ASTRALab

After Action Report

Project: Starlift MDO

Your Name: Max Luo

Your Email: mgl87@cornell.edu, max.luo@yahoo.com

Date last updated: 5/18/2024

Brief Problem Description:

Starlift is a multifaceted project attempting to establish a multi-spacecraft resource network within the space environment surrounding Earth and the moon for the United States government. With no existing system in place, one issue that must be addressed is how to design spacecraft for specific missions within these space environments to fulfill certain mission objectives.

A previous project by PhD student Giuliana Hofheins proposed a multidimensional design optimization (MDO) approach to minimizing the cost of an active debris removal program, aimed at deorbiting or otherwise disposing of 22 spent rocket bodies in sun-synchronous orbit (SSO). This MDO project considered certain design variables as parameters to be varied in search of the lowest-cost option, subject to certain constraints and constant-held variables to enforce realistic options for the proposed mission.

My project for the semester involved taking Giuliana's project and adapting it to be more modular for Starlift. This involved interpreting and deconstructing the existing code architecture, adapting it to accept and accommodate a wider variety of missions that may pertain to Starlift. Furthermore, because Giuliana's original code was so catered towards her specific debris removal mission, there

were numerous places that were locked in to hard-coded elements. A great amount of effort was spent in removing these hard-coded elements to better the modularity for future integration with Starlift as a whole.

Due to the large scope of the project, the current iteration of the MDO code is incomplete. While it is functional in that it retains the usability of Giuliana's original code and therefore gives the same results, it is not ready for full use in Starlift. This document will detail both the current functionality and overall architecture of the existing code, plus the steps that still need to be taken to ensure improved integration with Starlift's overall mission.

Documentation

Top-level architecture

To begin with, it is most useful to understand the current state of MDO. Figure 2 is a comprehensive flowchart of the way that variables within MDO are handled. The larger containers (MDOrun, GenerateParameter, etc.) reference existing .m MATLAB files, each serving either as an overall executor or a specific function. This diagram is purposefully obtuse and overly detailed; it shows a flow-through of variables that are sometimes hard-coded or specifically defined within specific code blocks, demonstrating the need for complete rebuilding. Not shown is the Cost block, which estimates the total spacecraft and program cost, or the optGA algorithm, which is a genetic algorithm that seeks the lowest cost given a set of parameters and design variables.

Figure 1a is a simplified diagram for the overall current code architecture, removing the complexity of cross-reliant variables from the diagram. Note the clear separation between parameters and design variables, and the tightly coded requirements that flow through into each subsequent coding block.

The proposed architecture to move towards is in Figure 1b, which removes the hierarchical structure between power, propulsion, and other modules to introduce greater flexibility in mission planning depending on whatever mission planners prefer to keep constant or variable. At the moment, MDO optimizes on total program cost. However, with careful preplanning of code modularity, MDO can be used to optimize on other optimizations, such as minimum fuel, minimum mass, or minimum time.

