To: Future Aircraft Designers for *Dawn*

From: Peter Sharpe

Subject: *Dawn* Optimization Code Documentation

Date: May 12, 2020

---

1 Overview & Summary

Much of the design of the Spring 2020 16.82 airplane, *Dawn*, was the result of a software tool that enabled large-scale multidisciplinary design optimization (MDO). Here, we provide documentation for the assumptions, models, and code architecture decisions that went into this *Dawn* Design Tool.

The purpose of this document is to compile only the most critical information needed to bring someone up to speed on the conceptual design and sizing methodology of *Dawn*.

Please note that this document is intended to accompany the slides from the Spring 2020 16.82 Critical Design Review, which are linked here for your convenience (accessible to anyone with MIT credentials). Figure from this slide deck are not reproduced inline to preserve brevity; however, we do refer to slides throughout this document as-needed to enhance readability.

For installation instructions, see the Appendix.

Contents

2 Key Information

2.1 Optimization Overview (Slide 17)

The *Dawn* Design Tool is somewhat unique in terms of aircraft design tools, because it simultaneously optimizes both aircraft design and the mission profile. This is an important feature for many aircraft (due to time-dependent parameters like fuel burn or different flight phases), but it is of critical importance for solar aircraft, where the energy input into the system is a direct function of time.

2.1.1 Aircraft Design

On the aircraft design side, we consider several hundred design variables. Among many others, these include:

- Wing geometry (parameterized as a two-section unswept wing)
- Tail geometry (parameterized as a one-section rectangular wing)
- Boom length
- Propeller diameter
- Battery capacity
- Solar panel area
- Motor rated power
- Wing internal structure (spar diameter profile, rib count, etc.)

We have several parameters to consider as well, including:

- Payload mass
- Number of booms
- Battery specific energy

- Margins

Jacobians involving these parameters tend to be small and dense (Jacobian sparsities $\approx 50\%$). Within these, there are often fully-dense blocks corresponding to individual discipline analyses.

2.1.2 Mission Profile

On the mission profile side, we consider several thousand design variables. Among many others, these include:

- Altitude
- Airspeed
- Throttle inputs
- Control surface inputs
- Angle of attack

The number of design variables here is high largely because each of these variables represents an infinite-dimensional design space. This is beacuse each variable is a function of time; in that regard, this is mathematically similar to an optimal control problem. We discretize through time using a direct collocation method integrated with a stabilized midpoint method ("the trapezoidal rule"). Accurate integration requires a minimum of approximately 100 points.

We have several parameters to consider as well, including:

- Minimum altitude
- Wind speed distribution
- Latitude
- Day of year

Jacobians involving these parameters tend to be large and sparse (again, corresponding to sparsity patterns typically seen in optimal control problems). This sparsity pattern means that the mission profile subproblem is typically "easier" to implement and solve compared to the aircraft design subproblem, despite its much larger size[1].

2.1.3 Objective Function (Slide 20)

The objective function of our optimization is typically to minimize wingspan.

This is somewhat of an unusual choice for objective function in aircraft design; a much more common choice is that of takeoff gross weight (TOGW). We choose wingspan as an objective as a means of mitigating risk in our design; historically the most common failure modes have been aerostructural in nature[2]. Wingspan is a better surrogate for aerostructural risk than TOGW, hence our choice for the former as an objective function.

[1]Though again, in practice, these are solved simultaneously.

[2]Examples in the past 20 years include NASA Helios, Google/Titan Solara 50, Facebook Aquila, and Airbus Zephyr.

2.1.4 Constraints

Constraints are by far the most interesting part of any optimization problem, due to the fact that most sensible MDO architectures will implement the majority of analyses (i.e. the core physics) as constraints.

Here, we follow the same pattern of implementing physics as constraints. The physics consist of approximately 150 models, corresponding to approximately 4,000 constraints. We do not attempt to detail these models to retain brevity in this document, but they are extensively documented where they appear in the code (often with links to individual further analysis by the author).

