Oregon Undergraduate Research Journal: McNair Special Issue
21.3 (2023) ISSN: 2160-617X (online)
ourj.uoregon.edu

Multidisciplinary Design Optimization of Portland State Aerospace Society (PSAS) Launch Vehicle 4

Aaron Casserly*

Abstract

Multidisciplinary Design Optimization is a field that enables the solution of challenging engineering problems involving multiple technical specializations and design/performance constraints. In this work, I optimize the design of the PSAS Launch Vehicle 4 (LV4). To that end, I evaluate different optimization approaches—such as RBFOpt Global Optimization, Nelder-Mead minimization, and Simplicial Homology Global Optimization with Nelder-Mead and COBYLA local minimization techniques, calculate structural analysis information for different stages of flight, outline a method of simulating fin "staging"—the dropping of a larger initial fin can at a certain altitude to reduce the required engine thrust and drag in the upper atmosphere and optimize fin parameters. I converged on the ideal design vector. This led to an apogee of 107 km with a 9.8 kN engine (realized with two 5 kN engines). Further debugging is required to resolve the apparent 120 km vehicle drift.

1. Introduction

The Portland State Aerospace Society (PSAS) Launch Vehicle 4 is the fourth iteration of their student-built rocket. PSAS is a feeder organization for those interested in working in spaceflight and we care about optimizing it to give young professionals experience in relevant problem solving to real-world problems, and to achieve the above 100 km apogee target while meeting performance constraints. The constraints given to the optimizer are real-world engineering constraints that must be met for this to be realized. Previous optimization efforts included a design vector involving various components of the rocket, and my approach to include fin parameters and scale them down when a near optimal design was converged on was an improvement. The goals of this research project were to improve and extend the existing Multidisciplinary Design Optimization (MDO) simulation code and to converge on a design vector satisfying engineering and performance constraints.

2. Methodology

Our approach minimizes the gross lift-off weight (GLOW) of LV4 without sacrificing apogee. Given the Tsiolkovsky rocket equation $\frac{\delta v}{v_e} = ln\frac{m_0}{m_f}$, where v_e represents the effective exhaust velocity, δv the total change in velocity, m_0 the GLOW and m_f the final dry mass of the rocket, we isolate m_0 and construct an objective function. This approach has previously been characterized:

> [M]inimizing GLOW while demanding a certain apogee is equivalent to simultaneously minimizing structural mass, maximizing engine performance, and balancing the conflicting goals of minimizing losses due to gravity and aerodynamics. Note

**Aaron Casserly (casser.aero@gmail.com) is a Jamaican immigrant and lawful permanent resident of the United States. He arrived in Oregon in April of 2019 and enrolled at the University of Oregon later that year. He is now a graduate with a Bachelor of Science in Mathematics and double minor in Physics and Computer Science with final GPA of 3.7/4. He plans to attend Northwestern University for a master's degree in electrical engineering in the Fall of 2024.*

> that the sources of this conflict are the incentive to expel propellant rapidly to avoid the cost of carrying propellant in a gravitational field and the incentive to reduce velocity in lower atmosphere since drag is proportional to air pressure and the square of velocity. (MDO Jupyter Notebook)

To represent our problem constraints, we use barrier and penalty functions. Both bind the generated rockets to a feasible region in the design space: barrier functions use an absolute constraint that may not be violated under any circumstances, whereas penalty functions slightly disincentivize convergence in sections of the design space far from a set of more lenient constraints.

Adding the objective, barrier, and penalty functions, we construct a pseudo-objective merit function, which takes in an array of values sufficient to describe the mathematical model of the rocket and its performance. Given that each evaluation of this merit function consists of simulating the rocket's trajectory, we are unable to use an optimization algorithm involving differentiation or a finite difference method. To navigate this limitation, we use a Nelder-Mead simplex method: a geometry-based optimization algorithm that does not perform well for higher dimensions but is satisfactory for our purposes. A genetic algorithm, which can handle higher-dimensional spaces, would also serve this function, but it incurs an additional computational cost.

The barrier and penalty functions are weighted by user-selected parameters before their addition to the objective function. Given that an overly low weighting will lead to the optimizer's neglect of the constraints and an overly high weighting to the optimizer ignoring the objective, we run an iterative sequence of Nelder-Mead optimizations, starting with very low weights and increasing them for every successive optimization. This gives the optimizer more global coverage of the design space early on so that it may find a suitable neighbourhood and then restricts its freedom once it has done so.

The All-at-Once (AAO) problem statement is a fundamental optimization problem from which all others may be derived. It includes an objective/pseudo-objective function to be minimized with respect to a design vector and subject to certain constraints. For this problem, we are estimating the optimal design vector, $\bar{x}$ according to $\lim_{n\to\infty} \mu_n = 0$, $\lim_{n\to\infty} \rho_n = \infty$, $f_n(\bar{x}) = m_{obj}(\bar{x}) + \mu_n \sum_i h_i(\bar{x}) + \rho_n \sum_j g_j(\bar{x})$.

$\bar{x}_n = min_{\bar{x}} f_n(\bar{x})$, $\lim_{n\to\infty} \bar{x}_n = \bar{x}^*$, where $\bar{x} = (m_{prop},\ \dot{m},\ p_e)$ is the design vector containing the total propellant mass, unadjusted propellent mass, mass flow rate, nozzle exit pressure, total tankage length, airframe diameter, airframe total length, GLOW, ballast mass, conical component of nosecone length, fin root chord, fin tip chord, fin sweep angle, fin span, and fin thickness. This information is necessary for the evaluation of the constraint functions, $h_{barrier} = 108401m < h < 151401m$, and $g_{penalty} = \left(F \le 6kN, LS \ge 22m/s, \frac{a_{max}}{g_0} \le 15g's, TWR \ge 2, L/D \le 21, \frac{p_e}{p_a} \ge 0.35\right)$. $h_{barrier}$ is the strict apogee constraint, and the looser penalty constraints are: Thrust (F)—the Electric Feed System (EFS) that deals with pressurizing before propellant injection is not feasible for powerful engines, Launch Speed (LS)—ensuring stable take-off, Thrust to Weight Ratio (TWR), Length to Diameter Ratio (L/D), maximum acceleration, and nozzle over-expansion.

The unaltered optimization code produces an .ork rocket file for further testing in OpenRocket, as seen in Figure 1.

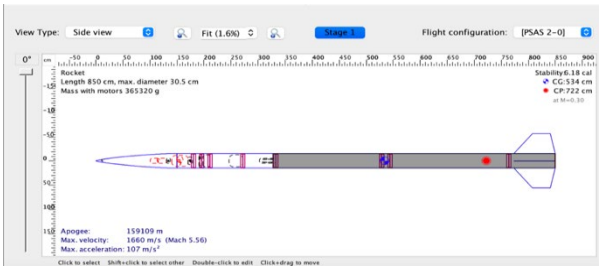


Figure 1. Example generated rocket.

It is then possible to run the simulation with settings emulating the launch site (WGS84 ellipsoid for Geodetic calculations), pictured in Figure 2.

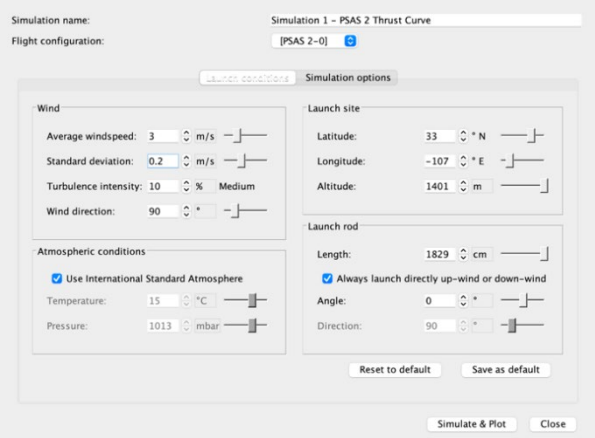


Figure 2. Settings to emulate the launch site.

These calculations produce a launch data graphic, pictured in Figure 3.

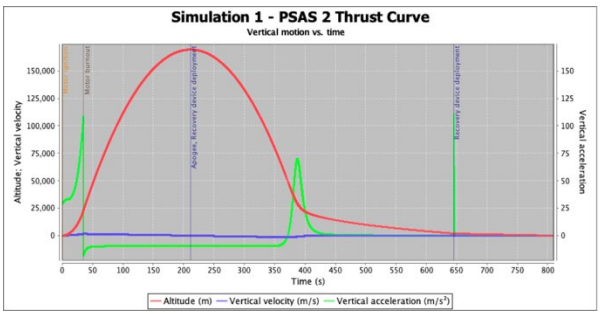


Figure 3. OpenRocket launch data graphic.

Review of uncertainty-based multidisciplinary design optimization methods for aerospace vehicles (Yao et al.) covers Uncertainty-Based Multidisciplinary Design Optimization (UMDO) theory and the cutting edge methods of that time. Throughout the lifecycle of the aerospace vehicle (design, manufacture, operation, disposal/repurposing), there exist many uncertainties related to the vehicle system itself, along with environmental and operational conditions. Before describing the UMDO procedure, several important definitions are given: uncertainty—incompleteness in knowledge and inherent variability of the system and operational environment; robustness—measure of insensitivity to variations in both the system and environment; reliability—likelihood of a component/system to perform intended function for a given period of time under the determined operating conditions; deterministic design optimization—process of obtaining optimal designs with all variables, models, parameters and simulations involved being deterministic; robust design optimization (RDO)—optimizing design such that it is insensitive to many variations; and reliability-based design optimization (RBDO)—obtaining optimal design while meeting reliability constraints. The combination of these last two, RDO and RBDO, is the basis for reliability-based robust design optimization (RBRDO): Find $\boldsymbol{x}$ minimizing $\tilde{f}(\boldsymbol{x},\boldsymbol{p}) = F\left(\mu_f(\boldsymbol{x},\boldsymbol{p}), \sigma_f(\boldsymbol{x},\boldsymbol{p})\right)$ subject to (s.t.) $P[\boldsymbol{g}(\boldsymbol{x},\boldsymbol{p}) \leq 0] \geq \boldsymbol{R}$, $\boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U$, where $\boldsymbol{x}$ represents the design variable vector, $\boldsymbol{p}$ represents the system constant parameter vector, $\boldsymbol{x}^L$ and $\boldsymbol{x}^U$ define the boundaries of the design space, μ_f and σ_f are the mean and standard deviation of the original optimization objective function, F is the reformulated optimization function with respect to μ_f and σ_f, $\mathbf{g}$ is the unequal constraint vector, P is the probability of the statement in brackets to be true, and $\boldsymbol{R}$ is the reliability vector related to this. Yao et al. provide illustrations related to RDO and RBDO, seen in Figures 4 and 5.

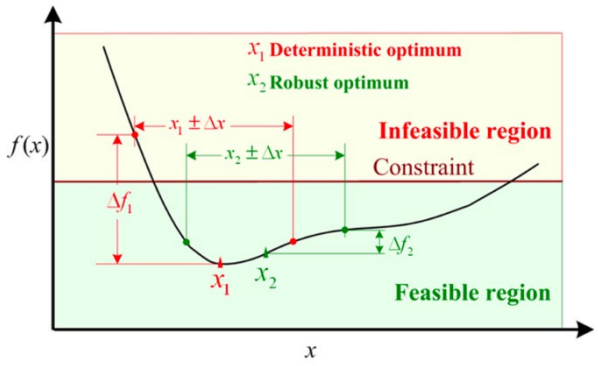


Figure 4. Graphical illustration of RDO.

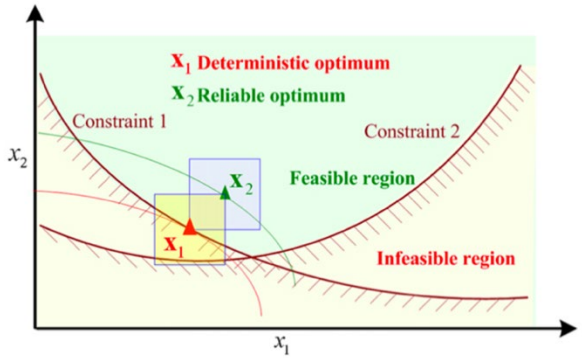


Figure 5. Graphical illustration of RBDO.

