80,
b=0),
                )
                fig_3d.show()

fig_3d.write_html("wing_3d.html")
                print("Saved: wing_3d.html")


                #
====================================
=======
                # 23b. Interactive 3D Pressure
Distribution (ΔCp)
                #
                # Uses VortexLatticeMethod
directly on the solved geometry with higher
                # panel resolution than the
optimizer used.  VLM gives the net pressure
                # difference ΔCp = Cp_lower
− Cp_upper at each panel, which is the
                # physically correct output of
inviscid panel theory.
                #
                # Visualization:
                #   • Upper surface colored by
−ΔCp (negative = suction = blue)
                #   • Lower surface colored by
+ΔCp (positive = pressure = red)
                #   • Diverging RdBu
colorscale centred at Cp = 0
                #   • Hover shows (x, y, z,
ΔCp) at each surface point
                #   • Chordwise and spanwise
Cp profiles as 2D subplots below the 3D view
                #
====================================
=======


                print("--- Pressure Distribution
(Cp via VLM) ---")
                try:
                    _N_SP  = 24   # spanwise
panels per half-wing
                    _N_CH  = 12   # chordwise
panels


                    # Rebuild solved airplane
(Python scalars only — no CasADi)
                    _fus_r_cp  =
config["root_chord_m"] *
config["fus_radius_frac"]
                    _fus_m_cp  =
(config["root_chord_m"]*(2/3)

*(1+_taper+_taper**2)/(1+_taper))
                    _fus_l_cp  = _fus_m_cp *
config["fus_length_frac"]
                    _sw_cp     =
_np.radians(_sweep)
                    _b2_cp     = _span / 2.0
                    _nf_cp     =
config["fus_nose_frac"]
```

```
            # All inputs are in US customary units as Raymer's
coefficients require.
            # The Raymer equality is enforced as an NLP
constraint; W_wing_N is an
            # optimizer variable so IPOPT resolves the W_wing →
W_gross → W_wing
            # algebraic loop simultaneously with all other
constraints.
            #
            # rho_cruise (not rho_sl) gives the correct structural
dynamic pressure.
            #
            # Equations (Raymer, "Aircraft Design: A Conceptual
Approach", 5th ed.):
            #   GA       Table 15.2  — light aircraft, trapezoidal
planform
            #   transport Table 15.46 — FAR Part 25 commercial
jets
            #   fighter   Table 15.25 — military fighter/attack, low-
AR planform
            #
========================================


            q_Pa    = 0.5 * rho_cruise * config["v_cruise_mps"]**2
            S_w_ft2 = S_w   * 10.7639
            W_fw_lb = config["W_fuel_N"] / 4.4482
            q_psf   = q_Pa  * 0.020885
            W_0_lb  = W_initial_N / 4.4482
            cos_qc  = np.cos(sweep_qc_rad)


            _raymer_eq = config["raymer_eq"]


            if _raymer_eq == "GA":
               # Raymer Eq 15.2 — General Aviation
               # W_w =
0.036·Sw^0.758·Wfw^0.0035·(AR/cos²Λ)^0.6·q^0.006
               #       ·λ^0.04·(100·t/c/cosΛ)^-0.3·(Nz·W0)^0.49
               raymer_W_lb = (
                  0.036
                  * S_w_ft2**0.758
                  * W_fw_lb**0.0035
                  * (AR / cos_qc**2)**0.6
                  * q_psf**0.006
                  * taper**0.04
                  * (100.0 * root_t_tc / cos_qc)**(-0.3)
                  * (config["N_z"] * W_0_lb)**0.49
               )


            elif _raymer_eq == "transport":
               # Raymer Eq 15.46 — Transport Jet
               # W_w =
0.0051·(Wdg·Nz)^0.557·Sw^0.649·AR^0.5·(t/c)^-0.4
               #       ·(1+λ)^0.1·cosΛ^-1·Scsw^0.1
               # Scsw (flap/aileron control surface area) ≈ 0.15·Sw
(typical transport)
               S_csw_ft2 = 0.15 * S_w_ft2
               raymer_W_lb = (
                  0.0051
                  * (W_0_lb * config["N_z"])**0.557
                  * S_w_ft2**0.649
                  * AR**0.5
                  * root_t_tc**(-0.4)
                  * (1.0 + taper)**0.1
                  * cos_qc**(-1.0)
                  * S_csw_ft2**0.1
               )


            elif _raymer_eq == "fighter":
               # Raymer Eq 15.25 — Military Fighter
               # W_w =
0.0103·(W0·Nz)^0.5·Sw^0.622·AR^0.785·(t/c)^-0.4
               #       ·(1+λ)^0.05·cosΛ^-1·Scsw^0.04
               # Scsw ≈ 0.10·Sw; K_rho = K_ws = 1.0 (no
composite/blend correction)
               print(f"D_wave  (Korn+Lock):
{_Dw_val:.1f} N  "

f"({100*_Dw_val/_D_grand_pre:.1f}%)")
               print(f"D_misc  (Raymer 12.6):
{_D_misc_val:.1f} N  "

f"({100*_D_misc_val/_D_grand_pre:.1f}%)")
               print(f"  CD_misc =
{config['CD_misc']:.4f}  (see preset for
breakdown)")
               print(f"D_total:
{_D_grand_pre:.1f} N")
               print()



               print("--- Fuselage Drag
Breakdown ---")
               print(f"Fuselage length:
{_fus_L:.2f} m   "
                     f"(MAC = {_mac_val:.2f} m  x
{config['fus_length_frac']:.1f})")
               print(f"Fuselage max diameter:
{2*_fus_R:.3f} m   "
                     f"(root chord  x
{config['fus_radius_frac']:.3f})")
               print(f"Fineness ratio (L/d):
{_fin_ratio:.1f}")
               _D_grand = _D_wing_N +
_D_fus_N + _Dw_val + _D_misc_val
               print(f"D_wing  (AeroBuildup):
{_D_wing_N:.1f} N
({100*_D_wing_N/_D_grand:.1f}%)")
               print(f"D_fus   (AeroBuildup):
{_D_fus_N:.1f} N
({100*_D_fus_N/_D_grand:.1f}%)")
               print(f"D_wave  (Korn+Lock):
{_Dw_val:.1f} N
({100*_Dw_val/_D_grand:.1f}%)")
               print(f"D_misc  (Raymer 12.6):
{_D_misc_val:.1f} N
({100*_D_misc_val/_D_grand:.1f}%)")
               print(f"D_total (all sources):
{_D_grand:.1f} N")
               print()



               print("--- Weight Buildup
(Raymer) ---")
               print(f"Gross Weight  (W₀):
{_W_init  / 4.448:.0f} lb  |  {_W_init:.0f} N")
               print(f"Wing Weight:
{_W_wing  / 4.448:.0f} lb  |  {_W_wing:.0f} N")
               print(f"Landing Weight:
{_W_final / 4.448:.0f} lb  |  {_W_final:.0f} N")
               print()
               # ── Weight Breakdown Chart
──────────────────────────

               try:
                  _wt_labels = ["Payload
&<br>Structure (W_fixed)", "Wing
Structure<br>(W_wing)", "Fuel<br>(W_fuel)"]
                  _wt_N      =
[config["W_fixed_N"], _W_wing,
config["W_fuel_N"]]
                  _wt_lb     = [v / 4.448 for v in
_wt_N]
                  _wt_pct    = [100.0 * v /
_W_init for v in _wt_N]
                  _wt_colors = ["#2e86c1",
"#1e8449", "#d35400"]
                  _fig_wt = go.Figure()
                  _fig_wt.add_trace(go.Bar(
                     x=_wt_labels, y=_wt_lb,
                     marker_color=_wt_colors,
                     text=[f"{v:.0f}
lb<br>({p:.1f}%)" for v, p in zip(_wt_lb, _wt_pct)],
                  _tf_cp     =
config["fus_tail_r_frac"]


                  _xsecs_cp = []
                  for _eta_cp in [0.0, 0.2, 0.4,
0.6, 0.8, 1.0]:
                     _eexp_cp  = _fus_r_cp /
_b2_cp
                     _eov_cp   = _eexp_cp +
_eta_cp*(1 - _eexp_cp)
                     _yp_cp    = _fus_r_cp +
_eta_cp*(_b2_cp - _fus_r_cp)
                     _wu_cp    = ((1-
_eov_cp)*_np.array(final_root.upper_weights)
                                 +  _eov_cp
*_np.array(final_tip.upper_weights))
                     _wl_cp    = ((1-
_eov_cp)*_np.array(final_root.lower_weights)
                                 +  _eov_cp
*_np.array(final_tip.lower_weights))