2.2 Optimization Methods (Slide 18)

In total, we typically consider approximately 2,500 variables and 4,000 constraints. The underlying physics of these constraints are generally nonlinear, nonconvex, and non-GP-compatible[3] - this is quite a handful!

We make this problem tractable using automatic differentiation, which reduces our problem's computational cost by several orders of magnitude. A detailed explanation of automatic differentiation is beyond scope, but I've written a helpful primer on my GitHub.

The core idea is that automatic differentiation allows us to find the gradient of a function of many variables.[4] Critically, *the cost of finding this gradient is independent of the dimensionality of the function*! We can also find higher-order derivatives (in particular, efficient optimization depends on second derivatives) for the same roughly-constant cost. This contrasts to the conventional finite-differencing, where the computational cost of finding a derivative scales as the product of the number of variables and the order of the derivative (to first approximation).

2.2.1 Limitations on Models due to Optimization Framework

One implication of this is that all models must be written in a differentiable framework. However, this is quite unrestrictive; essentially all engineering functions are differentiable. Even non-differentiable or non-continuous functions (such as the absolute value function or Heaviside step function) are admissible, as long as their non-differentiability or non-continuity is confined to a finite number of points. Due to this, most types of control flow that one would find in code are fair game (`for` loops, `if/then` statements, etc.). We emphasize that by no means is differentiability limited to basic arithmetic operators; differentiation through trigonometric expressions, linear solves, eigenvalue decompositions, and many other complicated operators are perfectly allowable.

The only function that comes to mind that would be mathematically-impossible to implement in a constraint is the Weierstrass function or other fractal functions [5].

[3]One possible optimization technique involves approximating physics by linear, convex, or GP-compatible constraints - for some constraints, this is trivial, but for others, this is a non-starter.

[4]Here, we restrict ourselves to scalar-valued functions of many variables, which make up nearly all *useful* functions.

[5]The author has yet to encounter a single practical function that cannot be implemented into a differentiable framework.

However, we do note that convergence can only be guaranteed for globally convex problems; as model non-differentiability, non-convexity, and nonlinearity increases, the likelihood of convergence difficulties increases [6].

2.3 Optimization & Code Architecture (Slide 19)

2.3.1 Optimization Architecture

There are many "disciplines" that drive the design of the vehicle; 7 key disciplines have been identified in this endeavor:

1. Atmosphere
2. Aerodynamics
3. Stability
4. Propulsion
5. Power Systems
6. Structures & Weights
7. Dynamics

The interconnections between these disciplines can be visualized using an Extended Design Structure Matrix (XDSM), a popular technique for visualizing MDO architectures. This is shown in Figure 1.

