DEVELOPMENT OF SOFTWARE TOOLSUITE FOR RAPID GENERATION OF SPACECRAFT REQUIREMENTS FROM MISSION CONSTRAINTS FOR SPACECRAFT PROPOSAL DEVELOPMENT

by

Eric Sahr

A Thesis Submitted to the Faculty of the

SYSTEMS AND INDUSTRIAL ENGINEERING DEPARTMENT

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2017

STATEMENT BY AUTHOR

SIGNED: Eric Sahr

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_________________________________ 08/17/2017
Roberto Furfaro Date
Associate Professor of Systems and Industrial Engineering

Contents

Table of Tables

Table of Figures

Abstract

The development, testing, and results of a software suite for automated development of spacecraft requirements is discussed. This software suite will enable mission scientists and engineers to rapidly develop spacecraft requirements from a previously-developed set of mission requirements. The software, written in MATLAB, is controlled by a Master Controller script, whose purpose is to accept inputs from the user and call subfunctions responsible for designing the various spacecraft subsystem requirements. The software was tested through the use of a series of arbitrarily-generated mission requirements, with the test results being examined for potential feasibility and reasonableness. Case studies are examined which show the efficacy of the software suite to accurately generate spacecraft requirements. The first case study examines a set of software-developed spacecraft requirements intended to meet the mission requirements of the Mars Reconnaissance Orbiter. The second case study examines a set of infeasible mission requirements to the planet Uranus, in an effort to demonstrate that the software will generate realistic, but infeasible, spacecraft requirements when the mission requirements are themselves infeasible. Both case studies generate reasonable spacecraft requirements as expected, with the direct comparison between the Mars spacecraft resulting in very similar preliminary spacecraft designs. This software suite will enable spacecraft scientists and engineers to quickly assess the feasibility of mission concepts and proposal designs through rapid development of spacecraft requirements.

1 Introduction

NASA and other space agencies currently use a proposal-based approach for the early development of potential mission and spacecraft concepts. Typically, a call for proposals will be issued by the respective agency, and various entities (including NASA field centers, academic institutions, and private companies) will spent significant time, money, and energy to develop a feasible mission proposal and spacecraft concept [1]. While much of this analysis is performed using advanced software tools, no software currently unifies the design of the various spacecraft subsystems to rapidly and iteratively develop a set of potential spacecraft requirements. This thesis aims to improve this early development process by automating the tasks associated with determining mission feasibility, allowing scientists and other non-engineering individuals to rapidly assess whether a particular mission concept is feasible with current spacecraft technology. If the mission concept is shown to be too unrealistic, the individual can rapidly make changes to ease the mission constraints, rather than wasting valuable resources exploring a mission concept that cannot be supported by current spacecraft technology.

The current method of developing mission proposals has the advantage of being highly accurate and reliable. Since engineers are intimately involved in the development of these spacecraft concepts, it is safe to argue that the overall mission goals and constraints have been carefully considered when sizing the spacecraft subsystems and developing spacecraft subsystem requirements. The disadvantage of this current method is the fact that a potential proposal must have the funding required to hire a staff of engineers to assess these feasibilities prior to being awarded a contract to fly the mission.

The scope of this software is solely in the development of the spacecraft subsystem requirements. The mission, budget, and schedule requirements are all elements that must be separately developed prior to using this tool. Additionally, the spacecraft trajectory must be separately developed outside of this tool.

The documentation associated with this software tool includes a discussion and justification of the software source code, followed by an explanation of the testing used to validate that source code. Two case studies are explored, one comparing the software output to the spacecraft requirements for the Mars Reconnaissance Orbiter, and another exploring infeasible mission constraints for a spacecraft traveling to Uranus. A discussion of potential errors as revealed by each of the case studies are discussed. Lastly, a list of potential future improvements to the software is outlined. The software source code appears as an appendix to this document.

2 System Architecture

2.1 Tools

2.1.1 MATLAB

The software suite is written entirely in MATLAB. MATLAB is a platform optimized for solving engineering and scientific problems rapidly. MATLAB, short for Matrix Laboratory, is developed and maintained by Mathworks, Inc [2]. This platform was used to develop and test all of the subfunctions discussed throughout the 'System Architecture' section. MATLAB has been used for the development of many scripts and functions in the spaceflight in the past, and is referenced by many undergraduate and graduate-level textbooks as the platform of choice for solving problems in the realm of spaceflight engineering.

2.2 Master Controller Script

The software suite is governed by a single master script, which takes in inputs from the user, calls subfunctions which design spacecraft subsystems, manages data between the subfunctions, and aggregates the subfunctions to build the initial spacecraft estimate.

2.2.1 System Requirements

The first task of the master controller script is to gather the necessary mission requirements. In a nominal proposal, the mission requirements will come from the science team, and spacecraft engineers involved with the proposal would design to those requirements. The master controller script serves a similar function. The master controller script requests the following mission requirements in order to arrive at the spacecraft requirements.

2.2.1.1 Requirements

2.2.1.1.1 Spacecraft Delta-V

This requirement is driven by the trajectory design of the potential mission. Since this is out of the scope of the current software suite, the master controller script requests this information from the spacecraft team. This parameter requests the Delta-v of the entire mission that will be handled solely by the spacecraft. Trajectory injection from the launch vehicle is excluded. This parameter is in the units of kilometers/second.

2.2.1.1.2 Destination Name

This requirement prompts the user to identify the destination object of the spacecraft. Due to the limitations of the software, currently only one destination can be identified. In the event a spacecraft is expected to visit multiple solar system objects, the user of the software should perform their own preliminary analysis to determine which destination will drive the spacecraft requirements. One of the suite's subfunctions contains pertinent data for the 8 planets, Pluto, and Earth's moon. The script also allows for Deep Space missions, where no particular destination is expected to drive requirements. Further details about the script handling destination data will appear in its own section.

2.2.1.1.3 Destination Minimum Range

This requirement is the minimum range of the spacecraft to the destination planet during nominal mission operations. This parameter is in kilometers.

2.2.1.1.4 Payload Mass

This requirement is the estimated mass of the spacecraft payload. This parameter does not include any spacecraft subfunctions, and should only include the estimate of the mass for any science instruments or other science payloads. This parameter is in kilograms.

2.2.1.1.5 Payload Power

This parameter represents the estimated power requirements of the science payload for the spacecraft during nominal peak science operations. This power requirement does not include any necessary spacecraft subsystems (ie thermal). This parameter is in watts.

2.2.1.1.6 Payload Volume

This parameter represents the estimated volume requirements for the science payload. This volume requirement does not include any necessary spacecraft subsystems. This parameter is in cm^3.

2.2.1.1.7 Payload Count

This parameter represents the number of science instruments on board the spacecraft. This parameter is used to estimate certain other requirements that will be levied on the spacecraft subsystems by the science payload.

2.2.1.1.8 Spacecraft Pointing Requirement

This requirement is the maximum rotation rate of the spacecraft in order to meet spacecraft safety and mission objectives. This requirement drives many attitude control requirement estimates. This parameter is in degrees per second.

2.2.1.1.9 Primary Mission Length

This parameter is the length of the spacecraft primary mission in days, not including outbound cruise to the destination.

2.2.1.1.10 Spacecraft Thermal Maximum

This parameter represents the maximum temperature that the spacecraft can endure. This would represent the component of the spacecraft who has the coldest maximum temperature, extrapolated to the entire vehicle. This parameter is in degrees Celsius.

2.2.1.1.11 Spacecraft Thermal Minimum

This parameter represents the minimum temperature that the spacecraft can endure. This would represent the component of the spacecraft which has the warmest minimum temperature, extrapolated to the entire vehicle. This parameter is in degrees Celsius.

2.2.1.1.12 Spacecraft Thruster Specific Impulse

This parameter sets the specific impulse of the thrusters on the spacecraft. The unit for this parameter is seconds. The software uses three different thruster sizes to accomplish the maneuvers that a spacecraft must typically accomplish. These three thruster sizes are described as ‘small’ (typically used for small attitude control system maneuvers), ‘medium’ (typically used for tasks where the attitude control system would not efficiently suffice, but where the large thrusters would be overpowered for the task at hand, such as some Trajectory Correction Maneuvers during Outbound Cruise phases of interplanetary missions), and ‘large’ (typically used for major maneuvers, such as orbit insertion at the destination of the spacecraft). The specific impulse for each of these three thruster sizes can be set separately from one another.

2.2.1.1.13 Spacecraft Flyby Flag

This parameter sets the “flyby flag.” This allows the user to decide whether the mission will be an orbiter at its destination, or a flyby mission of a particular destination. This flag affects how subfunctions execute. A flag of ‘0’ indicates the mission is an orbiter of the specified destination. A flag of ‘1’ indicates that the mission is a flyby mission of the specified destination. There are circumstances where it does not make sense to set the Flyby flag to 1, such as when the vehicle is intended as an Earth observer.

2.2.2 Subfunctions

Upon loading the requirements, the master script calls a series of subfunctions. These subfunctions perform preliminary analysis in sizing the mass and power requirements of the spacecraft subsystems needed in order to meet the mission requirements. Each subfunction is called, with Requirements data being passed to the subfunction. The subfunction performs its analysis, before passing updated requirements back to the main function. Once these requirements have been passed back, the main function updates its current estimates of the total mass and power requirements of the overall spacecraft, and passes these updated constraints to the next subfunction.

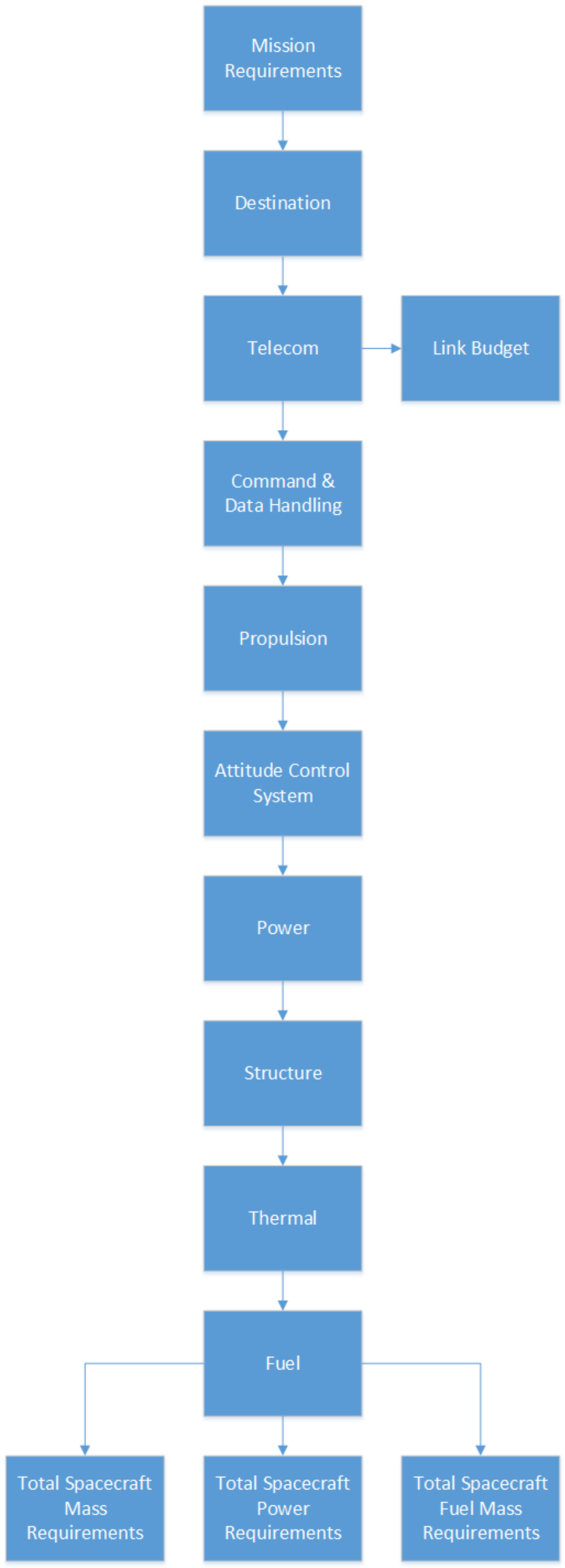

Figure 1 - A flow diagram illustrating the order of the subfunctions as called by the Master Controller Script, and the relationship between the Mission Requirements inputs, the subfunctions, and the outputs of Spacecraft Mass, Power, and Fuel Requirements.

A list and brief summary of each subfunction appears below:

2.2.2.1 Destination

This subfunction reads in the destination of the spacecraft as specified by the user, and loads in constants pertaining to that destination. An exhaustive list of these constants appears in the detailed description for this subfunction. These constants are passed back to the main function and loaded into the mission requirements passed to the subfunctions that design the spacecraft subsystems. If the destination has no data, the user will be prompted to provide this data about the destination prior to proceeding.

2.2.2.2 Telecommunications

This subfunction reads in mission requirements pertaining to range to Earth and data volume to design a preliminary telecommunications subsystem for the spacecraft. An optimization function balancing power required and antenna mass runs to optimize this subsystem. Total mass and power requirements for components of this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems. A link table is also produced.

2.2.2.3 Command & Data Handling

This subfunction reads in mission requirements as well as telecommunications capabilities to design a preliminary Command & Data Handling subsystem for the spacecraft. These requirements pertain to data volume and downlink bandwidth. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems.

2.2.2.4 Propulsion

This subfunction reads in mission requirements to design a preliminary Propulsion subsystem for the spacecraft. This subfunction does not design the fuel system (this is completed at the end of the Master

Controller script when the spacecraft mass and volume requirements can be better estimated), but rather only performs calculations regarding the number of thrusters required to meet mission requirements. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems.

2.2.2.5 Attitude Control

This subfunction reads in mission requirements to design a preliminary Attitude Control subsystem for the spacecraft. This subfunction reads in requirements pertaining to pointing, mission length, orbit insertion burn length, and estimated spacecraft mass/volume to design a preliminary attitude control subsystem. This subfunction also calculates the mass of the fuel required to perform the estimated attitude control burns. It does not calculate the mass or power requirements required to store the fuel. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems.

2.2.2.6 Power

This subfunction reads in mission requirements to design a preliminary Power system for the spacecraft. This subfunction reads in requirements pertaining to payload power, eclipse time at the destination, and distance from the sun. In the event that the spacecraft will travel far enough from the sun to make solar panels impractical, the subfunction will automatically perform calculations for an RTG-based power system instead. This is done for all spacecraft with destinations beyond Jupiter. This subfunction designs the solar panels and battery systems, and tabulates the mass and power requirements necessary for this subsystem. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems.

2.2.2.7 Structure

This subfunction reads in mission requirements to design a preliminary spacecraft structure to house the payload and other subsystems. This subfunction makes assumptions about the total volume of the spacecraft subsystems, in addition to reading in the payload volume requirements from the user. Knowing these requirements on the spacecraft, this subfunction designs a cylindrical spacecraft bus that could hypothetically house these components (with no assumptions made for the potential geometry or thermal constraints). The mass of the spacecraft structure is tabulated assuming a bus built from aluminum. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems. While a spacecraft structure could possibly have additional requirements for actuation, it is assumed that these are less than the science payload power requirements, and can be adequately handled by the power system as potential major actuation events would take place when the science payload was off.

2.2.2.8 Thermal

This subfunction reads in mission requirements, spacecraft requirements (so far), and destination constant parameters to design a preliminary thermal subsystem. The thermal subsystem is designed assuming that a passive system will be adequate for maintaining the spacecraft within acceptable thermal conditions at the destination. In the event that passive thermal systems are not sufficient to meet the spacecraft thermal requirements, the subfunction will automatically add spacecraft heaters to bring the thermal system into compliance with thermal requirements. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for other subsystems.

2.2.2.9 Fuel

Once all other subfunctions are sized and estimated, the final design subfunction is called which reads in mission requirements and spacecraft requirements to size an appropriate fuel system. The amount of fuel necessary to meet the requirements is tabulated, with additional margin added to account for spillage, loading error, and mass growth through the development lifecycle. A fuel system designed to store and deliver this fuel to the propulsion system is designed. Total mass and power requirements for the various components for this subsystem are tabulated and passed back to the main function for inclusion in calculations made for determining the final estimated spacecraft requirements.

2.3 Destination Subfunction

The Destination subfunction is the first subfunction executed off of the main script. This subfunction holds planetary destination data for Mercury, Venus, Earth, Earth's Moon, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto [3]. Additionally, the script is equipped to handle Deep Space and other destinations, but the user must provide additional data for the script to function properly.

```
function [Requirements] = Destination(Requirements)
```

The subfunction is written as a MATLAB function. It reads in the structure 'Requirements' which is first built in the Master Controller script. Once this subfunction is completed, it passes the structure 'Requirements' back to the Master Controller script.

```
if strcmp(Requirements.Destination.Name,'Mercury') == 1
    %Requirements.Destination.Name = 'Mercury';
    Requirements.Destination.IR = 4150;
    Requirements.Destination.Albedo = 0.106;
    Requirements.Destination.Radius = 2439.7;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (6.156 * 10^7);
    Requirements.Destination.SolarFlux = 9228;
    Requirements.Destination.mu = 2.2032 * 10^4;
    Requirements.Destination.SolarPressure = 3.05 * 10^-5;
```

Once the script has completed reading in the structure 'Requirements', it utilizes the parameter 'Requirements.Destination.Name' to determine which solar system destination the user desires. The subfunction performs a simple string comparison, if the names do not exactly match, it passes to the next elseif statement (representing a different possible destination). For destinations with known and consistent parameters (such as Mercury as shown above), the following parameters are hard-coded into the script:

2.3.1.1 Requirements.Destination.IR

This parameter represents the infrared radiation of the destination. It is in the units of Watts per meter squared [4].

2.3.1.2 Requirements.Destination.Albedo

The average albedo, the proportion of incident light reflected by a surface, of the destination, as observed by a hypothetical orbiting spacecraft [1].

2.3.1.3 Requirements.Destination.Radius

The radius of the destination, in kilometers [3].

2.3.1.4 Requirements.Destination.TransmitterMaxRange

The maximum range in kilometers of the spacecraft from the Earth at the destination. For Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto, this is determined by the hypothetical maximum distance at conjunction [3]. For these destinations, this is calculated with the following formula:

$$TransmitterMaxRange = EarthRangeToSun + DestinationRangeToSun$$

For the Earth, the TransmitterMaxRange is defined as Earth's Sphere of Influence radius.

For the Moon, the TransmitterMaxRange is defined as the Moon's maximum orbit distance from the Earth.

For Deep Space and other destinations, the user is prompted to enter the spacecraft's maximum range from the Earth.

```
Requirements.Destination.TransmitterMaxRange = input('What is the maximum range of the
destination to Earth in km?');
```

2.3.1.5 Requirements.Destination.SolarFlux

The solar flux at the destination, in W/m^2.

For Deep Space and user-specified destinations, the Solar Flux is calculated using the following equation: [5]

$$Solar\ Flux = \frac{Solar\ Luminosity}{Range\ To\ Sun}$$

2.3.1.6 Requirements.Destination.mu

The Standard Gravitational Parameter of the destination, in km^{3}s^{-2} [3].

2.3.1.7 Requirements.Destination.SolarPressure

The Solar Pressure at the destination, in (W-s)/m^3.

For all destinations, this is calculated by the following formula: [6]

$$Solar\ Pressure = \frac{Destination\ Solar\ Flux}{Speed\ of\ Light}$$

For destinations that are not hard-coded into the subfunction, the user is prompted to enter each piece of necessary data in order for the spacecraft subsystem requirement development to proceed.

```
else
    %Ask user to specify IR, Albedo, and Radius.
    Requirements.Destination.IR = input('What is the Orbit-Average IR of the destination in
W/m^2?');
    Requirements.Destination.Albedo = input('What is the Geometric albedo of the destination?');
    Requirements.Destination.Radius = input('What is the radius of the destination?');
    Requirements.Destination.TransmitterMaxRange = input('What is the maximum range of the
destination to Earth?');
    Requirements.Destination.SolarFlux = input('What is the solar flux of the destination?');
    Requirements.Destination.mu = input('What is the mu of the destination?');
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);
```

This section of the subfunction only runs if the destination as specified by the user does not match any of the above-mentioned destinations.

Once the Requirements for the destination is added to the 'Requirements' structure (which takes place at the end of each elseif statement), the 'Requirements' structure is passed back to the Master Controller script as an output.

2.4 Telecommunications Subfunction

Once all of the mission requirements have been loaded into the Master Controller, the script then begins to develop the spacecraft requirements. The first spacecraft requirements developed are the Telecommunications Requirements. This script utilizes a handful of assumptions about the telecommunications subsystem, all of which can be modified by the user if different assumptions are desired. These assumptions are: [1]

- Antenna Efficiency of 65%
- Antenna Frequency of 8.4 * 10^9 Hz
- Transponder weight of 7.6 kg
- Control Unit weight of 10.9 kg
- TWTA (traveling wave tube amplifier) weight of 6.2 kg
- RFS Components weight of 8 kg
- Medium Gain Antenna Weight of 2.1 kg
- Coax Cable weight of 7.8 kg
- Control Unit Power Requirement of 12.2 watts
- X-Exciter power requirement of 1.4 watts

- Receiver power requirement of 6.8 watts
- TWTA power requirement of 67 watts
- XS Down Converter power requirement of 2.1 watts

This subfunction is written as a MATLAB function, with the aforementioned ‘Requirements’ structure serving as the inputs, and TelecomMass and TelecomPower requirements as the output.

This subfunction calculates the Telecommunications subsystem requirements differently depending on the mass of the payload the spacecraft is expected to support. Very small payloads would potentially result in a very small spacecraft. This would cause the subfunction to generate unrealistically small antennae. To resolve this, the script was broken up into three different elseif branches.