_xsecs_cp.append(asb.WingXSec(

xyz_le=[_yp_cp*_np.tan(_sw_cp), _yp_cp, 0],

chord=config["root_chord_m"]*(1-(1-
_taper)*_eov_cp),

twist=config["twist_tip_deg"]*_eov_cp,

airfoil=asb.KulfanAirfoil(

name=f"cp{int(_eta_cp*100)}",

upper_weights=_wu_cp,
lower_weights=_wl_cp),
                     ))


                  _wing_cp =
asb.Wing(name="W_cp", symmetric=True,
xsecs=_xsecs_cp)
                  _fus_cp  =
asb.Fuselage(name="F_cp", xsecs=[

asb.FuselageXSec(xyz_c=[0.00*_fus_l_cp,0,0]
, radius=0.01*_fus_r_cp),

asb.FuselageXSec(xyz_c=[_nf_cp*_fus_l_cp,0
,0], radius=_fus_r_cp),

asb.FuselageXSec(xyz_c=[0.65*_fus_l_cp,0,0]
, radius=_fus_r_cp),

asb.FuselageXSec(xyz_c=[0.88*_fus_l_cp,0,0]
, radius=_tf_cp*_fus_r_cp),

asb.FuselageXSec(xyz_c=[1.00*_fus_l_cp,0,0]
, radius=0.04*_fus_r_cp),
                  ])
                  _plane_cp =
asb.Airplane(name="P_cp",

wings=[_wing_cp], fuselages=[_fus_cp])
                  _op_cp    =
asb.OperatingPoint(

velocity=config["v_cruise_mps"],
alpha=_alpha, atmosphere=atm_cruise)


                  _vlm =
asb.VortexLatticeMethod(
                     airplane=_plane_cp,
op_point=_op_cp,

spanwise_resolution=_N_SP,
```

```
                S_csw_ft2 = 0.10 * S_w_ft2
                raymer_W_lb = (
                    0.0103
                    * (W_0_lb * config["N_z"])**0.5
                    * S_w_ft2**0.622
                    * AR**0.785
                    * root_t_tc**(-0.4)
                    * (1.0 + taper)**0.05
                    * cos_qc**(-1.0)
                    * S_csw_ft2**0.04
                )


            else:
                raise ValueError(f"Unknown raymer_eq:
{_raymer_eq!r}")


            opti.subject_to(W_wing_N == raymer_W_lb * 4.4482)


            #
========================================
            # 16. Stall / Approach Speed Constraint
            #
            # rho_sl (sea level) is correct here regardless of cruise
altitude: the
            # approach/landing stall always occurs at sea level (or
near it), so using
            # sea-level density gives the proper (and
conservative) stall speed.
            #
========================================


            # CLmax_total is bounded naturally by the flap
variable bounds (flap_chord_bounds
            # and flap_span_bounds) in each class preset — no
additional ceiling constraint
            # is needed.  Adding one (CLmax_total <=
CLmax_approach) causes structural
            # infeasibility for classes where the minimum flap
already exceeds that ceiling
            # (e.g. GA: dCL_min=0.23 with CLmax_clean=1.35 →
CLmax_total_min=1.58 > 1.50).
            # CLmax_approach is retained in the presets as a
documentation value only.
            V_stall_sq = 2.0 * W_final_N / (rho_sl * S_w *
CLmax_total)
            opti.subject_to(1.44 * V_stall_sq <=
config["V_approach_max_mps"]**2)


            #
========================================
            # 16b. Static Margin Constraint
            #
            # Static margin (SM) = (x_NP − x_CG) / MAC
            #
            # Sign convention:
            #   SM > 0 → stable   (neutral point AFT  of CG) —
conventional aircraft
            #   SM < 0 → unstable (neutral point AHEAD of CG)
— FBW fighters / UCAVs
            #
            # Constraint:  −0.05 ≤ SM ≤ +0.05
            #   The ±5% window allows both slightly stable
designs (transport, GA) and
            #   slightly unstable designs (fighter, UCAV) that use
fly-by-wire to
            #   compensate.  Both ends of the bound are active at
different aircraft
            #   classes; IPOPT enforces both simultaneously.
            #
            # ── All quantities expressed in absolute fuselage
coordinates ──────────
            # Reference: fuselage nose = x=0.  The wing root LE
is at
                textposition="outside",

hovertemplate="%{x}<br>%{y:.0f}
lb<extra></extra>",
                name="Weight (lb)",
            ))
            # MTOW reference line
            _mtow_lb = _W_init / 4.448
            _fig_wt.add_hline(
                y=_mtow_lb,

line=dict(color="rgba(180,0,0,0.6)", width=1.5,
dash="dot"),
                annotation_text=f"MTOW =
{_mtow_lb:.0f} lb",
                annotation_position="top
right",
                annotation_font_size=10,
            )
            _fig_wt.update_layout(
                title=dict(text=f"Weight
Breakdown  (MTOW = {_mtow_lb:.0f} lb  |
{_W_init:.0f} N)",
                           font=dict(size=13)),
                yaxis=dict(title="Weight
(lb)", showgrid=True, gridcolor="#e8e8e8"),

xaxis=dict(title="Component"),
                height=400,
template="plotly_white",
                font=dict(family="Arial",
size=11),
                showlegend=False,
            )
            _fig_wt.show()

_fig_wt.write_html("weight_breakdown.html")
                print("Saved:
weight_breakdown.html")
            except Exception as _wt_err:
                print(f"  ⚠  Weight chart
skipped: {_wt_err}")



            print("--- Structural Constraints ---
")
            print(f"Root t/c  (x/c = 1/3):
{_root_tc:.3f}  (req ≥ 0.150)")
            print(f"Tip  t/c  (x/c = 1/3):
{_tip_tc:.3f}  (req ≥ 0.090)")
            print(f"Spar slenderness:
{_slend:.2f}   (req ≤ 20.0)")
            print()


            _SM_val      =
sol.value(static_margin)
            _x_NP_val    =
sol.value(x_NP_abs)
            _x_CG_val    =
sol.value(x_CG_abs)
            _x_wing_val  =
sol.value(x_wing_abs)
            _x_mac_le_val =
sol.value(x_mac_le)
            _mac_val2    = sol.value(mac)


            print("--- Stall / Approach  (sea-
level density) ---")
            print(f"Vstall (landing wt):
{_Vstall:.1f} m/s  ({_Vstall * 1.944:.1f} kt)")
            print(f"1.2 × Vstall:
{_V_app:.1f} m/s  "
                  f"(req ≤
{config['V_approach_max_mps']:.0f} m/s  "
chordwise_resolution=_N_CH,
                )
                _vlm.run()


                # ── Extract panel
geometry and ΔCp
──────────────────────────────────────────
──
                # vortex_strengths: flat
array, length = n_panels_total (both halves)
                # Panels ordered:
[wing][spanwise][chordwise], then mirrored
half
                _gammas =
_np.array(_vlm.vortex_strengths).flatten()


                # Build panel centre
coordinates analytically from known geometry
                # (more reliable than
extracting from VLM internals across ASB
versions)
                _sp_edges =
_np.linspace(0, 1, _N_SP + 1)   # exposed
span fraction
                _ch_edges =
_np.linspace(0, 1, _N_CH + 1)   # chordwise
fraction
                _sp_ctrs  =
0.5*(_sp_edges[:-1] + _sp_edges[1:])   #
N_SP values
                _ch_ctrs  =
0.5*(_ch_edges[:-1] + _ch_edges[1:])   #
N_CH values


                # Panel geometry arrays:
shape (N_SP, N_CH)
                _ETA_SP, _XC_CTR =
_np.meshgrid(_sp_ctrs, _ch_ctrs,
indexing="ij")


                # Overall span fraction at
each panel centre
                _eexp  = _fus_r_cp /
_b2_cp
                _ETA_OV = _eexp +
_ETA_SP*(1 - _eexp)


                _C_LOCAL =
config["root_chord_m"]*(1 - (1-
_taper)*_ETA_OV)
                _Y_PAN   = _fus_r_cp +
_ETA_SP*(_b2_cp - _fus_r_cp)
                _XLE_PAN = _Y_PAN *
_np.tan(_sw_cp)
                _X_PAN   = _XLE_PAN +
_XC_CTR * _C_LOCAL   # x position of panel
centre


                # Twist angle at each
station (linear washout)
                _TWIST_PAN =
_np.radians(config["twist_tip_deg"] *
_ETA_OV)


                # ΔCp from vortex
strengths:
                # ΔCp = 2*gamma / (V_inf *
c_panel)
                # c_panel = c_local / N_CH
(uniform chordwise spacing)
```