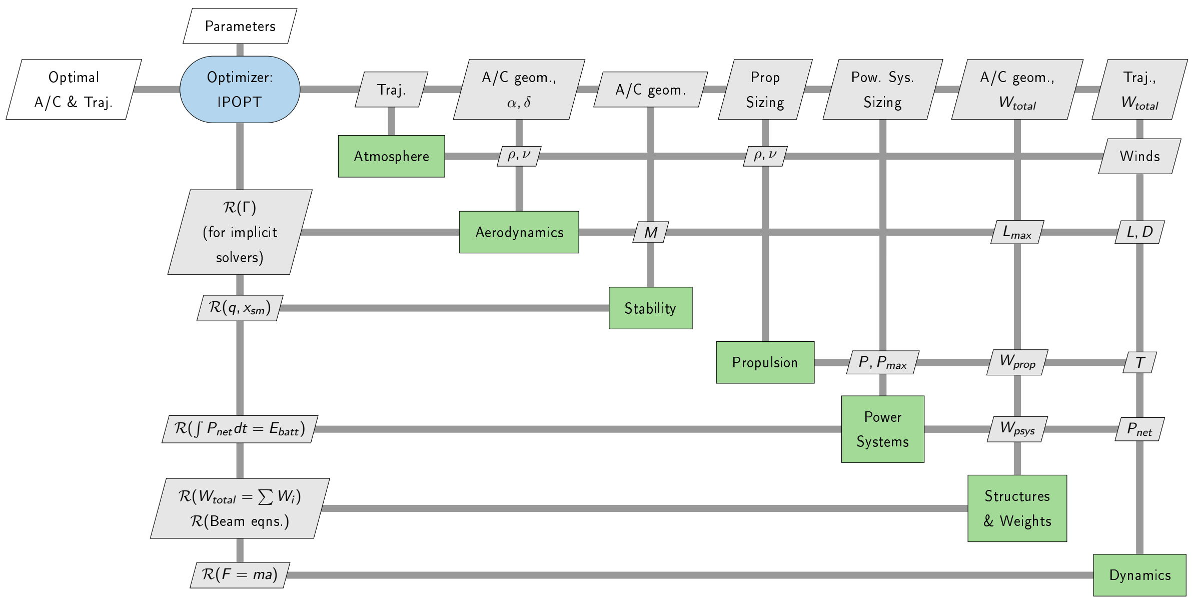

Figure 1: XDSM for the *Dawn* Design Problem

A key characteristic of the architecture described in Figure 1 is the complete lack of internal closure loops. Closure between disciplines is maintained at the optimizer level through the use of constraints; this approach is known in optimization literature as a Simultaneous Analysis and

[6]Within reason, problems caused by these factors can be mitigated by good initial guesses.

Design (SAND) approach. The SAND architecture offers significant benefits in terms of model flexibility and computational speed.

2.3.2 Code Architecture

One final note is that the entirety of the *Dawn* Design Tool is contained within a single file: `design_opt.py`. This code acts essentially as a script file, with no complicated external calls. The code is intentionally quite human-readable, even for Python beginners - and not by accident: there are nearly the same number of comments in this file as there are lines of actual executable code.

Multiple students from the class have been able to familiarize themselves with it and make contributions extremely quickly; please do not be afraid to dig into the code when references to code are made!

3 Models & Assumptions

Every optimizer is only as good as its models and assumptions. Here, we provide brief descriptions of the models and assumptions that went into each discipline analysis[7]. Many of these models are implemented into the open-source *AeroSandbox* where they are extensively documented; for this reason, descriptions in this document will generally be concise with links to detailed documentation.

3.1 Overall Assumptions & Margins (Slide 21)

The *Dawn* Design Tool aims to find the smallest-wingspan design that is capable of closing the mission requirements on the worst day specified. This worst-day is taken to be August 31 solar conditions with 99% wind conditions[8]. The aircraft must also ascend during the day preceding the worst day.

We typically assume a latitude of 49N, which corresponds to the northernmost extent of the continental United States. This would seem like the obvious sizing case for a solar aircraft; however, it turns out that this is not necessarily the case during the months of July and August. While higher latitudes do indeed result in the sun being lower in the sky (resulting in reduced solar power), they actually yield more hours of sunlight per day[9]. Because solar panels are relatively lightweight while batteries are heavy, it turns out that lower latitudes can easily size the solar aircraft during summer months. The author recommends that any design studies be performed at both 49N latitude and 26N latitude (the southernmost extent of the continental United States), as the trade between quality and duration of daily sunlight goes different ways in different regions of the design space.

To emphasize again: all data sources are extensively outlined in the linked code itself where used. More importantly, a full traceability chart has been included inline with the code with every key assumption. An example from the code itself is provided as follow - again, the code is intentionally human-readable!

[7]We limit ourselves to a discussion of only the key *engineering* assumptions that go into the code.

[8]This wind constraint actually turns out to not be tight, as the wingspan minimization drives up the wing loading sufficiently to keep the cruise speed above typical wind speeds.

