Nanosatellite Design with Design of Experiments, Optimization and Model Based Engineering

a project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Justin Ancheta

May 2018

approved by

Dr. Periklis Papadopoulos
Faculty Advisor

The Designated Project Advisor/Committee Approves the Project Titled

NANOSATELLITE DESIGN WITH DESIGN OF EXPERIMENTS,
OPTIMIZATION AND MODEL BASED ENGINEERING

by

Justin Ancheta

APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2018

Dr. Periklis Papadopoulos Department of Aerospace Engineering

ABSTRACT

NANOSATELLITE DESIGN WITH DESIGN OF EXPERIMENTS, OPTIMIZATION AND MODEL BASED ENGINEERING

by Justin Ancheta

Nano-satellite missions are an inexpensive tool used to perform scientific research functions and test new space technologies. From 2000 to 2016 there were 371 nanosatellite launches with 15.4% shown to complete their mission and another 22.6% which completed some aspect of their mission. The high failure rate of nano-satellites can be attributed to multiple factors such as system integration, non-space rated hardware, launch failure, and untested technology. By integrating design exploration, optimization and model-based system engineering techniques, some portion of nano-satellite risk may be mitigated.

This work presents a multi-disciplinary system exploration, design, and simulation approach for low-budget nano-satellite builders to get a better estimate of system performance before beginning the actual build. The use of design exploration minimizes the number of variables in optimization by finding which design variables have little options or impact on system performance and allows the design team to parameterize them. The system optimization process will give initial goals for the hardware mass, power and performance metrics which can then be simulated in a high-fidelity simulation to analyze mission and systems level requirements. This approach was implemented using a combination of open source tools and commercial programs. Open source packages such such as pyOpt and OpenMDAO were used to perform design space exploration and multi-objective optimization. MatLabs Simulink was used as the model based engineering program as it provides a format which would is readily available and familiar to students and engineering teams with little to no formal systems engineering background.

Design space exploration was performed by tools provided in the OpenMDAO package. Optimization was implemented by aggregating the constraints and objectives using the Kreisselmeier-Steinhauser method and using an unconstrained solver to minimize the envelope. The implementation of this was algorithm was provided by the pyOpt package.

The current level for the models for each system is currently in an early development stage which should be used with caution for actual system engineering. However this proposed approach can be easily expanded with higher-fidelity models, design equations, and parameterization based off of historical data which will result in an initial starting point for nano-satellite build teams who can then make design decisions based off of currently available technology, time and monetary constraints.

CONTENTS

LIST OF FIGURES

Figure

Nomenclature

^{0}k	Pointing knowledge of star tracker
ΔV	Change in velocity (impulse measurement)
λ	Transmitted signal wavelength (m)
λ_{max}	Maximum earth central angle (rad)
λ_{min}	Minimum earth central angle (rad)
μ	Standard Gravitational Parameter times the mass of the planet
ω	Argument of Periapsis
Ω	Longitude of the ascending node
$^{A}\omega^{B}$	Angular velocity of reference frame B in reference frame A
Φ	Gravitational Potential Function
ρ	Density, angular radius
θ	True anomaly
$\mathbf{a}$	Acceleration
A	Cross sectional area
a	Semi-major axis for elliptical orbit
BER	Bit error rate
C	Received power (W)
c	Speed of light in vacuum
C_D	Coefficient of drag
D	Data total (*bit*), or spacecraft residual dipole
e	Orbit eccentricity
E_b	Received energy per bit (W)
$EIRP$	Effective isotropic radiated power (W)
$\mathbf{F_i}$	Force of i

F	Fractional reduction in viewing time (Dimensionless between [0 and 1])
G	Gravitational Constant
G_r	Receiver gain (dB)
G_t	Transmitter gain (dB)
H_{max}	Maximum angular momentum capacity of wheel
i	Orbit inclination or incidence angle (context sensitive)
I_{sp}	Specific Impulse
$IFOV$	Instantaneous field of view (one pixel)
IPS	Instructions per second
J_n	Geopotential coefficient
k	Boltzmann's constant ($1.38064852 \times 10^{-23}$ m^2 kg $s^{-2} K^{-1}$)
L_a	Transmission path loss (dB)
L_l	Transmitter to antenna line loss (dB)
L_s	Space loss (dB)
M	Margin required to account for missed passes between [2,3]
M_e	Magnetic moment of earth (7.96e15 $Tesla \cdot m^3$
m_i	Mass of object i
$\dot{\mathbf{n}}$	Time derivative of vector $\mathbf{n}$
$\ddot{\mathbf{n}}$	Second time derivative of vector $\mathbf{n}$
N_o	Noise density (W)
P	Transmitter power (W)
P_{EOL}	Power at end of life
P_{SA}	Power of solar array
q	Reflectance factor
$\mathbf{r}_{ba}$	Vector distance from object a to b, same as ${}^a\mathbf{r}^b$

R	Data rate (bits/s)
R_E	Mean radius of the earth
R_r	Effective radius of isotropic transmitting aperture (m)
$SLOC$	Source line of code
T_{init}	Time required to initiate communications pass (s), suggested to be two minutes
T_{max}	Maximum time in view (s)
T_i	Torque
T_s	System noise temperature (K)
V_g	Ground velocity
W_f	Power flux density (W/m^2)
X_i	Power system efficiency

Chapter 1

INTRODUCTION

The following chapter will focus on the motivation for developing and extending current modular and open architecture systems and their supporting technologies and methods. The models for a space plug and play system are focused on standardizing a set of self-describing frameworks and interfaces for each subsystem such that system integration requires very little input from the system integrator. This project is a focus on mission development using a generalized model, and incorporating a modular redundancy system as a new system feature for plug-and-play satellites.

1.1 Motivation

Plug and play (PNP) technology and scalable modular/open architectures expand the operational capabilities for satellites and provide a means of risk mitigation for missions. From an operational standpoint a plug and play system will minimize the time necessary to integrate a system which will directly lead to rapid launch capabilities. In addition the ability to swap parts in orbit with on-board reserves or through a servicing platform will help reduce the chance of complete loss of a system. An open-architecture (OA) model will provide a framework based on plug and play concepts in order provide modular redundancy, rapid deployment of satellites, in flight servicing capability, as well as provide a new method for risk management.

1.1.1 Problem at hand

The two issues which encumbers rapid satellite development and launch are system integration errors and the risk mitigation techniques. The risk mitigation techniques are required to ensure a functional satellite will be delivered to the customer and operate in orbit, as such removing them from the development cycle is not possible. In order to reduce time in the development phase the issues encountered by system integration must be relieved. Adopting an OA/PNP system can minimize the time spent on both of these issues by providing a standard set of modular redundancy protocols, inter-system communication frameworks, mechanical subsystem module connection interfaces, and a common flight software package. Significant work has been done in developing system architecture standards and flight software framework already. The focus for this study is to investigate modular redundancy advancements for mission development. One such advancement is the incorporation of radiation tolerance methods as a new feature for the OA/PNP system models. The use of radiation tolerant methods combined with commercial off the shelf (COTS) hardware will minimize the cost of nano-satellites and lead times, this is significant due to the fact that many nano-satellites are developed by universities and budget constrained research teams. Minimizing time and cost will provide greater access for universities and research teams access to space exploration and research projects.

1.1.2 Why investigate problem

With the expansion of commercial and government endeavors in space, there will be an evolving role to provide servicing requests to prevent the loss of satellite assets. While possible the servicing of satellites are still costly and designing a system which can recover in the event of a subsystem failure should be considered first. Conventional life cycles for satellites typically occur in the time frame of up to 15 years from concept to launch and last for roughly about the same time in orbit[1]. For nano-satellites there is an even more dangerous gap over a year of development and less than a year for flight [2]. Often times these systems are custom made to order in cleanroom environments and vigorously tested at component, subsystem, and system levels. While the manufacturing process and rigorous testing is necessary to provide functional satellites, it prevents quick turnaround times for commercial, scientific and defense purposes. The most common commercial satellite bus is the SSL-1300 platform which fist flew in 1989 and averages about six launches per year [3]. Despite the flight heritage and expertise that was gained from years of working with the spacecraft bus it still difficult to interface the mission payloads to the existing spacecraft bus. A modular OA/PNP system incorporating n-modular redundancy will provide one method for minimizing time in the design-build-fly cycles, while simultaneously providing easier servicing of satellites. In the past servicing of satellites was provided by the space shuttle program as part of its normal operation profile [4]. With the ending of the current space shuttle program, there is very little that can be done to recover and repair satellites that are failing in orbit. With the use of modular PNP systems in flight repairs of satellites which have been damaged during operation can be repaired more easily and possibly autonomously via nano-satellites, removing the need to have a crew intercept and provide manual integration and configuration to a satellite. Another advantage of OA/PNP system is to provide rapid launch capabilities. The U.S. Department of Defense (DoD) created the Operationally Response Space (OSR) unit to focus on rapid deployment space systems to support DoD operations. In conjunction with the Air Force Research Laboratory (AFRL) the OSR developed the Space Plug and

Play Architecture (SPA) in pursuit of a six day design-build-fly cycle [5]. Over the next ten years NASA, OSR, and AFRL worked on the improving the avionics standard and developed a variety of demonstrations such as the Modular Space Vehicle, TacSat3, adaptive wiring harnesses and components of TechEd Sat proving the viability of the OA/PNP system [5]. While many demonstrations missions have proven the viability of the OA/PNP concept there is still significant advancement necessary to enable the full benefits of the architecture.