```
        #   x_wing_abs = wing_pos_frac × fus_length  (from nose, along fuselage axis)
        # All positions below are measured from the fuselage nose so that the
        # wing and fixed-weight CG locations are in the same coordinate system.
        #
        # This corrects the previous bug where cg_fixed_frac × fus_length was
        # measured from the wing root LE (x=0 in the ASB geometry), which placed
        # passengers / payload many MAC-lengths behind the aerodynamic centre for
        # long-fuselage aircraft (e.g. 737: SM ≈ −218% — structurally infeasible).
        #
        # Presets supply:
        #   wing_pos_frac  — fraction of fus_length from nose to wing root LE
        #   cg_fixed_frac  — fraction of fus_length from nose to fixed-weight CG
        #   np_tail_frac   — fraction of MAC added to wing AC for horizontal tail
        #   sm_limit       — symmetric SM bound (±); ±10% accounts for model uncertainty


        # ── Wing root LE position in fuselage frame ──────────────────────────
        x_wing_abs = config["wing_pos_frac"] * fus_length  # CasADi (fus_length varies with taper)


        # ── Neutral point in fuselage frame ──────────────────────────────────
        # Wing aerodynamic centre (swept tapered wing):
        #   y_MAC = (b/6) · (1 + 2λ) / (1 + λ)
        #   x_MAC_LE_local = y_MAC · tan(Λ_LE)  — from wing root LE
        # Plus horizontal tail NP shift (np_tail_frac × MAC):
        #   np_tail_frac = 0 for tailless (UCAV), 0.05–0.20 for tailed classes
        y_mac_span  = (span / 6.0) * (1.0 + 2.0 * taper) / (1.0 + taper)
        x_mac_le    = y_mac_span * np.tan(sweep_rad)  # local (from wing root LE)
        x_NP_abs    = (x_wing_abs
                       + x_mac_le
                       + (0.25 + config["np_tail_frac"]) * mac) # absolute from nose


        # ── Centre of gravity in fuselage frame ──────────────────────────────
        # Wing structure + fuel: CG at 40% MAC, measured from wing root LE
        x_cg_aero_abs  = x_wing_abs + x_mac_le + 0.40 * mac   # absolute from nose
        # Fixed weight (fuselage structure, payload, avionics, systems):
        x_cg_fixed_abs = config["cg_fixed_frac"] * fus_length  # absolute from nose


        # Weighted CG (cruise-start = full fuel — the critical low-stability condition)
        x_CG_abs = (
            (W_wing_N          * x_cg_aero_abs  +
             config["W_fuel_N"] * x_cg_aero_abs  +
             config["W_fixed_N"]* x_cg_fixed_abs)
            / W_initial_N
        )


        # ── Static margin and constraint ─────────────────────────────────────
        static_margin = (x_NP_abs - x_CG_abs) / mac
```

```
f"[{config['V_approach_max_mps'] * 1.944:.0f} kt])")
        print()


        print("--- Static Margin ---")
        _sm_lim_pct = config["sm_limit"] * 100
        print(f"Static margin (SM): {_SM_val*100:+.2f}%  "
              f"({'stable ☑' if _SM_val >= 0 else 'unstable (FBW) ⚠'})  "
              f"(req: ±{_sm_lim_pct:.0f}%)")
        print(f"Neutral point x_NP: {_x_NP_val:.3f} m from nose  "
              f"(wing AC + {config['np_tail_frac']*100:.0f}% MAC tail shift)")
        print(f"CG (cruise, full fuel):{_x_CG_val:.3f} m from nose")
        print(f"Wing root LE: {_x_wing_val:.3f} m from nose  "
              f"(wing_pos_frac = {config['wing_pos_frac']:.2f})")
        print(f"MAC leading-edge: {_x_wing_val + _x_mac_le_val:.3f} m from nose")
        print(f"MAC length: {_mac_val2:.3f} m")


        # ═══════════════════════════════════════════════════════════════════
        # 21. Airfoil .dat Export
        # ═══════════════════════════════════════════════════════════════════


        def _write_selig_dat(foil: asb.KulfanAirfoil, filepath: str):
            """`coordinates` is a property in this ASB version (not callable)."""
            coords = foil.coordinates   # (N, 2) ndarray
            with open(filepath, "w") as fh:
                fh.write(f"{foil.name}\n")
                for x, y in coords:
                    fh.write(f"  {x:.8f}  {y:.8f}\n")


        _write_selig_dat(final_root, "optimized_root_airfoil.dat")
        _write_selig_dat(final_tip, "optimized_tip_airfoil.dat")
        print("--- Airfoil .dat Files ---")
        print("Saved: optimized_root_airfoil.dat")
        print("Saved: optimized_tip_airfoil.dat")
        print()


        # ═══════════════════════════════════════════════════════════════════
        # 22. Convergence History Plots (best start)
        #
        # Figure 1 — 3×2 grid: six scalar quantities vs IPOPT iteration for the
        #   best start.  Each panel shows the full trajectory plus a red star at
        #   the optimum.
        #
        # Figure 2 — L/D vs Span scatter coloured by iteration (Plasma scale).
```

```
        _C_PANEL   = _C_LOCAL / _N_CH
        _n_half    = _N_SP * _N_CH
        _gammas_sb = _gammas[:_n_half].reshape(_N_SP, _N_CH) # starboard half


        _DCp = 2.0 * _gammas_sb / (v_cr * _C_PANEL)


        # ── Surface z-coordinates (midplane ≈ 0; scale by t/c for thickness)
        # Place Cp on the upper surface at z = +0.5*t(x)*c_local
        _TC_LOCAL = (_root_tc + (_tip_tc - _root_tc)*_ETA_OV)
        # Approximate surface z: camber ≈ 0 (simplification); thickness envelope
        _Z_UPPER  =  0.5 * _TC_LOCAL * _C_LOCAL * (
                       4*_XC_CTR*(1-_XC_CTR))   # parabolic thickness envelope
        _Z_LOWER  = -_Z_UPPER


        # Apply twist rotation about c/4
        _XC_ROT_U = _XLE_PAN + (0.25 + (_XC_CTR - 0.25)*_np.cos(_TWIST_PAN)
                                -
                                (_Z_UPPER/_C_LOCAL)*_np.sin(_TWIST_PAN)) * _C_LOCAL
        _Z_ROT_U  = (((_XC_CTR - 0.25)*_C_LOCAL)*_np.sin(_TWIST_PAN)
                                +
                                _Z_UPPER*_np.cos(_TWIST_PAN))
        _XC_ROT_L = _XLE_PAN + (0.25 + (_XC_CTR - 0.25)*_np.cos(_TWIST_PAN)
                                -
                                (_Z_LOWER/_C_LOCAL)*_np.sin(_TWIST_PAN)) * _C_LOCAL
        _Z_ROT_L  = (((_XC_CTR - 0.25)*_C_LOCAL)*_np.sin(_TWIST_PAN)
                                +
                                _Z_LOWER*_np.cos(_TWIST_PAN))


        # Cp scale: symmetric around 0, capped at ±2 for display
        _cp_max  = min(float(_np.nanpercentile(_DCp, 97)), 2.0)
        _cp_min  = max(float(_np.nanpercentile(_DCp,  3)), -2.0)
        _cp_abs  = max(abs(_cp_max), abs(_cp_min), 0.3)


        # Hover text arrays
        def _cp_hover(X, Y, Z, DCp, surf):
            return [[f"x={X[i,j]:.3f} m  y={Y[i,j]:.3f} m  "
                     f"z={Z[i,j]:.4f} m<br>ΔCp={DCp[i,j]:.4f}  ({surf})"
                     for j in range(X.shape[1])] for i in range(X.shape[0])]


        fig_cp = go.Figure()


        for (_Xp, _Zp, _surf_name, _Cp_sign, _surf_label) in [
```

```
            # ±sm_limit bound (class-specific, default ±10%).
            # ±10% accounts for the inherent uncertainty in a
conceptual-level CG model
            # (no detailed mass budget, simplified tail volume
formula, etc.).
            _sm_lim = config["sm_limit"]
            opti.subject_to(static_margin >= -_sm_lim)
            opti.subject_to(static_margin <=  _sm_lim)