The first branch is executed if the payload of the spacecraft is greater than 160 kg.

```
if Requirements.Payload.Mass > 160
syms x
f(x) = (AntennaEfficiency * (0.5*Requirements.Payload.Mass - 80 - 4 * x ^2) * (pi * x / ((3 *
10^8) / AntennaFrequency)^2));
df = diff(f,x);
AntennaRoots = solve(0 == df,x);
ARootsPos = double(AntennaRoots(AntennaRoots>0));
AntennaDiameter = max(0.4,ARootsPos);
LightweightAntennaMode = 0;
```

The script performs an optimization between antenna size and power requirements. This allows the script to reduce weight without putting unnecessary strain on the power system. This optimization is determined by the following equation: [1]

$$0 = Antenna\ Efficiency * \frac{1}{2} * Payload\ Mass - 80 - 4 * x^2 * \frac{\pi x}{(\frac{Speed\ of\ Light}{Antenna\ Frequency})^2} dx$$

This equation will result in multiple non-real answers. The script automatically parses the real answer, compares it to a minimum antenna size of 0.4 meters diameter, and sets the "LightweightAntennaMode" variable to 0.

The second branch is executed if the payload mass of the spacecraft is less than 27.999 kg.

```
elseif Requirements.Payload.Mass < 27.999
    AntennaDiameter = 0.4;
    LightweightAntennaMode = 1;
```

This branch assumes that a very small spacecraft will use a small, commercial-off-the-shelf antenna. Typically, these antenna have a diameter of about 0.4 meters [1]. This branch sets the "LightweightAntennaMode" variable to 1.

The last branch executes if the payload mass of the spacecraft is greater than 27.999 kg, but less than 160 kg.

```
else
    AntennaDiameter = 0.68;
    LightweightAntennaMode = 2;
```

Like the previous branch, this branch assumes that a small spacecraft will used a small, commercial-off-the-shelf antenna. This spacecraft will likely support a larger antenna, but still does not need a custom design. These antenna have a typical diameter of 0.68 meters [1]. This branch sets the "LightweightAntennaMode" to 2.

```
if LightweightAntennaMode == 0
    %Calculate weight using formula
    HGAAntennaWeight = max(2.1,2.89 * AntennaDiameter ^ 2 + 6.11 * AntennaDiameter - 2.59);
elseif LightweightAntennaMode == 1
    HGAAntennaWeight = 2.1;
elseif LightweightAntennaMode == 2
    HGAAntennaWeight = 7;
else
```

```
    %An error has occurred.
end
```

The “LightweightAntennaMode” automatically determines which calculation should be used to assess the mass of the antenna as designed.

When the mode is set to 0, this indicates that the Payload Mass was greater than 160kg, and a custom antenna was optimized for. In this mode, the subfunction performs a calculation to determine an estimated mass of the antenna based on the antenna diameter [1].

$$Estimated\ Antenna\ Mass = 2.89 * AntennaDiameter^2 + 6.11 * AntennaDiameter - 2.59$$

This equation was developed by examining diameter/mass ratios of antenna on previously flown spacecraft, and creating a best-fit equation to match that dataset [1].

When the LightWeightAntennaMode is set to 1, it indicates that a very small Payload Mass requirement was required, likely resulting in a small spacecraft. This mode uses a small commercial-off-the-shelf antenna, and thus the antenna weight is hard-coded into the function. This is also true when the LightweightAntennaMode is set to 2.

Lastly, if the LightweightAntennaMode is set to anything other than 0, 1, or 2 – the script aborts as an error has somehow occurred.

The script then calculates the power necessary to transmit using this antenna size. This equation is as follows: [1]

$$Power\ Transmitted = PayloadMass - 80 - 4 * AntennaDiameter$$

Next, the script takes the antenna information as designed, and adds it to the mass and power requirements for the rest of the telecommunications subsystem.

```
TelecomPower = ControlUnitPower + XExciterPower + ReceiverPower + PowerTransmitted + TWTAPower + 
XSDownConvertPower; %W
```

```
TelecomMass = TransponderWeight + ControlUnitWeight + TWTAWeight + RFSComponentsWeight +
HGAAntennaWeight + MediumGainAntennaWeight + CoaxCableWeight; %kg
```

Once these requirements are tabulated, they are passed as an output back to the Master Controller script. The Master Controller script then adds these numbers to the requirements of the spacecraft so far, which will inform further calculations in preliminary development of the spacecraft.

As an additional output, the subfunction also generates a nominal telecommunications link table, consisting of the pertinent information demonstrating the robustness of the telecommunications link between the spacecraft and the ground. Some of these values are assumptions, others are selected by the user, and the remainder are calculated from the rest of the link table values: [1]

- Frequency (Assumed: 8.4 GHz)
- Bit error rate (Assumed: 1 * 10^{-5})
- Range, km
- Symbol Rate (Assumed: 0.125 bps)
- Transmitter Power, watts
- Transmitter to antenna cable loss (Assumed: 0 dB)
- Transmitting Antenna Gain (Assumed: 67 dB)
- EIRP, dB
- Free Space Path Loss, dB
- Atmospheric Attenuation (Assumed: -0.17 dB)
- Polarization Loss (Assumed: -1 dB)
- Ground receiver gain (Assumed: 67 dB)
- Pointing loss (Assumed: -5 dB)
- Receiver cable loss (-1.95 dB)
- Total Received Power, dB

- System Noise Temperature (Assumed: 18 K)
- System Noise Density, dB/Hz
- Carrier power to total power ratio, dB
- Received carrier power, dB
- Carrier noise bandwidth (Assumed: 13 dB-Hz)
- Carrier signal to noise, dB
- Carrier signal to noise required by ground station (Assumed: 10 dB)
- Carrier link margin, dB
- Data power/total power ratio, dB
- Data power received, dB
- Data symbol rate, dB-Hz
- E_b/N_0 achieved, dB
- E_b/N_0 required (Assumed: 4.2 dB)
- Data link margin, dB
- Modulation Index, degrees

The Frequency, Bit Error Rate, Symbol Rate, Cable Loss, Antenna Gain, Atmospheric Attenuation, Polarization Loss, Spacecraft Antenna Gain, Pointing Loss, Received Cable Loss, Receiver Noise Temperature, Spacecraft Antenna Temperature, Carrier Noise Bandwidth, Carrier Noise Ratio Required, and E_b/N_0 Required, and Modulation Index are all assumed values in this subfunction. Future work would include enabling the user to fine-tune the telecommunications subsystem to better serve the requirements of the mission. For this software at this time, the default values for these parameters will suffice.

$$Spacecraft\ Range = Destination\ Transmitter\ Max\ Range$$

For computing the nominal link table, the spacecraft range is assumed to be the range from the Earth to the Destination at opposition.

$$Transmitter\ Power = Spacecraft\ Antenna\ Power$$

The optimization between antenna size and required transmitter power yields a value "Spacecraft Antenna Power" which is now used in the link table.

$$EIRP = Transmitter\ Power + Cable\ Loss + Antenna\ Gain$$

The EIRP (Effective Isotropic Radiated Power) is calculated as the summation of transmitter power, cable losses, and antenna gain [1].

$$Free\ Space\ Path\ Loss = 92.44 + 20 * \log(Frequency) + 20 * \log(Spacecraft\ Range)$$

The Free Space Path Loss is calculated by subtracting 20 times the log of the frequency, and 20 times the log of the maximum spacecraft range to the ground station from 92.44 [1].

$$Spacecraft\ Antenna\ Gain = 10 * \log(Antenna\ Efficiency * \frac{\pi * Antenna\ Diameter}{(\frac{Speed\ of\ Light}{Antenna\ Frequency})^2}$$

The Spacecraft Antenna Gain is 10 multiplied by the log of the Antenna Efficiency multiplied by π times the Antenna Diameter, divided by the Speed of Light divided by the antenna frequency, squared [7].

$$\begin{aligned} &Total\ Received\ Power \\ &\quad = EIRP - Free\ Space\ Path\ Loss + Atmospheric\ Attenuation + Polarization\ Loss \\ &\quad + Spacecraft\ Antenna\ Gain + Pointing\ Loss + Received\ Cable\ Loss \end{aligned}$$

The total received power is the summation of the EIRP, Free Space Path Loss, Atmospheric Attenuation, Polarization Losses, Spacecraft Antenna Gain, Pointing Losses, and Received Cable Losses [1].

$$System\ Noise\ Density = -228.6 + 10 * \log(System\ Noise\ Temperature)$$

The System Noise Density of the telecommunications link between the ground and the spacecraft is calculated as -228.6, plus 10 times the log of the cosine of the System Noise Temperature [1].

$$\frac{Carrier\ Power}{Total\ Power} = 20 * \log(\cos(Modulation\ Index))$$

The Carrier Power/Total Power value is calculated by multiplying 20 by the log of the Modulation Index [1].

$$Carrier\ Power\ Received = Total\ Received\ Power + \frac{Carrier\ Power}{Total\ Power}$$

The Carrier Power Received is the sum of the Total Received Power and the Carrier Power/Total Power values [1].

$$\begin{aligned}&Carrier\ Noise\ Ratio\ Received\\ &\quad = Carrier\ Power\ Received - System\ Noise\ Density - Carrier\ Noise\ Bandwidth\end{aligned}$$

The carrier noise ratio received value is the carrier power received, minus the system noise density, minus the carrier noise bandwidth [1].

$$\frac{Command\ Power}{Total\ Power} = 10 * \log(\sin(Modulation\ Index))$$

The Command Power/Total Power volume is 10 multiplied by the log of the sin of the modulation index.

$$Command\ Power\ Received = Total\ Received\ Power + \frac{Command\ Power}{Total\ Power}$$

The Command Power Received is the sum of the Total Received Power and the Command Power/Total Power [1].

$$Command\ Symbol\ Rate = -10 * \log(Symbol\ Rate) - 30$$

The Command Symbol Rate is 30 subtracted from 10 times the log of the Symbol Rate [1].

$$E_bN_0\ Achieved = Command\ Power\ Received + Command\ Symbol\ Rate - System\ Noise\ Density$$

The E_bN_0 Achieved value is the System Noise Density subtracted from the sum of the Command Power Received and the Command Symbol Rate [1].

$$Command\ Link\ Margin = \ E_bN_0\ Achieved - \ E_bN_0\ Required$$

The Command Link Margin is the E_bN_0 Achieved minus the E_bN_0 Required [1].

After completing these calculations, the subfunction has completed building the Link Table outlining the capabilities and margins of the Telecommunications subsystem. The subfunction ends and the script returns to the Master Controller script. Care must be taken when using the Telecom script, as it is possible with the present assumptions that negative margins will result. The user should consider altering characteristics of the telecom system (such as transmission frequency) to result in positive margin.

2.5 Command & Data Handling Subfunction

The Command & Data Handling subfunction is written as a MATLAB function, with the structure 'Requirements' being loaded as an input, and 'CDHMass' and 'CDHPower' as outputs.

This subfunction contains a number of hard-coded values, which serve as assumptions to the design of the subsystem. Future work on this subfunction would further optimize Command & Data Handling subsystem design to better match the needs of the spacecraft. These hard-coded values are: [1] [4]

- Independent Data Rate Equipment Mass: 30 kg
- Computer Mass: 2 kg
- Science Data Processor Mass: 15 kg
- Engineering Data Processor Mass: 10 kg
- Independent Data Rate Equipment Power: 20 watts

- Computer Power: 10 watts
- Engineering Data Processor Power: 5 watts

The subfunction designs small aspects of the overall Command & Data Handling subsystem. To determine the estimated power requirements of the Science Data Processor, the subfunction uses the following equation: [1]

$$Science\ Data\ Processor\ Power = 2 * Number\ of\ Payloads + 1$$

The data storage mass is calculated with the following equation: [1]

$$Data\ Storage\ Mass = 0.25 * Payload\ Data\ Storage\ Requirement$$

Lastly, the data storage power requirement is determined by the following equation: [1]

$$Data\ Storage\ Power = 1 * Payload\ Data\ Storage\ Requirement$$

Once all of these aspects of the subfunction have been determined, the requirements of the entire subsystem can be formulated:

```
CDHMass = IndependentDataRateEquipmentMass + ComputerMass + ScienceDataProcessorMass +
EngineeringDataProcessorMass + DataStorageMass;
CDHPower = IndependentDataRateEquipmentPower + ComputerPower + ScienceDataProcessorPower +
EngineeringDataProcessorPower + DataStoragePower;
```

These tabulated values are then passed back to the Master Controller script for inclusion in the overall spacecraft mass and power requirements so far. These values will be utilized to further design the remaining spacecraft subsystems.

2.6 Propulsion Subfunction

The Propulsion subfunction is responsible for designing the thrusters and engines that will be used to perform maneuvers with the spacecraft. The fuel system accompanying this subsystem is designed at a later step, as is the attitude control system.

The subfunction is written as a MATLAB function, with the structure ‘Requirements’ as an input. ‘PropMass’ and ‘PropPower’ are outputs.

The subfunction first determines the mass of each of the different types of thrusters the spacecraft will utilize. For large and medium thrusters, the equation governing this calculation is as follows: [1]

$$Thruster\ Weight = 0.34567 * Thruster\ Isp^{0.55235}$$

For small thrusters, it is assumed that the spacecraft will utilize off-the-shelf components, and the mass of the small thruster is hard-coded as 0.4 kg [1].

The subfunction then reads in the maneuver requirements from the requirements structure. These requirements are used to determine the number of thrusters of each type necessary to complete the various maneuvers as anticipated.

$$Large\ Thruster\ Count = \frac{Orbit\ Insertion\ Thrust\ Requirement}{Large\ Thruster\ Isp}$$

$$Medium\ Thruster\ Count = \frac{Trajectory\ Correction\ Maneuver\ Thrust\ Requirement}{Medium\ Thruster\ Isp}$$

Both of these equations round up to the next whole number using the MATLAB ‘ceil’ command, as a spacecraft logically cannot have a fraction of a thruster.

```
LargeThrusterCount = ceil(OIThrust/Requirements.Prop.LargeThruster);

MediumThrusterCount = ceil(TCMThrust/Requirements.Prop.MediumThruster);
```

The small thruster count is hard-coded at 12, as it is assumed that the small thrusters will be used for attitude control.

Once the requirements for the mass and count of all of the thrusters have been determined, they can be tabulated to determine the total mass of the Propulsion subsystem.

```
PropMass = LargeThrusterCount * LargeThrusterWeight + MediumThrusterCount * MediumThrusterWeight
+ SmallThrusterCount * SmallThrusterWeight;
```

The Power Requirements of the Propulsion subsystem are calculated using the following equation: [1]

$$\begin{aligned}&\mathit{Propulsion\ Subsystem\ Power\ Requirements}\\&= 0.01 * \frac{\mathit{Estimated\ Spacecraft\ Power\ Requirements\ (so\ far) - Payload\ Power\ Requirements}}{0.4}\end{aligned}$$

This equation assumes that the Propulsion system will utilize approximately one percent of the total spacecraft subsystem power requirements. The equation then assumes that 40% [1] of the subsystem power has been allocated so far, and extrapolates this assumption to determine an assumed final subsystem weight for the spacecraft.

Once mass and power for this subsystem have been determined, they are passed back to the Master Controller script for inclusion in overall spacecraft mass and power requirements. This information will be used to inform further subsystem development.

2.7 Attitude Control Subfunction

The subfunction generating the spacecraft Attitude Control System is written as a MATLAB subfunction. It passes in the structure ‘Requirements’ as an input, with ‘AttitudeMass’ and ‘AttitudePower’ as outputs.

The subfunction begins by pulling estimated spacecraft mass (so far in the design process) and extrapolating to determine a low-fidelity mass estimate. At this point in the subsystem design process, the subsystems designed so far represent approximately 39% of a spacecraft's final dry mass. Thus, the subfunction assumes that the rest of the spacecraft will follow this mass estimate by utilizing the following equation:

```
SpacecraftMass = 1/0.39 * Requirements.SpacecraftMass;
```

Once the mass is estimated for the purposes of designing the attitude control subsystem, the final spacecraft volume must be estimated to determine the moment arm of the attitude control subsystem. A subsystem packing estimate of 20 grams per cubic centimeter is used to estimate the final spacecraft volume. Additionally, an assumed requirement of a 10-inch packing envelope height is used [8].

$$Subsystem\ Volume\ (No\ Envelope) = \frac{Subsystem\ Mass}{Subsystem\ Packing\ Density}$$

$$Subsystem\ Radius\ (No\ Envelope) = \frac{3 * Subsystem\ Volume\ (No\ Envelope)^{\frac{1}{3}}}{4 *\ \pi}$$

$$Subsystem\ Volume\ Estimate$$
$$= \frac{4}{3} * \pi * (Subsystem\ Radius\ (No\ Envelope) + Packing\ Envelope\ Height)^3$$

$$Estimated\ Spacecraft\ Volume = \ Subsystem\ Volume\ Estimate + Payload\ Volume$$

This volume can then be used to estimate a moment arm for the final spacecraft estimate. This estimate assumes a spherical spacecraft in the calculation of a moment arm.

$$Estimated\ Spacecraft\ Radius = \ (\frac{3 * Estimated\ Spacecraft\ Volume}{4 *\ \pi})^{1/3}$$

This estimated radius is used as the moment arm of our attitude control system for the rest of this subsystem design.

The next step of the attitude control subsystem subfunction is to determine the thruster capability requirement, utilizing the spacecraft moment of inertia, number of thrusters in use for the maneuver, spacecraft moment arm, assumed maneuver to complete, and time to complete the maneuver [8].

$$Thruster\ Capability\ Requirement = \frac{8 * Spacecraft\ Moment\ of\ Inertia * \ \theta}{Number\ of\ thrusters * Moment\ Arm\ Distance * Time\ to\ Complete\ Maneuver^2}$$

Where theta represents the number of degrees that the attitude control system must rotate the spacecraft. The following factors are hard-coded for this analysis as assumptions:

- Theta – the number of degrees that the attitude control system must rotate the spacecraft (180 degrees) along one coordinate direction
- Number of thrusters – 2
- Time to Complete Maneuver – 30 seconds

Next, the subfunction will determine the amount of solar torque acting on the spacecraft. The spacecraft surface area is calculated from the assumed spherical volume, and a spacecraft reflectivity of 0.5 is assumed as a reasonable value for the spacecraft's reflectivity. The solar pressure at the spacecraft's anticipated distance from the sun is pulled from the Requirements structure.

$$Solar\ Torque = Solar\ Pressure * Spacecraft\ Surface\ Area * Moment\ Arm * (1 + Spacecraft\ Reflectivity$$

Next, the subfunction determines the amount of Momentum Buildup in the course of a single orbit. This is determined by calculating the orbit period from known values specific to the destination, as well as the orbit distance from the destination [8].

$$Orbit\ Period = \frac{2 * \pi}{Destination\ Standard\ Gravitational\ Parameter} * (Destination\ Radius + Height\ Above\ Destination\ Surface)^{3/2}$$

$$Momentum\ Buildup = Solar\ Torque * Orbit\ Period$$

Next, the subfunction determines how frequently momentum wheel desats will have to take place to keep the wheels from becoming saturated and losing spacecraft attitude control authority. It is assumed that a typical spacecraft reaction wheel can store 100 kg-m/s before becoming saturated [8].

$$Wheel\ Saturation = \frac{100}{Momentum\ Buildup}$$

$$Time\ Between\ Reaction\ Wheel\ Desats = Orbit\ Period * Wheel\ Saturation$$

The time between reaction wheel desats is calculated in order to determine how many reaction wheel desat maneuvers must be performed during the nominal science mission while in orbit at the destination [8].

$$Reaction\ Wheel\ Desat\ Count = \frac{Primary\ Mission\ Length}{Time\ Between\ Reaction\ Wheel\ Desats}$$

The subfunction then calculates how much total force will have to be expended in order to desaturate the reaction wheels over the course of the primary mission. The subfunction also calculates how much propellant is expended desaturating the wheels [8].

$$Reaction\ Wheel\ Force\ Required = \frac{Reaction\ Wheel\ Storage}{Number\ of\ thrusters * Moment\ Arm\ Radius}$$

The burn time spent for each reaction wheel desat is calculated [8].

$$Burn\ Time = \frac{Wheel\ Saturation}{Reaction\ Wheel\ Force\ Required}$$

Lastly, the total mass needed to desaturate the wheels over the course of the primary mission is determined by the following equation: [8]

$$\begin{aligned} &Total\ Reaction\ Wheel\ Desat\ Propellant\ Mass \\ &\qquad = Reaction\ Wheel\ Desat\ Count * \ Number\ of\ Thrusters \\ &\qquad * Reaction\ Wheel\ Force\ Required * Burn\ Time \end{aligned}$$

A series of constants in regards to the mass and power requirements of the attitude control system are hard-coded. These assumptions are: [1]

- Sun Sensor Power Requirement – 1 watt
- Star Tracker Power Requirement – 18 watts (per star tracker)
- Reaction Wheel Power Requirement – 21.4 watts (per wheel)
- Sun Sensor Mass – 0.5 kg
- Star Tracker Mass – 7 kg (per star tracker)
- Reaction Wheel Mass – 8.5 kg (per wheel)
- Star Tracker Count – 2
- Reaction Wheel Count – 4

The counts of star trackers and reaction wheels are set to 2 and 4 (respectively) as reasonable assumptions for minimizing weight requirements while still maintaining redundancy in the event of a hardware failure in these components.

```
StarTrackerCount = 2;
StarTrackerPower = StarTrackerCount * 18;
StarTrackerMass = StarTrackerCount * 7;

ReactionWheelCount = 4;
ReactionWheelPower = ReactionWheelCount * 21.4;
ReactionWheelMass = ReactionWheelCount * 8.5;
```

Lastly, the subfunction calculates the total mass and power requirements of the overall subsystem, before passing this information back to the Master Controller Script for inclusion in the development of further spacecraft subsystem estimates.

```
AttitudeMass = SunSensorMass + StarTrackerMass + ReactionWheelMass + PropMass;
AttitudePower = SunSensorPower + StarTrackerPower + ReactionWheelPower;
```

2.8 Power System Subfunction

The next subfunction called designs a preliminary power subsystem to meet the power needs of the spacecraft. It is written as a MATLAB function. The input for the function is the structure 'Requirements' while the outputs are 'PowerMass' and 'PowerPower.' These outputs represent the mass and power requirements for the subsystem, respectively.