1.2 Literature Review

The following section will cover some of the existing work related to spacecraft OA/PNP technologies, advances in technologies and methods, tolerance methods, as well as some companies which have developed intellectual property which support the OA/PNP model. Hardware adaptability is significant for this n-modular redundancy as such the use of FPGAs will be of interest as well.

1.2.1 FPGAs

Field programmable gate arrays (FPGAs) are the cornerstone of true system redundancy due to their ability to change functionality during normal and recovery operation modes. The ability for FPGAs to process large data streams in parallel also help alleviate the down-link budget. By providing more on-board processing for large data streams such as video and hyper-spectral imaging, the total data stream to the ground station is reduced [6]. While there are radiation hardened FPGAs they typically do not have the same performance as their terrestrial counterparts and have a higher end cost for the system developers [6]. FPGAs use a technique called scrubbing to effectively rewrite the FPGA code in whole or partially as a way to deal with environmental caused error. While effective in countering single event upsets (SEU), it cannot prevent latch ups or more physically damaging faults. Work done by Hane and Buerkle have shown that the use of radiation sensors in conjunction with multiple FPGAs can improve robustness for SEUs by up to ten times even in severe solar flares[7]. However there have been other studies which show that after the first spare, there are significant diminishing returns. The optimal situation for low earth orbits is enabling TMR with scrubbing using an additional spare which has a short activation time than the repair time of the original upset FPGA [8]. Other advancements in space based FPGA programming can be found in dynamic fault tolerance. FPGAs utilizing triple modular redundancy (TMR) can see system resources increase by over two times, this expands out significantly as you increase the number of modular redundancy systems. Jacobs et. al have found that implementing a reconfigurable fault tolerant framework provides over double the performance in low radiation environments over traditional TMR systems.

1.2.2 N-Modular Redundancy and Supporting Technologies

N-modular redundancy (NMR) is a significant topic for long-life missions as well as nano-satellite COTS bus systems. For deep space missions the ability for a spacecraft to make simply arrive to its mission location in an operating form is as challenge in itself. As a result many of these missions use custom spacecraft systems which are focused on robust operation instead of performance [9]. However robust systems which utilize NMR result in high power draw. On the other hand many nano-satellites utilize COTS systems which may not have significant fault tolerance by creating a NMR interface between the hardware and the rest of the system survival can be increased.

NMR is a methodology which provides fault masking as long as half plus one modules are operating correctly. Due to the module requirement however this is a resource intensive method which imposes significant power constraints for both FPGA and multi-core processing units. For satellites which implement higher than TMR tolerance schemes, power draw becomes a significant issue for thermal and computational resource management. One method of countering this is an online/offline energy management system. In this process half of the tasks are computed and compared by the voters. If all results are within tolerance of each other, then the second phase does not occur. If there is a distinction between various outputs, then a second cycle computers the other half of the tasked process and compares them [10]. Combining this method with the dynamic tolerance scheme can provide a significant increase in system resource reserves and minimize power draw. This is a significant advantage for deep space missions when extra power is required for communication and system life support.

Another fault recognition framework was developed which also takes into account mission objectives and policy requirements [11]. In this approach an adaptive decision process was created which balances bus safety and mission objectives as potential for handling detected faults and determining the best course of action. This process provides the option for spacecraft to delay in flight reconfiguration if there is an achievable mission objective which will not affect the long term health of the spacecraft, promoting autonomous decision making which is necessary for periods when the satellite is out of communication with ground networks.

Traditionally functions for a spacecraft have been passed down to specific subsystems and rarely have the ability to overlap due to size, weight and power requirements. A vector modeling approach of modularity mapping has been proposed which allows the mission designers to implement a satellite which provides little overlap of system function, or multiple levels of redundancy through the use of modular systems which can contain various functions [2]. In either case, the ability for the system as a whole to recover from failure remains a necessity. While micro-controllers and processors have very little ability to recover without a system restart, FPGAs have the ability to reconfigure in flight. This unique feature allows the use of subsystems to recover in the event of SEUs. For this reason FPGAs are used for core system functions, health checks and with the addition of TMR and scrubbing are used for full system recovery operating modes. Another event in the case of subsystem controller failure is the use of a secondary FPGA taking over the unusable FPGA. This can be done by use of a linear bus which uses a data bus structure which allows all distributed FPGAs communicate with all other hardware when necessary [12]. Previous attempts at this has been seen with the development of the X2000 at NASAs Jet Propulsion Lab. In the work done on the X2000 multiple levels of hardware and software fault protection implications were developed to integrate COTS into use for deep space missions. The X2000 team found that by implementing multiple levels of redundancy they were able to get comparable fault protection as specifically design hardware [9].

Another supporting technology was developed in conjunction with the SPA standard. Murray et. al developed an adaptive wiring manifold which replaced the custom wiring harness used for connecting satellites [13]. The design of these wiring panels provided a scalable interface which provided a way to connect arbitrary modules as well as provide a switchable pathway between them. This concept allows the ability to switch between failing hardware and potential reserves in flight. The second is the ability to add on additional hardware in flight. Combined with a self-describing framework this can provide true plug-and-play functionality.

1.2.3 Standardization

There have been two significant advances for the OA/PNP model since the early 2000s. The first is the introduction of the SPA developed by the AFRL and OSR, and the release of the NASA core Flight Executive (cFE). Both of these provide frameworks for the core mission software, firmware, and the hardware interfaces. The SPA standard has further developed into the modular open network architecture (MONARCH) which provided a scaled framework for large satellites over 1000 kg [14]. While many communication protocols have been standardized, there is still some room left for the engineers to design the system for minimizing negative performance interactions. The purpose of MONARCH was to compile the engineering efforts into a best-practice, and further specify the interfacing protocols. This removed work which would be lost on a system to system basis. It also further enhanced SPAs self-description and specification of data.

The second framework is the Goddard cFE. NASA Goddard Space Flight Center developed the framework as a way to reduce their internal development timelines and reduce mission cost. The software framework is provides a majority of the necessary functions for the command and data handling (C&DH) subsystem and interfaces with various real time operating systems [15]. The cFE package is a scalable flight qualified software environment which is used for flight as well as ground testing and mission simulation. The release to the general public for this software reduces the amount of the custom flight software which is required for the system designers, leading to reduced development cycle time [15].

1.2.4 Companies

Several companies have also begun implementing modular systems which are semi-open modules which can interface more readily with other systems. One such system is the Space Information Laboratories (SIL) Intelli-Avionics system [16]. This system is designed around minimizing the number system black boxes as a way to minimize overall system cost. The integration of similar systems into full system on modules reduces the complexity of dealing with multiple different hardware frameworks as well as having a single point of contact for a set of sub functions.

Similarly some companies are providing full turnkey platforms such as the Clyde Space CubeSat platforms [17]. By providing the interface requirements a customer can design and build a singular payload and integrate it into a fulling functional bus. Although it provides a convenient package it is not a one-size fits all open framework which can be modified by the customer.

WIP Some material is protected by NDA, working on what can still be shared.

1.3 Proposal

The purpose of this project is to develop a baseline nanosatellite design using design space exploration, optimization methods, and analysis using models in a simulation environment.

1.4 Methodologies

The general method for this project is as follows:

- Choose a set of nano-satellite missions and identify a bound of system and subsystem requirements based on common mission parameters.

- Using analytical design equations and estimations, coupled with regression of current hardware develop a sizing model for each core subsystem

- Perform a factorial analysis on the tradespace of each subsystem to define the key performance drivers.

- Optimize and simulation the design to perform a first order analysis of the feasibility of design capabilities.