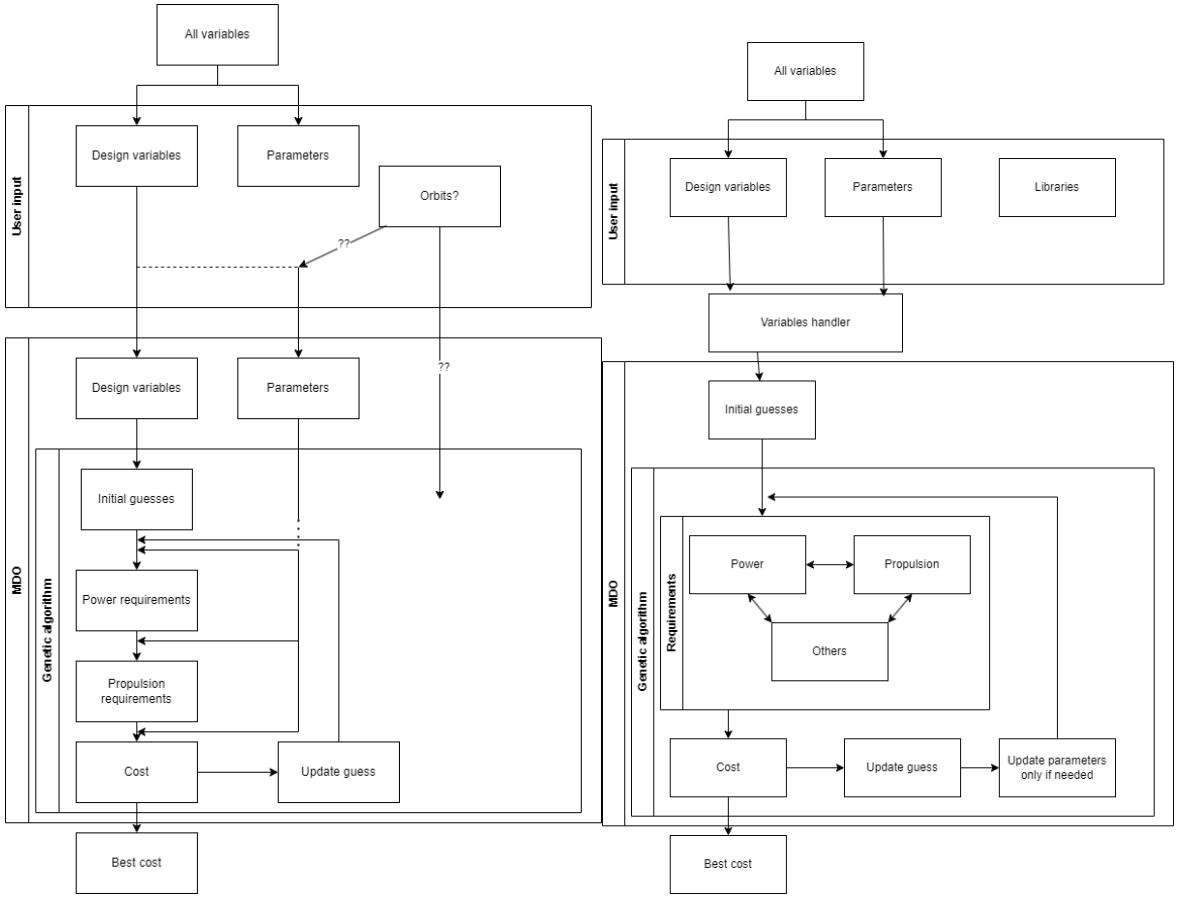


Figure 1a (left): Existing MDO code architecture from original Hofheins active debris removal mission.

Figure 1b (right): Proposed MDO code architecture to be used for Starlift.