The subfunction begins by assessing the power requirements at different phases of the mission. The nominal science phase requirements are determined by assuming that the spacecraft being designed will be similar to other spacecraft. This is performed by determining the power requirements of the spacecraft subsystems "so far", then extrapolating out to the rest of the spacecraft design. So far, the Master Controller script has designed the Telecommunications, Command and Data Handling, Propulsion, and Attitude subsystems for the spacecraft. These subsystems typically represent about 61% of the total power requirements for a spacecraft. Thus, we can extrapolate that the power requirement we have calculated "so far" will represent about 61% of the final spacecraft's power needs.

Next, the subfunction determines the maximum range to sun in order to determine whether it is appropriate to design solar panels or use radioisotope thermoelectric generators to provide power for the spacecraft.

If the range is greater than 8.155 * 10^8 kilometers (about Jupiter's distance from the sun), the script will automatically generate requirements for using a radioisotope thermoelectric generator to power the spacecraft. Since this is an early estimate and designing nuclear reactors is outside the scope of this software, two simple equations estimating mass and power requirements are used [1].

$$Power\ System\ Mass\ Requirements = (\frac{1}{4.93}) * Estimated\ Spacecraft\ Power\ Requirement$$

$$Power\ System\ Power\ Requirements = 0.1 * Estimated\ Spacecraft\ Power\ Requirement$$

In these instances, radioisotope thermoelectric generators are used as solar panels would have to be very large in order to realistically power a useful spacecraft past Jupiter’s orbit.

For spacecraft expected to be closer to the sun, a solar panel design is utilized by the subfunction. A number of losses are assumed in a solar panel system throughout the energy collection, storage, and distribution process. These estimated losses are outlined below. All of these are multiplicative factors on the overall power system output.

- Solar Panel to Power Loads Efficiency: 0.95
- Solar Panel to Battery Efficiency: 0.7
- Battery to Power Loads Efficiency: 0.96
- Radiation Degradation: 0.82
- UV Degradation: 0.98
- Thermal Degradation: 0.99
- Cell Mismatch Loss: 0.975
- Cell Resistance Loss: 0.99
- Contamination Loss: 0.99
- Shadow Loss: 1
- Temperature Adjustment Losses: 1

Other assumptions about the power subsystem are also made:

- Battery Depth of Discharge: 0.4
- Battery Discharge Voltage: 28 volts
- Array Pointing Error: 5 degrees
    - The equation for determining the loss of efficiency in the solar panels is governed by the equation cos(Array Pointing Error) [1].
- Cell Output: 0.172 watts per cell

If the spacecraft is in deep space, it will never meaningfully be in the shadow of any objects in the solar system. The subfunction automatically accounts for this in designing the battery system by setting the "Maximum Eclipse Time" variable to 0. This will be used later when designing the battery system.

If the spacecraft is going to a destination where it will have significant periods in shadow, the subfunction must determine this "eclipse time" in order to design a battery system to maintain the spacecraft while in shadow, as well as size the solar panels to handle the nominal spacecraft load and charge the batteries.

To determine the eclipse time of the spacecraft during nominal operations, the subfunction utilizes the following equations: [9]

$$alpha = \mathrm{asin}\left(\frac{Destination\ Radius}{Destination\ Radius + Destination\ Orbit\ Altitude}\right)$$

$$Orbit\ Period = 2 * \pi * \sqrt{\left(\frac{(Destination\ Radius + Destination\ Orbit\ Altitude)^3}{Destination\ Standard\ Gravitational\ Parameter}\right)}$$

Once the orbit period is determined, the subfunction can determine how much of that period is in shadow of the destination with the following equation: [9]

$$Maximum\ Eclipse\ Time = \frac{Orbit\ Period * \frac{alpha}{\pi}}{60}$$

With the knowledge of the eclipse time, the subfunction can now calculate the size of the battery system as well as how much energy needs to be stored for while the spacecraft is in shadow. The subfunction also calculates the time not in shadow, and the overall amount of power the solar panels must deliver in order to meet the science requirements and charge the batteries [1].

$$Time\ Not\ Eclipsed = Orbit\ Period - Maximum\ Eclipse\ Time$$

$$\begin{aligned} &Solar\ Panel\ Output \\ &= \frac{Science\ Power\ Requirement * Maximum\ Eclipse\ Time}{Battery\ to\ Load\ Efficiency * Solar\ Panel\ to\ Battery\ Efficiency * Time\ Not\ Eclipsed} \\ &+ \frac{Science\ Power\ Requirement}{Solar\ Panel\ to\ Load\ Efficiency} \end{aligned}$$

This equation sizes the minimum output of the solar panels. It does not indicate the requirement that the solar panels should be designed to, but rather their output at the end of the mission in a worst-case degradation scenario.

The theoretical solar panel cell output is assumed to be 0.172 watts. The actual solar panel cell output is described by the following equation: [1]

$$Actual\ Solar\ Panel\ Cell\ Output = Theoretical\ Solar\ Panel\ Cell\ Output * Degradation\ Factors$$

Degradation factors are determined by this equation: [1]

$$\begin{aligned} &Degradation\ Factors \\ &\quad = UVDegradation * Thermal\ Degradation * Cell\ Mismatch\ Loss \\ &\quad * Cell\ Resistance\ Loss * Contamination\ Loss * Shadow\ Loss \\ &\quad * (1 - Radiation\ Degradation) * Temperature\ Adjustment\ Factor \\ &\quad * Solar\ Intensity * Pointing\ Loss \end{aligned}$$

This aforementioned theoretical 0.172 watts per cell value comes from laboratory testing. The solar intensity degradation factor represents the reduced (or increased) solar intensity in the spacecraft's operating environment as compared to the laboratory conditions [1].

Calculation the actual output of the cell allows the subfunction to determine how many cells are needed to meet the power requirements of the spacecraft in sunlight [1].

$$Number\ of\ Cells = \frac{Solar\ Panel\ Output\ Requirement}{Actual\ Solar\ Panel\ Cell\ Output}$$

The solar panel cells are assumed to have a packing density of 1,100 cells per square meter [1].

$$Solar\ Array\ Area = \frac{Number\ of\ Cells}{Cell\ Density}$$

The subfunction can now calculate the battery system requirements, using the Science Power Requirements, the eclipsed time, and the aforementioned assumptions about the power system [1] [5].

$$Battery\ Capacity = \frac{Science\ Power\ Requirement * Maximum\ Eclipse\ Time}{Battery\ to\ Load\ Efficiency * Depth\ of\ Discharge * Discharge\ Voltage}$$

The subfunction has now calculated the requirements for the Battery system and the solar panels. The mass of the solar arrays and battery system can now be quantified using known relationships between power system components and component mass [1].

$$Solar\ Array\ Mass = 4kg * Solar\ Array\ Area$$

$$Battery\ Mass = \frac{1}{24}kg * Battery\ Capacity$$

Lastly, the subfunction calculates the mass and power requirements for the solar-panel-based power subsystem. It is assumed that the mass of power cabling, power distribution components, and other components will total 119.2 kilograms [1].

$$Power\ Subsystem\ Total\ Mass = Solar\ Array\ Mass + Battery\ Mass + 119.2$$

$$Power\ Subsystem\ Total\ Power\ Requirement = 0.1 * Estimated\ Spacecraft\ Power\ Requirement$$

The subfunction ends by passing the Power Subsystem Total Mass and Power Subsystem Total Power Requirement back to the Master Controller subfunction for inclusion in calculations for further spacecraft subsystems.

2.9 Structure Subfunction

The structure subfunction performs a preliminary design estimate of the structure that will contain the other spacecraft subsystems and the science payload. This subfunction was written as a MATLAB function. The input of the subfunction is the structure ‘Requirements.’ The outputs of the subfunction are the variables ‘StructureMass’, ‘StructurePower’, and ‘SubsystemVolume.’

The subfunction estimates the total volume expected to be occupied by the various spacecraft subsystems, and adds this to the science payload volume requirement. The subsystem volume is estimated as a function of subsystem mass. The subfunction also assumes a subsystem packing density of 20 pounds per square foot, and a packing envelope height of 10 inches. For spacecraft structure material, the subfunction assumes aluminum.

The formula for estimating the subsystem volume from the subsystem mass is as follows: [8]

$$Subsystem\ Volume\ (No\ Envelope) = \frac{Subsystem\ Mass}{Subsystem\ Packing\ Density}$$

$$Subsystem\ Radius\ (No\ Envelope) = \frac{3 * Subsystem\ Volume\ (No\ Envelope)^{\frac{1}{3}}}{4 *\ \pi}$$

$$Subsystem\ Volume\ Estimate$$
$$= \frac{4}{3} * \pi * (Subsystem\ Radius\ (No\ Envelope) + Packing\ Envelope\ Height)^3$$

Next, the subfunction estimates the amount of fuel that the mission will require [9]. This estimation is made solely for volume estimation purposes. Final design of the fuel system for the spacecraft is made in the fuel subfunction.

$$Estimated\ Fuel\ Need = Estimation\ Factor * Spacecraft\ Mass * (1 - e^{\left(\frac{Required\ Delta-V}{9.81*Isp}\right)})$$

It is assumed that the spacecraft will use hydrazine, which has a density of 1.01 g/cm^3. The propellant volume that will be loaded onto the spacecraft can now be estimated.

$$Propellant\ Volume = \frac{Estimated\ Fuel\ Need}{Propellant\ Density}$$

The total spacecraft volume is then estimated:

$$\begin{aligned} &Spacecraft\ Volume\ Estimate \\ &\qquad = Science\ Payload\ Volume + Subsystem\ Volume\ Estimate + Propellant\ Volume \end{aligned}$$

A structural thickness with safety factors is assumed, and the mass of the structure can be calculated: [1]

$$\begin{aligned} &Structure\ Mass \\ &\qquad = Aluminum\ Density * 2 * \pi * Spacecraft\ Radius * Spacecraft\ Length \\ &\qquad * Spacecraft\ Structure\ Thickness \end{aligned}$$

The structure subfunction makes no calculations for power requirements, and returns ‘0’ for StructurePower to the Master Controller script. The ‘StructureMass’ value and ‘SubsystemVolume’ calculations are passed as outputs to the Master Controller script for inclusion in further spacecraft subfunction calculations.

2.10 Thermal System Subfunction

The Thermal System subfunction is written as a MATLAB function. It accepts the structure 'Requirements' as an input from the Master Controller script, and passes 'ThermalMass' and 'ThermalPower' as outputs back to the Master Controller script.

The subfunction begins by setting the maximum and minimum acceptable thermal range from the Requirements structure. These are set manually by the user prior to running the script. This range represents the maximum and minimum temperatures that any subcomponent of the spacecraft can endure, with the assumption that this thermal range would be applied to the entire spacecraft as a preliminary estimate of thermal subsystem needs.

Assumptions are also made about 'alpha' and 'epsilon.' Alpha is the solar absorptivity of the surface, whereas epsilon is the infrared emissivity.

The power dissipation is the required power system output.

The Thermal subfunction utilizes an assumed sphere diameter to calculate possible thermal management needs. This calculation uses the Structure subfunction assumptions about Spacecraft Volume to determine a spherical equivalent spacecraft for making the thermal calculations.

$$Sphere\ Diameter = 2 * \sqrt[3]{\frac{3 * Spacecraft\ Volume\ Estimate}{4 * \pi}}$$

The subfunction then assesses where the "worst-case" spacecraft heating takes place. If the structure 'Requirements' has a "range to sun at destination" variable that is closer to the sun than Earth, the destination's thermal environment is used as the worst-case heating environment. Otherwise, the Earth's thermal environment is used.

Once this assessment is made, the subfunction loads the thermal variables at the worst-case heating location in order to analyze the potential worst-case heat load on the spacecraft.

The first calculation made is the “Worst-Case View Factor” which measures how much heat will be coming off of the destination into the spacecraft thermal environment. This is followed by determining the Reflectance Factor, a measure of the percentage of the energy reflected. These two variables are used to determine the Worst-Case Spacecraft Temperature. In these measurements, all variables pertaining to a specific planet refer to the planet with the worst-case thermal heating on the spacecraft. This can either be the Earth or the destination, depending on the destination’s range to the Sun [1].

$$Worst\ Case\ View\ Factor = 0.5 * (1 - \frac{(Spacecraft\ Altitude^2 + 2 * Spacecraft\ Altitude * Planet\ Radius)^2}{Spacecraft\ Altitude + Planet\ Radius})$$

$$Reflectance\ Factor = 0.657 + 0.54 * (\left(\frac{Planet\ Radius}{Planet\ Radius + Spacecraft\ Altitude}\right) - 0.196 * \left(\frac{Planet\ Radius}{Planet\ Radius + Spacecraft\ Altitude}\right)^2)$$

$$Worst\ Case\ Spacecraft\ Temperature\ Calculation\ Part\ 1 = (\left(\frac{Solar\ Flux * alpha}{4}\right) + (Worst\ Case\ IR * epsilon * Worst\ Case\ View\ Factor) + (Solar\ Flux * Albedo * alpha * Reflectance\ Factor * Worst\ Case\ View\ Factor) + (\frac{Maximum\ Power\ Dissipation}{\pi * Sphere\ Diameter}))$$

$$Worst\ Case\ Spacecraft\ Temperature = (\frac{Worst\ Case\ Spacecraft\ Temperature\ Calculation\ Part\ 1}{Stefan - Boltzman\ Constant * epsilon})^{\frac{1}{4}}$$

This series of calculations is repeated for the “Best Case” thermal scenario, representing the thermal conditions when the spacecraft is furthest from the sun.

Once the best case and worst case temperature scenarios are computed, the subfunction determines whether an additional heater is necessary to maintain the spacecraft thermal state, or if radiators will suffice to maintain the spacecraft thermal state [1].

$$Radiator\ Area = \frac{Worst\ Case\ Spacecraft\ Temperature - Thermal\ Max\ Requirement}{Stefan - Boltzman\ Constant * epsilon *\ Worst\ Case\ Spacecraft\ Temperature^4}$$

The temperature of the spacecraft with the best-case thermal environment (furthest from the sun) is computed to determine if additional heaters will be necessary in the coldest environment the spacecraft is expected to endure [1].

$$Radiator\ Temperature = (\frac{Best\ Case\ Spacecraft\ Temperature - Thermal\ Minimum\ Requirement}{Radiator\ Area * Stefan - Boltzman\ Constant * epsilon})^{\frac{1}{4}}$$

The subfunction then checks to see if this thermal criteria requires an active heater to keep the spacecraft within the required range [1].

$$Heater\ Check = Radiator\ Temperature - Thermal\ Minimum$$

If the Heater Check variable is less than 0, it is set to 0. Otherwise, a heater is designed and added to the spacecraft [1].

$$Heater\ Mass = 2\ kg/m^2$$

$$Heater\ Power\ Requirement = Heater\ Check$$

Once the passive thermal system and active heaters have been designed, the subfunction computes the total mass and power requirements of the thermal subsystem.

$$Thermal\ Mass = Radiator\ Mass + Heater\ Mass + Insulation\ Mass + Paint\ Mass + Foam\ Mass$$

$$Thermal\ Subsystem\ Power = Heater\ Power\ Requirement$$

The Radiator Mass is computed by: [1]

$$Radiator\ Mass = 0.03 * Radiator\ Area$$

The Insulation Mass is calculated by: [1]

$$Insulation\ Mass = 0.03 * Spacecraft\ Surface\ Area$$

The Paint Mass is calculated by: [1]

$$Paint\ Mass = 0.24 * Spacecraft\ Surface\ Area$$

The Foam Mass is computed by: [1]

$$Foam\ Mass = 64 * 0.075 * Spacecraft\ Volume$$

The Thermal Mass and Thermal Subsystem Power requirements are passed as outputs from the subfunction back to the Master Controller script for inclusion in further spacecraft subsystem design.

2.11 Fuel System Subfunction

The final spacecraft subsystem to be designed is the fuel subsystem. This subsystem is designed last so as to accurately compute final estimated spacecraft mass to determine the amount of fuel required to complete the mission requirements.

The subfunction is written as a MATLAB function. It accepts the structure ‘Requirements’ from the Master Controller script, and returns ‘FuelMass’ and ‘FuelPower’ to the Master Controller script for inclusion in the final spacecraft requirements design.

The subfunction utilizes the estimated spacecraft volume to determine the moment arm for any maneuvers. This information is pulled from the ‘Requirements’ structure, passed into the subfunction by the Master Controller script.

The subfunction assumes the efficiency of the thrusters in different use cases, specifically the actual specific impulse of the thruster when in steady thrust as compared to during pulsing thrust.

$$Isp_{steady} = 0.93 * Isp$$

$$Isp_{pulsing} = 0.5 * Isp$$

These “actual” specific impulses will be used for all further calculations pertaining to the fuel system in this subfunction.

The fuel system subfunction then computes the fuel requirements for using the previously-designed attitude control system to spin up the spacecraft to 5 rpm for the outbound cruise, and back to 0 rpm for orbit insertion at the destination [1]. While this may not be fully required for all missions, it is a good approximation of some of the maneuvers that will be expected of the spacecraft during any nominal mission.

$$Thruster\ Moment\ Arm\ Length = \sqrt[3]{\frac{3 * Spacecraft\ Volume}{4 * \pi}}$$

$$Rotation\ Rate = 5\ rpm$$

$$Thrust\ Required = \frac{Rotation\ Rate}{\left(\frac{Number\ of\ thrusters * Force\ Required * Moment\ Arm\ Length}{Isp_{steady}}\right)}$$

$$Propellant\ Required\ for\ This\ Maneuver = 2 * \frac{Number\ of\ Thrusters * Force\ Required * Moment\ Arm\ Length}{Isp_{steady}}$$

The subfunction then determines the propellant needed to maintain attitude control during a potential orbit insertion maneuver being made by the spacecraft. If the spacecraft's "Flyby Flag" is set to 1 (indicating that the spacecraft's mission is a flyby mission, and thus will not be making an orbit insertion maneuver), then all propellant needs for attitude control for this maneuver are set to 0. Otherwise, the subfunction determines the propellant needed through the following set of equations [1] [10].

$$Orbit\ Insertion\ Propellant\ Requirements\ (Braking\ Burn) = 2 * \frac{Number\ of\ thrusters * Force\ Required * \sqrt{\frac{2 * Isp_{steady} * omega}{Number\ of\ thrusters * Force\ Required * Moment\ Arm\ Length}}}{Isp_{steady}}$$

$$Orbit\ Insertion\ Prop\ Requirements\ (Attitude\ Contol) = 2 * \frac{Number\ of\ thrusters * Force\ Required * Orbit\ Insertion\ Time}{Isp_{pulsing}}$$

The next propellant requirement is the propellant required to maintain limit cycles during nominal science operations. A spacecraft attitude control system will have some error when performing maneuvers. This will result in a drift from the expected attitude of the spacecraft. Rather than constantly correcting for this error (which would use a significant amount of fuel), an error is allowed to propagate until the error reaches an unacceptable limit, in which case the attitude control system performs a maneuver to correct the error. The calculations for determining the amount of fuel to maintain this attitude state is discussed below.