Chapter 2

SPACE MISSION BACKGROUND THEORY

This chapter will overview the core concepts of the theory necessary to understand, simulate, and implement a general model for a nano-satellite. By generating a nano-satellite model, it is possible to develop a computational test-bench for emerging technologies and methods for use in space missions. This section will briefly overview the critical components necessary to generate the nano-satellite mission model, including the following modules:

(1) Orbit Mechanics and Numerical Techniques

(2) Space Environment Forces and Effects

(3) Subsystem Functionality

(4) Design Space Exploration

2.1 Orbital Mechanics and Numerical Techniques

In order to determine the environmental influences which would act as inputs to the general nano-satellite model, it is first necessary to determine the position in space that spacecraft is in. In order to determine the position and attitude, a simulation module will need to be constructed which provides accurate time, space, and orientation parameters. For conciseness only the general mechanics equations and numerical techniques will be dealt with in this section. Specific maneuver details and methods used for designing and analyzing missions are described in Appendix A to maintain focus on the governing mechanics for orbital mechanics.

2.1.1 Orbital Mechanics

There are three fundamental laws which describe the motion of planets around the Sun. The first two laws were discovered by Johannes Kepler 1609 and the third by Kepler again in 1619. They are as follows [18]:

(1) The orbit of a planet is an ellipse with the Sun as its focus.

(2) The line joining the planet to the sun sweeps out equal areas in equal time.

(3) The square of the period of a planet is proportional to the cube of its mean distance from the sun.

To describe the orbit shape and orientation the use of six Keplerian elements is needed. Keplerian elements are typically given to describe the shape of the orbiting body and include the eccentricity, semi-major axis, inclination, longitude of ascending node, argument of periapsis, and true anomaly.

- Eccentricity e is derived from the mathematical property of conic sections. Eccentricities from zero to less than one are ellipses. An eccentricity of one indicates a parabola, while anything higher than one is a hyperbolic conic section. The shape of the conic section defines the shape of the orbit.

- The semi major axis a is the length between the apoapsis and periapsis of the ellipse.

- Inclination i is the experienced tilt of the orbit plane about the ascending node with respect to the main body plane of reference.

- Longitude of the ascending node Ω orients the ascending node of the ellipse to where the orbit passes upwards through the reference plane. It is referenced against the vernal point of the main body.

- The argument of periapsis ω aligns the periapsis with respect to the ascending node of the orbit.

- True anomaly θ shows the angular position from periapsis of the orbiting body on the ellipse.

The elements are shown in figure 2.1. These elements can be mapped to the Cartesian grid and from the Cartesian grid to the Keplerian elements as shown in appendix A.

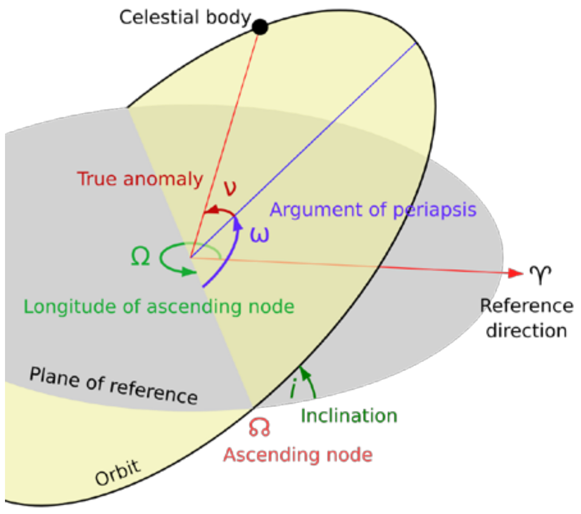


Figure 2.1: Keplerian Elements from Lasunncty / Wikimedia Commons CC-BY-SA-3.0

Using Newton's second law we can find the acceleration of the satellite in the reference inertial frame. There are a variety of forces which act on the satellite at any given time, however the focus of the gravitational forces will be discussed here. Perturbations such as thrust burns, atmospheric drag, solar pressure, and non-idealized gravity will be discussed in section 2.2. Assuming the satellite as a point mass we can separate the general equation of the satellite into the following equation.

$$\mathbf{F_{total}} = m_{sc}\mathbf{a}_{sc} \tag{2.1}$$

Due to the large impact of the gravitational force on the orbit compared to all others, the total force will be separated into $\mathbf{F_{grav}} + \mathbf{F_{other}}$. Equation 2.1 can also be rewritten more generally for any body in the reference frame. The general form of the equation is useful as it improves the accuracy of your spacecraft position due to taking into account changing mass.

$$\ddot{\mathbf{r}}_\mathbf{i} = \frac{\mathbf{F_{tot}}}{m_i} - \dot{\mathbf{r}}_\mathbf{i}\frac{\dot{m}_i}{m_i} \tag{2.2}$$

To find the gravitational force of all bodies on the spacecraft the following equation is used.

$$\mathbf{F_g} = -Gm_i \sum_{j=1,\ j\neq i}^{n} \frac{m_j}{r_{ji}^3}(\mathbf{r}_{ji}) \tag{2.3}$$

By neglecting non-gravitational forces and combining equations 2.2 and 2.3, a general orbit path can be found. As the mission design moves further into completion, higher fidelity can be found by adding in the necessary perturbations and expanding the number of bodies in the reference system. For the initial iteration of the orbit propagation multiple assumptions were made, these include:

- Only two bodies are involved in the problem.
- There are no significant forces on the spacecraft outside of gravity.
- All bodies are rigid and have a point mass at the center of gravity of the object.
- All burns (ΔV) are instantaneous and do not significantly change the mass of the spacecraft.

Further iterations will improve the fidelity of the model to include other perturbations such as multiple bodies, propulsive burns, drag, solar pressure, non-spherical gravity, and other significant environmental factors.

2.1.2 Reference Frames

Due to the complexity of the spacecraft dynamics it is important to define which reference frame we are working in and how they relate to one another. For Earth based orbits, the starting point for all equations of motion is the Earth-Centered Inertial (ECI) frame. This is the reference non rotating frame from which all other frames can be further found for our purposes.

The ECI frame is a Cartesian grid space which is defined by using $\mathbf{e\hat{c}_x}$, $\mathbf{e\hat{c}_y}$, $\mathbf{e\hat{c}_z}$. The $\mathbf{e\hat{c}_x}$ direction is bound by the intersection nodal vector of Earths equator and the orbital plane of Earth around the sun (the ecliptic) and is positive in the direction of the Vernal Equinox. Due to the precession of the Earth, the actual direction of the Vernal Equinox changes very slightly from year to year.The constant changing of this nodal vector creates a problem when standardizing the reference planes, as such the ECI frame is referenced against a specific epoch such as J2000 which was defined by the mean equator and equinox at the JD 2451548.0 $\mathbf{e\hat{c}_z}$ is the vector which moves through the North Pole, while $\mathbf{e\hat{c}_y}$ is defined simply by crossing $\mathbf{e\hat{c}_z}$ and $\mathbf{e\hat{c}_x}$ [19]. This frame is not entirely inertial however, but is non-rotating with respect to the stars, and has negligible amounts of acceleration allowing use as an inertial frame [20].

The Perifocal frame is effectively another inertial frame for Earth based satellites. Similar to the ECI frame it is a Cartesian grid space utilizing the vectors $\mathbf{\hat{P}}$, $\mathbf{\hat{W}}$, $\mathbf{\hat{Q}}$. In this frame $\mathbf{\hat{P}}$ is located in the direction of the periapsis of the orbit from the center of the orbiting body. The $\mathbf{\hat{W}}$ is perpendicular to the plane of orbit, which is the direction of the angular momentum vector $\hat{\mathbf{h}}$. $\hat{\mathbf{Q}}$ lies on the orbital plane with $\hat{\mathbf{P}}$ and completes the orthogonal basis for the reference frame. This frame allows use of the simplified orbit mechanics models such as the two body problem, as well as simplifying the transform from the Keplerian elements to the Cartesian grid space used for equations of motion.