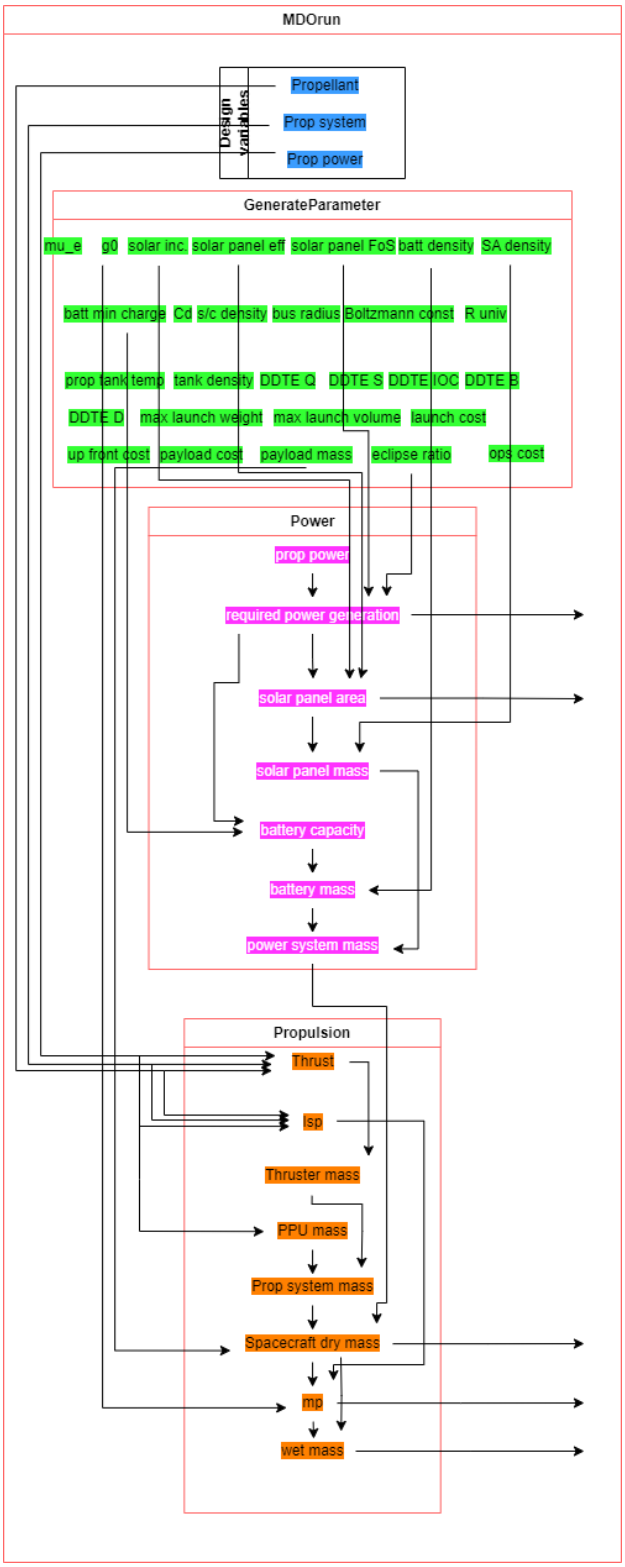


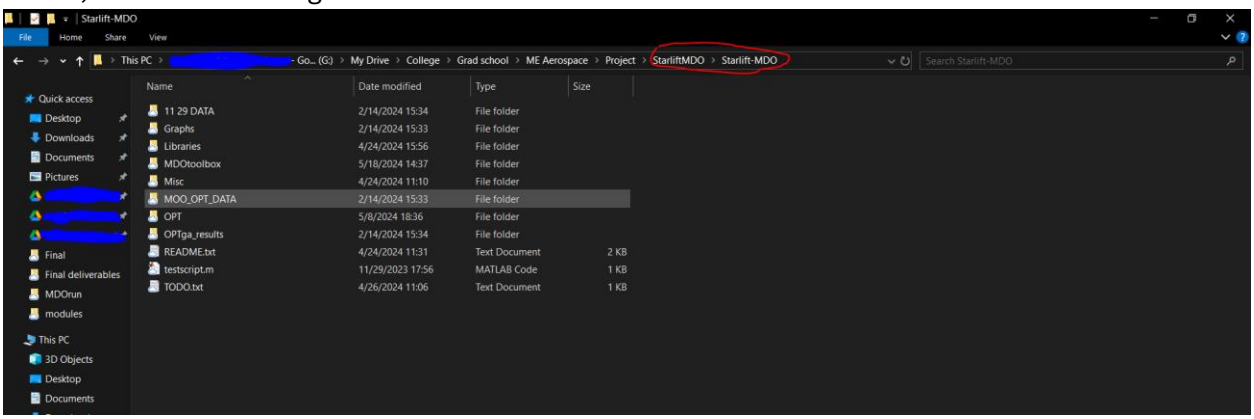
Figure 2: Comprehensive diagram of variables within the functions as shown in Figure 1a. Note the open-ended arrows, which flow into a not-shown Cost block.