The actual UMDO procedure organizes the elements involved in uncertainty-based design optimization: system optimization, system analysis, disciplinary analysis, and uncertainty analysis. The key to realizing UMDO for a large, complex system is efficiently arranging these elements into an execution sequence so that it may be implemented on a computer; the coupling relationship related to disciplinary analysis and computationally intensive system analysis make for a very time-consuming procedure. The computational burden of UMDO can be understood by the following modification to the RBRDO formulation: Find $\boldsymbol{x}$ minimizing $\tilde{f}(\boldsymbol{x},\boldsymbol{p},\boldsymbol{y}) = F\left(\mu_f(\boldsymbol{x},\boldsymbol{p},\boldsymbol{y}),\sigma_f(\boldsymbol{x},\boldsymbol{p},\boldsymbol{y})\right)$ s.t. $P[g_i(\boldsymbol{x},\boldsymbol{p},\boldsymbol{y}) \le 0] \ge \boldsymbol{R_i}$, $i = 1,2,\dots,n_g$, $\boldsymbol{x}^L \le \boldsymbol{x} \le \boldsymbol{x}^U$. $\boldsymbol{y}$ represents the intermediate state variables of the multidisciplinary analysis. We denote the output vector of disciplinary analysis i as $\boldsymbol{y}_i$, the coupling state vector output from disciplinary analysis i and input into disciplinary analysis j as $\boldsymbol{y}_{ij}$, the complete set of output vectors from discipline i coupled with other disciplines $\boldsymbol{y}_{i.}$, and the complete set of coupling state vectors input into disciplinary analysis i as $\boldsymbol{y}_{.i}$. With these conventions, we have $\boldsymbol{y} = [\boldsymbol{y}_i, i = 1,2,\dots,n_D]$, $\boldsymbol{y}_{.i} = [\boldsymbol{y}_{ji}, j = 1,2,\dots,n_D, j \ne i]$, $\boldsymbol{y}_{i.} = \boldsymbol{y}_{i.}(\boldsymbol{x}_i,\boldsymbol{p},\boldsymbol{y}_{.i})$ and $y_{i.} = [y_{ij}, j = 1,2,\dots,n_D, j \ne i]$. $\boldsymbol{x}_i$ is the local design variable vector of discipline i, and $\boldsymbol{p}$ is the system parameter vector. The paper provides a figure with information related to the coupling relationship for a three-discipline system, pictured in Figure 6.

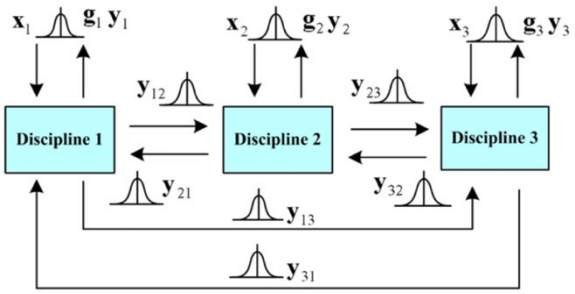


Figure 6. The coupling relationship of a three-discipline UMDO problem.

Yao et al. also provide an illustration of the conventional double-loop UMDO procedure, pictured in Figure 7.

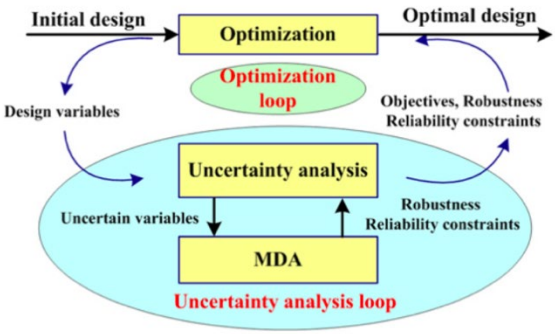


Figure 7. The conventional double-loop UMDO procedure.

I set out to review the existing open-source optimization code to analyze the dependencies and components of these procedures such that I would be able to isolate areas for potential improvements. In making modifications, I set out to compare simulated trajectory results and the optimized design vectors. The following is a list of planned improvements:

1. The majority of the Aerodynamics model is based on OpenRocket's source code, which is an oversimplification of reality. The first instance of a possible improvement to the Jupyter Notebook detailing the aerodynamics model is related to the fin-body interference coefficient. This can be improved using MIL-HDBK-762 (Handbook).
2. The optimization program generates designs based on a template OpenRocket

file. It will be necessary to update this to the latest airframe design, as the design fed into the optimizer is still based on the airframe of a previous iteration.

3. Implement fin "staging"; this will be done by dropping a large fin can. Essentially, larger fins enable a reduction in the required launch velocity, and, therefore, the engine size may be reduced. Dropping the can mid-flight will reduce the drag due to large fins. I will develop a method of simulating this effect.
4. Add to MDO capability via reading-in a database of aerodynamic coefficients created by CFD.
5. Improve the "UI" such that those unfamiliar with the code may set parameters and understand the process. Work on documentation and comments in relation to better user interaction.
6. Add structural analysis output such as weight, stresses, acceleration, acceleration in propellants, axial load down the rocket, and heatmap to show where the axial load is the highest.
7. If efforts to reduce engine weight have failed and we need a large engine (on the order of 10 kN), the use two 5 kN engines or four 2.5 kN engines would be optimal. This would reduce chamber pressure; additionally, lower-thrust engines are more feasible to build.
8. Compare efficiency/quality (merit function evaluation) of optimization approaches such as global optimization using RBFOpt, iterative Nelder-Mead, and Simplicial Homology Global Optimization (SHGO).

3. Results

3.1. Aerodynamics

The majority of the Aerodynamics model is based on source code from OpenRocket, which does not account for the additional complexity of reality. The first improvement I made in relation to this was to adjust the fin-body interference coefficient using MIL-HDBK-762. Using the slender-body theory approach (where the slenderness of the modelled body is used to create an approximation to the field surrounding it), the ratio of the fin normal force gradient—the resulting corrective force perpendicular to the z-axis of the rocket—in the presence of a cylindrical body compared to that of an isolated fin is given by $K_{F(b)} = \frac{\frac{2}{\pi}}{\left(1-\frac{d}{b}\right)^2}\left(\left(1+\frac{d^4}{b^4}\right)\left(\frac{1}{2}arctan\left[\frac{1}{2}\left(\frac{b}{r}-\frac{d}{b}\right)\right]+\frac{\pi}{4}\right)-\frac{d^2}{b^2}\left[\left(\frac{b}{d}-\frac{d}{b}\right)+2\,arctan\left(\frac{d}{b}\right)\right]\right)$, where r is half the fin span (b) and d is the body diameter, as pictured in Figure 8.

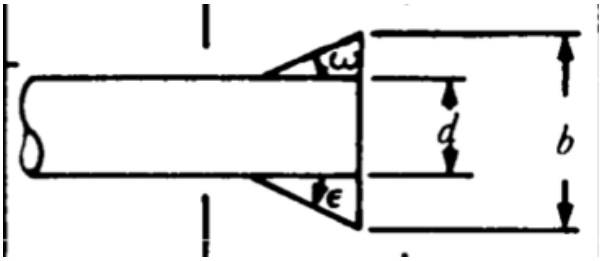


Figure 8. Fin span and body diameter of LV4.

The trajectory simulation component of the open-source code outputs the angle of attack—the difference between the rocket's z-axis and relative velocity vector—of the vehicle as a function of time, seen in Figure 9.

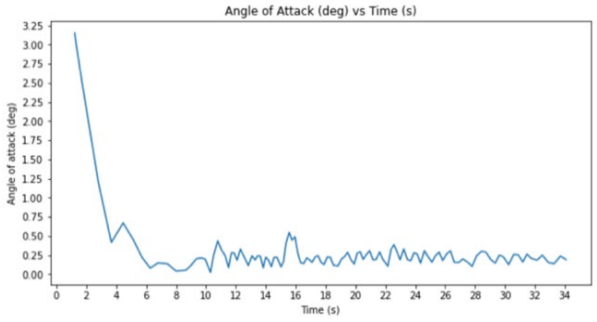


Figure 9. Angle of attack as a function of time.

The optimized design vector before and after this modification is represented in Figure 10.

```
6/06/2021 21:41:02
DESIGN VECTOR
----------------------------
design total propellant mass               = 242.0959 kg
design unadjusted propellant mass          = 237.0335 kg
design mass flow rate                      = 6.8475 kg/s
design nozzle exit pressure                = 106525.5088 Pa
total tankage length (after adjustment)    = 4.0814 m
design airframe diameter                   = 0.3048 m.
design airframe total length               = 8.6464 m.
design GLOW                                = 393.9780 kg
design ballast mass                        = 2.0000 kg
conical part of nosecone length            = 1.0923 m
design fin root chord                      = 0.7620 m
design fin tip chord                       = 0.3302 m
design fin sweep angle                     = 40.0000 deg
design fin span                            = 0.4064 m
design fin thickness                       = 3.1750 mm

3/04/2022 15:57:13
DESIGN VECTOR
----------------------------
design total propellant mass                = 246.6801 kg
design unadjusted propellant mass           = 241.5327 kg
design mass flow rate                       = 6.7373 kg/s
design nozzle exit pressure                 = 167633.7555 Pa
total tankage length (after adjustment)     = 4.1567 m
design airframe diameter                    = 0.3048 m.
design airframe total length                = 8.7216 m.
design GLOW                                 = 402.6394 kg
design ballast mass                         = 2.0000 kg
conical part of nosecone length             = 1.0923 m
design fin root chord                       = 0.7620 m
design fin tip chord                        = 0.3302 m
design fin sweep angle                      = 40.0000 deg
design fin span                             = 0.4064 m
design fin thickness                        = 3.1750 mm
```

Figure 10. Optimized design vector before and after fin-body interference modification.

The increased total propellant mass and gross lift-off weight indicate that increased resistance to the fin normal force is required. The higher nozzle exit pressure is related to the optimization program minimizing nozzle over-expansion (Monte).

An over/under-expanded nozzle is one in which the exit pressure is greater or lower than the atmospheric pressure. The combustion chamber generates high pressure, high temperature gas, and the ideal nozzle (shape and length optimized) converts this thermal energy into thrust as in Figure 11. An over-expanded nozzle is one in which the atmospheric pressure is greater than the exit pressure, which causes a pinching effect and decreases the efficiency of the nozzle as sections of the nozzle inner wall are not used to produce thrust. Figure 12 demonstrates an over-expanded nozzle. Under-expansion is the opposite: the atmospheric pressure is less than the nozzle exit pressure, which causes the flow to fan out after exiting the nozzle and results in inefficiency, as the expansion is not fully converted into thrust by the nozzle inner wall. Figure 13 demonstrates an under-expanded nozzle.

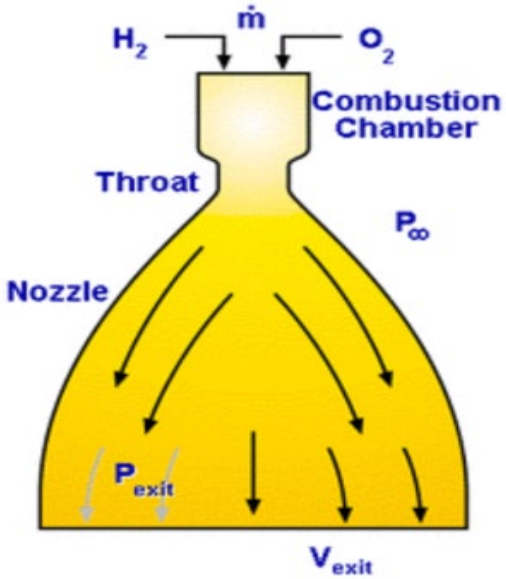


Figure 11. Nozzle and combustion chamber of LV4.

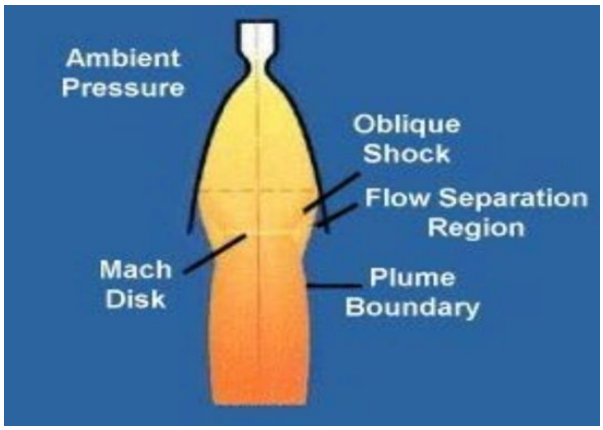


Figure 12. Over-expanded nozzle of LV4.