The first calculation is the duration cycle, how much time elapses between when a maneuver must be performed to correct the attitude state [1].

$$Cycle\ Duration = \frac{8 * Isp_{pulsing} * 1}{Number\ of\ thrusters * Moment\ Arm\ Length * Force\ Required * 0.5}$$

$$Total\ Cycles = \frac{Nominal\ Mission\ Length}{Cycle\ Duration}$$

$$Propellant\ Mass_{Per\ Limit\ Cycle} = 6 * \frac{Number\ of\ thrusters * Force\ Required * 0.03}{Isp_{pulsing} * g}$$

The fuel inventory for the Attitude Control system over the course of the mission can now be computed. An error margin of 3.5% is added to account for potential loading errors and trapped propellant [1].

$$\begin{aligned} &Propellant\ Inventory_{ACS} \\ &\qquad = \left(Propellant\ Mass_{Per\ Limit\ Cycle} + Orbit\ Insertion\ Prop\ Requirements \right. \\ &\qquad \left. + Spin\ Up\ Propellant\right) * Error\ Margin \end{aligned}$$

The volume of the Propellant Inventory is calculated: [1]

$$Propellant\ Inventory\ Volume_{ACS} = \frac{Propellant\ Inventory_{ACS}}{Propellant\ Density}$$

With this information, the subfunction can now design a tank system to contain the Propellant Inventory for the Attitude Control system [1]. These calculations assume a Hydrazine Blowdown Ratio of 4.5, as well as a Hydrazine Initial Pressure of 625 psi.

$$Initial\ ACS\ Ullage\ Volume = \frac{Propellant\ Inventory\ Volume_{ACS}}{Hydrazine\ Blowdown\ Ratio - 1}$$

The subfunction then designs the volume of the bladder in the tank that will contain the Attitude Control System fuel [1].

$$Radius_{ACSBladder} = \sqrt{\frac{0.75 * Propellant\ Inventory\ Volume_{ACS} + Initial\ ACS\ Ullage\ Volume}{\pi}}$$

$$Area_{ACSBladder} = 2 * \pi * Radius_{ACSBladder}^{2}$$

$$Volume_{ACSBladder} = 0.075 * Area_{ACSBladder}$$

$$\begin{aligned} Total\ Tank\ Volume_{ACS} \\ &= Propellant\ Inventory\ Volume_{ACS} + Initial\ ACS\ Ullage\ Volume \\ &+ Volume_{ACSBladder} \end{aligned}$$

Once the volume of the tank is determine by the subfunction, the weight of the ACS fuel tank can be calculated [1].

$$Tank\ Weight_{ACS} = 0.0116 * Hydrazine\ Initial\ Pressure * Total\ Tank\ Volume_{ACS}$$

This concludes the calculations for the tank for the Attitude Control System. The subfunction now repeats this process for the main propellant inventory for major spacecraft maneuvers. The equations are the same as for the Attitude Control system, with the only exception being the inventory of fuel for maneuvering the spacecraft [1] [9].

$$Main\ Engine\ Propellant\ Inventory = Spacecraft\ Total\ Dry\ Mass * (1 - e^{\frac{Delta{-}V\ Requirement}{9.81 * Isp}})$$

Lastly, the subfunction computes the mass of the overall fuel system to complete the dry mass calculations for the spacecraft as well as the wet mass of the fuel required to meet the maneuvering requirements throughout the mission. It is assumed that the plumbing of the spacecraft for the fuel system will have a dry mass of 20 kg [1].

$$Fuel\ System\ Mass = Tank\ Weight_{ACS} + Tank\ Weight_{Main\ Engine\ Fuel} + Plumbing$$

It is assumed that the fuel system will use minimal power when the spacecraft is in nominal science mode, and that the science power requirements will cover any potential needs for power by the fuel system when maneuvers are being performed. See the ‘Future Work’ section.

The subfunction then passes the Fuel System mass and Fuel system power variables back to the Master Controller function for final inclusion in the overall spacecraft design.

3 Results

Once the development of the software was completed, the output of the software needed to be verified to determine whether reasonable spacecraft would be produced over a wide range of conditions. Once this testing was completed, a case study spacecraft was modeled and compared to spacecraft data for a spacecraft specifically designed for those conditions. This allows us to determine whether the software reasonably estimates a spacecraft at a Phase A stage, as well as highlights any shortcomings in the software that should be improved upon as future work.

3.1 Testing

A separate script was developed in an effort to adequately test the software as developed and identify bugs or flawed assumptions. This script generated a series of varying requirements for different spacecraft missions, and produced an output report that allowed for a rapid assessment of whether the spacecraft estimates being produced were reasonable.

To test the software, several parameters were selected to vary, rapidly producing a large data set of potential mission designs that would be run through the software. These varying parameter sets were repeated for each of the default destinations that are included in the Destinations subfunction.

The selection of parameters to be varied are:

- Propulsion Delta-V Requirement

- Payload Mass Requirement
- Payload Power Requirement
- Destination Flyby/Orbit Flag

These parameters were selected as they were the most likely to represent stressing cases on the software, and were also the most likely to be modified by the user to match the needs of their mission design. Propulsion Delta-V strongly affects the fuel subsystem (as well as total fuel loaded) which is a strong driver of spacecraft mass. Payload mass and power are the most likely requirements to be modified by the user, and will strongly drive the overall mass and power requirements of the spacecraft as a whole. Lastly, if the spacecraft does not intend to orbit the destination body, many secondary branches on the subfunctions are explored. If the spacecraft is only performing a flyby, the design of many of the subsystems changes substantially. These varying cases are repeated for each default destination in the Destination subfunction, resulting in a large dataset of spacecraft designs to be examined. Ideally, many more (if not all) of the potential mission requirements would be varied in order to further explore the potential design space. However, due to the nature of the design of the testing function, this would rapidly result in an unwieldy number (hundreds of millions) of potential spacecraft designs. Some of these parameters varying do not result in a significant change to the final spacecraft design, and the analysis of subtly-different spacecraft numbering in the hundreds of millions cannot be realistically completed in any reasonable amount of time without further automation.

Separately, cases utilizing other branches that were not used as frequently (such as mission designs relying heavily on user input, such as deep space missions with no particular destination) were also separately exercised to explore the limits of the software and discover unnoticed bugs and other code errors.

3.1.1 Testing Methodology

The actual code written to test the software was quite simple. The testing code replaced the Master Controller script's requirements entry sections, with a large number of the spacecraft requirements being fixed for the entirety of the testing. For any parameters that needed to be varied to complete the testing, the script utilized a series of nested WHILE loops.

Each WHILE loop varied one parameter at a time. Within the last WHILE loop, as a function call of the original Master Controller script, which then completed the remaining analysis as normal. Once this function call was completed, the outputs were appended to a file for further manual analysis once the testing script was completed. The nature of the nested WHILE loops allowed us to greater explore the solution space of the software with ease. Each varied parameter was used in a spacecraft with each other varied parameter. In other words, thousands of spacecraft were rapidly estimated, with each one being very similar to the last with only one varied parameter. Each time the spacecraft delta-v requirement was varied, the remainder of the spacecraft remained fixed. This was repeated with each parameter until every combination of parameters had been tested.

While it was not feasible to examine each estimated spacecraft individually, this was not the purpose of our work at this stage of testing. This stage of testing allowed us to at-a-glance determine if the overall intent of our code was being executed, or if there were bugs or design flaws that were preventing this. We looked closely at the spacecraft mass and power requirements, as we believed this would be the easiest way to rapidly determine if reasonable spacecraft were being estimated by the software.

All in all, the testing of the software confirmed that the software was working as intended. However, there were a few circumstances where this was not the case. It was quickly noticed that the mass and power requirements for spacecraft going to the outer planets were rapidly ballooning well beyond what humanity would be reasonably capable of launching. In some cases, spacecraft with mass requirements greater than one million tons were being produced! While it was expected that the software may encounter edge cases that did not produce a reasonable spacecraft (in fact, mission design feasibility is the entire point of this software!), there were too many spacecraft being developed with this unrealistic mass requirement. Ultimately, it was discovered that a flawed unit conversion in the Attitude Control System subfunction was resulting in very large fuel requirements in order to maintain the needs of the spacecraft around large planets. This bug was present in all of the testing, but only manifested itself so severely around larger bodies. Once this bug (among a handful of others) was rectified, the testing indicated that the script would produce realistic spacecraft if the mission requirements were also realistic.

3.2 Case Study 1 – Mars Orbiter

In order to demonstrate the validity of this script, a Case Study was performed, comparing the software output to the Mars Reconnaissance Orbiter. To make this comparison, the mission requirements of the Mars Reconnaissance Orbiter were used as the mission requirements for the software. The software then generated a set of spacecraft requirements based on those mission requirements, which can now be compared to the spacecraft that was built and flown to Mars.

3.2.1 Mars Reconnaissance Orbiter

Launched in 2005 about a Lockheed Martin (now United Launch Alliance) Atlas V 401, NASA's Mars Reconnaissance Orbiter has spent the last decade capturing high-resolution imagery and science data of the Martian surface. The spacecraft has been helping scientists in the search for water on the surface of Mars, as well as serving as a communications link between NASA's rovers on the surface and the Earth [11]. The spacecraft supports six science payloads and three engineering instrument payloads. The spacecraft was designed to meet these mission requirements: [12]

- Payload Mass: 120 kg

- Propulsion System Delta-V requirement: 1.576 km/s
- Payload Power: 1400 W
- Primary Mission Length: 730 days
- Payload: 6 science instruments, 3 engineering payloads
- 255 km minimum science orbit altitude
- 160 Gb data storage

To support these mission requirements, the spacecraft team for the Mars Reconnaissance Orbiter designed the spacecraft with the following characteristics: [12]

- Total Mass: 2000 kg (925 kg dry)
- Hydrazine Monopropellant Propulsion System
- Total Power System Capability: 2000 W (at Mars, end of life)
- High-Gain Antenna Diameter: 3 meters
- 2-50 Amp-hour batteries

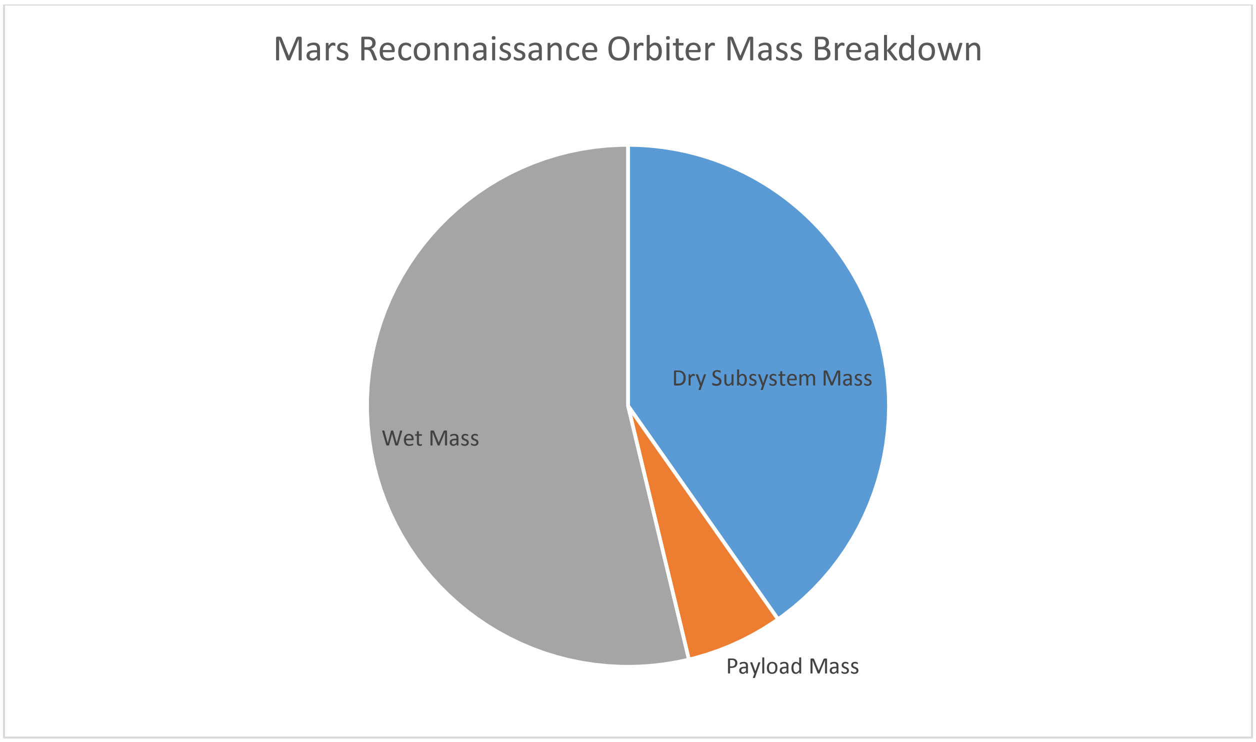

Figure 2 - Launch Mass Breakdown for MRO

For NASA's Mars Reconnaissance Orbiter, payload mass accounts for 6% of the total launch mass, the dry subsystem mass accounts for 40.25%, and the Wet Mass (Fuel) accounts for the remaining 53.75%. Respectively, these mass values were 120 kg for the Payload Mass, 805 kg for the Dry Subsystem mass, and 1075 kg for the wet fuel mass. The total mass of the spacecraft was 2000 kg at this stage in the estimation process.

The Mars Reconnaissance Orbiter was selected for this case study due to the fact that many of the spacecraft subsystems were designed in a way that the software is already well-equipped to handle. It utilizes a hydrazine monopropellant system, which is the default option for the propellant and fuel subfunctions which design those respective subsystems. The spacecraft utilizes solar panels due to its relative proximity to the Sun. This allows the case study to explore the development of the solar panel and battery systems. The spacecraft is in orbit around Mars, which allows the case study to explore much of the thermal subfunction for designing the thermal subsystem. MRO orbiting Mars also stresses the power subfunction further by eclipsing on each orbit, changing the sizing of the solar panels to allow for sufficient energy to charge the batteries, as well as the design of the batteries themselves. The propulsion system uses many of the same values on the spacecraft as are included as defaults in the software. Specifically, the thruster sizing is the same between the Mars Reconnaissance Orbiter as it is in the default software requirements. Due to the fact that the Mars Reconnaissance Orbiter is a NASA mission, it was also chosen as spacecraft mass, power, and fuel data was readily available for comparison to the software output.

3.2.2 Software-Generated Spacecraft Requirements

The software was given a set of mission requirements that would closely match the mission requirements of the Mars Reconnaissance Orbiter. Those requirements are:

- Delta-V: 1.576 km/s
- Destination Name: Mars
- Destination Min Range: 255 km
- Payload Mass Requirement: 120 kg
- Payload Power Requirement: 1400 W
- Payload Volume Requirement: 1 m^3
- Payload Count: 9
- Data Storage Requirement: 160 Gbit
- Pointing: 6 degrees/sec
- Primary Mission Length: 730 days
- Thermal Maximum: 25 degrees C
- Thermal Minimum: 2 degrees C
- Thruster Specific Impulse: 230 sec

- Large Thruster Thrust: 170 N
- Medium Thruster Thrust: 22 N
- Small Thruster Thrust: 0.9 N
- Flyby Flag: 0 (Spacecraft will orbit at destination)

No defaults in the subfunctions were modified for the design estimation of this spacecraft. All requirements were set at the Master Controller level. These mission-level requirements resulted in the following outputs from each of the subfunctions:

3.2.2.1 Destination Results

- Requirements.Destination.IR = 162
- Requirements.Destination.Albedo = 0.15
- Requirements.Destination.Radius = 3397
- Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (2.438 * 10^8)
- Requirements.Destination.SolarFlux = 586
- Requirements.Destination.mu = 4.282 * 10^4
- Requirements.Destination.SolarPressure = 2 * 10^-6

All of these requirements pertaining to the input Destination were passed successfully to the Master Controller script for inclusion in further subfunction analysis.

3.2.2.2 Telecommunications Results

- Telecommunications Subsystem Mass Requirement: 45.5 kg
- Telecommunications Subsystem Power Requirement: 126.78 W

3.2.2.2.1 Link Budget

Parameter	Value
Frequency, GHz	8.4
Bit error rate	1.0E-5
Range, km	393,400,000
Symbol rate, ksps	0.125
Transmitter Power, dB	37.28
Cable loss, dB	0
Antenna gain, dBi	67
EIRP, dB	104.28
Free space path loss, dB	282.82
Atmospheric attenuation, dB	-0.17
Polarization loss, dB	-1
S/C antenna gain, dBi	33.28
Pointing Loss, dB	-5
Receiver cable loss, dB	-1.95
Total Received Power, dB	-152.94
System noise temperature, K	18
System noise density, dB/Hz	-216.05
Carrier power/total power, dB	-5.347
Carrier Power Received, dB	-158.29
Carrier noise band width, dB-Hz	13
Carrier/noise ratio received, dB	44.76
Carrier/noise ratio required, dB	6
Carrier margin, dB	38.76
Command power/total power, dB	-1.499
Command power received, dB	-154.44
Command symbol rate, dB-Hz	-20.97
E_b/N_0 achieved, dB	40.64
E_b/N_0 required, dB	4.2
Command Link Margin, dB	36.44

Table 1 - Software-Developed Mars Mission Telecom Link Budget

3.2.2.3 Command & Data Handling Results

- Command & Data Handling Subsystem Mass: 97 kg
- Command & Data Handling Subsystem Power: 214 W

3.2.2.4 Propulsion System Results

- Propulsion System Mass: 63.41 kg
- Propulsion System Power: 8.52 W

3.2.2.5 Attitude Control System Results

- Attitude Control System Mass: 113.41 kg
- Attitude Control System Power: 122.6 W

3.2.2.6 Power System Results

- Power Subsystem Mass: 309.44 kg
- Power Subsystem Power Requirements: 77.36 W

3.2.2.7 Structure Results

- Structure Mass: 213.89 kg
- Structure Power: 0 W

3.2.2.8 Thermal System Results

- Thermal Mass: 28.4 kg
- Thermal Power: 289.77 W

3.2.2.9 Fuel System Results

- Fuel System Dry Mass: 79.30 kg
- Fuel System Power: 0 W

3.2.2.10 Mars Reconnaissance Orbiter Case Study Results Summary

- Total Spacecraft Mass: 2191 kg
  - Spacecraft Dry Mass: 1071 kg
  - Spacecraft Wet Mass: 1120 kg

- Total Spacecraft Power Requirement: 2239 W

This spacecraft, as designed, was capable of being launched atop the Atlas V, just like the spacecraft's real-life counterpart [13].

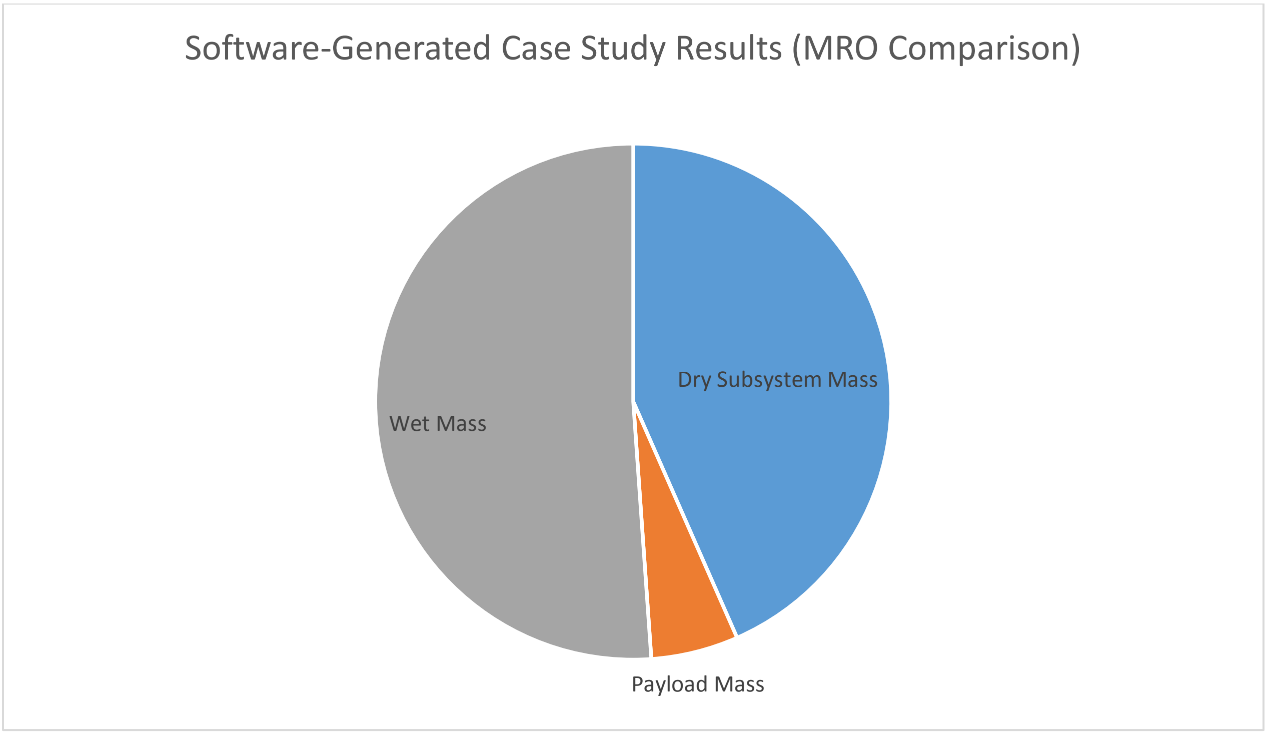

Figure 3 - Launch Mass Breakdown for Software-Generated Spacecraft

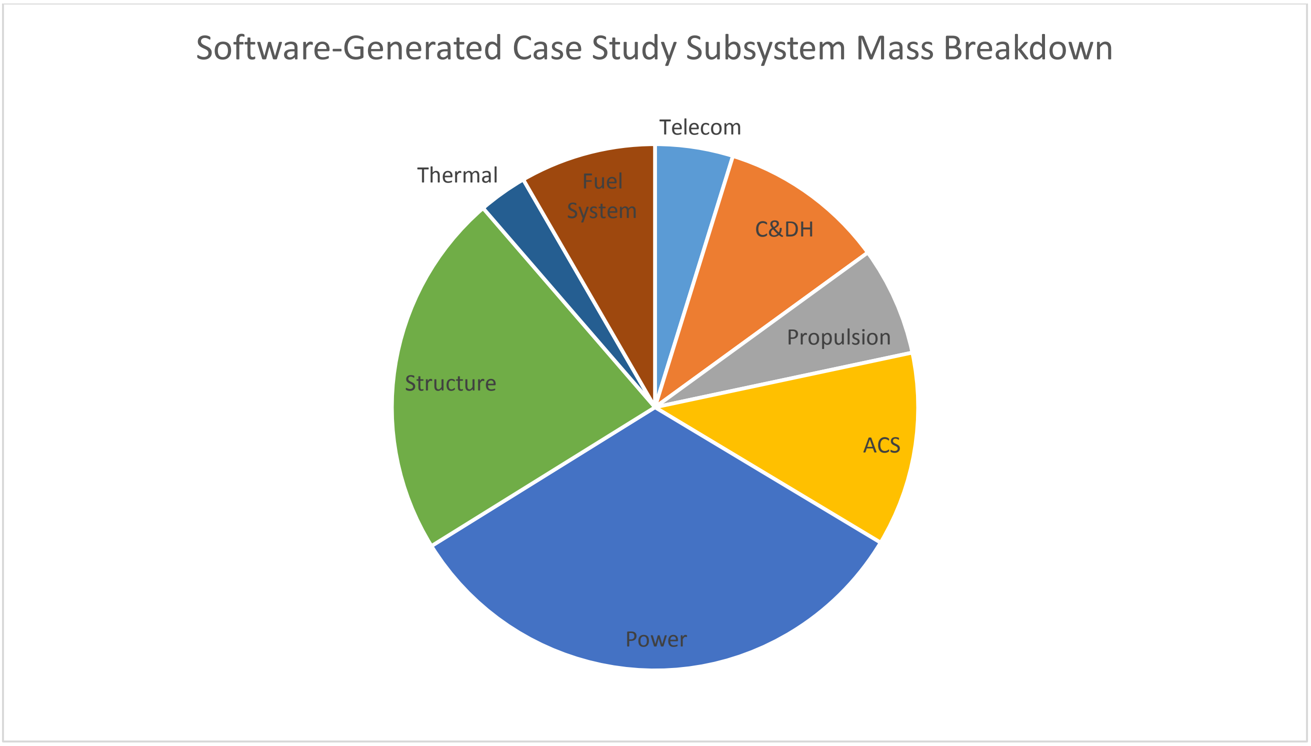

Figure 4 - Subsystem Mass Breakdown for Software-Generated Spacecraft

Subsystem	Mass Requirement (kg)	Subsystem Percentage	Expected Percentage
Telecom	45.5	5%	7%
C&DH	97	10%	7%
Propulsion	63.41	7%	8%
ACS	113.41	12%	10%
Power	309.44	33%	29%
Structure	213.89	23%	29%
Thermal	28.4	3%	3%
Fuel System	79.31	8%	8%