[9]At the poles, this culminates in the midnight sun effect, where the sun remains above the horizon for months at a time

```
solar_cell_efficiency = 0.25
# This figure should take into account all temperature factors,
# spectral losses (different spectrum at altitude), multi-junction effects, etc.
# Should not take into account MPPT losses.
# Kevin Uleck gives this figure as 0.205.
# This paper (https://core.ac.uk/download/pdf/159146935.pdf) gives it as 0.19.
# 3/28/20: Bjarni, MicroLink Devices has flexible triple-junction cells at 31%
#        and 37.75% efficiency.
# 4/5/20: Bjarni, "I'd make the approximation that we can get at least 25%
#        (after accounting for MPPT, mismatch; before thermal effects)."
# 4/13/20: Bjarni "Knock down by 5% since we need to account for things like
#        wing curvature, avionics power, etc."
# 4/17/20: Bjarni, Using SunPower Gen2: 0.223 from from
#        https://us.sunpower.com/sites/default/files/sp-gen2-solar-cell-ds-en-a4-160-506760e.
# 4/21/20: Bjarni, Knock down SunPower Gen2 numbers to 18.6% due to mismatch,
#        gaps, fingers & edges, covering, spectrum losses, temperature corrections.
# 4/29/20: Bjarni, Config slack: SunPower cells can do 21% on a panel level.
#        Area density may change; TBD
```

3.2 Margins (Slide 15)

To account for model uncertainty (and to a lesser extent, manufacturing uncertainty), we book-keep margin. The default margins included are a 25% margin on structural mass and a 5% margin on power generation to energy closure on the worst day.

3.3 Atmosphere & Wind (Slide 5, 7)

3.3.1 Atmosphere

Atmospheric data was pulled from fits to the popular 1976 COESA model. Data is considered valid for altitudes from sea level to 40 km. Data is fitted to enhance differentiability. For more information, see code and study.

3.3.2 Wind

Wind was modeled based on data from the ECMWF database. Data was fit for winds over the continential United States (CONUS) during the months of July and August. For more information, see code and study.

3.4 Aerodynamics

3.4.1 3D Aerodynamics (Slide 23)

Three approaches were coded and tested for 3D aerodynamics:

1. A workbook-style component-by-component lift and drag buildup (code)
2. A fully nonlinear lifting-line model that couples in profile drag, viscous decambering, local stall, etc. (code)

3. A vortex-lattice-method solver (code)

All are written from scratch in Python (see links above).

When swapping between different 3D aerodynamics methods, optimization solutions typically differ by only a few percent at most; this implies that these models cross-validate themselves quite well. Because of this, the first approach (workbook-style) was used as it is the simplest and fastest of the three.

3.4.2 2D Aerodynamics (Slide 24)

An airfoil known as `"HALE_03"` was created by Trevor Long and fixed *a priori* to the optimization run. Data was fit across the α-Re space using XFoil. Data was fit through the `AirfoilFitter` module of *AeroSandbox*, found here. Airfoil coordinate files are available here.

3.5 Structures & Weights (Slide 25)

Weights are built up using a collection of several dozen different models which are beyond scope here (all models are *extensively* documented inline in code in human-readable comments where used). However, we do briefly mention one key mass model, which is the wing weight model.

Wing secondary weight is empirical based on figures from the Daedalus project [1]. These figures are then multiplied by a factor of 1.3 to reflect the fragility and operational difficulties of Daedalus' exceptionally lightweight secondary structure [10].

Primary weight is based on a classical Euler-Bernoulli beam model that considers both bending and torsion. This model is available here. Spar material is assumed to be carbon fiber, and the geometry is assumed to be tube (for ease of manufacturing and superior torsional properties[11]). Loads are assumed to be static for the purposes of optimization, and we notionally assume a design load factor of 3 in an effort to mimic effects of dynamic loading. All designs are checked *a posteriori* via ASWing to ensure dynamic loading performance is sufficient.

3.6 Propulsion & Power Systems (Slide 22)

3.6.1 Battery

Batteries are assumed to have a specific energy of 450 Wh/kg as measured at the cell level; this is inline with projections from at least 3 commercial suppliers (one of which, Sion Power has been flown on the Airbus Zephyr solar aircraft already).