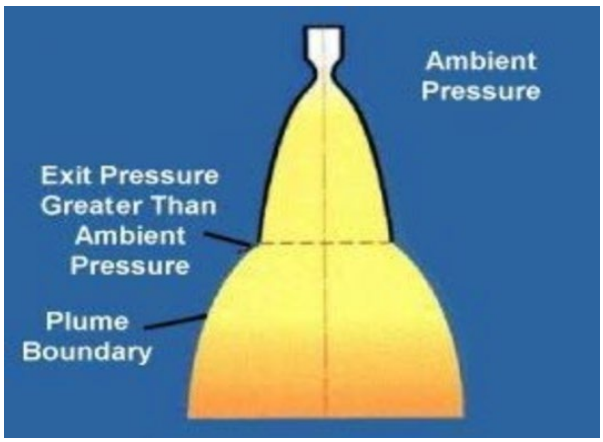


Figure 13. Under-expanded nozzle of LV4.

Ideally, we would like nozzle exit pressure to equal atmospheric pressure, and minimizing over-expansion is our best option. The higher exit pressure after increasing the accuracy of the fin-body interference coefficient indicates that the atmospheric pressure is higher, likely due to an increased fin normal force gradient.

In relation to enabling the MDO to read CFD data, I obtained a demo aerodynamics database—shown in Figure 14—with the goal of building an interpolation space in the variables.

```
Mach, qbar, pitch, yaw, roll, Cx, Cy, Cz, Cl, Cm, Cn
0.100,    631, 0.0, 0.0, 0.0, 0.381, 0.0, 0.0, 0.0, 0.0, 0.0
0.523,  15642, 0.0, 0.0, 0.0, 0.360, 0.0, 0.0, 0.0, 0.0, 0.0
0.963,  45734, 0.0, 0.0, 0.0, 0.451, 0.0, 0.0, 0.0, 0.0, 0.0
2.000, 110648, 0.0, 0.0, 0.0, 0.351, 0.0, 0.0, 0.0, 0.0, 0.0
2.850, 126219, 0.0, 0.0, 0.0, 0.294, 0.0, 0.0, 0.0, 0.0, 0.0
4.700,  85887, 0.0, 0.0, 0.0, 0.238, 0.0, 0.0, 0.0, 0.0, 0.0
```

Figure 14. Aerodynamic coefficients database.

I created a new notebook to complete this task and used the SciPy interpolate module (*Scipy.Interpolate.Interp1d — SciPy v1.8.1 Manual*) along with the built-in Python CSV module. Figure 15 displays the first from-scratch code contribution I made.

```
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
matrix = []
Dict = {}

with open('aero_database_demo.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for item in csv_reader:
        matrix.append(item)
    for k in range(len(matrix[0])):
        values = []
        for j in range(len(matrix)-1):
            value = float(matrix[j+1][k].strip())
            values.append(value)
        key = matrix[0][k].strip()
        Dict.update({key: values})
```

```
variables = Dict.keys()
print(variables)
```

```
dict_keys(['Mach', 'qbar', 'pitch', 'yaw', 'roll', 'Cx', 'Cy', 'Cz', 'Cl', 'Cm', 'Cn'])
```

```
zero = float(0)
for item in variables:
    Dict[item].append(zero)
    Dict[item].sort()
```

Figure 15. Code for processing database of Aerodynamic coefficients.

I initialize a list "matrix" to store the data along with a dictionary "Dict" to index into a variable's list of values. Using the csv module, we read in the data and construct a 2-dimensional matrix. We then use a nested structure to construct each variable's list of values, with which we populate the dictionary. We are now ready to perform interpolation, as shown in Figure 16.

qbar

```
x = Dict['Mach']
xnew = np.arange(0, max(Dict['Mach']), 1e-7)
y = Dict['qbar']
f = interpolate.interp1d(x, y, kind = 'cubic')
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.xlabel('Mach')
plt.ylabel('qbar')
plt.show()
```

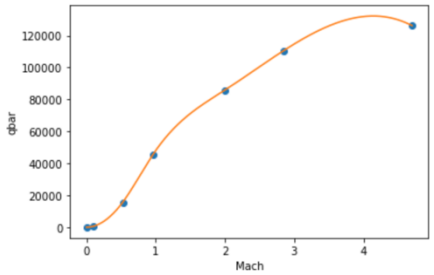


Figure 16. Cubic interpolation in the qbar variable.

We use the "interp1d" function to approximate a continuous function given our discrete data points. Figure 17 displays another example, this time with linear interpolation.

Cx

```
x = Dict['Mach']
xnew = np.arange(0, max(Dict['Mach']), 1e-7)
y = Dict['Cx']
f = interpolate.interp1d(x, y, kind = 'linear')
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.xlabel('Mach')
plt.ylabel('Cx')
plt.show()
```

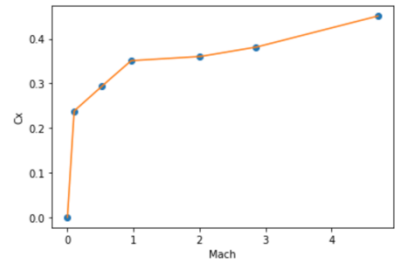


Figure 17. Linear interpolation in the Cx variable.

3.2. Optimization Method Efficiency/Quality Analysis

The three options for optimization that I considered were global optimization using

RBFOpt (black-box optimization), scipy.optimize's Nelder-Mead minimization, and Simplicial Homology Global Optimization (SHGO). It is important to note that our merit function is a combination of an objective function and a penalty function. This means that the constraints are captured by the penalty component and are not passed into the optimization methods directly. Therefore, we have in each case an unconstrained global optimization which approximates a constrained optimization.

The Nelder-Mead algorithm is designed to minimize a non-linear function $f: \boldsymbol{R}^n \to \boldsymbol{R}$ using function values at a few points in $\boldsymbol{R}^n$. It can be viewed as a simplex-based search algorithm. A simplex in $\boldsymbol{R}^n$ is defined as the convex hull of $n+1$ vertices. For example, a simplex in $\boldsymbol{R}^2$ is a triangle, while $\boldsymbol{R}^3$ would be a tetrahedron, shown in Figure 18.

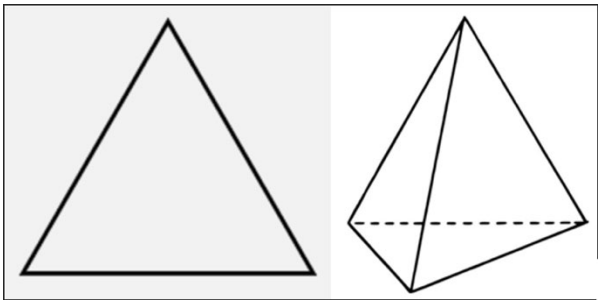


Figure 18. Simplexes in two- and three-dimensional space.

The method begins with a set of points $x_0, \dots, x_n \in \boldsymbol{R}^n$, which are the vertices of our simplex, and their merit function evaluations. The algorithm will then perform a series of transformations on the working simplex with the goal of decreasing the merit function evaluation at the vertices. This process is terminated when the absolute errors in the optimal design vector and its function evaluation are sufficiently small (*Minimize(Method='Nelder-Mead') — SciPy v1.8.0 Manual*). A simplification of the algorithm is the following:

1. Construct initial working simplex S.
2. Repeat until termination test is satisfied.
    a. Calculate termination test information (absolute error).
    b. If termination test is satisfied, transform the working simplex.
3. Return the best vertex of the current simplex S and the merit function evaluation.

We can construct the initial simplex by generating $n+1$ vertices $(x_0, \dots, x_n)$ around some input point in $\boldsymbol{R}^n$. For practical purposes, we use x_0 so that the algorithm may be restarted. The remaining n vertices are then generated to obtain a regular simplex, with all edges having the same length.

A key component is the simplex transformation algorithm, which consists of three stages:

1. Ordering to determine the worst (h), second-worst (s), and best (l) vertices in the current working simplex: $f_h = max_j f_j$, $f_s = max_{j \neq h} f_j$, $f_l = min_{j \neq h} f_j$.
2. Calculation of the centroid of the best side, which is opposite the h-vertex ($c := \frac{1}{n} \sum_{j \neq h} x_j$).
3. Computation of the new working simplex via transforming the current.

We attempt to replace the worst vertex using reflection, contraction or expansion with respect to the best side. The test points lie on the line from the worst point (x_h) to the centroid of the best side, as previously calculated. At most, two such points are calculated in each iteration. If successful, this accepted point becomes the new vertex of our working simplex. Otherwise, we shrink the simplex towards the best vertex (x_l), and it is necessary to compute n new vertices.

On testing the Nelder-Mead approach, I reduced the termination conditions to an absolute difference between optimal design vectors of 1 and an absolute difference between

merit function evaluations of 0.1. The results indicate that this will produce a locally feasible design; however, this is not ideal for the global optimization that we desire. Combining this with SHGO should improve results. The output of this is shown in Figure 19.

```
Iteration 1:
Optimization terminated successfully.
         Current function value: 0.458370
         Iterations: 278
         Function evaluations: 705
         Arithmetic errors (from violations of acceptable altitude window): 0
Propellant mass (kg): 223.0627674561624
Mass flow rate (kg/s): 5.084690802101526
Exit pressure (Pa): 111199.05113706988
Peak Altitude (km): 98.9836442931816
```

Figure 19. Iterative Nelder-Mead optimization output.

We can see here that the peak altitude is less than desired. Overall, this approach is time-intensive, given a more stringent termination condition, and will at best produce locally feasible designs. The limited design space exploration of the Nelder-Mead algorithm can be understood via Figure 20.

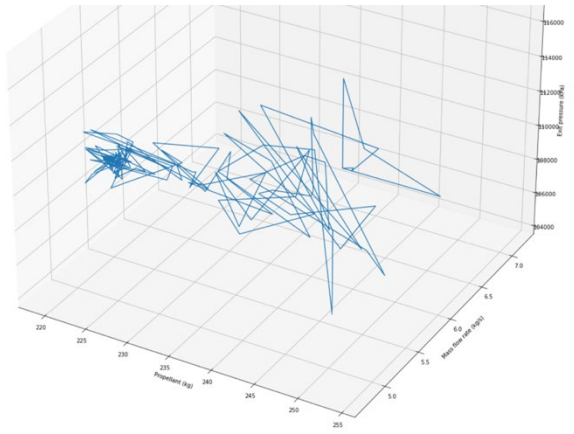


Figure 20. Iterative Nelder-Mead design space exploration.

The RBFOpt (*Coin-or/Rbfopt*) global optimization method provides excellent coverage of the design space and runs to completion in under two hours. To perform this optimization, we construct a black box using the RbfoptUserBlackBox class and execute RbfoptAlgorithm on it. The values comprising the two arrays in the definition of the black box were set based on a feasible range for the design variables. Figure 21 displays a snippet of our code.

```
bb = rbfopt.RbfoptUserBlackBox(len(init_array),
                               np.array([75, 1.5, 86346/4, 805000/2.5, 805000/2.5, 344738*1.5]),
                               np.array([375, 7, 86346*2.5, 2.068e6, 2.068e6, 4.137e6]),
                           np.array(['R']*len(init_array)), f)
settings = rbfopt.RbfoptSettings(minlp_solver_path=paths["bonmin_path"],
                                 nlp_solver_path=paths["ipopt_path"],
                                 max_evaluations=800, eps_impr=1.0e-4)
alg = rbfopt.RbfoptAlgorithm(settings, bb)
val, x, itercount, evalcount, fast_evalcount = alg.optimize()
```

Figure 21. RBFOpt Global Optimization code snippet.

This design variable range works in conjunction with Bonmin (*Bonmin*) (Basic Open-source Nonlinear Mixed Integer programming) to find a design vector that minimizes the merit function. With this approach, the majority of our constraints are satisfied, as seen in Figure 22.