Using MDO

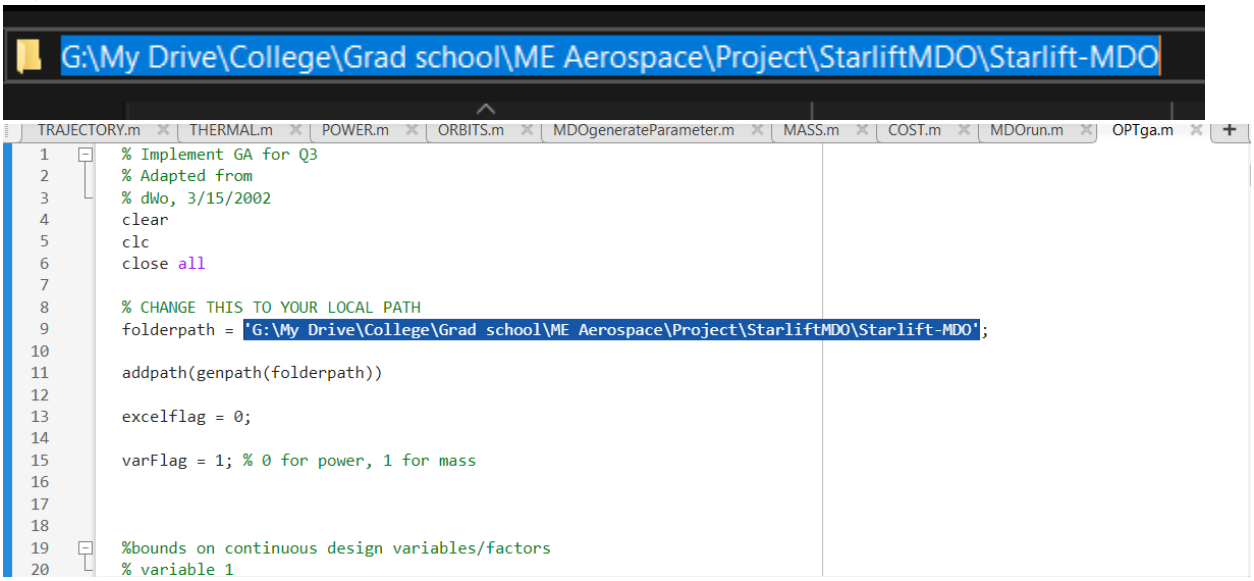
Currently, MDO is set up to borrow most of Giuliana's models for propulsion, mass, and cost set up for her active debris removal mission, based on a simple orbit transfer from low Earth orbit (LEO) to geosynchronous orbit (GEO). With parameters and design variables locked in, running MDO as-is should result in a proposed lowest cost for this specific mission.

To use MDO as-is, follow the following steps:

1. Create a local repository from which you will be working, based on Starlift's shared code repository. As of 5/18/2024, the home directory for this project is called StarliftMDO, which contains three separate versions of MDO. The active directory will be Starlift-MDO. In other words, we will be working in StarliftMDO\Starlift-MDO.



2. Navigate to \OPT, and open OPTga.m. Change folderpath to be your local path containing StarliftMDO\Starlift-MDO. If working on Windows, the easiest way is to copy the File Explorer address bar.



3. Run OPTga.m to acquire the lowest-cost estimate for a LEO to GEO mission.

Adjusting parameters and design variables

The intended way to adjust variables and/or design parameters is to do so in one centralized location, to emphasize either user readability and/or ease of integration with other systems and project groups. There are several places to make adjustments to these parameters and design variables, depending on one's specific use cases.

Currently, the specified design variables, taken from Giuliana's original code, are:

- Discretized selection of electric thruster (Hall thruster versus ion thruster)
- Thruster operating power, bounded between 500 W and 30,000 W
- Discretized selection of propellant choice (Xe, Kr, or Ar)

Note that these design variables are all in the propulsion space. At the moment, it is non-trivial to add modularity to discretized variables beyond these selections, due to a combination of MDO's cross-referenced nature and the coding/library infrastructure required to facilitate discretized selection. Improvements to this section will likely require further code restructuring; for the sake of familiarization with MDO, it is recommended to leave this section as-is for now.

All other parameters and variables are intended to be handled by MDOGenerateParameter.m and PARAMETERS_LIBRARY.xlsx. MDOGenerateParameter.m creates a MATLAB struct that contains all of the different constant-value parameters for use in the optimization genetic algorithm. Originally, this .m file was where all parameters were defined; now, all of these parameters are defined in the accompanying .xlsx file.

PARAMETERS_LIBRARY.xlsx is located under the Libraries folder. The function of each column is as such:

- MATLAB FIELD NAME: the name of struct field. For example, the solar panel efficiency is labelled as "n". If the struct name is "parameter", one would call the solar panel efficiency using "parameter.n".
- FIELD: a physical description of what the MATLAB field is. For example, "To" is not very descriptive for an end user, while "Propellant tank [temperature]" is.
- DEFAULT: a value that MDO will default to, if an initial value is not set in OVERRIDE.
- OVERRIDE: the value that MDO will use, if a value is provided. The combination of DEFAULT and OVERRIDE allows users to specify, in wide, general, terms, what they expect their mission(s) to mostly consist of. The OVERRIDE column forces a change in a parameter without having to modify code, or to manually keep track of deviations away from common parameter values between missions.
- OPTTARGET: an as-of-yet unused designator. Intended to be a binary 0/1 modifier indicating whether a parameter should be considered a design variable, or a constantly-held parameter.
- OPTBOUNDS: an as-of-yet unused designator. Intended to be used in conjunction with OPTTARGET. If OPTTARGET were to be equal to 1, OPTBOUNDS would specify the bounds on the variable space. Recall how for Giuliana's code, the thruster operating power was bounded between 500 and 30,000 W; a future version of MDO would have moved that definition to this column.
- PROPFLAG: an as-of-yet unused designator. Due to the complexity of the propulsion system and specifying it for any individual mission, PROPFLAG would have flagged any parameter

as having “special” considerations. Based on the flag value, MDO would have treated it differently, perhaps referencing specific table/library values and/or specific, edge-case functions. Due to simultaneous development of a propulsion library by other team members, PROPFLAG may be entirely removed from future versions.