Table 2 - Subsystem Mass Percentage Comparison for Mars Spacecraft

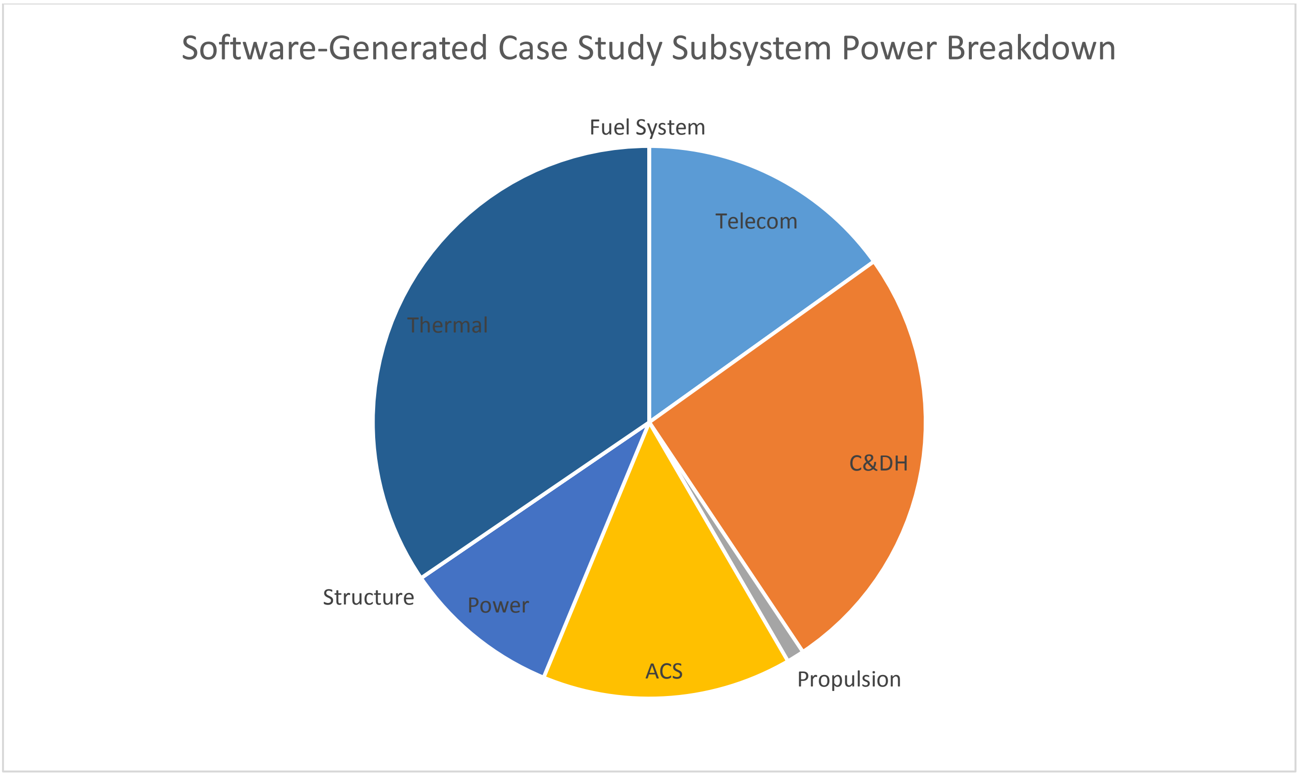

Figure 5 - Subsystem Power Breakdown for Software-Generated Spacecraft

Subsystem	Power Requirement	Subsystem Percentage	Expected Percentage
Telecom	126.78 W	15%	23%
C&DH	214 W	26%	17%
Propulsion	8.52 W	1%	1%
ACS	122.6 W	15%	20%
Power	77.36 W	9%	10%
Structure	0 W	0%	1%
Thermal	289.77 W	35%	28%
Fuel System	0 W	0%	1%

Table 3 - Subsystem Power Percentage Comparison for Mars Spacecraft

3.3 Case Study 2 – Unrealistic Spacecraft Estimate

In order to demonstrate whether the script will develop realistic spacecraft estimates, a Case Study was developed which feeds the software a set of unrealistic (with current technology) mission requirements. While the resulting spacecraft requirements should be realistic and reasonable, they should not be feasible, due to the infeasible nature of the mission requirements being imposed on the spacecraft requirements in this case.

To demonstrate the software, a series of unrealistic (but still hypothetically possible without current technology constraints) mission requirements is assembled. These mission requirements should strongly stress the spacecraft requirements so that a realistic but infeasible set of spacecraft requirements is developed. The mission requirements will include a very high spacecraft delta-v (to stress the propulsion and fuel systems, as well as raise the required spacecraft mass, which in turn will stress the attitude control system and structure), unreasonably stringent payload requirements (high mass and power requirements, narrow thermal constraints), a long primary mission length at the destination, and an unreasonably low science orbit altitude (stressing the thermal and attitude control subsystems).

In short, the requirements to be levied on the spacecraft system by the mission requirements are:

- Mission Delta-V: 15 km/s
- Mission Destination: Uranus
- Payload Mass Requirement: 750 kg
- Payload Power Requirement: 2500 Watts
- Payload Count: 5
- Spacecraft Thermal Maximum: 30 degrees Celsius
- Spacecraft Thermal Minimum: 25 degrees Celsius
- Primary Mission Length at Destination: 3,650 days

- Science Orbit Altitude: 25 kilometers

All other values are set to their defaults, and all assumptions made in the software remain unchanged for this case.

3.3.1 Software Generated Spacecraft Requirements

3.3.1.1 Destination Results

- Requirements.Destination.IR = 0.63
- Requirements.Destination.Albedo = 0.51
- Requirements.Destination.Radius = 25559
- Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (2.98 * 10^9)
- Requirements.Destination.SolarFlux = 4
- Requirements.Destination.mu = 5.793 * 10^6
- Requirements.Destination.SolarPressure = 1.24 * 10^-8

All of these requirements pertaining to the input Destination were passed successfully to the Master Controller script for inclusion in further subfunction analysis.

3.3.1.2 Telecommunications Results

- Telecommunications Subsystem Mass Requirement: 141.35 kg
- Telecommunications Subsystem Power Requirement: 739.66 W

3.3.1.2.1 Link Budget

Parameter	Value
Frequency, GHz	8.4
Bit error rate	1.0E-5
Range, km	$3.129 * 10^9$
Symbol rate, ksps	0.125
Transmitter Power, dB	650.17
Cable loss, dB	0
Antenna gain, dBi	67
EIRP, dB	717.17
Free space path loss, dB	300.83
Atmospheric attenuation, dB	-0.17
Polarization loss, dB	-1
S/C antenna gain, dBi	50.98
Pointing Loss, dB	-5
Receiver cable loss, dB	-1.95
Total Received Power, dB	459.19
System noise temperature, K	18
System noise density, dB/Hz	-216.05
Carrier power/total power, dB	-5.347
Carrier Power Received, dB	-453.84
Carrier noise band width, dB-Hz	13
Carrier/noise ratio received, dB	656.89
Carrier/noise ratio required, dB	6
Carrier margin, dB	650.89
Command power/total power, dB	-1.499
Command power received, dB	457.69
Command symbol rate, dB-Hz	-20.97
E_b/N_0 achieved, dB	652.77
E_b/N_0 required, dB	4.2
Command Link Margin, dB	648.57

Table 4 - Unrealistic Spacecraft Estimate Link Budget

3.3.1.3 Command & Data Handling Results

- Command & Data Handling Subsystem Mass: 63.25 kg
- Command & Data Handling Subsystem Power: 71 W

3.3.1.4 Propulsion System Results

- Propulsion System Mass: 63.41 kg
- Propulsion System Power: 20.26 W

3.3.1.5 Attitude Control System Results

- Attitude Control System Mass: 252.09 kg
- Attitude Control System Power: 122.6 W

3.3.1.6 Power System Results

- Power Subsystem Mass: 317.07 kg
- Power Subsystem Power Requirements: 156.31 W

3.3.1.7 Structure Results

- Structure Mass: 309.3 kg
- Structure Power: 0 W

3.3.1.8 Thermal System Results

- Thermal Mass: 34.45 kg
- Thermal Power: 244.41 W

3.3.1.9 Fuel System Results

- Fuel System Dry Mass: 286.17 kg
- Fuel System Power: 0 W

3.3.1.10 Unrealistic Spacecraft Case Study Results Summary

- Total Spacecraft Mass: 6259 kg
    - Spacecraft Dry Mass: 2217 kg
    - Spacecraft Wet Mass: 4042 kg

- Total Spacecraft Power Requirement: 3854 W

There are currently no existing spacecraft that could potentially launch a spacecraft of this weight on a direct transfer to Uranus. Significant weight reductions and refinements of the trajectory would be necessary [14].

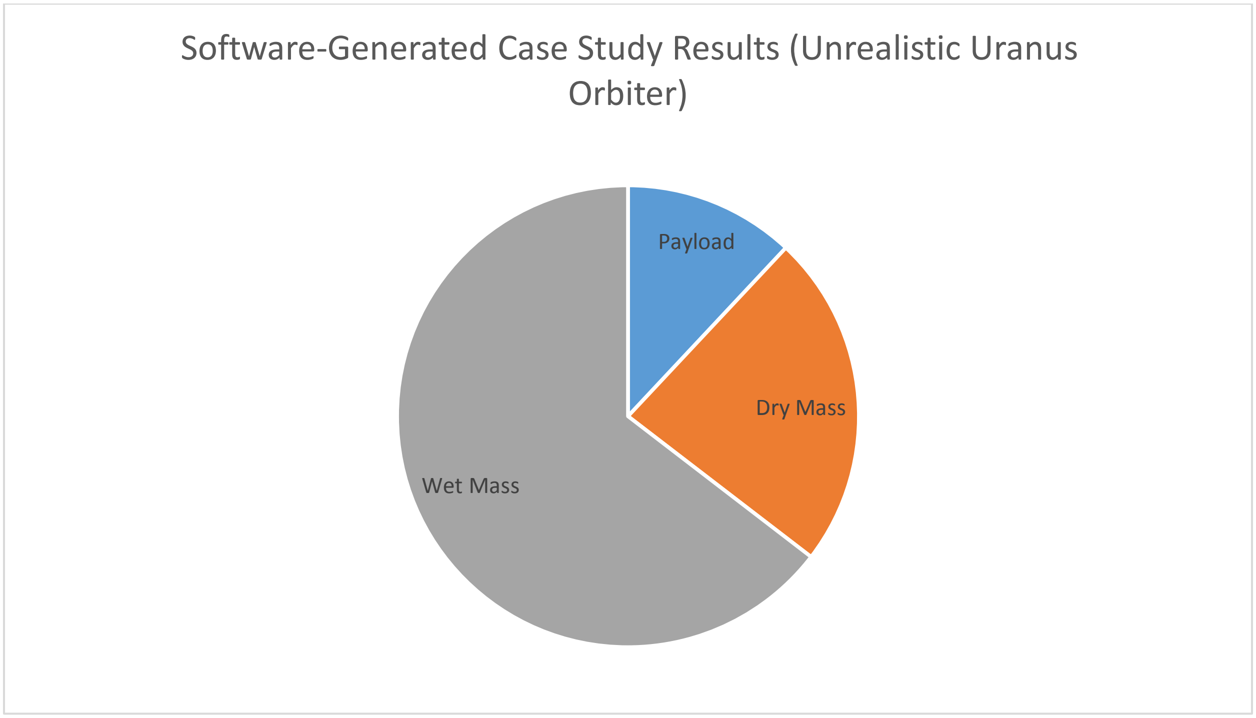

Figure 6 - Launch Mass Breakdown for the Unrealistic Mission Spacecraft

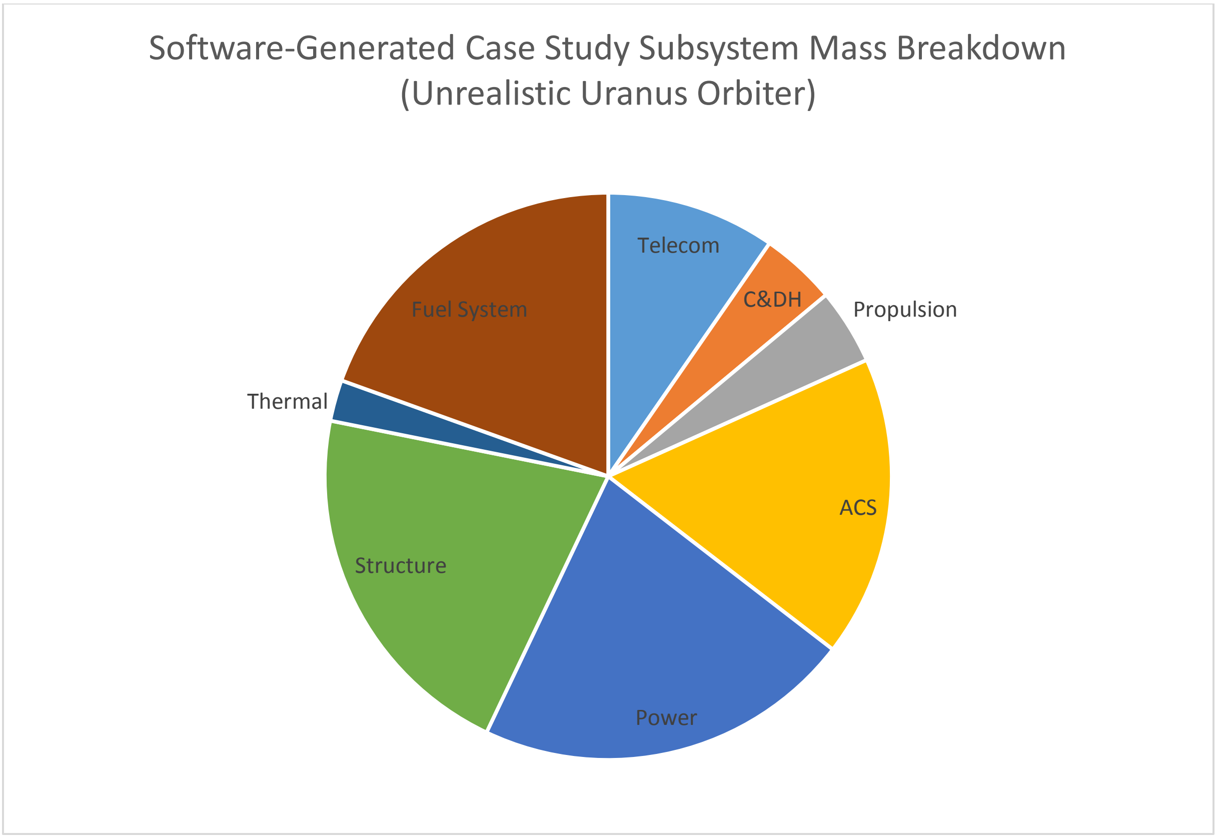

Figure 7 - Subsystem Mass Breakdown for the Unrealistic Mission Spacecraft

Subsystem	Mass Requirement (kg)	Subsystem Percentage	Expected Percentage
Telecom	141.35	10%	7%
C&DH	63.25	4%	7%
Propulsion	63.41	4%	8%
ACS	252.09	17%	10%
Power	317.07	22%	29%
Structure	309.3	21%	29%
Thermal	34.45	2%	3%
Fuel System	286.17	20%	8%

Table 5 - Subsystem Mass Percentages Comparison for the Unrealistic Mission Spacecraft

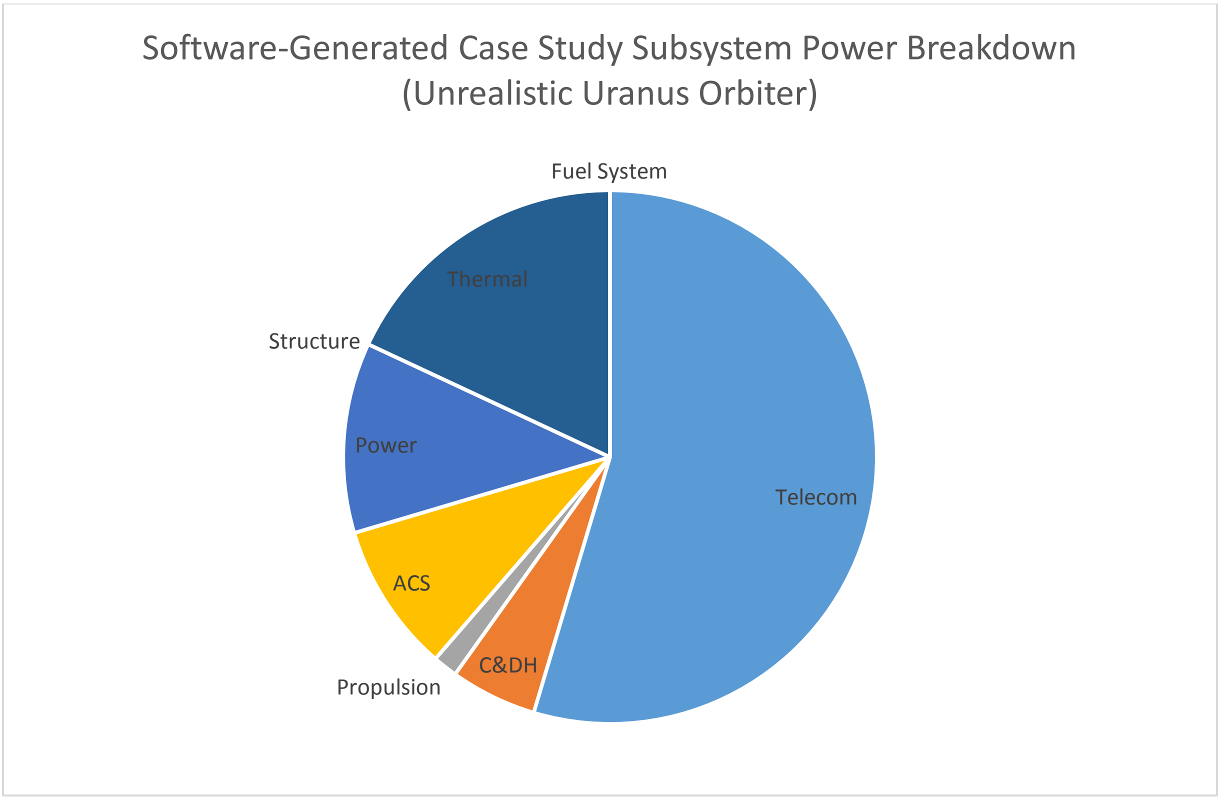

Figure 8- Subsystem Power Breakdown for the Unrealistic Mission Spacecraft

Subsystem	Power Requirement	Subsystem Percentage	Expected Percentage
Telecom	739.66	55%	23%
C&DH	71	5%	17%
Propulsion	20.26	1%	1%
ACS	122.6	9%	20%
Power	156.31	12%	10%
Structure	0	0%	1%
Thermal	244.41	18%	28%
Fuel System	0	0%	1%

Table 6 - Subsystem Power Percentages Comparison for the Unrealistic Mission Spacecraft

4 Discussion

After the completion of each of the case studies, we can now compare the results between the actual Mars Reconnaissance Orbiter spacecraft and the software-generated spacecraft designed to the same mission requirements. We will also examine the "Unrealistic Uranus Orbiter" spacecraft requirements to judge their feasibility in developing a spacecraft to meet these unrealistically high mission requirements.

4.1 Case Study 1 Discussion

The first case study compared the real-life Mars Reconnaissance Orbiter with a software-developed spacecraft that was designed to the same mission requirements. While there were many significant differences between the two spacecraft, the software-developed spacecraft requirements were similar to the subsystems on the Mars Reconnaissance Orbiter in many ways [12].