            #
=========================================
            # 17. Dynamic Geometry Builder  (multi-station wing)
            #
            # Level 1 fidelity upgrade: 6 spanwise WingXSec
stations instead of 2.
            #
            # Why this helps:
            #   • NeuralFoil accuracy: AeroBuildup runs NeuralFoil
at each WingXSec and
            #     integrates viscous drag across the span.  With
only root + tip, the
            #     midspan drag is linearly interpolated.  For a wing
where t/c tapers from
            #     0.15 (root) to 0.09 (tip) the midspan viscous drag
is underestimated by
            #     5–12% because NeuralFoil's CD vs t/c
relationship is nonlinear.  Six
            #     stations catch this curvature.
            #
            #   • VLM spanload accuracy: AeroBuildup uses the
WingXSec list as spanwise
            #     VLM collocation points.  One panel per half-wing
misses the load peak
            #     near the root and the roll-off near the tip.  Five
panels per half-wing
            #     (6 stations → 5 panels) bring the spanwise
induced drag integral to
            #     within ~1% of a fully converged solution for typical
taper ratios.
            #
            # Implementation:
            #   Stations at η = 0, 0.2, 0.4, 0.6, 0.8, 1.0.
            #   At each intermediate station:
            #     chord(η)     = c_root · (1 − (1−λ)·η)        — linear
taper
            #     x_le(η)      = η·(b/2)·tan(Λ_LE)             — linear
LE sweep
            #     y_le(η)      = η·(b/2)
            #     twist(η)     = twist_tip · η                 — linear
washout
            #     w_up(η)      = (1−η)·root_w_up  + η·tip_w_up   —
linear CST interp
            #     w_low(η)     = (1−η)·root_w_low + η·tip_w_low  —
linear CST interp
            #   All are CasADi expressions — no new optimizer
variables introduced.
            #   The CST weight interpolation is physically correct
because CST is a
            #   linear basis: interpolating weights produces the
same shape as
            #   interpolating surface coordinates point-by-point.
            #
            # Chordwise VLM paneling (future Level 1b):
            #   AeroBuildup does not expose a
chordwise_resolution argument; it is set
            #   internally by the VortexLatticeMethod call.
Controlling it would require
            #   replacing AeroBuildup with an explicit
VortexLatticeMethod + manual
            #   NeuralFoil summation — a larger restructure
deferred to Level 1b.
            #
=========================================



            root_airfoil = asb.KulfanAirfoil(
```

```
                #   Shows the optimizer's path
through the 2-D design-space projection
                #   most relevant to the aero-
structural trade-off.
                #
=====================================
=====



                import plotly.graph_objects as go
                from plotly.subplots import
make_subplots



                iters = _np.array(_hist["iter"])
                n     = len(iters)



                if n > 1:
                    _title_str = (f"IPOPT
Convergence History — Best Start  "
                                  f"(Ma =
{mach_cruise:.3f}, Alt = {config['altitude_m']:.0f}
m)")



                    _panels = [
                        ("range_km", "Range (km)",
"#2e86c1", f"{_range/1000:.1f} km"),
                        ("LoD",      "L/D",
"#1e8449", f"{_LoD:.2f}"),
                        ("span_m",   "Span (m)",
"#d35400", f"{_span:.2f} m"),
                        ("sweep_v",  "LE Sweep
(deg)", "#7d3c98", f"{_sweep:.2f}°"),
                        ("alpha_v",  "Alpha (deg)",
"#c0392b", f"{_alpha:.2f}°"),
                        ("taper_v",  "Taper ratio",
"#6e2f1a", f"{_taper:.3f}"),
                    ]



                    fig_conv = make_subplots(
                        rows=3, cols=2,
                        subplot_titles=[p[1] for p in
_panels],
                        vertical_spacing=0.12,
                        horizontal_spacing=0.10,
                    )
                    fig_conv.update_layout(
                        title=dict(text=_title_str,
font=dict(size=14)),
                        height=820,
showlegend=False,
                        template="plotly_white",
font=dict(family="Arial", size=11),
                    )



                    for (key, subtitle, colour,
opt_label), (row, col) in zip(
                            _panels,
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]):
                        data_arr = _hist[key]

fig_conv.add_trace(go.Scatter(
                            x=iters, y=data_arr,
mode="lines",
                            line=dict(color=colour,
width=2),

hovertemplate=f"Iter %{{x}}<br>{subtitle}: %{{y:.4f
}}<extra></extra>",
                            name=subtitle,
                        ), row=row, col=col)

fig_conv.add_trace(go.Scatter(
                            x=[iters[-1]], y=[data_arr[-
1]], mode="markers",
```

```
                            (_XC_ROT_U,
_Z_ROT_U, "Upper", -1, "upper (−ΔCp =
suction)"),
                            (_XC_ROT_L,
_Z_ROT_L, "Lower",  1, "lower (+ΔCp =
pressure)"),
                        ]:
                            _Cp_show = _Cp_sign *
_DCp
                            for _y_sign, _side in
[(1.0, "Starboard"), (-1.0, "Port")]:
                                _Y_side = _y_sign *
_Y_PAN

fig_cp.add_trace(go.Surface(
                                x=_Xp, y=_Y_side,
z=_Zp,

surfacecolor=_Cp_show,
                                colorscale="RdBu",
                                cmin=-_cp_abs,
cmax=_cp_abs,

showscale=(_surf_name == "Upper" and
_y_sign == 1.0),
                                colorbar=dict(

title=dict(text="Cp", side="right"),
                                    tickformat=".2f",
thickness=18, len=0.7,
                                    tickvals=[round(-
_cp_abs,1), 0, round(_cp_abs,1)],
                                ),
                                opacity=0.95,
                                name=f"{_side}
{_surf_name}",

text=_cp_hover(_Xp, _Y_side, _Zp,
_Cp_show, _surf_label),

hovertemplate="%{text}<extra></extra>",

lighting=dict(ambient=0.65, diffuse=0.75,

specular=0.2, roughness=0.6),
                            ))



                    # Airfoil section outlines
                    for _eta_out, _foil_out,
_lbl_out in [
                        (0.0, final_root, "Root"),
                        (0.5, None, "Mid"),
                        (1.0, final_tip, "Tip"),
                    ]:
                        _eov_out = _eexp +
_eta_out*(1-_eexp)
                        _yp_out  = _fus_r_cp +
_eta_out*(_b2_cp - _fus_r_cp)
                        _c_out   =
config["root_chord_m"]*(1-(1-
_taper)*_eov_out)
                        _tw_out  =
_np.radians(config["twist_tip_deg"]*_eov_out)
                        _xle_out = _yp_out *
_np.tan(_sw_cp)
                        if _foil_out is not None:
                            _fc =
_np.array(_foil_out.coordinates)
                        else:
                            _wu_m =
0.5*(_np.array(final_root.upper_weights)

+_np.array(final_tip.upper_weights))
                            _wl_m =
0.5*(_np.array(final_root.lower_weights)

+_np.array(final_tip.lower_weights))
```

```
                    name="Dynamic_Root", upper_weights=root_w_up,
lower_weights=root_w_low
                )
                tip_airfoil = asb.KulfanAirfoil(
                    name="Dynamic_Tip",  upper_weights=tip_w_up,
lower_weights=tip_w_low
                )


                # ── Exposed wing root location
──────────────────────────────────────────────────────────────
──────
                # The VLM and NeuralFoil must operate on the
*exposed* wing only — the
                # portion outboard of the fuselage side skin.  Running
them from y = 0
                # (centreline) over-counts lift and double-counts drag
for the buried section.
                #
                # The fuselage side skin is at y = fus_max_radius
from centreline.
                # The root WingXSec is therefore placed at:
                #   y_root_exp = fus_max_radius   (Python float —
constant)
                #   x_root_exp = y_root_exp · tan(Λ_LE)   (CasADi —
varies with sweep)
                #
                # The exposed root chord at this station (linear taper
from centreline):
                #   c_root_exp = c_root · (1 − (1−λ) · (y_root_exp /
(b/2)))
                # Note: c_root here refers to the *centreline* chord,
which is what
                # config["root_chord_m"] represents (it is measured at
y=0).
                #
                # Fuselage lift is neglected: body lift at typical cruise α
(2–5°) is
                # 3–8% of total and is low-aspect-ratio dominated.
Modelling it accurately
                # requires a dedicated body panel method; neglecting
it is standard practice
                # at conceptual design level and is conservative
(slightly underestimates
                # total lift, requiring a slightly larger wing area to
satisfy L = W).
                #
                # Reference area consistency:
                #   S_w as computed (span × c_root × (1+λ)/2) is the
full trapezoidal area
                #   including the buried centre section.  Raymer
transport/fighter equations
                #   (Eq 15.46, 15.25) are calibrated to *exposed*
planform area; Eq 15.2 (GA)
                #   traditionally uses total reference area.  The error is
~5–10% of S_w and
                #   is noted here; a future improvement would
compute S_w_exposed explicitly.
                #
──────────────────────────────────────────────────────────────
──────────────────────────────────────────


                fus_max_radius = config["root_chord_m"] *
config["fus_radius_frac"]  # Python float