- NOTES ON FIELDS: general notes for any field not covered in FIELD. Generally contains information on units and/or sources.
- OTHER NOTES: unused. Can be repurposed or deleted.

There is an additional tab in PARAMETERS_LIBRARY.xlsx named “propChoice”. This tab contains propellant selections and their molar masses. This page is intended to be expandable without significantly requiring additional development of MDO.

Note that the intent is for any variable with human influence to be placed in a library of some sort for ease of end-user adjustment. There are several values that are considered immutable. These are usually, if not always, physical constants, and are hard-coded into MDOGenerateParameter. Examples include Earth’s gravitational parameter, Earth’s radius, the Stefan-Boltzmann constant, etc.

Also note that hard-coded parameters have not been fully sterilized. For example, additional functions not found in Giuliana’s original code include a dedicated mass block and a more user-friendly trajectories block; one function that was removed from her original code is a thermal block. Because of the additional infrastructure created via the addition of new code blocks, many more parameters were hard-coded after the creation of these blocks, to maintain rapid code development without compromising functionality. Similarly, some parameters may be more appropriately placed in other blocks. Our group was not responsible for developing spacecraft trajectories, but having some trajectory model was useful in ensuring code functionality. As a result, there are several places where temporary measures were set in place, intended to be replaced in the full MDO version. Future versions of MDO should sterilize these hard-coded elements and temporary parameters to improve user modularity.

MATLAB code description

The above section describes the intended use of MDO. This section describes what each block literally does. The order taken will be the order in which code is executed, followed by addenda for code blocks that are deprecated or awaiting integration. The order is: MDOGenerateParameter → POWER → PROPULSION → COST, all of which are run through MDOrun, which itself is optimized over within OPTga.

MDOGenerateParameter

Reads the Excel file PARAMETERS_LIBRARY.xlsx, populating a struct that will flow through each subsequent code block. Also generates some immutable, constant values that are not designed to be altered by an end user. Creates a secondary struct intended for use by the propulsion block, which holds the name and molar mass of possible propellant selections. Currently, MDO assumes a continuous-thrust transfer from one circular orbit to another; altitudes and radii are therefore converted from km to m, before calculating the orbital periods of both of these orbits.

POWER

Reads the parameter struct created in MDOGenerateParameter, extracting relevant information. Calculates the relevant values expected of a power system, such as battery capacity, max power generation, system and subsystem masses, etc.

PROPULSION

Thus far, the most complex block. Attempting to reduce rigidity by moving several functions to helper .m files, based on the performance of Hall thrusters and DC ion thrusters. Uses inputs from power subsystem and known orbital requirements (e.g. momentum change) to calculate thrust and specific impulse of either Hall or DC ion thrusters. Because the original MDO was developed with electric propulsion in mind, the current propulsion block assumes electric propulsion (as opposed to chemical or hybrid systems). Thus, the propulsion block calculates important elements related strictly to propulsion as a generic concept (e.g. propellant mass, propulsion system mass), but also includes elements that arguably may be better suited to other sections (e.g. mass of the power processing unit), or would otherwise require some built-in logic based on the specific makeup of the target spacecraft design.

This section features significant hard-coded elements, and is in need of the most rework for full implementation with the rest of MDO and Starlift. In particular, with parallel efforts being made to create a library based on propulsion systems other than just electric systems, it becomes increasingly paramount to ensure flexibility in the specification of the propulsion system.

COST

While this section is among the longest, it is also the simplest. There is no one way to create a cost model, and cost models for different subsections are retrieved either from existing literature, commercial quotations, or engineering extrapolation. The cost module retrieves all information generated from the rest of MDO up to this point and processes the relevant information through the cost models it has for specific components. After considering the additional costs not related to the spacecraft hardware (e.g. R&D costs, launch costs, operation costs), the cost block tabulates all of these values for further post-processing by other functions.