	Mars Reconnaissance Orbiter	Software-Developed Spacecraft
Payload Mass (kg)	120	120
Dry Mass (kg)	805	951
Wet Mass (kg)	1075	1120
Total Spacecraft Mass (kg)	2000	2191
Power Requirement (W)	2000	2239

Table 7 - Comparison of Launch Mass Breakdown between MRO and Software-Generated Mars Spacecraft

With similar payload requirements, the remaining top-level requirements of the two spacecraft look very similar. The non-payload dry mass of each spacecraft has some deviation from the actual non-payload dry mass (about an 18% error), the wet mass is within 50 kilograms (about a 5% error), and the total mass is within 200 kg (about a 10% error). The power requirement is within 250 W, an error of about 10%. In general, the software-developed spacecraft appears to slightly overestimate the requirements of the real Mars Reconnaissance Orbiter. However, in systems that can be directly compared, there appear to be some key differences.

4.1.1 Error Due to Thruster Specific Impulse Differences

One notable difference between the two spacecraft is the fact that the wet fuel mass is similar between both spacecraft, even though the dry mass of the software-developed spacecraft requirement is higher than that of the Mars Reconnaissance Orbiter. This is due to the fact that the thruster specific impulse was not a specifiable parameter. The Mars Reconnaissance Orbiter design used a thruster specific impulse value of about 190 seconds, while the software utilizes a default parameter value of 230 seconds (considered a safe estimate). Due to this difference, the software slightly underestimated the fuel that would be required to complete the mission. An improvement to the software to rectify this error would be for the software to accept thruster specific impulse as a mission requirement when designing the spacecraft. This is part of a larger future effort to further improve the propulsion subfunction for improved user customization and greater fidelity in meeting mission requirements.

4.1.2 Error Due to Insufficient Antenna Requirements

The Telecommunications subsystem designed a telecommunications systems that met requirements with positive link margin, but the High-Gain Antenna is undersized in the software-generated spacecraft when compared to the Mars Reconnaissance Orbiter. There are a number of potential root causes that are not accounted for in the software. The first could be that the Mars Reconnaissance Orbiter utilizes a larger-than-required High Gain antenna in order to serve as a communications relay between the Earth and NASA’s landers and rovers on the Martian surface. This was not accounted for when inputting the mission requirements, but it is an option for the user to do so. This would require knowledge of the communications bandwidth necessary to accomplish this secondary goal. This knowledge could then trivially be included in the telecommunications downlink rate, which would size the antenna appropriately. However, it is important to note that under the current set of requirements levied by the mission design, the spacecraft's link budgets and margins are appropriate to complete the mission as specified.

4.1.3 Power Requirements Error

The Power Systems requirements of the two spacecraft have some non-negligible differences. The Mars Reconnaissance Orbiter requires 2000 W of power at Mars, [12] while the software-developed Mars orbiter requires a total of 2,239 Watts. There are a number of possibilities as to why this could be the case. The first is that the orbit of the Mars Reconnaissance Orbiter is not perfectly circular [12]. This could change the spacecraft's eclipse time, resulting in a smaller overall power requirement. This orbit different would also result in a longer orbit, with the spacecraft spending more time in full sun. Both of these factors could contribute to the spacecraft requiring a smaller power system in reality as compared to the software-developed spacecraft. Additionally, the subsystem design of the software-developed spacecraft does not exactly match MRO. Namely, the telecommunications system does not match between the real spacecraft and the software-developed spacecraft. The telecommunications subfunction optimizes between high gain antenna size and power required to maintain a positive link margin. It is possible that the engineers for MRO made a decision to include a larger antenna in exchange for reduced power requirements on the telecom system.

Lastly, it is possible that the lack of specificity in power requirements resulted in a larger-than-necessary power system. As mentioned elsewhere, the power requirements for each subsystem are assumed to all draw upon the power system simultaneously, when this may not be the case in reality. Improved specificity in the power needs from various subsystems at different times may reveal that the peak power requirements are not the sum of the power requirements for each individual subsystem. If this were the case, it would explain why the power requirements on the hypothetical software-developed spacecraft were higher than its real-world counterpart.

4.1.4 Other Sources of Error

There are a number of other potential error sources that could account for the differences between the two spacecraft designs. One such potential error source is the lack of specificity in other, less driving, requirements. There is significant documentation in regards to payload mass and power requirements, as well as spacecraft features such as solar panel area, power system capability, antenna diameter, and science orbit characteristics. However, there is little documentation on other constraints upon the spacecraft design, such as thermal limits and telecommunications subsystem design.

It is possible that the thermal constraints placed upon the spacecraft were poor assumptions, resulting in a spacecraft design estimation that cannot match the real-life orbiter. In order to improve this comparison for the purposes of validating the software, the best course of action would be determine what all of the necessary mission requirements were for the real-world spacecraft design, and use those requirements as inputs for the software. The software estimations could then be improved to better match real-world design choices and trade studies. An attempt to complete this process was made to validate the software at this stage, but more data is needed.

4.1.5 Other Unaccounted Assumptions

The real-world Mars Reconnaissance Orbiter was designed with a series of assumptions that are harder to quantify and account for in this software at the current time. One such example is the impact of the launch vehicle to the design of the spacecraft structure. The placement and layout of the launch vehicle adapter could have significant impact on the design of the structure, as well as the placement of many vehicle payloads and subsystems. Future work aims to minimize this uncertainty by including launch vehicle data in the analysis, improving this section of the requirements estimation. Additional launch vehicle data would also assist in determining the impact on the structure design of the launch vehicle g-loading profile.

Another example is the design impact of the fact that the real-world Mars Reconnaissance Orbiter performed a number of aerobrake maneuvers in order to circularize the orbit at Mars [12]. These maneuvers could have had significant impact on the layout of the spacecraft which cannot be easily quantified in this software. Further analysis of the realistic spacecraft layout would be necessary prior to finalizing the overall design.

4.1.6 Differences in Expected vs Actual Subsystem Breakdowns

4.1.6.1 Differences in Subsystem Mass Breakdown

Brown's Elements of Spacecraft Design textbook contains data regarding expected breakdowns of subsystem mass for interplanetary spacecraft [1]. This breakdown is detailed in the 'Expected Percentage' column of each of the subsystem mass breakdown comparisons. While not a hard rule or expectation, this breakdown of mass expectations allows us to examine whether the spacecraft as designed fits in with general spacecraft design practices. In general, the spacecraft subsystem mass for each of the subsystems closely compares with the expected percentages. These values are close enough to the statistical expectations that there is little likelihood of error or need for justification of these values. The most significant deviation is in the Structure subsystem. The spacecraft structure subsystem as designed requires 23% of the final spacecraft mass, while a typical spacecraft structure subsystem would require about 29% of the final spacecraft mass. This 6% difference may be a sign of the subfunction underdesigning the structure for high-stress instances, but could also potentially be explained by other subsystems having a higher-than-normal mass.

The power system exceeds mass expectations by about 4%, while the telecommunications subsystem falls under mass expectations by about 2%. Since the antenna diameter is computed as an optimization between the antenna mass and transmitter power, this expectation could be more closely matched (if desired by the software user) by reducing the transmitter power and increasing the antenna diameter.

4.1.6.2 Differences in Subsystem Power Breakdown

Brown's Element of Spacecraft Design textbook contains data regarding expected breakdowns of subsystem power requirements for interplanetary spacecraft [1]. This breakdown is specifically detailed in the 'Expected Percentage' column of each of the subsystem power breakdown comparisons. While not a hard rule or expectation for the power requirements of the spacecraft subsystems, this breakdown of expectations allows us to examine whether the spacecraft's power subsystem requirements are similar to those found in other spacecraft. Unlike the spacecraft mass, the subsystem power requirements vary more significantly from the expected norms. The Propulsion, Power, Structure, and Fuel System subsystems vary between 0-1% of expectations, but the Telecom, C&DH, Attitude Control, and Thermal subsystems vary 5-8% from their expected values. The C&DH and Thermal subsystem power estimates are 7% above a typical interplanetary spacecraft. The Attitude Control subsystem power requirement is about 5% below what is expected in a typical spacecraft, while the Telecommunications power requirement is about 8% below what is expected in a typical interplanetary spacecraft.

The Command & Data Handling subsystem estimate is likely off due to the fact that the subfunction estimating this subsystem does not capture enough detail. A number of parameters are hard-coded, and improving this subfunction's power requirement calculations would likely bring this subsystem more in line with power requirement expectations.

The Thermal subsystem estimate is likely off from the textbook expectation due to the subfunction's reliance on heaters, when improved insulation would serve the spacecraft design better. Additionally, the constraints on thermal conditions for the spacecraft were set arbitrarily, and may not reflect the actual needs of the spacecraft subcomponents. An improved knowledge of the actual thermal constraints of the spacecraft subsystems would allow us to determine a more accurate thermal subsystem, potentially reducing the thermal subsystem power requirements.

The Attitude Control subsystem estimate is likely below expectations because the current low-fidelity nature of the attitude control subsystem subfunction is likely resulting in a slight underestimate of the power requirements for the attitude control system. Improved fidelity in the hardware needs of the Attitude Control system may result in an improved subsystem design.

Lastly, the Telecommunications subsystem estimate falls well below the power expectations for this subsystem. This may be due to the fact that some of the components typically allocated under Telecommunications may be allocated under Command & Data Handling, reducing the power requirement of telecom and raising the power requirement for C&DH. Additionally, it is possible that the power requirement expectation is set assuming the usage of multiple antennae simultaneously, which is now assumed under the current telecommunications subsystem subfunction.

4.2 Case Study 2 Discussion

While this case study did not have a real-world equivalent spacecraft to compare to, this study provided value by demonstrating that when the software was given infeasible mission requirements, the software would generate realistic but infeasible spacecraft requirements. Due to the fact that there is no real-world spacecraft to compare this estimation to, we cannot perform a direct comparison to assess whether a realistic spacecraft was generated. However, we can examine the sizing of the subsystems and fuel requirements relative to the mission requirements to determine whether they are reasonable.

	Software-Developed Spacecraft
Payload Mass (kg)	750
Dry Mass (kg)	1288
Wet Mass (kg)	2014
Total Spacecraft Mass (kg)	4052
Power Requirement (W)	3854

Table 8 - Launch Mass Breakdown for Unrealistic Mission Spacecraft

This mission required a spacecraft capable of meeting a large payload power requirement, a large payload mass requirement, tight thermal margins, a long primary science mission, and a significantly high spacecraft delta-v. The results of the spacecraft requirements for this mission match these stringent expectations.

4.2.1 Excessive Telecommunications Link Budget Margin

The current Telecommunications system has significant excess link budget margin that is not necessary to meet the mission requirements. This is likely due to the optimization between antenna mass and transmitter power. Reducing the transmitter power (by considering trade studies beyond optimizing between mass and power, or even favoring a heavier telecommunications subsystem in exchange for reduced power requirements) would reduce the unnecessary power requirement necessary to operate the telecommunications subsystem. In turn, this would also reduce the mass of the power system itself, further improving the feasibility of the spacecraft. This reduction in mass would also result in reduced mass for the ACS, fuel tanks, and total wet mass, solving additional problems currently present in the estimated design.

4.2.2 Excessive Fuel Requirements

The mission has excessive and unreasonable delta-v requirements, causing multiple issues with this spacecraft design. The high delta-v requirement requires an enormous wet mass, over 4000 kg! This is not a reasonable expectation for the spacecraft, and the user should consider reducing the overall spacecraft mass as well as refining the trajectory to reduce the delta-v requirement. The addition of gravity assists or the utilization of low-energy trajectories would significantly improve the feasibility of this mission. Reducing the wet mass would reduce the mass requirements of the fuel tanks necessary to store this fuel, as well as reduce the requirements of the attitude control system in order to maintain pointing requirements while maneuvering such a heavy spacecraft. Reducing mass will also allow that mass to be re-allocated to the telecommunications subsystem, reducing the power requirements and easing the expectations on the power system as well.

4.2.3 Unreasonable Mass Requirements for Launch Vehicle

As it stands, the current mass requirements for the spacecraft are too high for any existing launch vehicle to successfully launch to Uranus [14] [15]. The user should consider reducing the mission requirements most responsible for increasing the spacecraft mass, specifically the payload mass and power requirements, as well as the unrealistically high delta-v requirements. This will serve the dual purpose of reducing the spacecraft volume while also bring the mass down enough to consider existing launchers for this mission.

4.2.4 Differences in Expected vs Actual Subsystem Breakdowns

4.2.4.1 Differences in Subsystem Mass Breakdown

The Telecom, C&DH, Propulsion, and Thermal subsystems all were designed with mass requirements well in line with the mass expectations set forth in the Brown textbook. All of these subsystems had allocations with 4% of the expected value. However, the Attitude Control, Power, Structure, and Fuel subsystems all were designed with mass requirements differing significantly from typical interplanetary subsystem mass breakdown expectations.

The Attitude Control System was designed with a mass allocation of 17%, 7% more than expected. This may be due to the large mass requirements imposed by the spacecraft payload. This requires a larger attitude control system to maintain the pointing requirements, resulting in an increased mass. Additionally, the mission design requires the spacecraft to orbit very close to the atmosphere of Uranus. This results in additional torques that require the Attitude Control system to correct, increasing the fuel mass necessary to maintain a steady attitude throughout each orbit.

The Power subsystem was designed with a mass allocation of 22%, 7% less than expected. This Power subsystem mass allocation is likely due to the fact that the Fuel subsystem is cannibalizing the mass percentage allocation in order to store the large amount of fuel necessary to meet the mission delta-v requirements. The mass of the Power subsystem is reasonable given the mission requirements imposed upon the subsystem.

The Structure subsystem was designed with a mass allocation of 21%, 8% less than expected. Like the power subsystem, the mass percentage allocation is likely being cannibalized by the Fuel subsystem. The Structure subsystem mass allocation is in line with expectations.

The Fuel subsystem was designed with a mass allocation of 20%, 12% more than expected. This is likely due to the very large wet fuel mass required to meet mission delta-v requirements. This results in a large fuel tank necessary to hold the fuel, much larger than typically expected. However, this mass allocation is reasonable given the large amount of fuel the system must store.

4.2.4.2 Differences in Subsystem Power Breakdown

The percent allocations for power requirements on the software-design spacecraft are far off from many expectations. The Telecommunications subsystem is designed with an allocation totaling 55% of the spacecraft’s subsystem power. A typical interplanetary spacecraft usually requires only about 23% of the subsystem power. This is likely due to the fact that the telecommunications subsystem is designed to optimize between antenna mass and transmitter power. This optimization at long range back to Earth likely required a large antenna, and this optimization resulted in significantly overpowering the communications system. Upon completing this automated analysis, the user should return to the design and reduce the Telecommunications subsystem capabilities so a reasonable link budget is utilized. This will reduce the power requirements of the Telecommunications subsystem and bring the power percentage allocations for the entire spacecraft more in line with expectations.

The Command & Data Handling, Attitude Control, and Thermal subsystems all have power percentage allocation requirements far below what is expected. This is likely due to the Telecommunications subsystem cannibalizing all of the percent allocations to meet a large power requirement. The overall power requirements of these systems are reasonable when the mission requirements levied upon the spacecraft are considered.

The Power Requirement power allocation is within 2% of the expected allocation, the power requirement for the Power system is reasonable.

5 Future Work

While the software suite so far represents a step forward in using software to perform spacecraft initial design estimation tasks, there is substantial work to go to improve this tool to further automate these early estimation tasks and perform mission feasibility studies.

5.1 Command and Data Handling Subsystem Estimation Improvement

Currently, the subfunction responsible for handling the Command and Data Handling subsystem could be improved in order to better match the subsystem to the spacecraft being designed. Currently, this subfunction has fixed parameters for many aspects of the Command and Data Handling subsystem, which may not fully anticipate spacecraft needs. Several improvements could be made, namely accounting for designing a command and data handling subsystem that is lightweight (for very small spacecraft) or allowing for user input of requirements that would drive the overall subsystem design. These improvements would be integrated with the telecommunications subsystem subfunction to better develop both subsystems more accurately for a spacecraft design.

5.2 Structure Power Improvements

The current Structure subsystem script assumes that any power requirements for the structure subsystem would be less stringent than the overall power requirements for the rest of the spacecraft during nominal operations. This also necessitates the assumption that the structure subsystem will only draw power while other subsystems do not need to do so. This does not reflect potential realistic scenarios, such as actuation of various structural elements such as booms or antenna deployments. These improvements could also be implemented in the mission planner improvement (see below) which would allow the user to set requirements at different phases of the mission with more granularity.

5.3 SPICE Toolkit Integration

Integration of the SPICE Toolkit would greatly expand the accuracy of the spacecraft development estimation as well as further automate many analysis tasks that are still left to the user. The inclusion of SPICE integration would allow the software to automatically determine many values (such as range to sun, or range to destination) that may constrain the spacecraft design. In conjunction with loading a potential mission trajectory, SPICE Toolkit integration would allow for analysis of potential constraining circumstances that the user had not initially considered. Additionally, SPICE Toolkit integration would slim down much of the software's source code, as it would no longer have to perform analysis for many small pieces of data that SPICE could perform on its own.

5.4 Trajectory

The addition of trajectory information to the software would better allow the software to estimate spacecraft requirements more realistically given the mission constraints. Currently, the software only knows the final destination of the spacecraft, which may not be the most stressing case on the spacecraft. For example, a trajectory could be designed that would take the spacecraft very close to the sun, or very far past the destination's orbit of the sun during outbound cruise. (The most apparent case of this flaw would be a low-energy trajectory with a very long time-of-flight, well past the destination's orbit of the sun). The addition of trajectory information into the script would combine well with the future implementation of the mission planner feature, which would allow the user to specify constraining requirements during each phase of a specific trajectory. Ideally, this trajectory would be input into the system as an ephemeris file, and the software suite would use SPICE to feed information into the subfunctions for thermal and power system constraints.

5.5 Mission Planner

The addition of a mission planner would improve the software suite by allowing the user to set mission requirements with improved granularity. This would allow the software to better determine what the driving requirements of the mission are on the spacecraft design. For example, there are many obvious and common scenarios where a subsystem will have a large power draw, but this draw would not be necessary during many nominal science operations when the science payload will be drawing power. Currently, the software suite has no way of knowing whether a particular subsystem will draw power while other subsystems are also drawing power. Due to this lack of knowledge, the assumptions must be made. Either the power requirement will be higher than is truly required, or the software must make a potentially incorrect assumption that the subsystems will not need power at the same time, and set the power requirements to match accordingly. The mission planner would allow the user to set a series of low-fidelity mission phases, specifying requirements for specific mission subsystems during each of these mission phases. These phases would be identified by specifying the start and end date for each of these phases. These could be as short as an insertion maneuver, all the way up to long-term outbound cruise operations. This mission planner would also allow for gravity assists or flybys of other objects, potentially allowing the user to set additional mission constraints based on the environment of a flyby on the way to the final destination.

5.6 Launch Vehicle Selection

With the implementation of the mission planner and trajectory features into the software suite, the next step of automating the spacecraft requirements process would be to identify potential launch vehicles that would be suitable to the spacecraft and trajectory. This feature would consist of a look-up table (much like the Destination subfunction) which would then rule out any launch vehicles unsuited to the spacecraft (ie, unable to lift a heavier spacecraft into the desired orbit). This would allow the user to rapidly determine whether the mission requirements are feasible given current launch vehicle technology and cost constraints. This feature would also allow the structure subfunctions to make subtle improvements to the structure design to meet the requirements of the launch vehicle adapter, while also redesigning the structure to meet the requirements imposed on the spacecraft by the g-loading of the launch vehicle. This addition would be accomplished by determining which launch vehicles had a shroud volume sufficient for accommodating the volume of the spacecraft. The subfunction would then search for minimum departure C3s for the launch date range specified by the user to the destination, and determine which launchers (if any) were capable of delivering that C3 with the current spacecraft mass. Any launch vehicles that could accommodate the spacecraft volume as well as meet the C3 requirements would be presented to the user for further analysis outside the software.

5.7 Spacecraft Development and Operations Cost Estimate

Another feature to be developed in the future would be the addition of a cost estimating tool. During this phase of proposal development, the proposal developers are acutely concerned with the potential cost of their spacecraft design. A cost estimating model would allow the user to rapidly estimate the potential cost of a mission design and the spacecraft that would be able to meet that mission's constraints. This cost estimating tool would compare the cost of the designed spacecraft with the cost of subsystems on other spacecraft with similar capability. These estimates for each subsystem would then be summed for a design and development estimate for the overall spacecraft. Once the spacecraft development cost has been estimated, the cost estimating subfunction could work in conjunction with the mission planner subfunction to estimate the staffing levels (and thus cost) of operating the spacecraft during all phases of the mission. This tool could accept a specific budget input from the user, and assess potential budget risk based on the spacecraft and mission design, as well as inputs from the user specifying factors that could affect budget risk.

5.8 Project Management

With the addition of some assumptions about subsystem complexity and cost, a subfunction for the software could be developed which would estimate schedule needs for design and development of the spacecraft. This would allow the user to determine whether a particular launch date for a specific trajectory is feasible given the schedule risk for developing the spacecraft as estimated. Working in conjunction with the trajectory subfunction, the Project Management subfunction could assess schedule risk for meeting a specific launch date. This would be supported by additional inputs from the user to determine factors that could affect schedule risk.

5.9 Fuel and Propulsion Subsystem Development Improvements

There are a number of potential improvements that could be made to the Fuel and Propellant subfunctions to better estimate a series of spacecraft requirements. Specific potential improvements are allowing the user to select a preferred fuel type (currently, hydrazine is assumed for both the attitude control system and the main thrusters), more realistically model the power requirements for the fuel system, and implement alternative propulsion methods such as electric propulsion.