There are also a variety of non-Newtonian reference frames which are used in order to help determine the attitude of the spacecraft. The first is the orbital reference frame which is found as a set of vectors originating from the point of the spacecraft. The $\hat{\mathbf{O}}_\mathbf{x}$ axis is in the direction of the spacecrafts velocity vector while $\hat{\mathbf{O}}_\mathbf{z}$ vector is in the direction of the central body. A third vector, $\hat{\mathbf{O}}_\mathbf{y}$ is the orthogonal cross product between the $\hat{\mathbf{O}}_\mathbf{z}$ and $\hat{\mathbf{O}}_\mathbf{x}$ vectors. The spacecraft roll, pitch, and yaw angles can then be found by the rotation of the spacecraft reference frame about the orbital reference frame. The spacecrafts reference frame is based on the geometry of the spacecraft and can be arbitrarily decided. Further reference frames for any deployed satellite structures can be found by referencing from the spacecraft main body frame. For the purpose of this project all body frames will be referenced as $\mathbf{SC}_\mathrm{x}$, $\mathbf{SC}_\mathrm{y}$, $\mathbf{SC}_\mathrm{z}$ with appropriate reference to the specific deployed structure.

Earth also has two more useful reference frames which are the Earth Centered Earth Fixed (ECEF) and East, North, Up (ENU). ECEF has X,Y,Z axis which are set at the center of mass of the earth and point to the 0° latitude, 0° longitude, and true north respectively. The ENU frames are based at the surface of the Earth at a local ground station and have the X,Y,Z axis defined by East, North, and directly upward respectively. The combination of these inertial and non-Newtonian reference frames allows for relative ease for determining spacecraft attitude, ground station tracking, and adjusting for time delays based off the rotation of the Earth.

2.1.3 Numerical Integration Techniques

The equations of the orbiting bodies are governed by gravitational attraction with minor perturbations from external forces and the space environment. Due to their transient nature, it is necessary to numerically solve them. There are many choices for methods when it comes to numerical integration, only three of the common families will be covered.

Runge-Kutta

A commonly used technique is the Runge-Kutta method due to a combination of its ease of use and adaptive step sizing [18]. Generally speaking the algorithm is as follows [19]:

$$\text{For } \dot{y} = f(t, y) \text{ with initial condition } y(t_0) = y_0 \tag{2.4}$$

A step size (h) greater than zero is then chosen and implemented in the following process.

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{2.5}$$

$$k_s = f(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1})) \tag{2.6}$$

For a general embedded Runge-Kutta method, the Butcher Tableau contains the coefficients for the above equation and is of the form shown in table 2.1.

In these tables the bottom b^* row is the higher order coefficients used for the error estimator. In adaptive methods the error is determined by the equation 2.7. This is used to change the step size based on the user defined tolerance.

$$e_{n+1} = y_{n+1} - y^*_{n+1} = h \sum_{i=1}^{s} (b_i - b^*_i) k_i \tag{2.7}$$

Adams-Bashforth

Table 2.1: Butcher Tableau general form

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
$.$	$.$		$\ddots$		
c_s	a_{s1}	a_{s2}	$\cdots$	$a_{s,s-1}$	
	b_1	b_2	$\cdots$	b_{s-1}	b_s
	b_1^*	b_2^*	$\cdots$	b_{s-1}^*	b_s^*

It can be seen that the Runge-Kutta method is a single-step process which evaluates the function f multiple times per step in the solution, this typically leads to a longer computational time as it can not take advantage of previous steps history. Other methods for orbit propagation can be found in families of numerical techniques which use predictor-corrector schemes. These methods attempt to find a solution using a multiple previous steps to approximate a time step solution and then correct it using a corrector. By requiring multiple historical steps, it is not always convenient to use the predictor-corrector schemes, one solution is to propagate the necessary steps using a one-step method such as Runge-Kutta, and then move forward in the solution with the multi-step method. One of the best known predictor-corrector methods is the Adams-Moulton method which uses the Adams-Bashforth predictor and the Moulton corrector formula. Equations 2.8, 2.9, and 2.10 show the predictor, corrector and estimated error formulas for stepping from time n to n+1. Note that the term $h^5 \frac{d^5x(\xi)}{dt^5}$ is the truncation error term and is generally omitted due to not knowing the truncation error [18].

$$x^P_{n+1} = x_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) + \frac{251}{720}h^5\frac{d^5x(\xi)}{dt^5} \tag{2.8}$$

$$x^C_{n+1} = x_n + \frac{h}{24}(9f(t_{t+1}, x^P_{n+1}) + 19f_n - 5f_n - 1 + f_{n-2}) + \frac{19}{720}h^5\frac{d^5x(\xi)}{dt^5} \tag{2.9}$$

$$\frac{19}{270}\left|x^C_{n+1} - x^P_{n+1}\right| \tag{2.10}$$

Gauss Jackson

The Gauss-Jackson method is a multistep predictor corrector which has been used since the 1960s to estimate the position of tracked objects in space. The drawback of this method is that for an estimation of the N^{th} order, $N+1$ points must be known before hand. Like the Adams-Bashforth method this can be initiatied using a small time step Runge-Kutta varation and then propagated using this method [21, 22, 23]. It has been shown that the Gauss Jackson method has a lower global error than traditional Runge-Kutta 4 for near circular orbits, however RK4 has a lower global error than highly elliptical orbits [24]. Implementation of the Gauss-Jackson method can be found in [22].

2.2 Space Environment and Perturbing Forces

Many assumptions made above will not work as a mission is planned out in detail. In reality, burns to change and maintain orbits take time and expel mass from the spacecraft. Atmosphere imparts drag when in low earth orbit, the sun is a constant source of solar pressure which affects trajectory, and the main body of orbit is usually not a perfect sphere. General methods for including these perturbations in the orbit model will be found in the subsequent subsections and will be used in conjunction with equation 2.2 to improve the fidelity of space mission modeling.

2.2.1 Finite Burns

Most satellites which require long term orbit maintenance use some form of chemical propulsion or electric propulsion. In these cases the satellites propulsion system expels some amount of propellant in order to adjust its course. The thrust generated by the propulsion systems is assumed to be on axis with the direction of travel and can be modeled adding the following equations to equation 2.2[25]:

$$\mathbf{F}_{\mathbf{Thrust}} = T\frac{{}^{R_0}\mathbf{V}^{SC}}{|{}^{R_0}\mathbf{V}^{SC}|} \tag{2.11}$$

$$\dot{m} = \frac{-T}{I_{sp}g_0} \tag{2.12}$$

2.2.2 Atmospheric Drag

While in low earth orbits, the atmosphere still imparts some forces on the spacecraft. While very small, it can add up over time depending on the altitude of the spacecraft. The general method to include this force inside of perturbations is to use use the satellites coefficient of drag (C_D), cross sectional area perpendicular to the velocity vector, and density from the chosen atmospheric model in order to calculate the drag. NRLMSISE-00 is an empirical model of earths atmosphere which is an extension of MSISE-90 which expanded the previous data set to include satellite drag [26]. Other models exist for the atmospheric density, such as: 1988 MET model, COSPAR International Reference Atmosphere 1986 (CIRA), and U.S. Standard Atmosphere. For the simulations purpose, the NRLMSISE-00 model was chosen in order due to including empirical data from satellites. Similar to thrust, drag is along the direction axis of travel, allowing us to formulate the equation for aerodynamic drag the same way as thrust. The following equations can be added to the perturbation forces in order to add in atmospheric drag. The model used to estimate density for our cases can be found in Appendix A.

$$\mathbf{F}_{\mathbf{D}} = -\frac{1}{2}\rho|{}^{ATM}\mathbf{V}^{SC}|^2 C_D A\frac{{}^{ATM}\mathbf{V}^{SC}}{|{}^{ATM}\mathbf{V}^{SC}|} \tag{2.13}$$

$${}^{ATM}\mathbf{V}^{SC} = {}^{EC}\mathbf{V}^{SC} - {}^{EC}\mathbf{V}^{ATM} = {}^{EC}\mathbf{V}^{SC} - ({}^{EC}\omega^{Earth} \times {}^{EC}\mathbf{r}^{Q}) \tag{2.14}$$

2.2.3 Solar Radiation Pressure

Solar pressure is a force which is negligible compared to other perturbation forces under 800km altitude [1]. The following formula can be used to find the force caused by solar pressure. A is the cross sectional area exposed to the sun,r is an empirical reflection factor. An r value of 0 completely absorbs, while a value of 1 represents total reflection.

$$\mathbf{F_{SP}} = -4.5 \times 10^{-6}(1 + r)A \tag{2.15}$$

2.2.4 Oblate Earth

The original assumption under the first implementation of the gravitational force was a center mass point. However because the mass of the central body is usually non uniform, adjustments must be made. The well defined gravitational potential function, Φ, uses geopotential coefficients in order to modify the point mass gravitational force on the spacecraft. The general potential function is given as:

$$\Phi = \frac{\mu}{r}\left[1 - \sum_{n=2}^{\infty} J_n \left(\frac{R_E}{r}\right)^n P_n(\sin(\psi))\right] \tag{2.16}$$

Where J_n is the geopotential coefficient, ψ is the geodetic latitude, P_n is the Legendre polynomials [1, 19]. In the Cartesian system this can be re-written using equation 2.17 [27].