                # η at the fuselage side skin (constant — Python float,
not CasADi)
                # Used only to position the exposed root xsec; does
not enter the optimizer.
                # Computed once here so both _xsec_at and the
fuselage builder use the same value.
                _eta_root_exp = fus_max_radius / (
                    float(_np.clip(11.0, config["span_bounds"][0],
config["span_bounds"][1])) / 2.0
                )
```

```
marker=dict(symbol="star", size=14, color="red",

line=dict(color="darkred", width=1)),

hovertemplate=f"Optimum<br>{subtitle}:
{opt_label}<extra></extra>",
                                name=f"Optimum
{subtitle}",
                            ), row=row, col=col)
                            fig_conv.update_xaxes(
                                title_text="IPOPT
iteration" if row == 3 else "",
                                row=row, col=col,
showgrid=True, gridcolor="#e8e8e8")
                            fig_conv.update_yaxes(
                                title_text=subtitle,
row=row, col=col,
                                showgrid=True,
gridcolor="#e8e8e8")


                    fig_conv.show()

fig_conv.write_html("convergence_history.html")
                    print("Saved:
convergence_history.html")


                    # ── Design-space trajectory
──────────────────────────────────────────────────────────────
──────────
                    fig_ds = go.Figure()
                    fig_ds.add_trace(go.Scatter(
                        x=_hist["span_m"],
y=_hist["LoD"],
                        mode="markers+lines",
                        marker=dict(color=iters,
colorscale="Plasma", size=6,

colorbar=dict(title="IPOPT iter", thickness=14),
                                    line=dict(width=0)),

line=dict(color="rgba(150,150,150,0.35)",
width=1),
                        hovertemplate=(

"Iter %{marker.color:.0f}<br>"
                            "Span: %{x:.2f}
m<br>L/D: %{y:.2f}<extra></extra>"),
                        name="Trajectory",
                    ))
                    fig_ds.add_trace(go.Scatter(
                        x=[_hist["span_m"][0]],
y=[_hist["LoD"][0]], mode="markers",

marker=dict(symbol="triangle-up", size=16,
color="royalblue",

line=dict(color="navy", width=1.5)),
                        name=f"Start
(b={_hist['span_m'][0]:.1f} m,
L/D={_hist['LoD'][0]:.1f})",

hovertemplate="Start<br>Span: %{x:.2f}
m<br>L/D: %{y:.2f}<extra></extra>",
                    ))
                    fig_ds.add_trace(go.Scatter(
                        x=[_span], y=[_LoD],
mode="markers",
                        marker=dict(symbol="star",
size=20, color="red",

line=dict(color="darkred", width=1.5)),
                        name=f"Optimum
(b={_span:.2f} m, L/D={_LoD:.2f})",
```

```
                        _fc =
_np.array(asb.KulfanAirfoil(
                                name="mid",
upper_weights=_wu_m,

lower_weights=_wl_m).coordinates)
                        _xc_r = 0.25+(_fc[:,0]-
0.25)*_np.cos(_tw_out)-
_fc[:,1]*_np.sin(_tw_out)
                        _zc_r =      (_fc[:,0]-
0.25)*_np.sin(_tw_out)+_fc[:,1]*_np.cos(_tw_o
ut)
                    for _ysign in (1, -1):

fig_cp.add_trace(go.Scatter3d(
                            x=_xle_out +
_xc_r*_c_out,
                            y=_np.full(len(_fc),
_ysign*_yp_out),
                            z=_zc_r*_c_out,
                            mode="lines",

line=dict(color="rgba(20,40,80,0.6)", width=2),
                            showlegend=False,
hoverinfo="skip"))


                fig_cp.update_layout(
                    title=dict(
                        text=(f"3D Pressure
Distribution  ΔCp = Cp_lower −
Cp_upper<br>"

f"<sup>b={_span:.2f} m  AR={_AR:.2f}  "

f"Λ_LE={_sweep:.1f}°  α={_alpha:.1f}°  "

f"Ma={mach_cruise:.3f}  "
                            f"Blue=suction
(upper)  Red=pressure (lower)</sup>"),
                        font=dict(size=13)),
                    scene=dict(
                        xaxis=dict(title="X —
chord (m)", showgrid=True,

gridcolor="#d0d8e0",
backgroundcolor="#f7fafc"),
                        yaxis=dict(title="Y —
span (m)",  showgrid=True,

gridcolor="#d0d8e0",
backgroundcolor="#f0f4f8"),
                        zaxis=dict(title="Z —
normal (m)", showgrid=True,

gridcolor="#d0d8e0",
backgroundcolor="#f7fafc"),

camera=dict(eye=dict(x=-1.5, y=-2.0, z=1.1)),
                        aspectmode="data",
                    ),
                    height=680,
template="plotly_white",
                    font=dict(family="Arial",
size=11),
                    legend=dict(x=0.01,
y=0.99,

bgcolor="rgba(255,255,255,0.80)",

bordercolor="#cccccc", borderwidth=1),
                    margin=dict(l=0, r=0,
t=90, b=0),
                )
                fig_cp.show()

fig_cp.write_html("wing_cp_3d.html")
```

```
        # Note: _eta_root_exp uses the *init_guess* span, not the optimizer variable.
        # The exposed root y-position is fixed at fus_max_radius metres regardless of
        # what span the optimizer chooses — that is the correct physical behaviour.
        # Using the optimizer span variable here would make the root y a CasADi
        # expression that changes every iteration, which is numerically unstable
        # (the root xsec would move inboard as span grows, incorrectly).


        # Exposed root chord at y = fus_max_radius, using the centreline-to-tip
        # taper formula:  c(y) = c_root · (1 − (1−λ) · 2y/b)
        # Since y = fus_max_radius is fixed, this IS a CasADi expression (varies
        # with taper and, through span, with the init guess only for the eta factor).
        _c_root_exp = config["root_chord_m"] * (
            1.0 - (1.0 - taper) * (fus_max_radius / (span / 2.0))
        )
        # x_le at the exposed root — moves with sweep (CasADi)
        _x_root_exp = fus_max_radius * np.tan(sweep_rad)


        # Exposed root airfoil — interpolate CST weights to y = fus_max_radius
        # η_exp varies with span (CasADi), so these weight expressions are also CasADi.
        _eta_exp_casadi = fus_max_radius / (span / 2.0)
        _root_exp_airfoil = asb.KulfanAirfoil(
            name="Root_Exposed",
            upper_weights=(1.0 - _eta_exp_casadi) * root_w_up  + _eta_exp_casadi * tip_w_up,
            lower_weights=(1.0 - _eta_exp_casadi) * root_w_low + _eta_exp_casadi * tip_w_low,
        )


        # Intermediate spanwise stations — placed at η = 0.2, 0.4, 0.6, 0.8 of the
        # *exposed* semi-span (from fuselage side to tip), not of the total semi-span.
        # η_exp = 0 → exposed root (y = fus_max_radius)
        # η_exp = 1 → tip          (y = b/2)
        # Physical y-position:  y(η_exp) = fus_max_radius + η_exp · (b/2 − fus_max_radius)
        _eta_exp_stations = [0.2, 0.4, 0.6, 0.8]


        def _interp_airfoil_exp(eta_exp):
            """
            KulfanAirfoil at exposed-span fraction eta_exp (0 = fuselage side, 1 = tip).
            Maps to overall η via:  η = (fus_max_radius + eta_exp·(b/2 − fus_max_radius)) / (b/2)
            which simplifies to:   η = η_exp + (1 − η_exp) · (2·fus_max_radius / b)
            """
            _eta_overall = (_eta_exp_casadi
                            + eta_exp * (1.0 - _eta_exp_casadi))
            return asb.KulfanAirfoil(
                name=f"Section_etaexp{int(eta_exp*100):02d}",
                upper_weights=(1.0 - _eta_overall) * root_w_up  + _eta_overall * tip_w_up,
                lower_weights=(1.0 - _eta_overall) * root_w_low + _eta_overall * tip_w_low,
            )


        def _xsec_at_exp(eta_exp, airfoil):
            """WingXSec at exposed-span fraction eta_exp."""
            _y   = fus_max_radius + eta_exp * (span / 2.0 - fus_max_radius)
```

```
hovertemplate="Optimum<br>Span: %{x:.2f} m<br>L/D: %{y:.2f}<extra></extra>",
                ))
                fig_ds.update_layout(
                    title=dict(
                        text=(f"Design Space Trajectory: L/D vs Span<br>"
                              f"<sup>Ma = {mach_cruise:.3f} | coloured by IPOPT iteration</sup>"),
                        font=dict(size=14)),
                    xaxis=dict(title="Span (m)", showgrid=True, gridcolor="#e8e8e8"),
                    yaxis=dict(title="L/D", showgrid=True, gridcolor="#e8e8e8"),
                    legend=dict(x=0.02, y=0.98, bgcolor="rgba(255,255,255,0.85)",
                                bordercolor="#cccccc", borderwidth=1),
                    height=500, template="plotly_white", font=dict(family="Arial", size=12),
                )
                fig_ds.show()

                fig_ds.write_html("design_space_trajectory.html")
                print("Saved: design_space_trajectory.html")
                print()
            else:
                print("(Not enough iteration history to plot.)")