5.9.1 User Options to Select Fuel Types

Currently, hydrazine is assumed to be the fuel of choice for the attitude control systems and main engines. In future updates, the user would be allowed to select different fuel types (both monopropellants and bipropellants, with appropriate tankage and plumbing design). This would be accomplished with a look-up table similar to the Destination subfunction, which would contain pertinent data on each of the fuel types for the other subfunctions to utilize.

5.9.2 Improving Power Requirements for Particular Fuel Subsystems

The current subfunction designing the fuel subsystem does not compute any power requirements for the fuel subsystem. This is rationalized by assuming that the fuel subsystem will not require peak power needs at the same time as nominal science operations (ie, the spacecraft will likely not perform a maneuver or other critical fuel tasks while in the midst of science operations and peak power needs from the payload). With this in mind, it is assumed that the power system will be able to deliver power to the fuel system as needed when other subsystems do not require this power. This could be improved by adding logic that would calculate the power requirements for a fuel subsystem, as well as determining when those power requirements occurred. This would work in tandem with the future implementation of the Mission Planner subfunction which would allow the user to more accurately constrain the needs of the spacecraft over the life of the mission.

5.9.3 Implementation of Electric Propulsion

Another future implementation for the fuel and propulsion subsystems would be the inclusion of electric propulsion. Currently, the software only allows for the inclusion of chemical propulsion. Not all missions or mission proposals call for chemical propulsion, and the addition of electric propulsion to the software suite would allow for more robust and realistic analysis of a greater number of use cases. This addition would require substantial rework on many of the subsystem subfunctions, as this change would affect the design of many of the spacecraft subsystems and would greatly change many constraining cases.

5.10 Improved Antenna Customization

The current Telecom subfunction allows for limited customization of the antennae to be used on the spacecraft. The low and medium gain antennae are chosen by default, with no options for customization by the user. The High Gain Antenna design is an optimization of the EIRP (Equivalent Isotropically Radiated Power) between antenna size and power requirements. Providing the user with additional customization options would allow them to further refine the spacecraft requirements to better support the intended mission design. For example, customizing the telecommunications subsystem could allow the user to implement requirements secondary to the primary science mission, such as serving as a communications relay for other spacecraft in the vicinity of the intended destination of the spacecraft being designed.

5.11 Automatic Thermal Constraints

A future implementation for the Thermal subfunction will be the addition of automatic thermal constraints for various spacecraft subsystems. The current implementation of the software requires the user to know the thermal constraints of the hardware they will be using. Expecting the user to know this information or to perform the analysis to arrive at that information is not always a safe assumption. In the future, one of the outputs of each of the subsystem subfunctions would be the implementation of thermal constraints being driven by hypothetical components being used for that subfunction. This would better inform the decisions made by the thermal subsystem subfunction, while reducing the workload on the user. The user would still have the option to override the thermal constraints as set by the software in the event the software analysis does not meet the fidelity requirements for the specific mission.

5.12 Graphical User Interface

The addition of a Graphical User Interface (GUI) would allow users to more easily implement their mission designs prior to running the software. This would assist in preventing errors, and would reduce the amount of time spent inputting data into the code. The current software requires knowledge of MATLAB and how it handles data. With a GUI, a user with potentially some to zero knowledge of MATLAB could input the necessary constraints and mission requirements with ease. The addition of a GUI could also allow the user to easily pull data from the software for further manual analysis or inclusion in a proposal. This data would include the spacecraft requirements, telecommunications link budgets, Mass Equipment Lists, and Power Equipment Lists.

5.13 Improved Margin Determination

Further refinement of spacecraft mass and power margins must be included in a future build of the software. This addition of margin would allow for growth of mass and power needs as the spacecraft design is further developed. The current software does not assume mass growth in all subsystems, only in the fuel requirements. Inclusion of user-definable mass and power growth margins will allow the user to more confidently predict the final requirements of the spacecraft.

6 Conclusion

The development and testing of a software suite for streamlining the spacecraft proposal process is discussed. Justification for the source code is detailed, with an explanation of how each subfunction serves the overall spacecraft design process. Two case studies are explored, with each showing that the software demonstrates the ability to accurately meet spacecraft requirements when compared to a real-world example, as well as demonstrating that the software will not generate a feasible spacecraft design if it is not provided a reasonable mission concept as an input. Potential errors revealed by the analysis performed for the case studies are also discussed. The software has the potential to assist spacecraft scientists and engineers in the rapid development of spacecraft requirements to assess potential mission feasibilities.

7 Appendices

7.1 APPENDIX A - Software Code

7.1.1 Master Controller Script

```
function [Requirements] = AutomatedSpacecraftDesign(Requirements)

%This script calls many other subfunctions in an effort to prepare a
%preliminary spacecraft design document.

%Mission Requirements
Requirements = struct;
Requirements.Maneuver.DeltaV = 4.3; %km/s
Requirements.Destination.Name = input('Enter a destination.','s');
Requirements.DestinationMinRange = 200; %Orbit/Flyby Minimum Altitude, km
Requirements.Payload.Mass = 80; %Payload Mass, kg
Requirements.Payload.Power = 70; %Payload Power, W
Requirements.Payload.Volume = 20000; %Payload Volume, cm^3
Requirements.Payload.Count = 2;
Requirements.Payload.DataStorage = 25; %Gbit
Requirements.Pointing = 6; %degrees/sec
Requirements.SpacecraftMass = Requirements.Payload.Mass;
Requirements.SpacecraftPower = Requirements.Payload.Power;
Requirements.OIThrust = 1200; %N
Requirements.PrimaryMissionLength = 360; %days
Requirements.Thermal.Max = 25; %degrees C
Requirements.Thermal.Min = 2; %degrees C
Requirements.SpecificImpulse = 230; %sec
Requirements.Prop.LargeThruster = 170; %N
Requirements.Prop.MediumThruster = 22; %N
Requirements.Prop.SmallThruster = 0.9; %N
Requirements.FlybyFlag = 0;

%Fetch destination information.
[Requirements] = Destination(Requirements);
Requirements.SunMaxRange = Requirements.Destination.TransmitterMaxRange - 1.496 * 10^8;
Requirements.EarthMaxRange = Requirements.Destination.TransmitterMaxRange;

%Calculate telecommunications system
[TelecomMass, TelecomPower] = Telecom(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + TelecomMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + TelecomPower;

%Calculate command & data handling
[CDHMass, CDHPower] = CDH(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + CDHMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + CDHPower;

%Calculate propulsion system
[PropMass, PropPower] = Prop(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + PropMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + PropPower;

%Calculate attitude control
[AttitudeMass, AttitudePower] = Attitude(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + AttitudeMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + AttitudePower;

%Calculate power system
[PowerMass, PowerPower] = Power(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + PowerMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + PowerPower;

%Calculate structure
[StructureMass, StructurePower, SpacecraftVolume] = Structure(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + StructureMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + StructurePower;
Requirements.SpacecraftVolume = SpacecraftVolume;

%Calculate thermal system
[ThermalMass, ThermalPower] = Thermal(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + ThermalMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + ThermalPower;

%Calculate fuel system
[FuelSystemMass, FuelSystemPower, WetFuelMass] = Fuel(Requirements);
Requirements.SpacecraftMass = Requirements.SpacecraftMass + FuelSystemMass;
Requirements.SpacecraftPower = Requirements.SpacecraftPower + FuelSystemPower;

TotalSpacecraftMass = Requirements.SpacecraftMass + WetFuelMass;
```

```
TotalSpacecraftPower = Requirements.SpacecraftPower;

end
```

7.1.2 Destination Subfunction

```
function [Requirements] = Destination(Requirements)

if strcmp(Requirements.Destination.Name,'Mercury') == 1
    Requirements.Destination.IR = 4150;
    Requirements.Destination.Albedo = 0.106;
    Requirements.Destination.Radius = 2439.7;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (6.156 * 10^7);
    Requirements.Destination.SolarFlux = 9228;
    Requirements.Destination.mu = 2.2032 * 10^4;
    Requirements.Destination.SolarPressure = 3.05 * 10^-5;

elseif strcmp(Requirements.Destination.Name,'Venus') == 1
    Requirements.Destination.IR = 153;
    Requirements.Destination.Albedo = 0.65;
    Requirements.Destination.Radius = 6051.8;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (1.087 * 10^8);
    Requirements.Destination.SolarFlux = 2586;
    Requirements.Destination.mu = 3.24859 * 10^5;
    Requirements.Destination.SolarPressure = 8.77 * 10^-6;

elseif strcmp(Requirements.Destination.Name,'Earth') == 1
    Requirements.Destination.IR = 231;
    Requirements.Destination.Albedo = 0.367;
    Requirements.Destination.Radius = 6378.14;
    Requirements.Destination.TransmitterMaxRange = 30000;
    Requirements.Destination.SolarFlux = 1353;
    Requirements.Destination.mu = 3.986 * 10^5;
    Requirements.Destination.SolarPressure = 4.59 * 10^-6;

elseif strcmp(Requirements.Destination.Name,'Mars') == 1
    Requirements.Destination.IR = 162;
    Requirements.Destination.Albedo = 0.15;
    Requirements.Destination.Radius = 3397;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (2.438 * 10^8);
    Requirements.Destination.SolarFlux = 586;
    Requirements.Destination.mu = 4.282 * 10^4;
    Requirements.Destination.SolarPressure = 2 * 10^-6;

elseif strcmp(Requirements.Destination.Name,'Jupiter') == 1
    Requirements.Destination.IR = 13.5;
    Requirements.Destination.Albedo = 0.52;
    Requirements.Destination.Radius = 71492;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (8.155 * 10^8);
    Requirements.Destination.SolarFlux = 50;
    Requirements.Destination.mu = 1.266 * 10^8;
    Requirements.Destination.SolarPressure = 1.7 * 10^-7;

elseif strcmp(Requirements.Destination.Name,'Saturn') == 1
    Requirements.Destination.IR = 4.6;
    Requirements.Destination.Albedo = 0.47;
    Requirements.Destination.Radius = 60268;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (1.505 * 10^9);
    Requirements.Destination.SolarFlux = 15;
    Requirements.Destination.mu = 3.794 * 10^7;
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);

elseif strcmp(Requirements.Destination.Name,'Uranus') == 1
    Requirements.Destination.IR = 0.63;
    Requirements.Destination.Albedo = 0.51;
    Requirements.Destination.Radius = 25559;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (2.98 * 10^9);
    Requirements.Destination.SolarFlux = 4;
    Requirements.Destination.mu = 5.793 * 10^6;
    Requirements.Destination.SolarPressure = 1.24 * 10^-8;

elseif strcmp(Requirements.Destination.Name,'Neptune') == 1
    Requirements.Destination.IR = 0.52;
    Requirements.Destination.Albedo = 0.41;
    Requirements.Destination.Radius = 24764;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (4.48 * 10^9);
    Requirements.Destination.SolarFlux = 2;
    Requirements.Destination.mu = 6.809 * 10 ^ 6;
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);

elseif strcmp(Requirements.Destination.Name,'Pluto') == 1
    Requirements.Destination.IR = 0.5;
    Requirements.Destination.Albedo = 0.3;
    Requirements.Destination.Radius = 1195;
    Requirements.Destination.TransmitterMaxRange = (1.496 * 10^8) + (7.38 * 10^9);
    Requirements.Destination.SolarFlux = 1;
    Requirements.Destination.mu = 9 * 10^2;
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);

elseif strcmp(Requirements.Destination.Name,'Moon') == 1
    Requirements.Destination.IR = 430;
    Requirements.Destination.Albedo = 0.12;
    Requirements.Destination.Radius = 1737.4;
    Requirements.Destination.TransmitterMaxRange = 390836;
    Requirements.Destination.SolarFlux = 1353;
    Requirements.Destination.mu = 4902.79;
    Requirements.Destination.SolarPressure = 4.59 * 10^-6;

elseif strcmp(Requirements.Destination.Name,'DeepSpace') == 1
    Requirements.Destination.IR = 0;
    Requirements.Destination.Albedo = 0;
    Requirements.Destination.Radius = 0;
    Requirements.Destination.TransmitterMaxRange = input('What is the maximum range of the
destination to Earth in km?');
    Requirements.Destination.SolarFlux = (3.828 *
10^26)/(Requirements.Destination.TransmitterMaxRange^2);
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);

else
    %Ask user to specify IR, Albedo, and Radius.
    Requirements.Destination.IR = input('What is the Orbit-Average IR of the destination in
W/m^2?');
    Requirements.Destination.Albedo = input('What is the Geometric albedo of the destination?');
    Requirements.Destination.Radius = input('What is the radius of the destination in km?');
    Requirements.Destination.TransmitterMaxRange = input('What is the maximum range of the
destination to Earth in km?');
    Requirements.Destination.SolarFlux = input('What is the solar flux of the destination?');
    Requirements.Destination.mu = input('What is the standard gravitational parameter of the
destination in km^3-s^-2?');
    Requirements.Destination.SolarPressure = (Requirements.Destination.SolarFlux)/(2.998*10^8);

end
```

7.1.3 Telecommunications Subfunction

```
function [TelecomMass, TelecomPower] = Telecom(Requirements)

AntennaEfficiency = 0.65;
AntennaFrequency = 8.4 * 10^9;
c = 2.98 * 10^8;

if Requirements.Payload.Mass > 160
syms x
f(x) = (AntennaEfficiency * (0.5*Requirements.Payload.Mass - 80 - 4 * x ^2) * (pi * x / ((3 *
10^8) / AntennaFrequency)^2));
df = diff(f,x);
AntennaRoots = solve(0 == df,x);
ARootsPos = double(AntennaRoots(AntennaRoots>0));
AntennaDiameter = max(0.4,ARootsPos);
LightweightAntennaMode = 0;

elseif Requirements.Payload.Mass < 27.999
    AntennaDiameter = 0.4;
    LightweightAntennaMode = 1;

else
    AntennaDiameter = 0.68;
    LightweightAntennaMode = 2;

end

if LightweightAntennaMode == 0
    %Calculate weight using formula
    HGAAntennaWeight = max(2.1,2.89 * AntennaDiameter ^ 2 + 6.11 * AntennaDiameter - 2.59);
elseif LightweightAntennaMode == 1
    HGAAntennaWeight = 2.1;
```

```
elseif LightweightAntennaMode == 2
    HGAAntennaWeight = 7;
else
    %An error has occurred.
end

PowerTransmitted = Requirements.Payload.Mass - 80 - 4 * AntennaDiameter;

TransponderWeight = 7.6;
ControlUnitWeight = 10.9;
TWTAWeight = 6.2;
RFSComponentsWeight = 8;
MediumGainAntennaWeight = 2.1;
CoaxCableWeight = 7.8;

ControlUnitPower = 12.2;
XExciterPower = 1.4;
ReceiverPower = 6.8;
TWTAPower = 67;
XSDownConvertPower = 2.1;


TelecomPower = ControlUnitPower + XExciterPower + ReceiverPower + PowerTransmitted + TWTAPower +
XSDownConvertPower; %W
TelecomMass = TransponderWeight + ControlUnitWeight + TWTAWeight + RFSComponentsWeight +
HGAAntennaWeight + MediumGainAntennaWeight + CoaxCableWeight;

ModulationIndex = 1; %radians
Frequency = AntennaFrequency/10^9; %GHz
BitErrorRate = 1 * 10^-5;
Range = Requirements.Destination.TransmitterMaxRange; %km
SymbolRate = 0.125; %bps
TransmitterPower = PowerTransmitted;
CableLoss = 0;
AntennaGain = 67; %34 m DSN
EIRP = TransmitterPower + CableLoss + AntennaGain;
FreeSpacePathLoss = 92.44 + (20*log10(Frequency)) + (20*log10(Range));
AtmosphericAttenuation = -0.17; %dB
PolarizationLoss = -1; %dB
SCAntennaGain = 10 * log10(AntennaEfficiency * ((pi *
AntennaDiameter)/((c)/(AntennaFrequency)))^2); %dBi
PointingLoss = -5; %dB
ReceivedCableLoss = -1.95; %dB
TotalReceivedPower = EIRP + (-1*FreeSpacePathLoss) + AtmosphericAttenuation + PolarizationLoss +
SCAntennaGain + PointingLoss + ReceivedCableLoss;
SystemNoiseTemperature = 18;
SystemNoiseDensity = -228.6 + 10 * log10(SystemNoiseTemperature); %dB/Hz
CarrierPowerTotalPower = 20 * log10(cos(ModulationIndex)); %dB
CarrierPowerReceived = TotalReceivedPower + CarrierPowerTotalPower; %dB
CarrierNoiseBandwidth = 13; %dB-Hz
CarrierNoiseRatioReceived = CarrierPowerReceived - SystemNoiseDensity - CarrierNoiseBandwidth;
%dB
CarrierNoiseRatioRequired = 6; %dB
CarrierMargin = CarrierNoiseRatioReceived - CarrierNoiseRatioRequired; %dB
```

```
CommandPowerTotalPower = 20 * log10(sin(ModulationIndex)); %dB
CommandPowerReceived = TotalReceivedPower + CommandPowerTotalPower; %dB
CommandSymbolRate = -10 * log10(SymbolRate) - 30; %dB-Hz
EbNoAchieved = CommandPowerReceived + CommandSymbolRate - SystemNoiseDensity; %dB
EbNoRequired = 4.2; %dB
CommandLinkMargin = EbNoAchieved - EbNoRequired; %dB

end
```

7.1.4 C&DH Subfunction

```
function [CDHMass, CDHPower] = CDH(Requirements)

IndependentDataRateEquipmentMass = 30;
IndependentDataRateEquipmentPower = 20;

ComputerMass = 2;
ComputerPower = 10;

ScienceDataProcessorMass = 15;
ScienceDataProcessorPower = 2 * Requirements.Payload.Count + 1;

EngineeringDataProcessorMass = 10;
EngineeringDataProcessorPower = 5;

DataStorageMass = 0.25 * Requirements.Payload.DataStorage;
DataStoragePower = 1 * Requirements.Payload.DataStorage;

CDHMass = IndependentDataRateEquipmentMass + ComputerMass + ScienceDataProcessorMass +
EngineeringDataProcessorMass + DataStorageMass;
CDHPower = IndependentDataRateEquipmentPower + ComputerPower + ScienceDataProcessorPower +
EngineeringDataProcessorPower + DataStoragePower;

end
```

7.1.5 Propulsion Subfunction

```
function [PropMass, PropPower] = Prop(Requirements)

LargeThrusterWeight = 0.34567 * Requirements.Prop.LargeThruster^0.55235;
MediumThrusterWeight = 0.34567 * Requirements.Prop.MediumThruster^0.55235;
SmallThrusterWeight = 0.4;

OIThrust = Requirements.OIThrust;
LargeThrusterCount = ceil(OIThrust/Requirements.Prop.LargeThruster);

TCMThrust = 22 * 6;
MediumThrusterCount = ceil(TCMThrust/Requirements.Prop.MediumThruster);

SmallThrusterCount = 12;
```

```
PropMass = LargeThrusterCount * LargeThrusterWeight + MediumThrusterCount * MediumThrusterWeight
+ SmallThrusterCount * SmallThrusterWeight;
PropPower = 0.01 * ((Requirements.SpacecraftPower - Requirements.Payload.Power)/0.4);

end
```

7.1.6 Attitude Control System Subfunction

```
function [AttitudeMass, AttitudePower] = Attitude(Requirements)

SpacecraftMass = 1/0.39 * Requirements.SpacecraftMass;
SubsystemPacking = 320.4; %kg/m^3
PackingEnvelopeHeight = 0.254; %meters
SubsystemVolumeWithoutEnvelope = ((Requirements.SpacecraftMass -
Requirements.Payload.Mass))/(SubsystemPacking);
SubsystemRadiusWithoutEnvelope = ((0.75 * SubsystemVolumeWithoutEnvelope)/(pi))^(1/3);
SubsystemVolume = (4/3) * pi * (SubsystemRadiusWithoutEnvelope + PackingEnvelopeHeight)^3;
SpacecraftVolume = SubsystemVolume + Requirements.Payload.Volume;
SpacecraftRadius = ((3 * SpacecraftVolume)/(4 * pi)) ^ (1/3);

%Determine the thruster capability
MOI = 0.5 * (SpacecraftMass * SpacecraftRadius^2);
theta = 180;
n = 2;
LeverArm = SpacecraftRadius;
t_b = 30;
F = (8 * MOI * theta)/(n * LeverArm * t_b^2);

%Determine the amount of solar torque acting on the spacecraft
SolarPressure = Requirements.Destination.SolarPressure;
Area = (1/2) * (4 * pi * SpacecraftRadius ^ 2); %Area facing sun
L = SpacecraftRadius; %Spacecraft moment arm
q = 0.5; %Spacecraft reflectivity
SolarTorque = SolarPressure * Area * L * (1 + q);

%Determine the momentum buildup over one orbit
OrbitPeriod = ((2 * pi)/(sqrt(Requirements.Destination.mu))) * ((Requirements.Destination.Radius
+ Requirements.DestinationMinRange) ^ (3/2));
MomentumBuildup = SolarTorque * OrbitPeriod;

%Time Between Desats
WheelStorage = 100;
WheelSaturate = WheelStorage/MomentumBuildup;
DesatTime = OrbitPeriod * WheelSaturate;
DesatCount = (Requirements.PrimaryMissionLength * 86400)/DesatTime;

%Desat Force
F = WheelStorage/(n * L);

%Desat Prop Usage
BurnTime = WheelSaturate/F;
PropMass = DesatCount * (n * F * BurnTime)/290;
```

```
SunSensorPower = 1;
SunSensorMass = 0.5;

StarTrackerCount = 2;
StarTrackerPower = StarTrackerCount * 18;
StarTrackerMass = StarTrackerCount * 7;

ReactionWheelCount = 4;
ReactionWheelPower = ReactionWheelCount * 21.4;
ReactionWheelMass = ReactionWheelCount * 8.5;

AttitudeMass = SunSensorMass + StarTrackerMass + ReactionWheelMass + PropMass;
AttitudePower = SunSensorPower + StarTrackerPower + ReactionWheelPower;

end
```