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)          = 5.2142 deg
L/D ratio (c.f. < 25.0)                        = 27.6179
fin flutter ratio (c.f. > 1.0)                          = 27.2560
Sommerfield criterion (c.f. pe/pa >= 0.35)     = 1.6490
max acceleration (c.f. < 15.0)                 = 6.2303 gs
TWR at lift off (c.f. > 2.0)                  = 1.9012
Lowest stability margin caliber (c.f. > 2.0) = 2.9854
speed when leaving launch rail (c.f. > 22.0)  = 25.9781 m/s
altitude at apogee (c.f. > 105.0)              = 97.1917 km
LFETS LOX velocity (c.f. < 9.144)     = 5.5099 m/s
LFETS IPA velocity (c.f. < 9.144)     = 4.6270 m/s
design thrust (ground level) (c.f. < 10000)    = 10653.9372 N
```

Figure 22. RBFOpt Global Optimization Constraint Satisfaction.

Simplicial Homology Global Optimization (SHGO), in conjunction with Nelder-Mead minimization, provides good results with a completion time of about 2 hours. The theoretical advantages of SHGO are guaranteed when the objective function is Lipschitz smooth (objective function is continuous, convex, and smooth); however, if this is not the case, the algorithm will converge to the global optimum if the default "simplicial" sampling method is used (*Scipy.Optimize.Shgo — SciPy v1.8.0 Manual*). SHGO is a general-purpose global optimization algorithm that approximates the homology groups of a complex built on a hypersurface that is homeomorphic (similar in form) to a complex on the objective function. This facilitates approximations of locally convex subdomains in the search space (multidimensional space consisting of design vector parameters and their constraints) and provides an excellent visual tool for characterising and solving higher-dimensional black box optimization problems. The complex is created using sampling points

within the feasible search space as vertices. The algorithm is best suited to finding all local minima of an objective function with a computationally expensive evaluation (such as ours, which involves simulating the trajectory of the design).

Using the sampled points of an objective function as vertices, this method constructs a simplicial complex. The resulting directed subgraph contains the set of all 1-chains from the elements of $\mathcal{H}^1 \in \mathcal{H}$ and enables the finding of minimizer pools (Endres et al.) Sperner's lemma enables us to approximate the domains of stationary points for our objective function in the feasible search space, denoted by Ω. The homology groups produced from the construction of $\mathcal{H}$ will be invariant given an adequate sampling set. It follows that for the given sampling set of vertices $\mathcal{H}^0 \in \mathcal{H}$, we are guaranteed to extract the optimal minimiser pool. The algorithm has four steps:

1. Sampling point generation of N vertices in the search space from which 0-chains of $\mathcal{H}^0$ are constructed.
2. Triangulation of the vertices to construct the directed simplicial complex $\mathcal{H}$.
3. Construction of the minimiser pool using Sperner's lemma.
4. Local minimization using the starting points defined in the minimiser pool (Nelder-Mead method).

Given a set of sampling points $\mathcal{P}$, we wish to describe a discrete mapping $h: \mathcal{P} \longrightarrow \mathcal{H}$ that will provide a simplicial approximation for the surface of the merit function. To begin, we need to formally define the set of vertices forming the 0-chains of the simplicial complex and the edges forming the 1-chains of $\mathcal{H}$. The following are useful definitions:

1. χ is the set of sampling points created by a sampling sequence in a bounded hyperrectangle (rectangle generalized to higher dimensions).
2. The set $\mathcal{P} = [\boldsymbol{x}\epsilon\chi|\boldsymbol{g}(\boldsymbol{x}) \geq 0]$ describes a set of points within the feasible set Ω.
3. Given an objective function $f, \mathcal{F}$ represents the set of scalar outputs mapped by the objective function $f: \mathcal{P} \longrightarrow \mathcal{F}$ in relation to a sampling set $\mathcal{P} \subseteq \Omega \subseteq \mathbb{R}^n$.
4. If $\mathcal{H}$ is a directed simplicial complex, then $\mathcal{H}^0 \coloneqq \mathcal{P}$ is the set of all vertices of $\mathcal{H}$.
5. Given a set of vertices $\mathcal{H}^0$, we construct the simplicial complex $\mathcal{H}$ by a triangulation connecting every vertex in $\mathcal{H}^0$. This supplies a set of undirected edges E.
6. $\mathcal{H}^1$ is a set constructed by directing every edge in E. This is done by selecting a vertex $v_i \in \mathcal{H}^0$ and connecting to another vertex v_j by an edge within E. This edge is directed as $v_i v_j$ from v_i to v_j if and only if the merit function evaluation at the former is lesser than the latter. It is directed as $v_j v_i$ from v_j to v_i if and only if the merit function evaluation at the former is greater than the latter. In these cases, we have $\partial(v_i v_j) = v_j - v_i$ and $\partial(v_j v_i) = v_i - v_j$. The case in which $f(v_i) = f(v_j)$, with neither v_i nor v_j already being a minimizer, we use the rule that "the incidence direction of the connecting edge is always directed towards the vertex that was generated earliest by the sampling point sequence." If v_i is not connected to another vertex v_k, then our convention will be to leave $v_i v_k$ undefined, with $\partial(v_i v_k) = 0$. The higher dimensional simplices $\mathcal{H}^k, k = 2, 3, \ldots n+1$ may be directed in an arbitrary direction to complete the construction of the complex $h: \mathcal{P} \longrightarrow \mathcal{H}$. This will be used to find the minimiser pool for the local minimization starting

points required by the algorithm.

7. v_i is a minimiser if and only if all edges connected to v_i are directed away from v_i; formally, that is $\partial(v_i v_j) = (v_{j\neq i} - v_i) \vee 0 \; \forall v_{j\neq i} \in \mathcal{H}^0$. The set of all minimisers is the minimiser pool $\mathcal{M}$.
8. The star of a vertex v_i $[st(v_i)]$ is the set of all points Q s.t. every simplex containing Q contains v_i.
9. The k-chain $C(\mathcal{H}^k)$, $k = n + 1$ of simplices in st (v_i) results in a boundary cycle $\partial(C(\mathcal{H}^{n+1}))$ with $\partial\left(\partial(C(\mathcal{H}^{n+1}))\right) = \emptyset$. The bounds of the domain defined by s.t. (v_i) form the faces of $\partial(\mathcal{H}^{n+1})$.

To place these constructions in a practical context, we minimize the Ursem01 function in two dimensions, which is defined as:

$$\min \quad f(\boldsymbol{x}) = -sin(2x_1 - 0.5\pi) - 3cos(x_2) - 0.5x_1, x \in \Omega = [0,9] \times [-2.5, 2.5]$$

A plot of this function with its three local minima is shown in Figure 23.

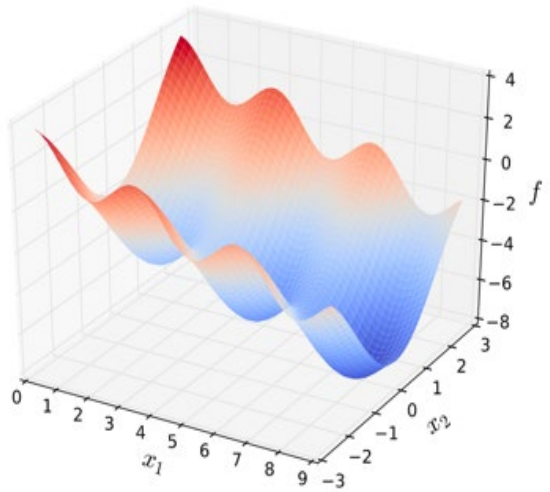


Figure 23. The Ursem01 Function.

The set $\mathcal{P}$ contains $N = 15$ sampling points from the 2-dimensional Sobol sequence. Figure 24 contains a mapping of the objective function values.

$$f: \begin{bmatrix} v_0 = (0.0, -2.5) \\ v_1 = (4.6, 0.0) \\ v_2 = (6.9, -1.25) \\ v_3 = (2.3, 1.25) \\ v_4 = (3.45, -0.625) \\ v_5 = (8.05, 1.875) \\ v_6 = (5.75, -1.875) \\ v_7 = (1.15, 0.625) \\ v_8 = (1.725, -0.9375) \\ v_9 = (6.325, 1.5625) \\ v_{10} = (8.625, -2.1875) \\ v_{11} = (4.025, 0.3125) \\ v_{12} = (2.875, -1.5625) \\ v_{13} = (7.475, 0.9375) \\ v_{14} = (5.175, -0.3125) \end{bmatrix} \rightarrow \begin{bmatrix} f_0 = 3.403 \\ f_1 = -6.275 \\ f_2 = -4.0651 \\ f_3 = -2.208 \\ f_4 = -3.3429 \\ f_5 = -4.051 \\ f_6 = -1.493 \\ f_7 = -3.674 \\ f_8 = -3.591 \\ f_9 = -2.191 \\ f_{10} = -2.606 \\ f_{11} = -5.062 \\ f_{12} = -0.601 \\ f_{13} = -6.239 \\ f_{14} = -6.044 \end{bmatrix}$$

Figure 24. Objective function values using sampling points from Sobol sequence.

From Definition 4 above, we have $\mathcal{H}^0$ from $\mathcal{P}$. Definition 5 enables us to construct $\mathcal{H}$ using Delaunay triangulation to find a set of connected edges. The edges are then directed according to Definition 6. Definition 7 enables us to find the minimiser set, which in this case is $\mathcal{M} = \{v_1, v_7, v_{13}\}$. Figure 25 is the resulting structure, which highlights the domain of s.t. (v_1).

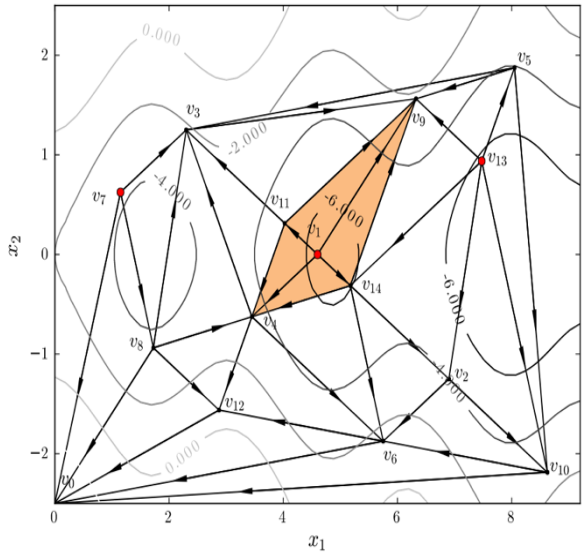


Figure 25. A directed complex $\mathcal{H}$—asimplicial approximation for an objective function.

Increasing the sampling size to $N = 150$ and repeating the procedure produces the complex in Figure 26.

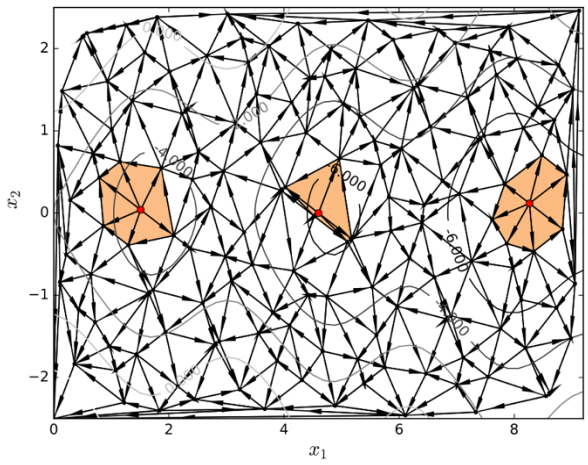


Figure 26. A directed complex $\mathcal{H}$—asimplicial approximation for an objective function with 150 vertices.

This has different minimiser vertices that are better approximations to the local minima, but $|\mathcal{M}|$ is unchanged. This points to the SHGO property: if the number of initial sampling points is adequate, $|\mathcal{M}|$ ceases to grow with increasing N, which provides a heuristic for the number of sampling points needed to approximately map the local minima of a merit function.

3.3. User Interface and Documentation Improvements

In relation to making the workings of the code more understandable, I added the following prior to the code block containing the three main techniques: The optimization approaches are RBFOpt Global Optimization, Iterative Nelder-Mead, and Simplicial Homology Global Optimization using Nelder-Mead at the local minima. RBFOpt produces results in about 2 hours, depending on your machine. The two array arguments passed to the RbfoptUserBlackBox class define the bounds of the black box and correspond to minimum and maximum feasible values for the design vector. Iterative Nelder-Mead does take a while; however, in the iterate function in the above code block, you may change the “xatol” and “fatol” parameters to relax the termination condition. These correspond to the absolute error in the design vector and its merit function evaluation between iterations such that the optimization will terminate. Simplicial Homology is my preferred method as it finds approximations to local minima and then uses iterative Nelder-Mead at each of these to find the global minimum. This method is theoretically guaranteed to find the global minimum when using the 'simplicial' sampling method. However, for a merit function as complex as ours (involving trajectory simulation), it is inefficient. The “sobol” sampling method will approximate the global minimum with an execution time of about 2 hours. I have also added comments to the code related to setting (black box/design vector) boundaries.

3.4. Initial Design Modifications

In coordination with Hayden Reinhold from the PSAS airframe team, I have updated the initial template.ork OpenRocket file to approximate the current design. This involved modifying component weights and lengths, along with using an approximate thickness to model our isogrid plate bulkheads as having uniform density, demonstrated in Figure 27.

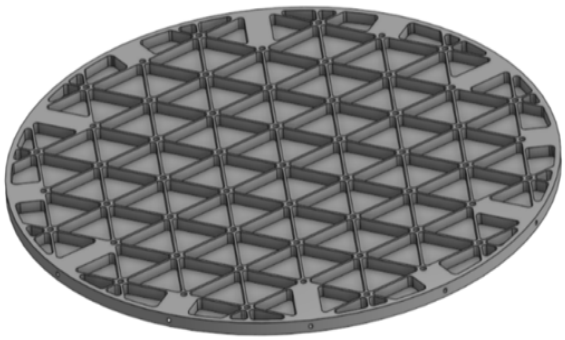


Figure 27. Isogrid plate bulkhead.

The updated initial design fed into the optimizer produced the diagram found in Figure 28.

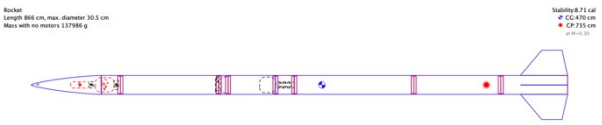


Figure 28. Updated initial design.

The main changes were made to the weight/length of Nosecone, Electrical Recovery System (ERS), N2 tank/Reaction Control System (RCS), Avionics/Camera module, Liquid Oxygen (LOX) tank, and fin can. A SHGO simulation with this updated model resulted in global coverage of the design space, as in Figure 29.

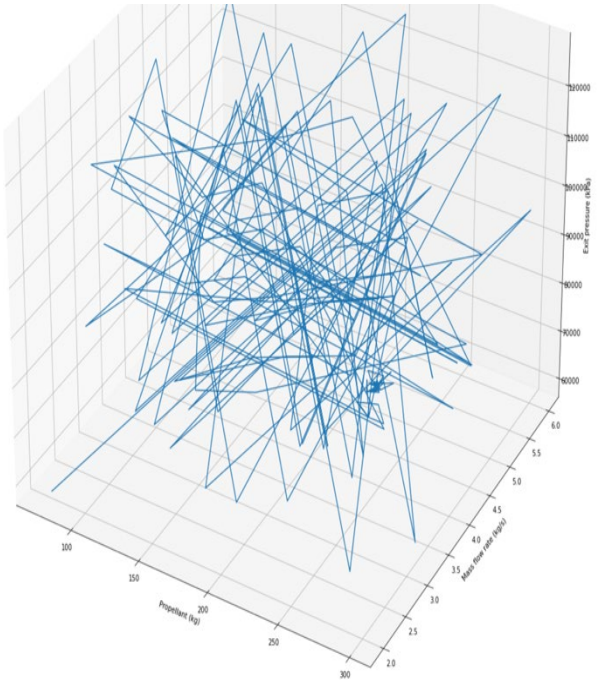


Figure 29. SHGO coverage of design space with updated design.

We met the majority of our constraints; however, manifesting a 10.6 kN engine poses a problem, displayed in Figure 30.