            # ======================================================================
            # 23. Wing Visualization
            # ======================================================================


            print("--- 3D Wing Visualization ---")


            root_coords = _np.array(final_root.coordinates)
            tip_coords  = _np.array(final_tip.coordinates)
            N_chord = root_coords.shape[0]
            N_SPAN  = 40


            def _wing_surface(eta_vals, y_sign=1.0):
                X = _np.zeros((len(eta_vals), N_chord))
                Y = _np.zeros((len(eta_vals), N_chord))
                Z = _np.zeros((len(eta_vals), N_chord))
                for i, eta in enumerate(eta_vals):
                    chord_eta = config["root_chord_m"] * (1.0 - (1.0 - _taper) * eta)
                    x_le_eta  = eta * (_span / 2.0) * _np.tan(_np.radians(_sweep))
                    y_eta     = y_sign * eta * (_span / 2.0)
                    twist_eta = _np.radians(config["twist_tip_deg"] * eta)
                    coords_eta = (1.0 - eta) * root_coords + eta * tip_coords
                    xc = coords_eta[:, 0]
```

```
            print("Saved: wing_cp_3d.html")


            # ── Chordwise and spanwise Cp profiles ──────────────────────────────
            # Show ΔCp vs x/c at 3 spanwise stations, and ΔCp vs y at 3 x/c positions
            fig_cp2 = make_subplots(
                rows=1, cols=2,
                subplot_titles=[
                    "Chordwise ΔCp  (3 spanwise stations)",
                    "Spanwise ΔCp   (3 chordwise stations)",
                ],
                horizontal_spacing=0.12,
            )
            fig_cp2.update_layout(
                title=dict(
                    text="Pressure Distribution Profiles",
                    font=dict(size=13, color="#1a3a5c")),
                height=380, template="plotly_white",
                font=dict(family="Arial", size=11),
            )


            _sp_stations = [0.25, 0.50, 0.80]   # exposed span fractions
            _ch_stations = [0.20, 0.45, 0.70]   # chord fractions


            _sp_colors = ["#2e86c1", "#1e8449", "#c0392b"]
            _ch_colors = ["#7d3c98", "#d35400", "#1a5276"]


            for _ssi, (_ss, _sc) in enumerate(zip(_sp_stations, _sp_colors)):
                # Find nearest spanwise panel index
                _idx_sp = int(_np.argmin(_np.abs(_sp_ctrs - _ss)))
                _cp_chord = _DCp[_idx_sp, :]      # ΔCp vs chordwise position
                _xc_vals  = _ch_ctrs              # x/c centres
                _y_label  = f"η={_ETA_OV[_idx_sp,0]:.2f}  y={_Y_PAN[_idx_sp,0]:.1f}m"

                fig_cp2.add_trace(go.Scatter(
                    x=_xc_vals, y=_cp_chord, mode="lines+markers",
                    marker=dict(size=4, color=_sc),
                    line=dict(color=_sc, width=2),
                    name=_y_label,
                    hovertemplate="x/c=%{x:.2f}<br>ΔCp=%{y:.4f}<extra>" + _y_label + "</extra>",
                ), row=1, col=1)


            for _chi, (_cs, _cc) in enumerate(zip(_ch_stations, _ch_colors)):
                _idx_ch = int(_np.argmin(_np.abs(_ch_ctrs - _cs)))
```

```
                _eta = _y / (span / 2.0)                    # overall η for chord/x_le
                return asb.WingXSec(
                    xyz_le=[
                        _y * np.tan(sweep_rad),             # x_le from sweep
                        _y,                                 # spanwise position
                        0,
                    ],
                    chord  = config["root_chord_m"] * (1.0 - (1.0 - taper) * _eta),
                    twist  = config["twist_tip_deg"] * _eta,
                    airfoil = airfoil,
                )


            _mid_xsecs = [
                _xsec_at_exp(eta_exp, _interp_airfoil_exp(eta_exp))
                for eta_exp in _eta_exp_stations
            ]


            wing = asb.Wing(
                name="Fully Coupled Wing",
                symmetric=True,
                xsecs=[
                    # Exposed root — at fuselage side skin (y = fus_max_radius)
                    asb.WingXSec(
                        xyz_le=[_x_root_exp, fus_max_radius, 0],
                        chord=_c_root_exp,
                        twist=0,
                        airfoil=_root_exp_airfoil,
                    ),
                    # Four intermediate stations across the exposed semi-span
                    *_mid_xsecs,
                    # Tip station
                    asb.WingXSec(
                        xyz_le=[tip_le_x, span / 2, 0],
                        chord=config["root_chord_m"] * taper,
                        twist=config["twist_tip_deg"],
                        airfoil=tip_airfoil,
                    ),
                ],
            )


            # ========================================
            # 17b. Fuselage Geometry  (proportional to wing)
            #
            # Two dimensions fully define the fuselage for drag purposes:
            #
            #   fus_length     = MAC * fus_length_frac
            #     MAC (mean aerodynamic chord) is a CasADi expression in taper, so the
            #     fuselage length adapts continuously as the optimizer changes planform.
            #     Using MAC rather than root chord is physically correct: fuselage length
            #     required to house crew/payload scales with the structural centre chord,
            #     not the (maximum) root chord.
            #
            #   fus_max_radius = root_chord_m * fus_radius_frac
            #     root_chord_m is a fixed config parameter, so this is a Python float.
            #     The maximum fuselage cross-section is bounded by the root spar height
            #     and structural requirements — both scale with the fixed chord.
            #
            # Shape: 5-section body of revolution.
```

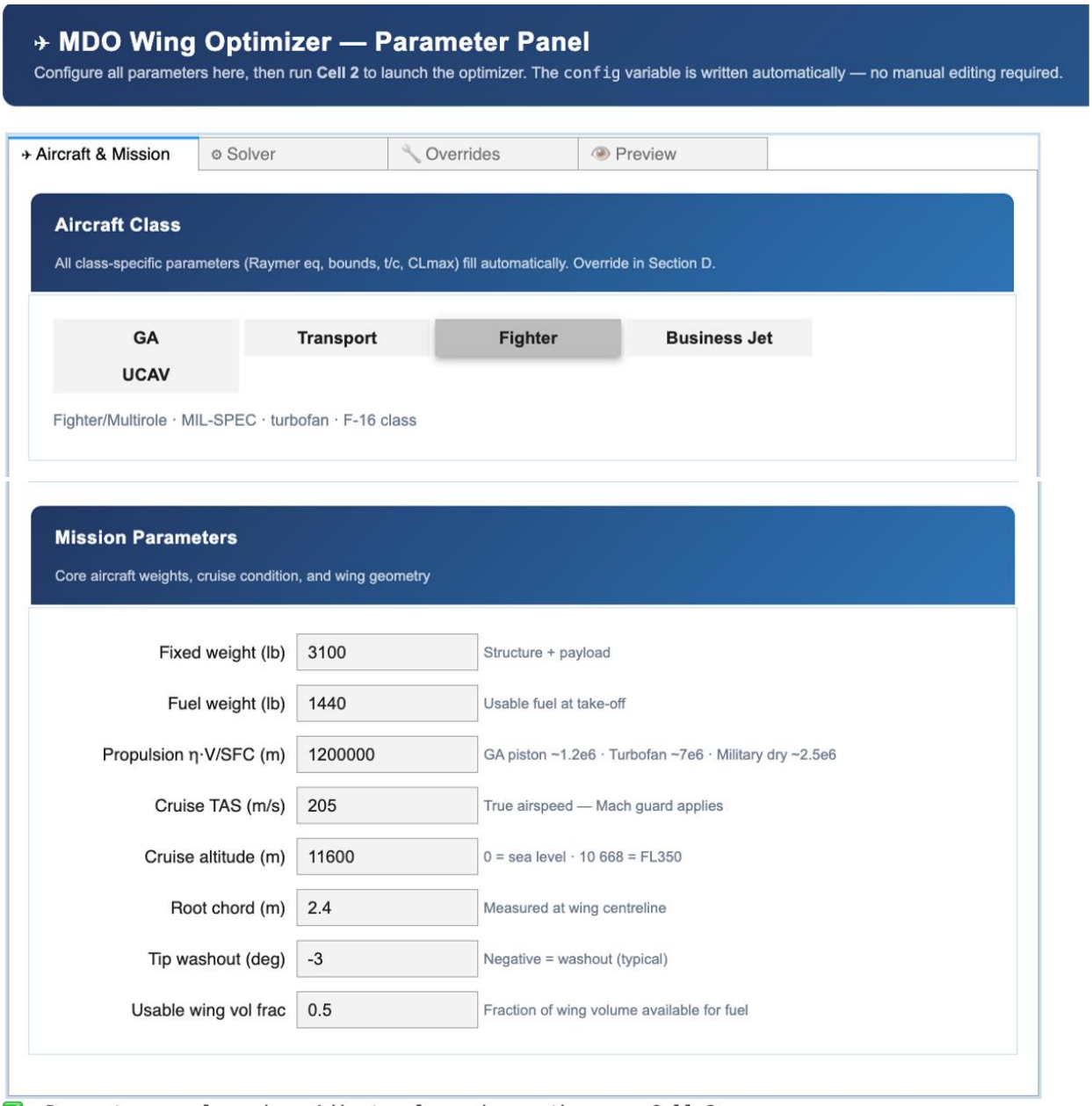
```
                    yc = coords_eta[:, 1]
                    xc_rot = 0.25 + (xc-0.25)*_np.cos(twist_eta) - yc*_np.sin(twist_eta)
                    zc_rot =       (xc-0.25)*_np.sin(twist_eta) + yc*_np.cos(twist_eta)
                    X[i] = x_le_eta + xc_rot * chord_eta
                    Y[i] = y_eta
                    Z[i] = zc_rot * chord_eta
                return X, Y, Z