7.1.7 Power Subfunction

```
function [PowerMass, PowerPower] = Power(Requirements)

SciencePowerRequirement = (Requirements.SpacecraftPower - Requirements.Payload.Power) / 0.61;

SolarRange = Requirements.Destination.TransmitterMaxRange - (1.496 * 10^8);
if SolarRange < (9.652 * 10^8)
    SolarFlag = 1;
else
    SolarFlag = 0;
end

if SolarFlag == 1
    %Use solar panels
    Solar2Loads = 0.95;
    Solar2Battery = 0.7;
    Battery2Load = 0.96;

    if strcmp(Requirements.Destination,'DeepSpace') %Deep space will never meaningfully eclipse,
NightMax = 0
        NightMax = 0;
    else
        %Nighttime
        alpha = asin(Requirements.Destination.Radius/(Requirements.Destination.Radius +
Requirements.DestinationMinRange));
        OrbitPeriod = (2 * pi * sqrt((Requirements.Destination.Radius +
Requirements.DestinationMinRange)^3/(Requirements.Destination.mu)))/60;
        NightMax = (OrbitPeriod * ((alpha)/pi))/60;
    end

        %%Solar Array Requirements
        T_sun = OrbitPeriod - (NightMax * 60);
        P_sa = ((SciencePowerRequirement * NightMax)/(Solar2Battery * Battery2Load * T_sun)) +
((SciencePowerRequirement)/(Solar2Loads));
```

```
        %%Battery System Requirements
        DOD = 0.4;
        DischargeVoltage = 28; %volts
        BatteryCapacity = (SciencePowerRequirement * NightMax) / (Battery2Load * DOD *
DischargeVoltage);

        %%Calculate size and mass of solar panels
        RadiationDegradation = 0.18;
        ArrayTemp = 120;
        ArrayError = 5;
        CellOutput = 0.172; %W/cell
        UVDegradation = 0.98;
        ThermalDegradation = 0.99;
        CellMismatchLoss = 0.975;
        CellResistanceLoss = 0.99;
        ContaminationLoss = 0.99;
        ShadowLoss = 1;
        TempAdjustment = 1;
        SolarIntensity = SolarRange/(1.496 * 10^8);
        PointingLoss = cos(ArrayError);
        PowerCellActual = CellOutput * (UVDegradation * ThermalDegradation * CellMismatchLoss *
CellResistanceLoss * ContaminationLoss * ShadowLoss * (1 - RadiationDegradation) * TempAdjustment
* SolarIntensity * PointingLoss);
        NumberOfCells = P_sa/PowerCellActual;
        CellDensity = (10000/8) * 0.88;
        SolarArrayArea = NumberOfCells/CellDensity;

        %%Mass and power estimates for subsytems
        SolarArrayMass = 4 * SolarArrayArea;
        BatteryMass = BatteryCapacity / 24;

        PowerMass = SolarArrayMass + BatteryMass + 63.7 + 55.5;
        PowerPower = 0.1 * SciencePowerRequirement;

else
    %Use RTGs
    PowerMass = SciencePowerRequirement / 4.93;
    PowerPower = 0.1 * SciencePowerRequirement;
end

end
```

7.1.8 Structure Subfunction

```
function [StructureMass, StructurePower, SubsystemVolume] = Structure(Requirements)

%Compute subsystem volume
SubsystemPacking = 320.4; %kg/m^3
PackingEnvelopeHeight = 0.254; %meters
SubsystemVolumeWithoutEnvelope = ((Requirements.SpacecraftMass -
Requirements.Payload.Mass))/(SubsystemPacking);
SubsystemRadiusWithoutEnvelope = ((0.75 * SubsystemVolumeWithoutEnvelope)/(pi))^(1/3);
SubsystemVolume = (4/3) * pi * (SubsystemRadiusWithoutEnvelope + PackingEnvelopeHeight)^3;
```

```
%Aluminum Values
AlDensity = 2700;

%Fuel Needs Estimate
FuelEstimationFactor = 2;
MassEstimate = Requirements.SpacecraftMass/0.61;
FuelEstimate = (FuelEstimationFactor * MassEstimate) * (1 - exp(-
(Requirements.Maneuver.DeltaV*1000)/(9.81 * Requirements.SpecificImpulse)));
PropDensity = 1010;
PropVolume = (FuelEstimate/PropDensity);% * 3.531 * 10^-5;

%Total Spacecraft Volume
InstVolume = Requirements.Payload.Volume * 1 * 10^-6;% * 3.531 * 10^-5;
TotalSpacecraftVolume = SubsystemVolume + PropVolume + InstVolume; %m^3

%Estimate the structure mass.
R = (TotalSpacecraftVolume/pi)^(1/3);
SCThickness = 0.005;

StructureMass = AlDensity * ((pi * (R + SCThickness)^3) - TotalSpacecraftVolume);
StructurePower = 0;

end
```

7.1.9 Thermal Subfunction

```
function [ThermalMass, ThermalPower] = Thermal(Requirements)

%Establish requirements
ThermalMax = Requirements.Thermal.Max;
ThermalMin = Requirements.Thermal.Min;
alpha = 0.32;
epsilon = 0.8;
PowerDissipation = Requirements.SpacecraftPower;

SphereDiameter = 2 * (((3 * Requirements.SpacecraftVolume)/(4 * pi))^(1/3));

%Determine "worst-case" heating
if strcmp(Requirements.Destination,'Mercury') == 1
    WorstCaseIR = Requirements.Destination.IR;
    WorstCaseAlbedo = Requirements.Destination.Albedo;
    WorstCaseRadius = Requirements.Destination.Radius;
    WorstCaseSolarFlux = Requirements.Destination.SolarFlux;
    SpacecraftAltitude = Requirements.DestinationMinRange;

elseif strcmp(Requirements.Destination,'Venus') == 1
    WorstCaseIR = Requirements.Destination.IR;
    WorstCaseAlbedo = Requirements.Destination.Albedo;
    WorstCaseRadius = Requirements.Destination.Radius;
    WorstCaseSolarFlux = Requirements.Destination.SolarFlux;
    SpacecraftAltitude = Requirements.DestinationMinRange;
```

```
else %Use Earth as "worst case" heating otherwise
WorstCaseIR = 237;
WorstCaseAlbedo = 0.3;
WorstCaseRadius = 6378;
WorstCaseSolarFlux = 1371;
SpacecraftAltitude = 200;

end

WorstCaseViewFactor = 0.5 * (1 - (((SpacecraftAltitude^2 +
2*SpacecraftAltitude*WorstCaseRadius)^0.5)/(SpacecraftAltitude + WorstCaseRadius)));
ReflectanceFactor = 0.657 + 0.54 * (WorstCaseRadius/(WorstCaseRadius + SpacecraftAltitude)) -
0.196 * (WorstCaseRadius/(WorstCaseRadius + SpacecraftAltitude))^2;
sigma = 5.67 * 10^-8; %Stefan-Boltzman Constant
WorstCaseSpacecraftTemp = ((((WorstCaseSolarFlux * alpha)/4) + (WorstCaseIR * epsilon *
WorstCaseViewFactor) + (WorstCaseSolarFlux * WorstCaseAlbedo * alpha * ReflectanceFactor *
WorstCaseViewFactor) + (PowerDissipation/(pi * SphereDiameter^2)))/(sigma * epsilon)) ^ (1/4);

%Determine "best-case" heating
if strcmp(Requirements.Destination,'Mercury') == 1
Earth.IR = 237;
Earth.Albedo = 0.3;
Earth.Radius = 6378;
WorstCaseSolarFlux = 1371;
SpacecraftAltitude = 100;

elseif strcmp(Requirements.Destination,'Venus') == 1
Earth.IR = 237;
Earth.Albedo = 0.3;
Earth.Radius = 6378;
WorstCaseSolarFlux = 1371;
SpacecraftAltitude = 100;

else %Use Destination as "best case" heating otherwise
    BestCaseIR = Requirements.Destination.IR;
    BestCaseAlbedo = Requirements.Destination.Albedo;
    BestCaseRadius = Requirements.Destination.Radius;
    BestCaseSolarFlux = Requirements.Destination.SolarFlux;
    SpacecraftAltitude = Requirements.DestinationMinRange;

end

BestCaseViewFactor = 0.5 * (1 - (((SpacecraftAltitude^2 +
2*SpacecraftAltitude*BestCaseRadius)^0.5)/(SpacecraftAltitude + BestCaseRadius)));
ReflectanceFactor = 0.657 + 0.54 * (BestCaseRadius/(BestCaseRadius + SpacecraftAltitude)) - 0.196
* (BestCaseRadius/(BestCaseRadius + SpacecraftAltitude))^2;
sigma = 5.67 * 10^-8; %Stefan-Boltzman Constant
BestCaseSpacecraftTemp = ((((BestCaseSolarFlux * alpha)/4) + (BestCaseIR * epsilon *
BestCaseViewFactor) + (BestCaseSolarFlux * BestCaseAlbedo * alpha * ReflectanceFactor *
BestCaseViewFactor) + (PowerDissipation/(pi * SphereDiameter^2)))/(sigma * epsilon)) ^ (1/4);

Q_w1 = WorstCaseSpacecraftTemp - ThermalMax;
RadiatorArea = Q_w1/(sigma * epsilon * WorstCaseSpacecraftTemp ^ 4);
```

```
Q_w2 = BestCaseSpacecraftTemp - ThermalMin;
WorstCaseRadiatorTemp = (Q_w2/(RadiatorArea * sigma * epsilon))^(1/4);

HeaterCheck = WorstCaseRadiatorTemp - ThermalMin;
HeaterPowerReq = max(0,HeaterCheck);
if HeaterPowerReq == 0
    HeaterMass = 0;
else
    HeaterMass = 2;
end

r = (Requirements.SpacecraftVolume/pi)^(1/3);
h = r;
SurfaceArea = (2 * pi * r * h) + (2 * pi * r ^ 2); %m^2
InsulationMass = 0.03 * SurfaceArea; %kg
PaintMass = 0.24 * SurfaceArea; %kg
FoamMass = 64 * 0.075 * Requirements.SpacecraftVolume;
RadiatorMass = 0.03 * RadiatorArea;

ThermalMass = RadiatorMass + HeaterMass + InsulationMass + PaintMass + FoamMass;
ThermalPower = HeaterPowerReq;
end
```

7.1.10 Fuel Subfunction

```
function [FuelSystemMass, FuelSystemPower, WetFuelMass] = Fuel(Requirements)

SpacecraftVolume = Requirements.SpacecraftVolume;

Isp = Requirements.SpecificImpulse;

Isp_steady = 0.93 * Isp;
Isp_pulsing = 0.5 * Isp;

%Fuel requirements for spin-up to 5 RPM from 0 and back to 0.
F = Requirements.Prop.SmallThruster; %Thruster thrust value
n = 2; %Number of thrusters in the maneuver
L = ((3 * SpacecraftVolume)/(4 * pi)) ^ (1/3); %Spacecraft radius
omega_b = 5; %rpm
t_b = omega_b/((n*F*L)/(Isp_steady));
SpinUpDownProp = 2 * (n*F*t_b)/(Isp_steady);

%Fuel requirements for attitude control during orbit insertion
if Requirements.FlybyFlag == 0
    %Check if the mission will have an orbit insertion, or merely a flyby
    if strcmp(Requirements.Destination.Name,'DeepSpace')
        OIProp = 0;
        OIProp2 = 0;
    else
        %Calculate orbit insertion attitude control fuel
        F = Requirements.Prop.SmallThruster;
        omega = 60;
        t_b = sqrt((2 * Isp_steady * omega)/(n * F * L));
        OIProp = 2 * (n * F * t_b)/(Isp_steady);

        OrbitInsertionLength = 27; %minutes
        P_w = 60 * OrbitInsertionLength * 2;
        OIProp2 = 2*((n * F * P_w)/Isp_pulsing);
    end
else
    OIProp = 0;
    OIProp2 = 0;
end

%Fuel requirements for limit cycle characteristics
F = Requirements.Prop.SmallThruster;
Cycle_Duration = (8 * Isp_pulsing * 1)/(n * L * F * 0.5);

%%%Number of cycles over the course of the mission.
TotalCycles = Requirements.PrimaryMissionLength/Cycle_Duration;

%%%Mass of propellant requires to maintian three limit cycles
ThreeLimitCyclePropMass = 6 * ((n * F * 0.03)/(Isp_pulsing * 9.8067));

%Total ACS Fuel Inventory
ACSHydrazinePropInventory = (SpinUpDownProp + OIProp + OIProp2 + TotalCycles *
ThreeLimitCyclePropMass) * 1.035;

HydrazineBlowdownRatio = 4.5;
HydrazineInitialPressure = 625 * 6.89476;

ACSHydrazineVolume = ACSHydrazinePropInventory/1.01;
InitialACSUllageVolume = ACSHydrazineVolume/(HydrazineBlowdownRatio - 1);

BladderRadius = sqrt((0.75 * (ACSHydrazineVolume + InitialACSUllageVolume))/(3.141));
Area_Bladder = 2 * 3.141 * BladderRadius ^ 2;
BladderVolume = 0.075 * Area_Bladder;
TotalACSHydrazineVolume = ACSHydrazineVolume + InitialACSUllageVolume + BladderVolume;

%Tank Weight
ACSTankWeight = 0.0116 * HydrazineInitialPressure * TotalACSHydrazineVolume/1000;

DryMass = Requirements.SpacecraftMass + ACSTankWeight + ACSHydrazinePropInventory;
FuelEstimate = DryMass;
MainHydrazinePropInventory = (DryMass + FuelEstimate) * (1 - exp(-
(Requirements.Maneuver.DeltaV*1000)/(9.81 * Requirements.SpecificImpulse)));
MainHydrazineVolume = MainHydrazinePropInventory/1.01;
InitialMainUllageVolume = MainHydrazineVolume/(HydrazineBlowdownRatio - 1);

BladderRadius = sqrt((0.75 * (MainHydrazineVolume + InitialMainUllageVolume))/(3.141));
Area_Bladder = 2 * 3.141 * BladderRadius ^ 2;
BladderVolume = 0.075 * Area_Bladder;
TotalMainHydrazineVolume = MainHydrazineVolume + InitialMainUllageVolume + BladderVolume;
MainTankWeight = 0.0116 * HydrazineInitialPressure * TotalMainHydrazineVolume/1000;

Plumbing = 20;
FuelSystemMass = ACSTankWeight + MainTankWeight + Plumbing;
```

```
FuelSystemPower = 0;
WetFuelMass = ACSHydrazinePropInventory + MainHydrazinePropInventory;

end
```

7.1.11 Test Script

```
%This script will rapidly produce a series of test requirements.
Requirements = struct; %Test Requirements
Requirements.Maneuver.DeltaV = 4.3; %km/s
% Requirements.Destination.Name = input('Enter a destination.','s');
Requirements.DestinationMinRange = 200; %Orbit/Flyby Minimum Altitude, km
Requirements.Payload.Mass = 80; %Payload Mass, kg
Requirements.Payload.Power = 70; %Payload Power, W
Requirements.Payload.Volume = 20000; %Payload Volume, cm^3
Requirements.Payload.Count = 2;
Requirements.Payload.DataStorage = 25; %Gbit
Requirements.Pointing = 6; %degrees/sec
Requirements.SpacecraftMass = Requirements.Payload.Mass;
Requirements.SpacecraftPower = Requirements.Payload.Power;
Requirements.OIThrust = 1200;
Requirements.PrimaryMissionLength = 360; %days
Requirements.Thermal.Max = 25;
Requirements.Thermal.Min = 2;
Requirements.SpecificImpulse = 230;
Requirements.Prop.LargeThruster = 170;
Requirements.Prop.MediumThruster = 22;
Requirements.Prop.SmallThruster = 0.9;
Requirements.FlybyFlag = 0;
n = 0;
Requirements.Destination.Name = 'Mercury';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);
                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Venus';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Earth';
```

```
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Mars';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;
```

```
                                  [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                  fileID = fopen('Mass.txt','a');
                                  fmt = '%5d\n';
                                  fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                  fclose(fileID);

                                  fileID = fopen('Power.txt','a');
                                  fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                  fclose(fileID);

                                  FlybyTest = FlybyTest + 1;
                              end
                  PayloadPowerTest = PayloadPowerTest + 100;
              end
              PayloadMassTest = PayloadMassTest + 25;
          end
      DeltaVTest = DeltaVTest + 2
      toc
  end

  Requirements.Destination.Name = 'Jupiter';
  DeltaVTest = 0.5;
  while DeltaVTest < 10
      tic
      Requirements.Maneuver.DeltaV = DeltaVTest;
          PayloadMassTest = 10;
          while PayloadMassTest < 200
              Requirements.Payload.Mass = PayloadMassTest;
              PayloadPowerTest = 50;
              while PayloadPowerTest < 1000
                  Requirements.Payload.Power = PayloadPowerTest;
                              FlybyTest = 0;
                              while FlybyTest < 2
                                  Requirements.FlybyFlag = FlybyTest;

                                  Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                  Requirements.SpacecraftPower = Requirements.Payload.Power;

                                  [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                  fileID = fopen('Mass.txt','a');
                                  fmt = '%5d\n';
                                  fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                  fclose(fileID);

                                  fileID = fopen('Power.txt','a');
                                  fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                  fclose(fileID);

                                  FlybyTest = FlybyTest + 1;
                              end
                  PayloadPowerTest = PayloadPowerTest + 100;
              end
              PayloadMassTest = PayloadMassTest + 25;
```

```
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Saturn';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Neptune';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
```

```
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end

Requirements.Destination.Name = 'Uranus';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);
                                FlybyTest = FlybyTest + 1;
                            end
                    PayloadPowerTest = PayloadPowerTest + 100;
                end
                PayloadMassTest = PayloadMassTest + 25;
            end
        DeltaVTest = DeltaVTest + 2
        toc
end

Requirements.Destination.Name = 'Pluto';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                    PayloadPowerTest = PayloadPowerTest + 100;
                end
                PayloadMassTest = PayloadMassTest + 25;
            end
        DeltaVTest = DeltaVTest + 2
        toc
end

Requirements.Destination.Name = 'Moon';
DeltaVTest = 0.5;
while DeltaVTest < 10
    tic
    Requirements.Maneuver.DeltaV = DeltaVTest;
```

```
        PayloadMassTest = 10;
        while PayloadMassTest < 200
            Requirements.Payload.Mass = PayloadMassTest;
            PayloadPowerTest = 50;
            while PayloadPowerTest < 1000
                Requirements.Payload.Power = PayloadPowerTest;
                            FlybyTest = 0;
                            while FlybyTest < 2
                                Requirements.FlybyFlag = FlybyTest;

                                Requirements.SpacecraftMass = Requirements.Payload.Mass;
                                Requirements.SpacecraftPower = Requirements.Payload.Power;

                                [Requirements] = AutomatedSpacecraftDesign(Requirements);
                                fileID = fopen('Mass.txt','a');
                                fmt = '%5d\n';
                                fprintf(fileID,fmt,Requirements.SpacecraftMass);
                                fclose(fileID);

                                fileID = fopen('Power.txt','a');
                                fprintf(fileID,fmt,Requirements.SpacecraftPower);
                                fclose(fileID);

                                FlybyTest = FlybyTest + 1;
                            end
                PayloadPowerTest = PayloadPowerTest + 100;
            end
            PayloadMassTest = PayloadMassTest + 25;
        end
    DeltaVTest = DeltaVTest + 2
    toc
end
```

8 References

[1] C. D. Brown, Elements of Spacecraft Design, Reston, VA: American Institute of Aeronautics and Astronautics, 2002.

[2] MathWorks, "MATLAB," 01 08 2017. [Online]. Available: https://www.mathworks.com/products/matlab.html?s_tid=hp_products_matlab.

[3] P. K. Seidelmann, Explanatory Supplement to the Astronomical Almanac, Sausalito, CA: University Science Books, 1992.

[4] J. R. Wertz, Space Mission Analysis and Design, El Segundo, CA: Microcosm Press, 1999.

[5] H. D. Young, Sears and Zemansky's University Physics with Modern Physics, San Francisco, CA: Pearson Education, Inc, 2008.

[6] J. Wright, Space Sailing, Abingdon: Routledge, 1992.

[7] M. D. Griffin, Space Vehicle Design, Reston, VA: American Institute of Aeronautics and Astronautics, 2004.

[8] R. Furfaro, *Class Notes from SIE 552, 2/28/17,* Tucson, 2017.

[9] H. D. Curtis, Orbital Mechanics for Engineering Students, Oxford: Elsevier, Ltd, 2014.

[10] G. P. Sutton, Rocket Propulsion Elements, Hoboken, NJ: John Wiley & Sons, Inc, 2010.

[11] National Aeronautics and Space Administration, "Mars Reconnaissance Orbiter Mission Overview," National Aeronautics and Space Administration, 2006. [Online]. Available: https://mars.nasa.gov/mro/mission/overview/. [Accessed 1 August 2017].

[12] M. D. Johnston, "The Mars Reconnaissance Orbiter Mission," IEEE, Pasadena, CA, 2003.

[13] United Launch Alliance, "Atlas V Launch Services User's Guide," United Launch Alliance, Centennial, CO, 2010.

[14] United Launch Alliance, Delta IV Launch Services User's Guide, Centennial, CO: United Launch Alliance, 2013.

[15] J. Shan, *Interplanetary Trajectories,* Toronto: York University.