```
CONSTRAINTS
----------------------------
Largest angle of attack (c.f. < 8.0)         = 5.2807 deg
L/D ratio (c.f. < 25.0)                       = 27.2469
fin flutter ratio (c.f. > 1.0)                          = 26.5080
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.7082
max acceleration (c.f. < 15.0)                = 6.4626 gs
TWR at lift off (c.f. > 2.0)                 = 1.8866
Lowest stability margin caliber (c.f. > 2.0) = 2.2576
speed when leaving launch rail (c.f. > 22.0)  = 25.7842 m/s
altitude at apogee (c.f. > 105.0)              = 95.6293 km
LFETS LOX velocity (c.f. < 9.144)     = 5.5395 m/s
LFETS IPA velocity (c.f. < 9.144)     = 4.6314 m/s
design thrust (ground level) (c.f. < 10000)    = 10580.0105 N
```

Figure 30. SHGO constraint satisfaction with updated design.

Our apogee estimate is slightly conservative, so 95.6 km is excellent. Trajectory information indicates a successful launch is possible with the design in Figure 31.

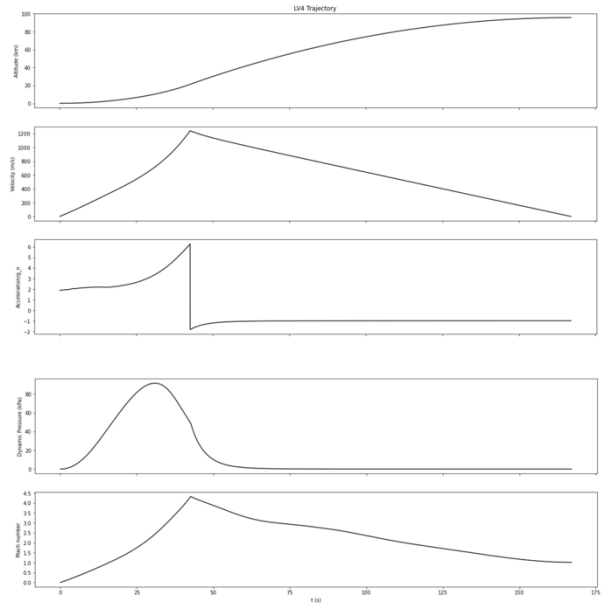
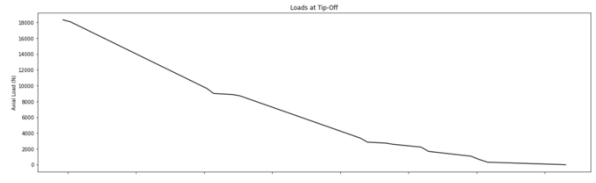
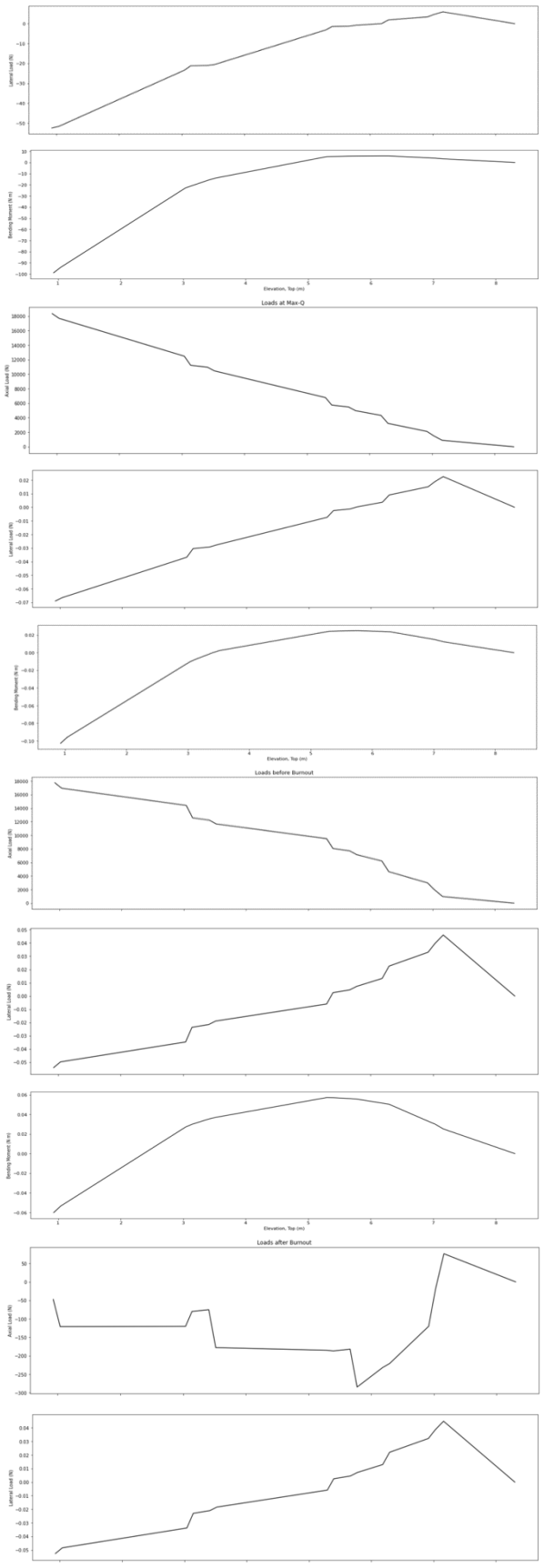


Figure 31. LV4 Trajectory Information with SHGO Approach.

3.5. Structural Analysis Output

Related to the goal of improving the MDO via providing structural analysis output, I added a new notebook that ported relevant code from the structural model notebook. This code contains a structural analysis function that calculates the axial and lateral loads along with bending moments at launch (tip-off), maximum aerodynamic pressure (max Q), and before and after engine burnout. The axial forces consist of friction along the body, parasitic drags related to each passthru module, and drag coefficient contributions. Lateral load is calculated via summing normal forces, and the bending moment is calculated considering the shear at the top of each component along with the lateral load at the middle. Using the structural plot function in the Display_Information notebook, the main MDO notebook now outputs the graphs seen in Figure 32.





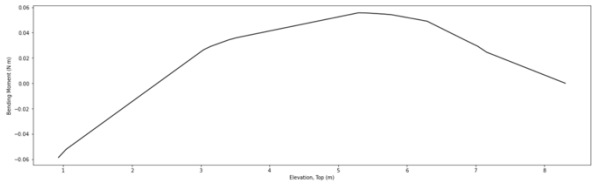


Figure 32. Structural analysis information added to MDOoutput.

3.6. Other Modifications

In order to make weight reductions easier, Peter McCloud (a scientist affiliated with NASA Aerothermodynamics) suggested that a pie chart with the mass breakdown be added to the MDO output. Figure 33 displays a code snippet and Figure 34 the result.

Mass breakdown: Propellant, Propulsion Modules, Pressurant (N2) and Airframe

```
labels = 'Propellant', 'Pressurant(N2)', 'Propulsion', 'Airframe', 'Other'
propellant = sim.m_prop[0]
pressurant = sim.LV4.rcs_tank.gas_mass
propulsion = 4.24 # from post-flight mass budget
airfrm_diameter = sim.LV4.diameter
airfrm_length = sim.LV4.length
volume = ((np.pi * ((airfrm_diameter/2) ** 2) * airfrm_length)
          - (np.pi * (((airfrm_diameter/2) - AIRFRAME_THICKNESS) ** 2) * airfrm_length))
airframe = volume * 2700 # density of 6061-T6 aluminum in kg/m^3
total = sim.LV4.GLOW
other = total - (propellant + pressurant + propulsion + airframe)
sizes = [(propellant/total) * 100, (pressurant/total) * 100, (propulsion/total) * 100, (airframe/total) * 100, (other/total) * 100]
explode = (0, 0.1, 0.12, 0, 0)

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```

Figure 33. Mass breakdown code added.

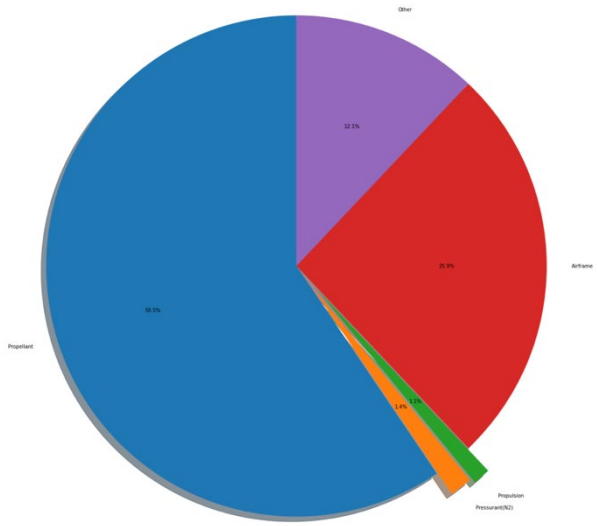


Figure 34. Mass breakdown pie chart.