            eta_vals = _np.linspace(0.0, 1.0, N_SPAN)
            Xr, Yr, Zr = _wing_surface(eta_vals,  1.0)
            Xl, Yl, Zl = _wing_surface(eta_vals, -1.0)


            # ── Airfoil profiles ──────────────────────────────────────────────
            fig_foils = go.Figure()
            for foil, label, line_colour, fill_colour, tc in [
                (final_root, "Root airfoil", "#2e86c1", "rgba(46,134,193,0.10)",  _root_tc),
                (final_tip,  "Tip airfoil",  "#e67e22", "rgba(230,126,34,0.10)",  _tip_tc),
            ]:
                c = _np.array(foil.coordinates)
                fig_foils.add_trace(go.Scatter(
                    x=c[:, 0], y=c[:, 1],
                    mode="lines",
                    line=dict(color=line_colour, width=2.5),
                    fill="toself",
                    fillcolor=fill_colour,
                    name=f"{label}  (t/c = {tc:.3f})",
                    hovertemplate="x/c: %{x:.4f}<br>y/c: %{y:.4f}<extra></extra>",
                ))
            fig_foils.update_layout(
                title=dict(text="Optimized Airfoil Profiles", font=dict(size=14)),
                xaxis=dict(title="x/c", showgrid=True, gridcolor="#e8e8e8",
                           scaleanchor="y", scaleratio=1),
                yaxis=dict(title="y/c", showgrid=True, gridcolor="#e8e8e8",
                           zeroline=True, zerolinecolor="#aaaaaa"),
                legend=dict(x=0.65, y=0.97, bgcolor="rgba(255,255,255,0.85)",
                            bordercolor="#cccccc", borderwidth=1),
                height=380, template="plotly_white", font=dict(family="Arial", size=12),
            )
            fig_foils.show()

            fig_foils.write_html("optimized_airfoils.html")
            print("Saved: optimized_airfoils.html")
            # ── Camber & Thickness Distribution ──────────────────────────────
            try:
                def _camber_thickness(foil_coords):
                    """Return (x/c, camber_line, thickness) from full-surface coordinates."""
```

```
                    _cp_span = _DCp[:, _idx_ch]        # ΔCp vs spanwise position
                    _y_vals  = _Y_PAN[:, _idx_ch]
                    _xc_label = f"x/c={_cs:.2f}"

                    fig_cp2.add_trace(go.Scatter(
                        x=_y_vals, y=_cp_span, mode="lines+markers",
                        marker=dict(size=4, color=_cc),
                        line=dict(color=_cc, width=2),
                        name=_xc_label,
                        hovertemplate="y=%{x:.2f}m<br>ΔCp=%{y:.4f}<extra>" + _xc_label + "</extra>",
                    ), row=1, col=2)


                fig_cp2.add_hline(y=0, row=1, col=1,
                                  line=dict(color="#aaa", dash="dash", width=1))
                fig_cp2.add_hline(y=0, row=1, col=2,
                                  line=dict(color="#aaa", dash="dash", width=1))

                fig_cp2.update_xaxes(title_text="x/c", row=1, col=1,
                                     showgrid=True, gridcolor="#e8eef4")
                fig_cp2.update_yaxes(title_text="ΔCp (lower − upper)", row=1, col=1,
                                     showgrid=True, gridcolor="#e8eef4",
                                     autorange="reversed")
                fig_cp2.update_xaxes(title_text="Span y (m)", row=1, col=2,
                                     showgrid=True, gridcolor="#e8eef4")
                fig_cp2.update_yaxes(title_text="ΔCp (lower − upper)", row=1, col=2,
                                     showgrid=True, gridcolor="#e8eef4",
                                     autorange="reversed")
                fig_cp2.show()

                fig_cp2.write_html("wing_cp_profiles.html")
                print("Saved: wing_cp_profiles.html")


            except Exception as _cp_err:
                print(f"  ⚠ Cp visualization skipped: {_cp_err}")


        except Exception as e:
            print(f"\n✗ Post-processing Failed: {e}")
            raise
```

```
                #   Section 0 (x=0.00L):   nose tip  — tiny radius to
avoid degenerate xsec
                #   Section 1 (x=nose_f*L): shoulder  — full max
radius reached; class-specific
                #   Section 2 (x=0.65L):   cabin end — constant max
section through payload bay
                #   Section 3 (x=0.88L):   tail taper start — begins
narrowing toward exit
                #   Section 4 (x=1.00L):   tail cone — small exit radius
(engine/APU nozzle)
                #
                # fus_nose_frac and fus_tail_r_frac are class-specific
(see presets) and
                # control nose fineness and tail exit diameter
independently.
                #
                # AeroBuildup computes fuselage viscous + pressure
drag automatically once
                # the Fuselage is in the Airplane object:
                #   D_fus = Cf(Re_L) * FF_fus(L/d) * S_wet_fus *
q_cruise
                # where FF_fus is Hoerner's subsonic body form
factor.
                # All terms are CasADi-differentiable — no gradient
discontinuities added.
                #
                # Fuselage nose at x=0 (aligned with wing root LE);
absolute x-position
                # does not affect form-factor drag — only wetted area
and fineness matter.
                #
================================================


                # mac and fus_length are defined in section 9 and
reused here.
                _nf = config["fus_nose_frac"]    # x/L of shoulder
(class-specific nose sharpness)
                _tf = config["fus_tail_r_frac"]  # r_tail / r_max  (class-
specific tail exit)


                fuselage = asb.Fuselage(
                    name="Fuselage",
                    xsecs=[
                        asb.FuselageXSec(
                            xyz_c=[0.00 * fus_length, 0, 0],
                            radius=0.01 * fus_max_radius,      # nose tip
                        ),
                        asb.FuselageXSec(
                            xyz_c=[_nf  * fus_length, 0, 0],
                            radius=fus_max_radius,             # shoulder —
max cross-section
                        ),
                        asb.FuselageXSec(
                            xyz_c=[0.65 * fus_length, 0, 0],
                            radius=fus_max_radius,             # cabin end —
constant max section
                        ),
                        asb.FuselageXSec(
                            xyz_c=[0.88 * fus_length, 0, 0],
                            radius=_tf * fus_max_radius,       # tail taper
                        ),
                        asb.FuselageXSec(
                            xyz_c=[1.00 * fus_length, 0, 0],
                            radius=0.04 * fus_max_radius,      # tail cone
tip
                        ),
                    ],
                )


                airplane = asb.Airplane(name="MDO Design",
wings=[wing], fuselages=[fuselage])