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \frac{3J_2\mu R_e^2}{2R_s^7} \cdot \begin{bmatrix} 5Z^2 - X(X^2 + Y^2 + Z^2) \\ 5Z^2 - Y(X^2 + Y^2 + Z^2) \\ 5Z^2 - 3Z(X^2 + Y^2 + Z^2) \end{bmatrix} \tag{2.17}$$

2.2.5 Magnetic Sphere and Ionizing Radiation

The geomagnetic field for Earth can be calculated using the world magnetic model (WMM) which is distributed by NOAAs National Centers for Environmental Information. Updated coefficients are updated every five years, and can be assumed to be linearly time varying across the coefficients five year lifespan. The magnetic model can be used to assist the attitude determination system to determine the current pointing of the spacecraft. Due to time limitations this will not be implemented in this project, however suitable sources for this model can be found from NOAAs NCEI website. By knowing the satellites lattitude, longitude, altitude and date the WMM software can predict the components of the magnetic field [28].

CREME96 is a suite of programs which can be used to model the ionizing radiation environment of near Earth orbits. Similar to the magnetic sphere, the inclusion of this model will not be implemented for the initial simulation due to the time constraints. In future work it will be used to model the chance of single event upsets to create a realistic SEU test for redundancy models.

Chapter 3

SUBSYSTEM BASELINE DESIGN

This section will cover the principle design equations that will be used in the design exploration, optimization and models simulation. Each section has a set of design methods which provide general sizing and performance rates, a set of possible objectives to optimize around, as well as a variety of system and mission variables which influence the output of the design equation sets. For the example mission we have a nanosatellite at some altitude and inclination in a circular orbit. The objective is to provide short wave infrared imagery for analysis of identifying geological material.

3.1 Attitude Determination and Control System

For the attitude determination and control system (ADCS) model to work the initial state, mass moment of inertia, inertia, external torques, and desired pointing angle must be known. For each time iteration through the ADCS model will determine the power required to point to the correct position, and produce any updated variables that it has changed. For the purpose of design exploration and sizing a set of equations will be used to determine the estimated torques on a nanosatellite and a regression analysis of ADCS hardware will be performed in order to estimate the size, weight, and power draw for this subsystem. Due to time limitations only the system sizing and optimization will be performed for this subsystem. The system model in the simulation will only support power draw as a way to estimate battery state of charge over the course of the satellites life. Future work will be needed to incorporate multiple control modes and the required six degree of freedom model to simulate realistic satellite behavior. That is outside the scope of this project.

3.1.1 Estimated Torques

Estimating the total worst-case disturbance torques allow the system designer to get basic information on the size and complexity of the ADCS. In this case there are four main components of the torques: gravity, solar, magnetic and aerodynamic. The following equations allow the estimated toques (in $N \cdot m$) to be calculated. The angle, θ, is the maximum deviation of the z-axis from the local vertical in radians.The angle, i, is the incidence angle of the sun on the spacecraft. The moments of inertia are in $kg \cdot m^2$. All length units in this subsection are meters. The reflectance factor, q, lies between 0 and 1 and can be estimated as 0.6 [1]

$$T_{gravity} = \frac{3\mu}{2R^3}|Iz - Iy|sin(2\theta) \tag{3.1}$$

$$T_{solar} = \frac{W_f}{c}A_s(1 + q)cos(i)\ (c_{pressure} - c_{gravity}) \tag{3.2}$$

$$T_{mag} = \frac{2DM_e}{R^3} \tag{3.3}$$

Note that D is the residual dipole (in $A \cdot m^2$) of the spacecraft, and M_e is the magnetic moment of earth ($7.96 * 10^{15}\ tesla \cdot m^3$).

$$T_{aero} = \frac{1}{2}\rho V^2 A_s C_d\ (c_{pressure} - c_{gravity}) \tag{3.4}$$

3.1.2 Regression Analysis of ADCS Systems

The output report of the ADCS code has been included at the end of this report. In it the regression model figures to analyze data can be seen, as well as R^2

values and residuals plotted against fitted values. A quick summary of the results includes the following findings:

- Mass of reaction wheels is related to total momentum capacity, power is related to wheels maximum torque.
- For wheels there is little correlation between the Z dimension and either max torque or momentum capacity due to the range of models selected.
- The X and Y dimensions of reaction wheels are related to the momentum capacity of the wheel. This is due to increasing the inertia of the wheels directly increases the total momentum capacity.
- Mass and power of magnetic torque rods average to a constant. The physical dimensions are correlated to the rods dipole however.
- IMUs and Magnetometers have a wide variety of power, weight and size. More models should be gathered before a reasonable estimate can be made.

3.1.3 Physical Parameter Estimation Models

H_{max} is given as maximum momentum capacity in $N \cdot m \cdot s$. τ_{max} is the maximum torque of the wheel in $N \cdot m$

$$M_{Wheel}(kg) = 1.666 \cdot H_{max} + 0.1216 \tag{3.5}$$

$$P_{Wheel}(W) = 0.466 \cdot H_{max} + 0.5106 \tag{3.6}$$

$$X_{Wheel}(mm) = 20.55 \cdot ln(H_{max}) + 120.4 \tag{3.7}$$

$$Y_{Wheel}(mm) = 20.21 \cdot ln(H_{max}) + 118.0 \tag{3.8}$$

$$Z_{Wheel}(mm) = 23.61 \cdot ln(H_{max}) + 100.2 \tag{3.9}$$

Dipole is given as $A \cdot m^2$

$$M_{Mgtqr}(kg) = 0.001 \cdot Dipole + 0.3457 \tag{3.10}$$

$$P_{Mgtqr}(W) = 0.0502 \cdot Dipole + 0.399 \tag{3.11}$$

$$X_{Mgtqr}(mm) = 1.087 \cdot Dipole + 17.65 \tag{3.12}$$

$$Y_{Mgtqr}(mm) = 0.8574 \cdot Dipole + 14.47 \tag{3.13}$$

$$Z_{Mgtqr}(mm) = 29.18 \cdot Dipole + 87.24 \tag{3.14}$$

The pointing knowledge, $\circ_k$, is in degrees.

$$M_{ST}(kg) = -7296 \cdot \circ_k^2 + 31.79 \cdot \circ_k + 2.735 \tag{3.15}$$

$$P_{ST}(W) = -4592e4 \cdot \circ_k^2 + 750 \cdot \circ_k + 5 \tag{3.16}$$

$$X_{ST}(mm) = -6071 \cdot \circ_k + 196.3 \tag{3.17}$$

$$Y_{ST}(mm) = -6500 \cdot \circ_k + 200.3 \tag{3.18}$$

$$Z_{ST}(mm) = -1.536e4 \cdot \circ_k + 387 \tag{3.19}$$

No reasonable estimation models could be found with the IMUs or magnetometers. Because of this the overall margin of the ACDS should be increased in the baseline to allow for the inclusion of these items if necessary.

3.2 Command and Data Handling

The command and data handling (CDH) subsystem is the most critical component of nanosatellites as it controls the timing for functions for all on-board operations as well as communication between the end user and the satellite in orbit. This subsystem can not be adequately sized using parametric equations due the complexity of processing components, board formats, and rapid miniaturization of hardware for space. To provide an adequate estimate of size, power and weight for this subsystem a regression analysis was performed. The selection process shown in Space Mission Design and Analysis provide a method to estimate the complexity of the system which is a possible candidate for the sizing estimate variable. The adapted table is shown in table 3.1.

3.2.1 OBC Requirements Sizing

The on-board computer (OBC) is a complex system which lies outside the scope of this project. As such a down select process is required in order to meet the OBC requirements. The performance requirements are based on the software requirements, which are outlined in this section. It is assumed that NASAs core flight software suit will be the base of the nanosatellite software and that additional modules will double the total logical lines of code. This results in a total SLOC count of 78.6K lines [29], with the language assumption of C. The underlying operating system, FreeRTOS is around roughly 10k SLOC. Using the tables from SMAD we can safely estimate the instructions per second as

$$IPS = 28 \cdot SLOC \tag{3.20}$$

3.3 Communications

The communications subsystem is the only interface that the satellite has with the ground facility and is used for tracking, telemetry, command, mission data transfer and ranging. The design of the communications architecture is based on the combination of satellite, relays, and ground station resources which are available in the mission design. The current architecture chosen is a single ground station located in San Jose, with a single low altitude nanosatellite placed in an orbit which passes overhead. This was chosen due to the simplicity of the architecture and to minimize the overall cost of the proposed design. All of the design equations for the communications subsystem and the communications architecture are from the Space Mission Analysis and Design book unless otherwise noted [1].