We also assume an allowable battery depth of discharge of 85%. This figure is on the more optimistic side, but still within the range of typical values seen within the literature. We feel comfortable with a figure of 85% (as opposed to a more conventional 80%), as our battery power demands are much less aggressive than those for other electric aircraft. In particular, many electric aircraft must carry reserve power, as their possible failure events typically occur towards the end of a mission where limited glide range leaves few options. Furthermore, many electric aircraft incorporate VTOL or STOL operations, which create large power draws near landing (while batteries are nearly fully drained - their most vulnerable state for high power draws).

[10]Figure chosen based on personal correspondence with Prof. Drela.
[11]Torsion sizes the spar primarily, not bending.

We assume a battery cell packing factor of 75%. This reflects the fraction of the battery pack, by mass, that consists of raw cells. This figure is fairly typical of solar aircraft, according to personal correspondence with Ed Lovelace[12].

We assume a battery charging and discharging efficiency of 97.5% (i.e. round-trip efficiency $\approx$ 95%).

3.6.2 Solar Cells

We assume a 25.0% realizable efficiency on the solar panels. This figure takes into account all temperature factors, spectral losses (different spectrum at altitude), multi-junction effects, etc. It does not take into account MPPT losses, which are bookkept as a further efficiency of 96.15%.

The solar cells are assumed to have an area density of $0.35\,\mathrm{kg\,m^{-2}}$.

Both of the figures above correspond to cells from MicroLink Devices; specifics are detailed by the Propulsion subteam.

3.6.3 Propeller

The propeller model is based on dynamic disc actuator theory and calibrated to data from QProp. Specifically, we assume a coefficient of performance of 90%.

4 Contact Information

For all other questions not answered in this document, please feel free to reach me at `pds@mit.edu`.

5 Appendix

5.1 Code Setup

Here, we list the steps required to run the cross-platform *Dawn* Design Tool on your computer.

The *Dawn* Design Tool is written in Python, and it consists of modules built on top of *AeroSandbox*. Accordingly, a Python distribution (v3.7+) is required. The standard Python download is sufficient. However, to enhance installation extensibility, the author strongly recommends the use of the Anaconda Distribution of Python, which is an open-source Python distribution that includes over 1,500 packages for scientific computing and data science.

[12]Power systems engineer for Aurora's Odysseus solar aircraft.

5.1.1 *AeroSandbox* Setup

AeroSandbox is the sole direct dependency of the *Dawn* Design Tool˙*AeroSandbox* is an open-source Python package created by the author. A succinct description is found in its `README`:

> "At its heart, AeroSandbox is a collection of end-to-end automatic-differentiable models and analysis tools for aircraft design applications. This property of automatic-differentiability dramatically improves performance on large problems; design problems with thousands or tens of thousands of decision variables solve in seconds on a laptop. Using AeroSandbox, you can simultaneously optimize an aircraft's aerodynamics, structures, propulsion, mission trajectory, stability, and more."

As of the time of writing, *Dawn* Design Tool has been verified to be compatible with *AeroSandbox* v2.2.4. *AeroSandbox* uses semantic versioning, so compatibility can be expected through all v2 releases.

AeroSandbox can be installed from the Python Package Index (PyPI) through the use of the `pip` command (installed with every Python distribution). Open a Python console and enter the following:

```
pip install aerosandbox<=2
```

Once this completes without errors, *AeroSandbox* has been installed correctly. If errors happen, check the continuous integration logs for help identifying the source of the error.

5.1.2 *Dawn* Design Tool Setup

The *Dawn* Design Tool is available to anyone with MIT credentials through the internal MIT GitHub Enterprise server under the repository name `pds/Design_Opt_HALE`. Download the repository by cloning with Git; the clone targets are listed as follows:

- `git@github.mit.edu:pds/Design_Opt_HALE.git` via SSH protocol
- `https://github.mit.edu/pds/Design_Opt_HALE.git` via HTTPS protocol

References

[1] Juan R. Cruz. Weight Analysis of the Daedalus Human Powered Aircraft. 1989.