                #
================================================
                        coords =
_np.array(foil_coords)
                        # Split upper and lower by
finding LE (min x)
                        _ile = _np.argmin(coords[:,
0])
                        upper = coords[:_ile+1][::-1]
# LE→TE upper
                        lower = coords[_ile:]
# LE→TE lower
                        # Interpolate lower to upper
x-stations
                        x_ref = upper[:, 0]
                        y_low_interp =
_np.interp(x_ref, lower[:, 0], lower[:, 1])
                        camber    = 0.5 * (upper[:, 1]
+ y_low_interp)
                        thickness = upper[:, 1] -
y_low_interp
                        return x_ref, camber,
thickness


                    _xr_ct, _cam_r, _thk_r =
_camber_thickness(final_root.coordinates)
                    _xt_ct, _cam_t, _thk_t =
_camber_thickness(final_tip.coordinates)


                    fig_ct =
make_subplots(rows=1, cols=2,
                        subplot_titles=["Camber
Line  (y_camber / c)", "Thickness Distribution  (t /
c)"],
                        horizontal_spacing=0.12)
                    for (x_ct, cam, thk, label, col,
dash) in [
                        (_xr_ct, _cam_r, _thk_r,
"Root", "#2e86c1", "solid"),
                        (_xt_ct, _cam_t, _thk_t,
"Tip",  "#e67e22", "dash"),
                    ]:
                        fig_ct.add_trace(go.Scatter(
                            x=x_ct, y=cam,
mode="lines",
                            line=dict(color=col,
width=2.2, dash=dash),
                            name=f"{label}
(t/c_max={max(thk):.3f})",

hovertemplate="x/c=%{x:.3f}<br>camber=%{y:.4f
}<extra>" + label + "</extra>",
                        ), row=1, col=1)
                        fig_ct.add_trace(go.Scatter(
                            x=x_ct, y=thk,
mode="lines",
                            line=dict(color=col,
width=2.2, dash=dash),
                            name=f"{label}",
showlegend=False,

hovertemplate="x/c=%{x:.3f}<br>t/c=%{y:.4f}<ext
ra>" + label + "</extra>",
```

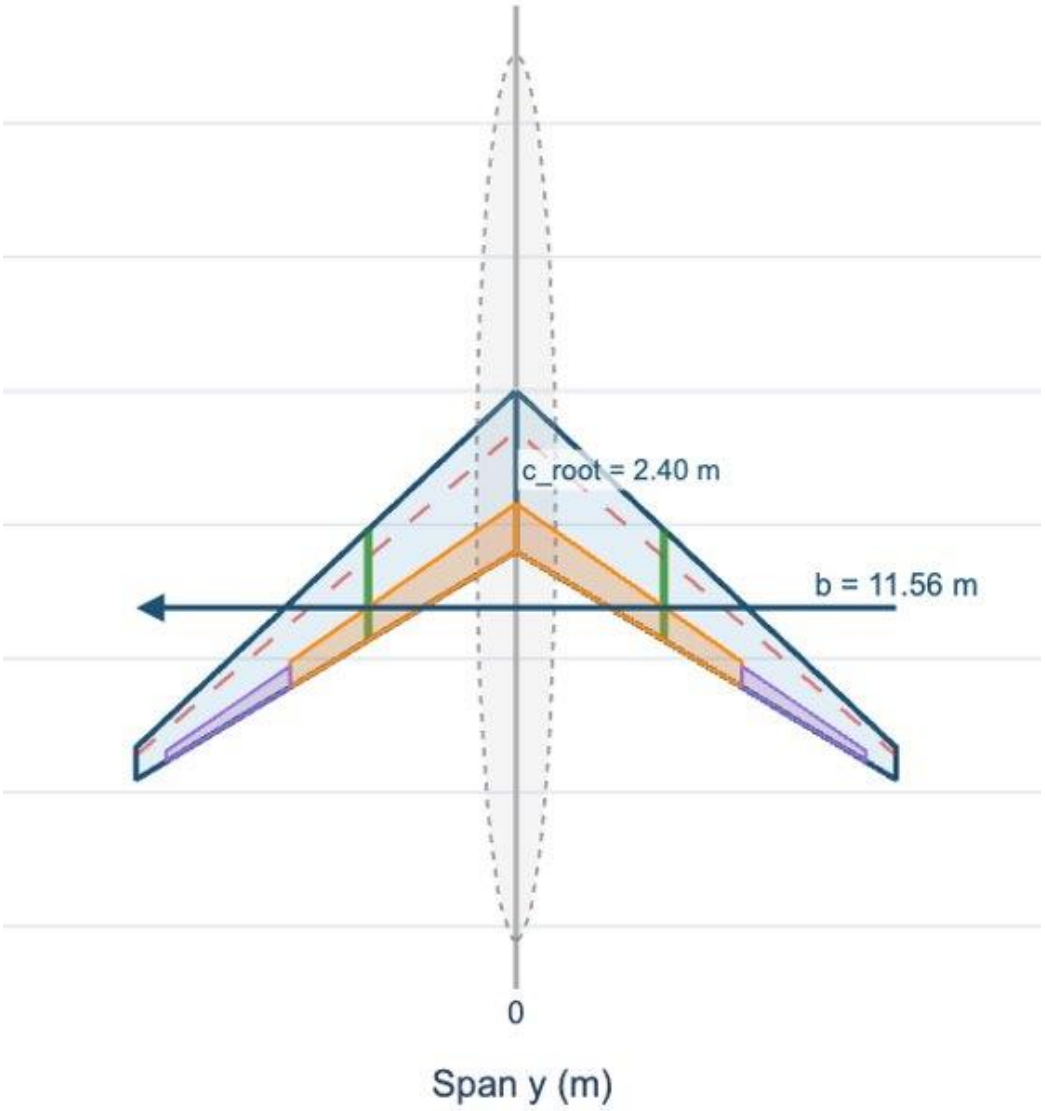
```
            # 18. Aerodynamics & Breguet Objective
            #
            # atm_cruise is passed to OperatingPoint so
NeuralFoil sees the correct
            # Reynolds number: Re = ρ·V·c / μ.  At altitude, both ρ
and μ change, and
            # only using the correct atmosphere captures the net
Re reduction that
            # affects NeuralFoil's transition and drag predictions.
[#6]
            #
==========================================


            op_point = asb.OperatingPoint(
                velocity=config["v_cruise_mps"],
                alpha=alpha,
                atmosphere=atm_cruise,          # altitude-aware
atmosphere  [#6]
            )
            aero_results = asb.AeroBuildup(airplane=airplane,
op_point=op_point).run()


            opti.subject_to(aero_results["L"] == W_initial_N)
```

# C-2 Optimizer User Interface

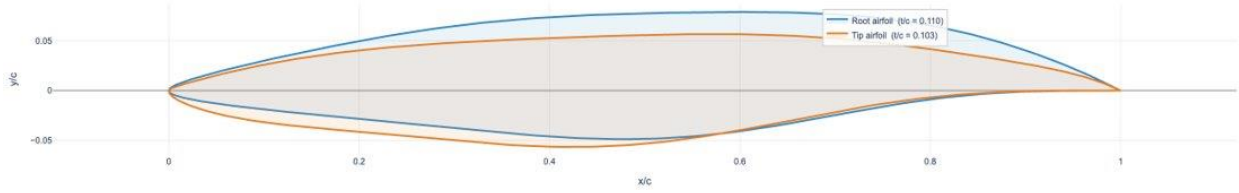



✅ `Parameter panel ready.  Adjust values above, then run Cell 2.`

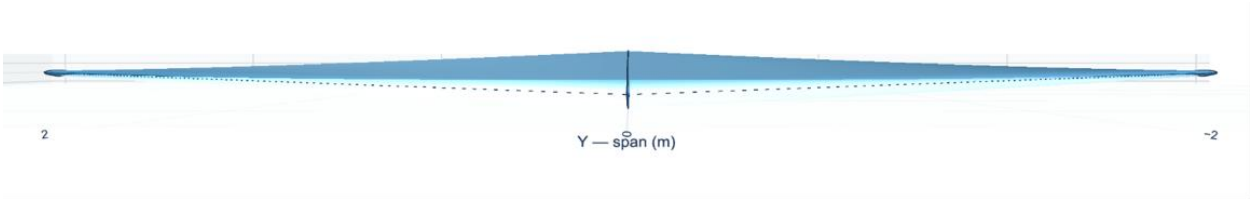
## C-3 Optimizer Test Run Planform




## C-4 Optimizer Test Run 2D Root and Tip Airfoils




**C-5 Optimizer Test Run 3D Front View**


2
0
-2
Y — span (m)