Because cost models evolve with time and technical development, ensure that the cost models used in this block are sufficiently up to date.

MDOrun

The main function that executes all of the previous .m files. Requires design variables as an input, before generating a parameter struct via MDOGenerateParameter and passing all of the above values to each subsequent function. Note how MDOGenerateParameter is called *inside* the function, suggesting another potential issue wherein computer memory and executable instructions are being reserved for it with every iteration. One optimization here would be to reconfigure the calling of MDOGenerateParameter such that it does not have to be newly generated with each call.

MDOrun reports tabulated costs of subsystems and related (e.g. launch, R&D, etc.), plus the estimated mission time and spacecraft mass. MDOrun itself is iterated within the genetic algorithm that seeks optimized space program costs.

OPTga

While MDOrun processes the information handed to it, OPTga is the one that hands the information to MDOrun. It begins by defining the scope of the design variables, before issuing initial values to populate its first generation. The genetic algorithm itself resembles a rough approximation of ecological processes: a population of possible designs are considered, with the traits of the best-performing individuals combined to evaluate the fitness of the next generation. This process is repeated for as many generations as the user sees fit. The final deliverable is the lowest cost reported after the execution of all of these generations.

There are four values that can be changed in OPTga. Because these values are more pertinent to code execution than to spacecraft development, it is appropriate to have them be configured in code as opposed to a user-friendly interface. Specifically, these values control the way that the genetic algorithm functions:

- deltaX: The step size taken to perturb any design variable (a “mutation”). This step size is scaled based on the requirements of the specific variable.
- NG: the number of generations that OPTga will iterate over.
- mrate: The mutation rate. Indicates how often a mutation happens within an individual. The mutation rate specifies how often a mutation happens, while deltaX specifies by how much an individual mutates.
- N: population size per generation.

The vast majority of this function derives from open-source functions elsewhere on Mathworks. It is recommended to not modify this function any more than necessary, unless you are intimately familiar with genetic algorithms and optimization algorithms.

THERMAL

Deprecated function from old code. Specified the mass and mass fractions of a thermal control system for the active debris removal mission.

TRAJECTORY

Deprecated function from old code. Calculated orbital parameters and total momentum change required for individual spacecraft in charge of active debris removal mission.

ORBITS

Temporary function designed as placeholder for later integration with orbital mechanics block. Reads a starting and ending altitude, calculating the momentum change required for either an impulsive maneuver or a low-thrust, continuous maneuver. Designed to be replaced for better integration with the rest of the Starlift team.

MASS

Currently disconnected from the remainder of MDO. Intended to provide a single collection point for all subsystem masses, similar to how COST is a single collection point for all subsystem costs. Currently, mass is handled piecemeal by individual code blocks. By collecting all mass in one place, MDO would be able to more easily handle optimization for mass as opposed to having to search for all individual mass expressions. Having both a cost and a mass block would allow a user to switch optimization target from one to the other. Currently, MASS relies on work done from a 1991 NASA paper written by Kent M. Price, David Pidgeon, and Alex Tsao. This function is currently nonfunctional with many hard-coded elements. It is recommended to offload references to mass from other system blocks to MASS, especially if the flow of variables and information from block to block can be made more flexible.

Future steps and final notes

With the flexibility requested by US government agencies of Starlift, plus the inherent systems engineering challenges of designing any particular spacecraft for flexible mission types, designing and developing effective MDO code proves useful. However, the need for a wide variety of missions and spacecraft drives MDO to be equally flexible. Thus, further development of MDO should always maintain awareness of user modularity and flexibility, even if at the expense of concision. Currently, MDO has moved strongly towards being more modular by emphasizing user interactivity and decision-making before first code execution. Recommended actions are as follows:

1. Fully establish block structure to facilitate easy flow of information and variables.
    a. Ensure cross-compatibility within requisite team members.
2. Minimize/reduce all hard-coded, rigid structures.
3. Implement better, flexible approach more similar to Figure 1b than to current structure.
4. Update models on mass, cost, etc.

Further, informal documentation can be found within the comments of each code block, many of which describe the issues within. To thoroughly tackle future development of Starlift MDO, it is recommended to thoroughly understand not just this report, but also every comment made within every code block.

Summary

MDO is functional, in that it works and generates an output.

MDO is approaching user modularity, but is not yet there.