3.3.1 Data Transmitted

In order to ensure that as much data gathered by a scientific satellite is capable of being transferred back to Earth, it is important to ensure that the communications links have a sufficient received energy-per-bit to noise density ratio. This ratio is typically design around for a value between five and ten for most missions [1]. This ratio is used to predict the bit error rate depending on modulation technique and can increase (or decrease) the number of passes required for a complete data download. As many nanosatellites do not have the ability to perform orbit maintenance it is important to ensure as much data is sent back to Earth in each pass as the satellite will eventually burn up in the planets atmosphere, resulting in permanent loss of any data not transmitted. The total data downloaded can be estimated from equation 3.21.

$$D = \frac{R(FT_{max} - T_{init})}{M} \tag{3.21}$$

Where the fractional reduction in viewing time, F, is defined by the orbit geometry in equation 3.22

$$F \overset{\text{def}}{=} \frac{1}{\lambda_{max}} acos\left(\frac{cos(\lambda_{max})}{cos(\lambda_{min})}\right) \tag{3.22}$$

3.3.2 Received Energy per Bit to Noise Density

$$\frac{E_b}{N_o} = \frac{P\,L_l G_t L_s L_a G_r}{k T_s R} \tag{3.23}$$

At first glance the ratio terms are not intuitive of the underlying design variables so a derivation and example calculation of this is performance parameter is provided in the appendices. This is used to determine which modulation scheme to use to satisfy a bit error rate requirement.

3.3.3 Communication Size

The pointing loss, L_θ, in dB can be calculated as a function of pointing error and beam width.

$$L_\theta = -12 \left(\frac{err}{\theta}\right)^2 \tag{3.24}$$

Where θ is the half power beam width. The estimated size of the helix antenna can then be found using

$$\theta = \frac{52}{\sqrt{(\pi D)^2 L/\lambda^3}} \tag{3.25}$$

Where D and L are in meters and λ is in GHz. It should be noted that for the best gain the following inequality should be followed.

$$0.8 \le \frac{\pi D}{\lambda} \le 1.2 \tag{3.26}$$

Another option for the microwave frequency is the patch antenna which has a design region given as follows [30]

$$\{0.003\lambda < t < 0.05\lambda\} \cap \left\{\frac{\lambda}{3} < h < \frac{\lambda}{2}\right\} \tag{3.27}$$

3.3.4 Design Variables for subsystem

From the architecture design and communication subsystem sizing equations we have the following input variables which will be analyzed for cause-effect relations to determine which design choices will make the largest impact. There are design variables from the hardware side, as well as the missions orbit. The following hardware design variables have been identified based on the preceding design equations and will first be analyzed in design of experiments to determine which will become parameters and which will remain variables for optimization.

- Data rate R
- Antenna max rotation (Θ_{tx})
- Transmission antenna power (P)
- Trans/Receiver gains (G_i)
- System Noise temperature(T_s)
- Frequency (f)
- Modulation technique

The following variables have been identified as factors in the sizing of the communication subsystems but are more closely related with other subsystem, mission and environmental variables. As such there effect will be analyzed however their respective size drivers will be based on the subsystem or mission parameter in question.

- Altitude
- Ground station location

- Weather losses (for worst case scenarios)

Hardware variables include transmitter power antenna gains, system temperature, line loss,and bit rate.

3.3.5 Data Transmitted

Data rate of the system (bps), Time in view, and fractional viewing time are the variables in play. The hardware component is simply bps and transmitting antenna rotation (power, time in view). The mission orbit selection and ground station location will determine time in view, fractional viewing time.

3.4 Electrical Power System (EPS)

The EPS needs to have knowledge of the solar cell position, battery position, and current draw. If the current draw is too high the EPS must reset the entire system to prevent hardware failure. If the charge of the batteries needs to increase, the solar panels will need to be adjusted and a new output of their position must be had for the ADCS. This section is divided into two parts, power generation and power storage.

3.4.1 Solar Power Generation

Determining the solar array size is dependent on the time the satellites time in different power modes, time in sunlight, and solar cell type. The cell type determines both the efficiency of the satellite panels as well as the degradation percentage per year due to environment and usage. The selected cell material was GaAs due to superior performance towards end of life. It is assumed that the solar array incidence angle is at worse within 30 degrees of the sun light. Estimating the total power of the solar array for generation can be found using equation 3.28.

$$P_{sa} = \frac{\left(\frac{P_e T_e}{X_e} + \frac{P_d T_d}{X_d}\right)}{T_d} \tag{3.28}$$

The power in the eclipse orbit and daylight orbit are dependent on the satellites mission objectives and should be worst case scenarios. The efficiencies X_i are dependent on the power regulation type but range from 60% to 85% typically [1]. The beginning of life power is cell and manufacturing/integration dependent. This is estimated as

$$P_{BOL} = P_0 \cdot I_d \tag{3.29}$$

For a typical GaAs cell this estimated at181 $\frac{W}{m^2}$. The end of life power can be found by extrapolating the estimated degradation per year over the lifetime of the satellite, assuming no failures in the solar cell this is give as follows:

$$P_{EOL} = P_{BOL} \cdot (1 - \frac{degradation}{yr})^{lifetime} \tag{3.30}$$

From the end of life power we can size the solar array such that it will generate enough power at the last year of its life and is given as the ratio $\frac{P_{sa}}{P_{EOL}}$. The estimated mass is can be estimated as

$$M_{sa} = \frac{P_{EOL}}{M_{sp}} \tag{3.31}$$

typically the specific mass is around 14 to 47 $\frac{W}{kg}$ at the end of life.

3.4.2 Power Storage

The batteries must be charged in the daylight period and used during the eclipsed period which leads. The estimated necessary battery capacity can be found with equation 3.32.

$$B_{cap} = \frac{P_{ecl} T_{ecl}}{(DOD) N\, n} \tag{3.32}$$

Where DOD is the estimated depth of discharge, N is the number of batteries and n is the transmission efficiency of the battery. The mass of the batteries can be estimated from the power density of the battery material type and some margin for cabling. It is assumed that anything less than 30 Amps for the total system draw can be around 0.05kg for the total connections [1].

3.5 Payload

For this satellite mission an infrared imager was chosen as the optical payload. Because the payload drives the requirements for the satellite bus it is important to find a locally optimal baseline payload before evaluating the design of the bus. In this case the sizing process for a passive optical payload was adapted from the SMAD Firesat example. The scaling ratios and constants provided by the design equation were applied to the Argus 1000 Infrared spectrometer to compare to modern small satellite remote sensor payloads.

3.5.1 Payload Design Size Estimates

The initial baseline payload can be estimated by estimating eight design variables based off of mission requirements, trade studies and historical data. The design variables and typical levels are discussed in the next subsection. For the preliminary investigation it is assumed that the satellite in question has a near circular orbit ($e \approx 0$). The baseline design estimate is provided from SMAD.

After the orbit has been decided the payload can begin being designed. For the preliminary design we can assume a circular orbit and a spherical Earth. We begin by determing the orbit period, in minutes, for the circular orbit which can be found with equation 3.33.

$$P\ (\text{min}) = 1.658669\text{e-}04 \cdot \sqrt{R_E + Altitude} \tag{3.33}$$

It will be useful later on to determine the ground velocity of the spacecraft over the Earth. In our case we can simply the ground velocity to be constant (assuming no perturbations for our orbit) and simplifies to the following.

$$V_G = \frac{2\pi \;\; R_E}{P\ (\text{sec})} \tag{3.34}$$

The angular radius, ρ, seen from the satellite to the target is the angle from the satellite which goes from the subsatellite point to the outer horizon of the Earth. This can be calculated for with equation 3.35

$$\rho = sin \frac{R_E}{R_E + Altitude} \tag{3.35}$$

The Earth central angle, λ is the angular radius as seen from the center of the earth between the subsatellite point and the target. The region viewable by the spacecraft can also be defined from the center of the earth as λ_0. The sum of λ_0 and the satellites angular radius is equal to 90°.

$$90^\circ = \lambda_0 + \rho \tag{3.36}$$

The distance between the satellite and the outer horizon of the earth can now be calculated.