The only non-trivial component of this is calculating the airframe mass. The volume of the tube was first calculated given the existing airframe length, diameter, and thickness

variables. Given the 2700 kg/m^3 density of 6061-T6 aluminum, the mass is calculated. The pie chart was constructed with reference to the matplotlib documentation (*Basic Pie Chart — Matplotlib 3.5.2 Documentation*).

Another issue is related to the engine thrust constraint. It is not feasible for the engine to have a thrust much greater than 6 kN. Since the thrust constraint is lenient, I experimented with setting the CONS_THRUST parameter to 5.5 kN. Figure 35 contains the results using Simplicial Homology with the simplicial sampling method (guaranteed optimality) and COBYLA local minimization.

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 6.4060 deg
L/D ratio (c.f. < 25.0)                       = 31.9445
fin flutter ratio (c.f. > 1.0)                        = 33.6321
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.6618
max acceleration (c.f. < 15.0)                = 6.9777 gs
TWR at lift off (c.f. > 2.0)                  = 1.1000
Lowest stability margin caliber (c.f. > 2.0) = 4.0010
speed when leaving launch rail (c.f. > 22.0)  = 20.0737 m/s
altitude at apogee (c.f. > 105.0)              = 90.2172 km
LFETS LOX velocity (c.f. < 9.144)    = 5.5849 m/s
LFETS IPA velocity (c.f. < 9.144)    = 4.7673 m/s
design thrust (ground level) (c.f. < 5500)    = 9192.0443 N
```

Figure 35. Constraint satisfaction with 5.5 kN engine thrust constraint.

The result is a roughly 9 kN engine with 90 km apogee. Reducing the constraint to 5.4 kN, I found that the optimal apogee was a scant 14 km. The sweet spot for the constraint is 5.49698 kN, which produces an apogee of 83 km with an engine thrust of 9 kN, shown in Figure 36.

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 6.4671 deg
L/D ratio (c.f. < 25.0)                       = 31.9447
fin flutter ratio (c.f. > 1.0)                        = 34.5250
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.6618
max acceleration (c.f. < 15.0)                = 6.8493 gs
TWR at lift off (c.f. > 2.0)                  = 1.0711
Lowest stability margin caliber (c.f. > 2.0) = 3.9664
speed when leaving launch rail (c.f. > 22.0)  = 19.8191 m/s
altitude at apogee (c.f. > 105.0)              = 83.6215 km
LFETS LOX velocity (c.f. < 9.144)    = 5.5057 m/s
LFETS IPA velocity (c.f. < 9.144)    = 4.6997 m/s
design thrust (ground level) (c.f. < 5496.98)    = 9065.8888 N
```

Figure 36. Constraint satisfaction with optimized engine thrust constraint.

Building a launch rail greater than 10 meters is difficult; therefore, I ran the simulation with the LAUNCH_TOWER parameter set to 9.8 m, shown in Figure 37.

Launch Parameters

```
# Launch constants
# Vertical Launch Area at Spaceport America
LAUNCH_SITE_ALT = 1381 # m, altitude of launch site above sea level, from freemaptools.com/elevation-finder.htm
LAUNCH_TOWER    = 9.8 # launch rail height in m
LAUNCH_SITE_LOC = [32.9406637137444, -106.92170420379703] # dec deg N, E from google maps
AZ_PERTURB = 158.19822500605724
EL_PERTURB = 0.3421796303215663
```

Figure 37. Launch Parameters with reduced launch rail length.

The optimal CONS_THRUST parameter in this case is somewhere between 5599.6 and 5599.7. Figure 38 displays the results for those two options.

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 4.9819 deg
L/D ratio (c.f. < 25.0)                       = 19.2188
fin flutter ratio (c.f. > 1.0)                        = 28.9802
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 1.4887
max acceleration (c.f. < 15.0)                = 3.2883 gs
TWR at lift off (c.f. > 2.0)                  = 2.0768
Lowest stability margin caliber (c.f. > 2.0) = 3.2218
speed when leaving launch rail (c.f. > 22.0)  = 20.0714 m/s
altitude at apogee (c.f. > 105.0)              = 14.2009 km
LFETS LOX velocity (c.f. < 9.144)    = 3.2942 m/s
LFETS IPA velocity (c.f. < 9.144)    = 2.8119 m/s
design thrust (ground level) (c.f. < 5599.6)    = 5664.9528 N

CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 6.6632 deg
L/D ratio (c.f. < 25.0)                       = 31.9341
fin flutter ratio (c.f. > 1.0)                        = 33.4302
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.6618
max acceleration (c.f. < 15.0)                = 7.2042 gs
TWR at lift off (c.f. > 2.0)                  = 1.2110
Lowest stability margin caliber (c.f. > 2.0) = 3.8711
speed when leaving launch rail (c.f. > 22.0)  = 15.3504 m/s
altitude at apogee (c.f. > 105.0)              = 77.8307 km
LFETS LOX velocity (c.f. < 9.144)    = 5.8874 m/s
LFETS IPA velocity (c.f. < 9.144)    = 5.0255 m/s
design thrust (ground level) (c.f. < 5599.7)    = 9672.8167 N
```

Figure 38. Constraint satisfaction to find sweet spot of CONS_THRUST parameter.

The apogee is dismal for the first and the engine thrust is too high for the second. With a 9.8 m launch rail, along with the CONS_THRUST parameter set to 5599.67, we have an apogee of roughly 83 km and engine power of 9.8 kN. The result is still too powerful of an engine, so we need to experiment with the initial fin parameters to reduce the required thrust. We will investigate the velocity off the launch rail related to the engine thrust. The "event_manager" function shown in Figure 39 contains the desired parameter and Figure 40 shows the related launch velocity analysis code.

```
if (not rkt.off_tower and np.linalg.norm(new[1] - state_list[0][0][1]) >= LAUNCH_TOWER and
    np.linalg.norm(state[1] - state_list[0][0][1]) < LAUNCH_TOWER) or rkt.tip_off_steps != 0:
    if rkt.tip_off_steps == 0:
        rkt.off_tower    = True
        rkt.tower_index  = len(state_list)
        rkt.launch_speed = np.linalg.norm(state[3])
```

Figure 39. Velocity off launch rail (rkt.launch_speed parameter).

```
%run OpenRocket_Interface.ipynb

init_array = [M_PROP, MDOT, P_E, LOX_TANK_P, IPA_TANK_P, ENG_P_CH]
ipa_wt, of_ratio, p_ch, Tc, MW, gamma, _ = propellant_optimizer(init_array[5])
sim = trajectory(False, 0, 0, 0, 0, 0, 0, 0, init_array[0], init_array[1], init_array[2],
                 THROTTLE_WINDOW, MIN_THROTTLE,
                 RCS_MDOT, RCS_P_E, RCS_P_CH,
                 BALLAST, 0.762, 0.3302, 0.6981317007977318, 0.4064, 0.003175, CON_NOSE_L, #enter larger fin parameters
                  init_array[3], init_array[4], RIB_T, NUM_RADL_DVSNS,
                 AIRFRM_IN_RAD, ipa_wt, of_ratio, init_array[5], Tc, gamma, MW,
                 [0, 0, AS_PERTURB, EL_PERTURB, True, 0, 0, 0, 0, 0, 0, True],
                         0.025, False, 0.045, False, False)

[-2.14689996e-04  1.22471226e-04  3.49119566e+00]

print(sim.LV4.launch_speed, "m/s")
print(sim.thrust[0], "N")

23.47499314763607 m/s
14134.541128133678 N
```

Figure 40. Launch velocity analysis code.

With the standard fins and unoptimized initial design vector, we have a launch velocity of 23.47 m/s and engine thrust of 14 kN. We would like to reduce the engine thrust to around 10 kN, which means the optimized design should be < 7 kN. The plan was to drop a rectangular fin can, meaning that only the fin root and tip chord lengths would be changed. Increasing the fin root and tip chords by 8 m, the velocity off the launch rail decreases to 19.3 m/s; however, the engine thrust remains unchanged at 14 kN.

3.7. Fin Staging/Optimization

Fin geometry is defined by the root, tip, sweep angle, semispan, and thickness, as in Figure 41.

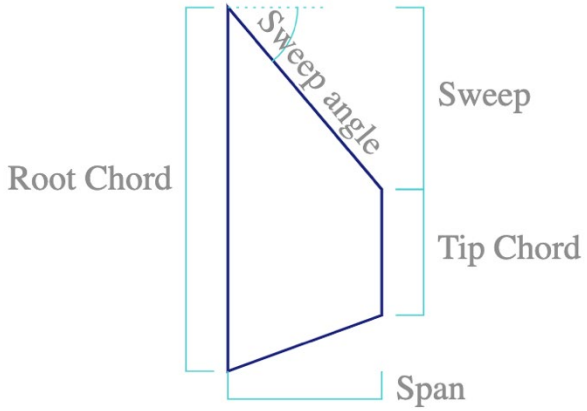


Figure 41. Typical Fin Nomenclature (Barrowman Package — Barrowman 0.0.1 Documentation).

To simulate dropping a fin can mid-flight, we create a second rocket object with new fin parameters. The idea here is to reduce the required engine thrust by starting out with larger fins, and then the smaller fins will reduce drag in the upper atmosphere. The function pictured in Figure 42 performs fin resizing.

```
def fin_resizing(LV4, root, tip, sweep, span, thickness, mass_red):
    new_rocket = LV4
    new_rocket.fin.root = root
    new_rocket.fin.tip = tip
    new_rocket.fin.sweep_angle = sweep
    new_rocket.fin.semispan = span
    new_rocket.fin.thickness = thickness
    new_rocket.mass = new_rocket.mass - mass_red
    return new_rocket
```

Figure 42. Fin resizing function.

In the trajectory simulation notebook, I have modified the event_manager, time_step, integration, and trajectory functions to accommodate fin staging, in particular the code seen in Figure 43.

```
if fin_staging and (state[1][2] >= stage_drop_ECEF):
    rkt = fin_resizing(rkt, root, tip, sweep, span, thickness, mass_red)
```

Figure 43. Fin staging condition in event manager function.

The stage_drop_ECEF—Earth-centred, Earth-fixed coordinate system ('Earth-Centred, Earth-Fixed Coordinate System')—parameter determines the altitude at which we drop the fin can module. To convert from altitude to ECEF, the function pictured in Figure 44 is used.

```
def altitude_to_ECEF(alt):
    return (alt + 6.371e3)

drop_ECEF = altitude_to_ECEF(50) # enter drop altitude here
```

Figure 44. Altitude to ECEF for entry in trajectory function.

The trajectory function now includes parameters related to fin staging, seen in Figure 45.

```
def trajectory(fin_staging, stage_drop_ECEF, stage_root, stage_tip, stage_sweep, stage_span, stage_thickness,
               mass_red, m_prop, mdot, p_e,
               throttle_window, min_throttle,
               rcs_mdot, rcs_p_e, rcs_p_ch,
               ballast, root, tip, sweep, span, thickness, con_nose_l,
               tank_p_o, tank_p_f, rib_t, num_radl_dvsns,
               airfrm_in_rad, ipa_wt, of, p_ch, T_ch, ke, mm, perturbations,
               dt, adaptive=False, tol=0.005, descend=False, early_return=False, recovery=False):
```

Figure 45. Trajectory function arguments.

Figure 46 displays the documentation added to the main MDO notebook.

```
Fin Staging - the first 8 arguments of the trajectory function are:
1) The boolean variable fin_staging, indicating whether or not you would like to simulate this
2) stage_drop_ECEF, enter drop altitude in kilometers into altitude_to_ECEF function
3) stage_root, root chord of new fin
4) stage_tip, tip chord " "
5) stage_sweep, sweep angle " "
6) stage_span, semispan " "
7) stage_thickness, thickness " "
8) mass_red, mass of initial larger fin can
```

Figure 46. Fin staging simulation instructions.

To test the code, we drop the ECEF at 50 km and leave the fin parameters unchanged, with zero mass reduction, as seen in Figure 47.

```
trajectory(True, drop_ECEF, 0.762, 0.3302, 0.6981317007977318, 0.4064, 0.003175, 0
```

Figure 47. Fin staging test arguments.

The simulation produces the same output as was produced prior to adding this functionality, indicating that the code is sound.

To find the ideal altitude for dropping the larger fin can, we must find the point at which the smaller fins are passively stable. McCloud suggested that this be done when the ratio of the center of pressure to the center of mass is around 2. The stability_margin in the MDO calculates this information, shown in Figure 48.