$$D_{max} = R_E \cdot tan(\lambda_0) \tag{3.37}$$

The elevation,E, from the observer on the ground to the satellite is the angle from the target to the satellite in space. The incidence angle is the angle from the normal of the targets surface to the satellite. The sum of the incidence angle and elevation angle is 90°. By defining a maximum incidince angle (or minimum elevation angle), we can compute the maximum angle in view and rhe maximum earth central angle for the satellite using equations 3.38 and 3.39 respectively. It is important to see that the swath width is simply twice the maximum earth central angle.

$$\eta_{max} = sin^{-1}(cos(90 - IA_{max}) \cdot sin(\rho)) \tag{3.38}$$

$$\lambda_{max} = 90^\circ - (90^\circ - IA_{max}) - \eta_{max} \tag{3.39}$$

Using the calculated values the slant range to the target can be calculated using equation 3.40

$$R_S = R_E \cdot \left(\frac{sin(\lambda_{max})}{sin(\eta_{max})} \right) \tag{3.40}$$

The maximum along track ground sampling distance, Y_{max}, is a design variable which is a spatial resolution parameter. This is the worst ground sample distance at the edge of the swath. Using this design parameter we can find the instantaneous FOV for a single pixel using equation 3.41.

$$IFOV = \frac{Y_{max}}{R_S} \cdot \frac{180^\circ}{\pi} \tag{3.41}$$

From Y_{max} it is possible to also determine the across track max ground sample distance through a simple trigonometric expressions.

$$X_{max} = \frac{Y_{max}}{cos(IA_{max})} \tag{3.42}$$

The best spatial resolution of the optical payload (viewing the target directly under the satellite) can be found as well using right triangles.

$$X, Y_{min} = IFOV \cdot Altitude \cdot \frac{\pi}{180^\circ} = 2 \cdot Altitude \cdot tan\left(\frac{IFOV}{2}\right) \tag{3.43}$$

Data rates can now be found by determining the number of pixels that are along the track (this is highest when the optical sensor has the highest nadir angle), the number of swaths recorded in one second, the number of pixels. These can be found in equations 3.44, 3.45, and 3.46 respectively.

$$Z = \frac{2 \cdot \eta_{max}}{IFOV}^{C} \tag{3.44}$$

$$Z_A = \frac{V_G}{Y} \cdot 1sec \tag{3.45}$$

$$Z = Z_C \cdot Z_A \tag{3.46}$$

The encoding bits per pixel is a design parameter based on your dynamic range necessary for data acquisition. After this is decided the data rate may be found as shown in equation 3.47

$$DR = Z \cdot \frac{Bit}{Pixel} \tag{3.47}$$

There are two other key objectives for the payload design, sensor dwell time and aperture diameter. The dwell time can be increased or decreased based on the number of pixels the instrument has, however the dwell time must remain above the detection time constant of the payload. The estimated integration period for the pixel can be found in 3.48, where N_m is the number of pixels on the instrument scanner along the track.

$$T_i = \frac{Y_m ax \cdot N_m}{V_g \cdot Z_C} \tag{3.48}$$

The aperture diameter can be estimated from the pixel width (d), image quality factor (Q), focal length (f), and operating wavelength (λ_{op}). The focal length can be estimated with equation 3.49 and the payload aperture diameter by equation 3.50

$$f = \frac{Altitude \cdot d}{X_m in} \tag{3.49}$$

$$D_{aperture} = \frac{2.44\lambda_{op} \cdot f \cdot Q}{d} \tag{3.50}$$

The ratio of the aperture diameter of the payload and a previously flown instrument can be compared and used or estimating the size. The ratio, R_{AD}, is also used to determing the scaling parameter K for the instrument. If the ratio is less than 0.5, the scaling parameter is 2. Otherwise the scaling parameter can be left as 1. If X_o, Y_o, Z_o denote the linear dimensions of the previous hardware, and variables kg_o and W_o denote the mass and power as well. The estimated physical parameters can be found using the following five equations:

$$X_{new} = R_{AD} \cdot X_o \tag{3.51}$$

$$Y_{new} = R_{AD} \cdot Y_o \tag{3.52}$$

$$Z_{new} = R_{AD} \cdot Z_o \tag{3.53}$$

$$kg_{new} = K \cdot R_{AD} \cdot kg_o \tag{3.54}$$

$$W_{new} = K \cdot R_{AD} \cdot W_o \tag{3.55}$$

The five physical parameters, data rate, ground sample distance, and image quality factors are all possible objectives for payload optimization.

3.5.2 Design and Optimization Variables

The following design variables are used in the design space exploration and optimization routines.

- Altitude (km)
- Maximum Incidence Angle (θ)
- Maximum Along Track Ground Sample Distance (m)
- Bit Per Pixel (Interger $\geq$ 1)
- Pixels for Instrument (Interger)
- Width of Square Detector (μm)

- Image Quality Factor (0.5 to 2)
- Operating Wavelength (λ)

Typical altitudes for altitude lie between 300 and 700 km for typical nanosatellites. The maximum incidence angle is the maximum angle expected for which the payload will be used. It is relative to the normal of the local ground at the target sight. The incidence angle and the spacecraft elevation angle (from the ground station) sum to 90 degrees. The maximum along track ground sample distance is the spatial requirement at the maximum earth central angle (i.e. maximum off nadir coverage). This can be visualized the ground sample distance at the edge of the swath, with the ground sample distance decreasing as you get close to nadir. Bit per pixel is an integer for the number of bits for the pixel data.

The pixels for instrument variable is the number of pixels along one dimension (along track) in the optical payload for whisk broom type scanners. Width of square detector is the physical dimensions for an individual pixel in the optical system. In the case of CMOS sensors this can be as small as $2\mu m$. Larger pixel sizes such as those on the Lunar Image Spectrometer on the SELENE mission are 40 μm also exist. The image quality factor is a scaling function to reduce the image quality or due multiple samples. It directly influences the data rate by scaling the total number of bits in the image. It is suggested that a factor of 1.1 be used as the nominal good image quality factor [1]. Finally the operating wavelength is the wavelength chosen for the optical payload. In our case it is para-metricized at $3\mu m$, the short infrared band.

3.6 Redundancy Model

The triple module redundancy (TMR) model is comprised of three majority voters, three minority voters, and three inverted tri-state models, and an error detection phase. The three inputs are sent to the three majority voters in parallel. These three majority voters then provide the most common input based on the following truth table shown in table 3.2. Similarly the minority voter (truth table in table 3.3) provides the least common output as an output which is used by the tri-state buffer which is used to suppress erroneous outputs.

The majority voters output (Y) is sent as the primary channel (P) input for the minority voter. R1 and R2 are from the other respective channels. If the primary input (P) is part of the majority then the inverted tri-state (an active low) is active and the data is allowed to pass through. If the primary channel is not part of the majority then the tri-state buffer prevents the data from being passed through allowing all outputs to be the correct input as long as two faults have not occurred in the logic module. The fault detection block is a combination of gates which will generate a high signal in order to send a reset command to the FPGA. This is a combination of XOR and OR gates based in the incoming signals before and after the majority voters as simple use for the current design. The test of the TMR module as well as the TMR module setup is shown in Appendix C. A switching algorithm has been developed to move data between failed modules in a predetermined time frame however the specifics can not be shown due to it being intellectual property. The redundancy test setup can be viewed in Appendix C as well.

Table 3.1: Adapted system complexity table from [1]

Requirement or Constraint	System Complexity		
	Simple	Typical	Complex
Processing Commands:			
Command Rates	50 CMD/s	50 CMD/s	≥ 50 CMD/s
Computer Interface	None	Computer or Stored Commands	Yes
Stored Commands	None		Not Needed
Number of Channels	<200	300-500	>500
Processing of Telemetry Data			
TLM Rates	<4 Kbps	4-64 Kbps	64 - 256 Kbps
Payload Data	None	1-200 kbps	10 Kbps - 10 MBps
Computer Interface	None	None	Yes
Number of Channels	<200	400-700	>500
Other			
Mission Time Clock	None	Included	Included
Watchdog	None	Included if OBC chosen	Yes
ADCS Function	None	None	Yes
Bus Contraints	Single Unit	Up to multiple units	Integrated or Distributed
Total Ionizing Dose	<2kRads	2-50 kRads	50kRads - 1MRad

Table 3.2: Majority Voter Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.3: Minority Voter Truth Table

P	R1	R2	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0

Chapter 4

SYSTEM EXPLORATION, DESIGN OF EXPERIMENTS AND SIMULATION MODELS

4.1 Requirements Generation

Each nanosatellite has a set of design and mission requirements which much be fulfilled in order to be considered a successful mission. The generation of these requirements are based on inputs from a combination of stakeholders including designers, researchers, launching companies, and regulatory agencies. For the purpose of this project two documents will be created. The first is the Certification Acceptance Requirements Documents, which is a physical document detailing the traceability of all system and mission requirements. The second is an internal requirements module which can be used to provide quick checks on system level designs and will eventually be combined into the simultaneous analysis and design optimization process.The requirements module is integrated with other system and subsystem models and provide some amount of tracability as well as automatic documentation for theoretical designs. The module will provide a quick check for the feasibility of the design but are not a substitute of actual verification and validation of the physical hardware which is to be flown. The major limitation of the requirement module is the fidelity of the subsystem modules.