```
stability_margin = (rkt.CoM[2] - update[1][5][6]) / rkt.diameter
```

Figure 48. Center of mass vs. center of pressure metric.

Figure 49 contains the code to find the ideal drop time.

Stability Margin: Find ideal drop time

```
# given 0.025 time step calculate the time at which we should drop the fins
def find_drop_time(stability_list):
    closeness = []
    for item in stability_list:
        closeness.append(abs(item - 2.0))
    index = closeness.index(min(closeness))
    return (index * 0.025)
```

Figure 49. Time after launch to drop the fin can.

Now, the MDO has a component that simulates performance with the smaller fins (post dropping) to find the altitude at which they are passively stable, as seen in Figure 50.

```
smaller_fin_sim = trajectory(False, 0, 0, 0, 0, 0, 0, 0, x[0], x[1], x[2],
          THROTTLE_WINDOW, MIN_THROTTLE,
          RCS_MDOT, RCS_P_E, RCS_P_CH,
          BALLAST, 0.762, 0.3302, 0.6981317007977318, 0.4064, 0.003175, CON_NOSE_L,
           x[3], x[4], RIB_T, NUM_RADL_DVSNS,
          AIRFRM_IN_RAD, ipa_wt, of_ratio, x[5], Tc, gamma, MW,
          [0, 0, AZ_PERTURB, EL_PERTURB, True, 0, 0, 0, 0, 0, True],
                    0.025, False, 0.045, False, False)

drop_time = find_drop_time(smaller_fin_sim.stability_margin)
time_diff = []
for item in smaller_fin_sim.t:
    time_diff.append(abs(item - drop_time))
min_index = time_diff.index(min(time_diff))

def altitude_to_ECEF(alt):
    return (alt + 6.371e3)

drop_ECEF = altitude_to_ECEF(smaller_fin_sim.alt[min_index])
```

Figure 50. Finding the ideal drop altitude.

The code shown in Figure 51 prints the altitude at which the fins are dropped, as well as the Mach number at that point.

```
time_diff_2 = []
for item in sim.t:
    time_diff_2.append(abs(item - drop_time))
min_index_2 = time_diff_2.index(min(time_diff_2))
print("Fin drop altitude: ", smaller_fin_sim.alt[min_index]/1000, "km")
print("Mach number upon fin dropping: ", sim.Ma[min_index_2])
```

```
Optimization done!
[-1.79217336e-04  1.02235629e-04  3.99217627e+00]
[-1.79217336e-04  1.02235629e-04  3.99217627e+00]
Fin drop altitude:  0.007434751872904598 km
Mach number upon fin dropping:  0.02662397778539716
```

Figure 51. Displaying fin drop altitude and the Mach number at that point.

In Figure 52, a plot of the stability margin versus time has been added in order to double-check the fin drop logic and find ideal fin sizing. This process, shown in Figure 53, involved modifications to the Trajectory_Simulation, Display_Information, and MDO notebooks, as the rocket_plot function required an additional parameter “stability” for the stability margin list.

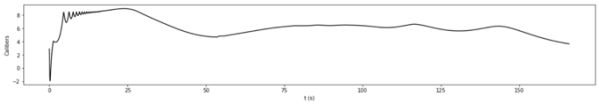


Figure 52. Calibers vs Time plot in MDO Notebook.

```
def rocket_plot(t, alt, v, a, F, q, Ma, stability,
```

```
ax7.plot(t[start:end], stability[start:end], 'k')
ax7.yaxis.major.locator.set_params(nbins=10)
ax7.set_ylabel("Calibers")
ax7.set_xlabel("t (s)")
```

Figure 53. Modifications to Display_Information enabling stability (calibers) vs. time plot.

The initial fluctuation (especially that of the negative section of the graph) does not seem to be physical; rather, when the wind is turned on in the trajectory simulation, and before the rocket has left the launch rail, there is an interval during which the wind velocity dominates the rocket’s velocity, meaning that the angle of attack is ~ 45 degrees or more. This interferes with the update [1][5][6] (indexing into multidimensional array) parameter in Fig. 48, which is the center of pressure and the seventh element of the array returned by the aero function in the Aerodynamics_Model notebook, as in Figure 54.

```
return np.array([force_body, torque_body, v0, dyn_press, Ma, alpha, CoP[2]
```

Figure 54. Array returned by the aero function in the Aerodynamics Model.

This explanation has been verified by turning off the wind and viewing the stability plot, seen in Figure 55.

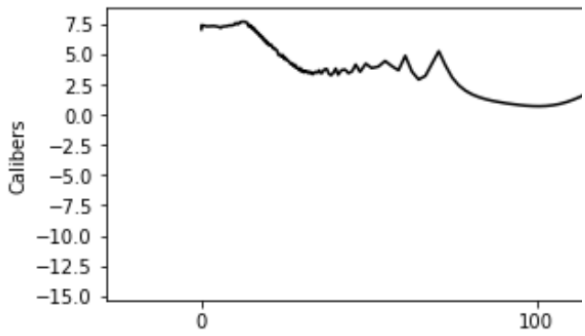


Figure 55. Stability plot with wind turned off (now strictly positive).

A plot of the first five seconds of the stability metric was added, given the fluctuating behaviour, with a line indicating the time at which the rocket has left the launch rail, as in Figure 56.

```
def Init_Stability(sim):
    off_rail = sim.t[sim.tower_index]
    plt.plot(sim.t[0:200], sim.stability_margin[0:200], 'k')
    plt.axvline(x=off_rail)
    plt.ylabel("Calibers")
    plt.xlabel("t (s)")
    plt.show()
    return None
```

Figure 56A. Stability metric code.

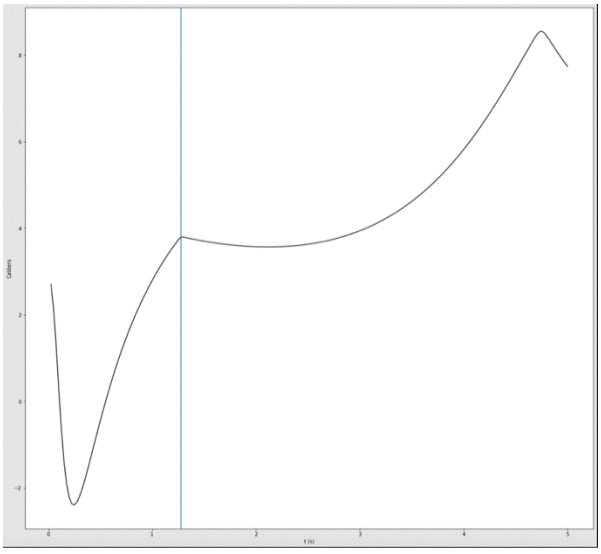


Figure 56B. Stability metric: first five seconds post-launch.

Given that the stability is quite high without fin staging, I reduced the fin root and tip chords by 0.2 m. The result, shown in Figure 57, is a minimum just above 2.

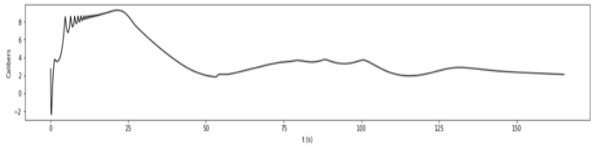


Figure 57. Calibers vs. time with reduced fin length.

On adding the fin root and tip chords to the design vector along with boundaries for these values incentivizing a larger root than tip chord, SHGO produces the design parameters pictured in Figure 58.

```
DESIGN VECTOR
-----------------------------
design total propellant mass                = 306.5331 kg
design unadjusted propellant mass           = 299.9971 kg
design mass flow rate                       = 5.4960 kg/s
design nozzle exit pressure                 = 57563.9675 Pa
total tankage length (after adjustment)     = 5.1717 m
design airframe diameter                    = 0.3048 m.
design airframe total length                = 9.7366 m.
design GLOW                                 = 445.9791 kg
design ballast mass                         = 2.0000 kg
conical part of nosecone length             = 1.0923 m
design fin root chord                       = 0.6531 m
design fin tip chord                        = 0.4723 m
design fin sweep angle                      = 40.0000 deg
design fin span                             = 0.4064 m
design fin thickness                        = 3.1750 mm
```

Figure 58. Optimized fin root and tip chords.

The stability was still quite high for this iteration, so McCloud recommended that the semispan be added to the design vector, shown in Figure 59.

```
[M_PROP, MDOT, P_E, LOX_TANK_P, IPA_TANK_P, ENG_P_CH, FIN_ROOT, FIN_TIP, FIN_SEMISPAN]
```

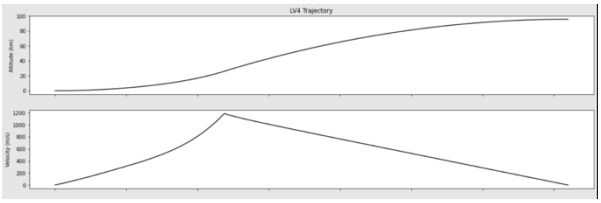
Figure 59. Modified design vector variables.

Figure 60 displays the boundaries given to SHGO.

```
shgo(f, bounds=[*zip([75,2,86346/1.5, 805000/2,805000/2, 344738*1.5, 0.8, 0.5, 0.15],
                     [300,6,86346*1.5,2.068e6, 2.068e6, 4.137e6, 1.8, 0.9, 0.9])],
```

Figure 60. Boundaries on design vector values given to SHGO.

Figure 61A–C contains the results.



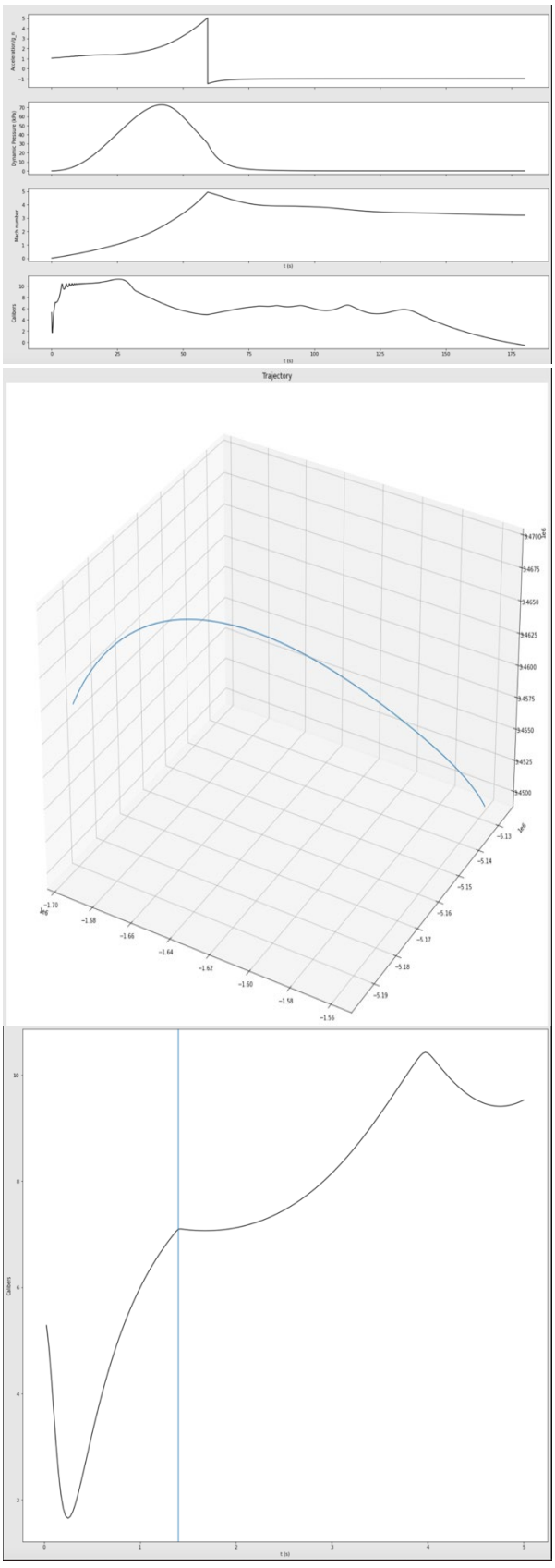


Figure 61A. Trajectory and stability metric: optimized fins.

```
DESIGN VECTOR
-----------------------------
design total propellant mass                = 306.2610 kg
design unadjusted propellant mass           = 299.7172 kg
design mass flow rate                       = 4.9497 kg/s
design nozzle exit pressure                 = 57563.9985 Pa
total tankage length (after adjustment)     = 5.1669 m
design airframe diameter                    = 0.3048 m.
design airframe total length                = 9.7318 m.
design GLOW                                 = 445.9012 kg
design ballast mass                         = 2.0000 kg
conical part of nosecone length             = 1.0923 m
design fin root chord                       = 0.5145 m
design fin tip chord                        = 0.1640 m
design fin sweep angle                      = 40.0000 deg
design fin span                             = 0.6942 m
design fin thickness                        = 3.1750 mm
```

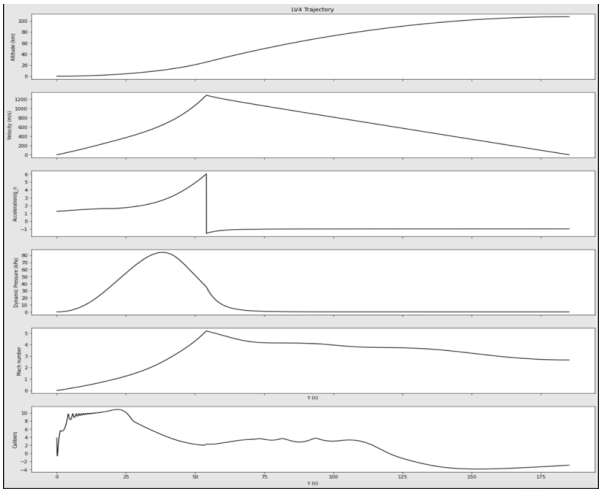
Figure 61B. Design vector: optimized fins.

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 6.9743 deg
L/D ratio (c.f. < 25.0)                       = 31.9285
fin flutter ratio (c.f. > 1.0)                          = 19.1983
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.6618
max acceleration (c.f. < 15.0)                = 6.7540 gs
TWR at lift off (c.f. > 2.0)                 = 1.0464
Lowest stability margin caliber (c.f. > 2.0) = 4.8811
speed when leaving launch rail (c.f. > 22.0)  = 14.3173 m/s
altitude at apogee (c.f. > 105.0)              = 95.5472 km
LFETS LOX velocity (c.f. < 9.144)    = 5.4400 m/s
LFETS IPA velocity (c.f. < 9.144)    = 4.6436 m/s
design thrust (ground level) (c.f. < 5800)    = 8961.1662 N
```

Figure 61C. Constraint satisfaction information indicating acceptable apogee with 9 kN engine: optimized fins.

Here, we achieve feasible fins and a 95 km apogee with a 9 kN engine. The root and tip chords are relatively small, making for a reasonable altitude (less drag), and the larger span corresponds to greater stability, which contributes to a reduced engine thrust.

The minimum stability is still quite high, so we scale down the optimized fin parameters to 80%, reducing the fin area. The result is an apogee of 107 km with a 9.8 kN engine, seen in Figure 62.



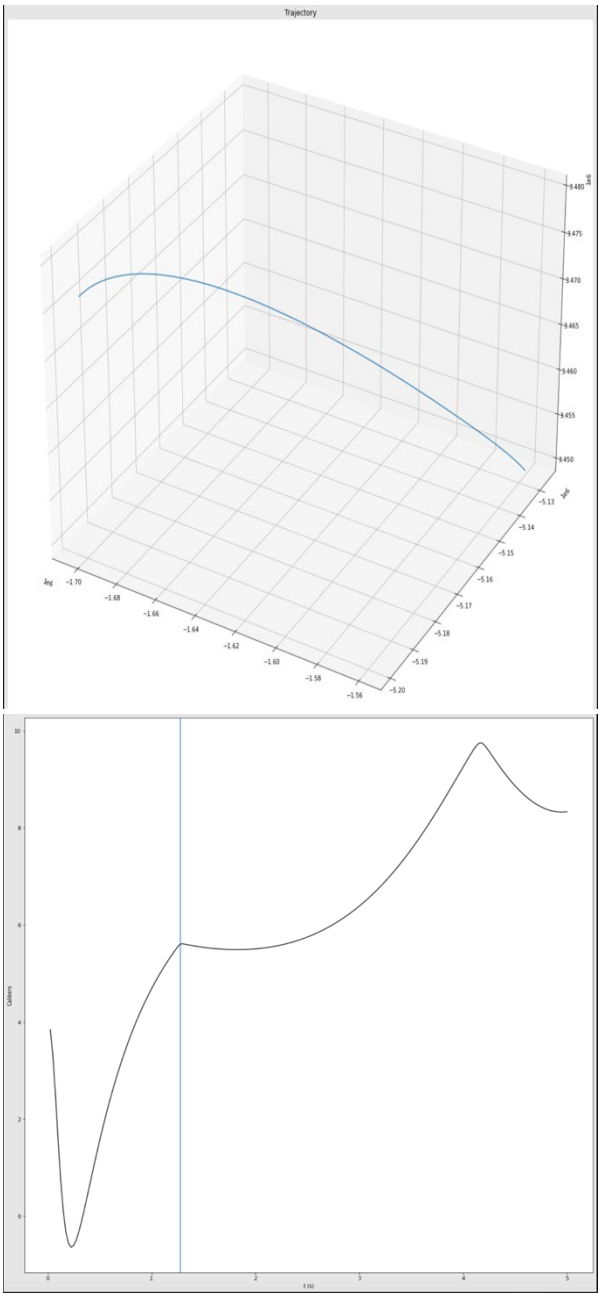


Figure 62A. Trajectory and stability metric: optimized fins scaled down to 80%.

```
DESIGN VECTOR
-----------------------------
design total propellant mass                = 306.2930 kg
design unadjusted propellant mass           = 299.7606 kg
design mass flow rate                       = 5.4331 kg/s
design nozzle exit pressure                 = 57563.9918 Pa
total tankage length (after adjustment)     = 5.1676 m
design airframe diameter                    = 0.3048 m.
design airframe total length                = 9.7326 m.
design GLOW                                 = 442.9361 kg
design ballast mass                         = 2.0000 kg
conical part of nosecone length             = 1.0923 m
design fin root chord                       = 0.4116 m
design fin tip chord                        = 0.1312 m
design fin sweep angle                      = 40.0000 deg
design fin span                             = 0.5554 m
design fin thickness                        = 3.1750 mm
```

```
CONSTRAINTS
-----------------------------
Largest angle of attack (c.f. < 8.0)         = 6.4908 deg
L/D ratio (c.f. < 25.0)                       = 31.9310
fin flutter ratio (c.f. > 1.0)                        = 24.5095
Sommerfield criterion (c.f. pe/pa >= 0.35)    = 0.6618
max acceleration (c.f. < 15.0)                = 7.6165 gs
TWR at lift off (c.f. > 2.0)                 = 2.2574
Lowest stability margin caliber (c.f. > 2.0) = 2.0206
speed when leaving launch rail (c.f. > 22.0)  = 15.5879 m/s
altitude at apogee (c.f. > 105.0)              = 107.6219 km
LFETS LOX velocity (c.f. < 9.144)    = 5.9712 m/s
LFETS IPA velocity (c.f. < 9.144)    = 5.0971 m/s
design thrust (ground level) (c.f. < 5599.67)    = 9805.8223 N
```

Figure 62B. Design vector and constraint satisfaction information indicating ideal apogee with 9.8 kN engine: optimized fins scaled down to 80%.

This is the ideal design. The 9.8 kN thrust may be realized by using two 5 kN engines. What remains unexplained is the 120 km vehicle drift (calculated by considering the coordinates at launch and apogee, as in Figure 63).

```
ADDITIONAL INFORMATION
-----------------------------
started at (lat, long, height): (32.94066362617013, -106.92170468292086, 1385.0087711460915)
ended at (lat, long, height): (32.607725193057234, -108.13965149845598, 108890.64253619876)
```

Figure 63. Coordinates at launch and apogee.

Setting the wind parameter to False in the trajectory function increases drift to 121 km, which means there is either an error in the MDO or the wind does not contribute to the drift. Therefore, I have added a plot of the angle of attack from when the rocket leaves the launch rail to just before engine burnout, pictured in Figure 64.

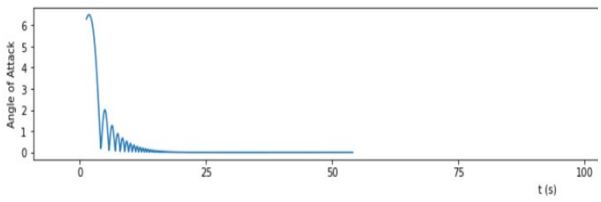


Figure 64. Angle of attack from leaving the launch rail to just before engine burnout.

Pictured is a non-zero angle of attack; however, it is not significant enough to constitute a 120 km drift. One potential hypothesis is that the drift is due to the combination of the Coriolis acceleration and the wind; however, upon setting these variables to zero and false, respectively, the drift actually increased to 126 km, which means that there must be some computational error in the MDO notebook.

4. Conclusion

In this work, I set out to improve and extend the existing open-source Multidisciplinary Design Optimization (MDO) simulation code. I evaluated different optimization approaches such as standard global optimization with RBFOpt, Nelder-Mead with different local minimization approaches, and Simplicial Homology Global Optimization. I found the SHGO approach to be the most effective and converged on the ideal design vector upon including fin parameters and scaling down given stability margin information. The 120 km vehicle drift was caused by some computational error in the notebook. This has been confirmed by running trajectory simulations with the design variables returned by the MDO, which indicate that the wind is the cause of the large vehicle drift. Further debugging and testing are required. However, the convergence of that ideal design vector serves as a theoretical guidance to the PSAS engineering team going forward.

Acknowledgements

Thank you to PSAS mentors Andrew Greenberg, Cory Gillette, Peter McCloud, and Max Eltzroth for their guidance and expertise throughout the duration of the project. Thanks also to Christabelle Dragoo and Denise Elder from the UO McNair team for their support.

Bibliography

Barrowman Package — Barrowman 0.0.1 Documentation. https://open-aerospace.github.io/barrowman/barrowman.html.

Basic Pie Chart — Matplotlib 3.5.2 Documentation. https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html.

Bonmin. 2014. COIN-OR Foundation, 2022. *GitHub*, https://github.com/coin-or/Bonmin.

Coin-or/Rbfopt. 2015. COIN-OR Foundation, 2022. *GitHub*, https://github.com/coin-or/rbfopt.

Design of Aerodynamically Stabilized Free Rockets, Military Handbook.

'Earth-Centered, Earth-Fixed Coordinate System'. *Wikipedia*, 6 Mar. 2022. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Earth-centered,_Earth-fixed_coordinate_system&oldid=1075493103.

Endres, Stefan C., et al. 'A Simplicial Homology Algorithm for Lipschitz Optimisation'. *Journal of Global Optimization*, vol. 72, no. 2, Oct. 2018, pp. 181–217. *Springer Link*, https://doi.org/10.1007/s10898-018-0645-y.

Martins, Joaquim R. R. A., and Andrew B. Lambe. 'Multidisciplinary Design Optimization: A Survey of Architectures'. *AIAA Journal*, vol. 51, no. 9, 2013, pp. 2049–75. *American Institute of Aeronautics and Astronautics*, https://doi.org/10.2514/1.J051895.

Minimize(Method='Nelder-Mead') — SciPy v1.8.0 Manual. https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html#optimize-minimize-neldermead.

Monte, Vaughn. *Over-Under Expanded Nozzle - Propulsion 1 - Aerospace Notes*. https://aerospacenotes.com/propulsion-1/over-under-expanded-nozzle/.

Scipy.Interpolate.Interp1d — SciPy v1.8.1 Manual. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html.

Scipy.Optimize.Shgo — SciPy v1.8.0 Manual. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.shgo.html.

Shyy, Wei, et al. 'Global Design Optimization for Aerodynamics and Rocket Propulsion Components'. *Progress in Aerospace Sciences*, vol. 37, no. 1, Jan. 2001, pp. 59–118. *ScienceDirect*, https://doi.org/10.1016/S0376-0421(01)00002-1.

Yao, Wen, et al. 'Review of Uncertainty-Based Multidisciplinary Design Optimization Methods for Aerospace Vehicles'. *Progress in Aerospace Sciences*, vol. 47, no. 6, Aug. 2011, pp. 450–79. *ScienceDirect*, https://doi.org/10.1016/j.paerosci.2011.05.001.