4.1.1 Certification Acceptance Requirements Document

The requirements created for this project have been developed based on input from John Hines, Reine Ntone, and Tyler Woods. They have been collected into a certification and acceptance requirements document (CARD) located at the end of Appendix 2. The CARD is a single integrated document which provides all necessary requirements and objectives necessary for mission completion. It also provides a breakdown of which party is responsible for the completion of the requirement, the verification method, and any necessary verification documents. The CARD for the proposed mission is provided in the appendices.

The CARD is a table of requirements which shows the requirement I.D., a description, traceability (documents, etc.), dictates what party is responsible, how the verification will be shown, what documents are linked to the verification, and any comments the requirement may pose to the subsystem teams, systems engineers, management, or customers. Verification of requirements being met are done through testing (T), review of design (D), analysis / memo (A), and inspection (I).

4.2 Design Space Exploration and Optimization

This section will cover design space exploration techniques and possible methods of analysis in order to determine which variables will have the most effect.

4.2.1 Design Space Exploration and Design of Experiments

Spacecraft system designs complex and difficult to design due to the large set of design variables that are needed to find near optimal solutions. Typically the design variable inputs are in a range of $O(10^5 \sim 10^7)$ and are coupled with other design variables [31]. Due to time limiatations and the inability to validate against hardware, we will focuse on the smaller set, $O(10^1)$ set of variables in the baseline design. The goal of these large data set optimizations is not to find a global optimum, but rather converge towards the optimal solution in order for designers to effectively move the end design towards their specific goal. In order to minimize the time of the optimization routines in OpenMDAO a design of experiment analysis is performed in order to set variables with little effect on the system design to a single parameter based on historical data and designer intuition. Using OpenMDAOs design of experiment (DoE) driver, parametric models of each subsystem have been created and were analyzed for a full factorial statistical analysis. The full factorial statistical analysis is a method which analyzes each variable (a factor) at multiple values (levels) in order to understand how the outcome (response variable) changes. The benefit of a full factorial analysis is the ability to detect interaction between multiple design variables with respect to the outcome. Design variables which have limited effect on the outcome or have limited options to begin with may be set to specified values based on the reality of the situation. For example there are only so many ground stations which the designer may have to choose from, or the payload may already exist and the designer is then required to make as little modifications to it as possible to minimize performance loss of the payload.

4.2.2 Design of Experiments Analysis

When first analyzing a DoE, it is important to code the data into a useful format. For our purpose a linear transform is used to map the different levels of each factor into a range of -1 to 1. This has two benefits when analyzing the data. The first is that there is no change to the shape of the distribution of independent variables which is important when you need to verify assumptions for statistical models such as analysis of variance (ANOVA). Secondly the transformed factors in full factorial design matrix has all orthogonal columns [32]. This is important as any experimental design which is orthogonal allows each factor to be evaluated independently of each other. It has bonus effect which makes interaction plots significantly more readable, but that has no analytical benefit. The linear transform applied is given in equation 4.3 and its inverse function in 4.4. Let $\mathbf{U_i}$ be the vector of the i-th factor which has the level for each experiment. $X_{i,H}$ and $X_{i,L}$ represent the highest and lowest level respectively. We can then define the scaling constants for the transform as follows:

$$a_i = \frac{X_{i,H} + X_{i,L}}{2} \tag{4.1}$$

$$b_i = \frac{X_{i,H} - X_{i,L}}{2} \tag{4.2}$$

$$\mathbf{Y_i} = \frac{\mathbf{U_i} - a_i}{b_i} \tag{4.3}$$

$$\mathbf{U_i} = \mathbf{Y_i} * b_i + a_i \tag{4.4}$$

By ensuring orthogonality using statistical methods such as ANOVA and graphical methods such as matrix scatter plots, we can analyze the main and interaction effects between design variables on each response output. The main graphical methods that were used were the main effect and interaction effect scatter plots, response distribution histograms and box plots. In most cases this showed immediate results of what design factor impacted the response outputs as main effects, which set of design variables had two-factor interaction, and in some cases that the response variable was indifferent to the factor. Using this knowledge, trade studies for specific design variables (e.g. operational wavelength), and general intuition specific factors were set to parameters based on historical data or constants taken from the SMAD book. This reduced the number of iterations for the optimizer function and reduced the overall optimization time.

For design spaces which do not have clear results from graphical techniques, an ANOVA test can be performed. For this test to be applied there are a six assumptions which must be met. While some assumptions are robust to how well your data fits to it, assumption 5 is required to be able to substantiate any claim from this test. The assumptions are listed as follows:

(1) Response variable is continuous

(2) Independent variable should have two or more categorical independent groups

(3) There should be independence of observations

(4) No significant outliers

(5) Response variable (residuals) should be normally distributed for each category of the independent variable

(6) Homogeneity of variances. (Variances of response groups should be within 2 times the lowest)

The general model applied to to the data set (in this case two factors (i and j) is given as

$$Y_{ijk} = \mu + \alpha_i + \beta_j + E_{ijk} \tag{4.5}$$

Where for response value at (i,j), $Y_{i,j,k}$ the mean value, a factor effect is found (e.g. the i-th factor has the overall effect α_i), and some error term, E_{ijk} is computed. The predicted values for each observation then becomes:

$$\hat{Y}^{ijk} = \hat{\mu} + \hat{\alpha}_i + \hat{\beta}_j, \text{ with residual } R_{ijk} = Y_{ijk} - \hat{\mu} - \hat{\alpha}_i - \hat{\beta}_j \tag{4.6}$$

For a design of experiment with no replication, (i.e. $Y_{ijk} = \mu_{ij} + E_{ijk}, \quad k = 1$) there are some small things to make note of. The validity of an N-Way ANOVA with no replication significantly decreases and are listed below [33]. For assumptions that there may be significant interaction effect.

- For fixed factors (i,j = 1,...,n)
    - Model 1 n-ANOVA can be performed with caution
    - Model 2 n-ANOVA can be performed with higher type II error
    - Interaction effects can not be tested
- For random factors (i,j = 1,...,n)
    - Factors A and B can be tested as factor I MS / remained MS
    - Interactions can not be tested
- For mixed models (A is fixed and B is random)
    - Factor A can be tested
    - Factor B can not be tested
    - There is no interaction test available.

In the case of correctly assuming that there is no interaction effect, all factors may be tested for Factor (I) MS / remainder MS. The ANOVA process is fairly lengthy to describe in this report and sufficient examples may be found online or in engineering statistics textbooks. For the purpose of this study, the ANOVA process was handled by MATLABs statistical toolbox. Model validation can be performed by analyzing the normal probability plot and run sequence plot of residuals, as well as a scatter plot of the predicted values against the residuals [32]

4.2.3 Optimization

While there are many optimization methods they can be broadly categorized into methods which use gradients and those that do not. Gradient free optimizer suffer computational performance as the number of design variables increase due to the the use of design variables across the entire range of the design space [34]. Gradient optimization methods allow for designers to assess the system sensitivity to various design variables and select a range for design options. For the purpose of this project the OpenMDAO tools will be used. OpenMDAO is a set of optimization methods implemented in Python to allow researchers to analyze complex (high design variable sets) systems by grouping separate components into processes which can be run in parallel to satisfy a set of constraints.

The OpenMDAO approach assigns each variable a residual which is then grouped to form a set of nonlinear system of equations [35]. Hwang has shown that by driving the residuals to zero that a system of nonlinear equations can be formed which allow the computation of total derivatives. The unifying derivatives equations is shown in equation 4.7.

$$\frac{\partial R}{\partial u}\frac{du}{dr} = I = \frac{\partial R}{\partial u}^{T}\frac{du}{dr}^{T} \tag{4.7}$$

Using the computed total derivatives it is possible then to use the methods in OpenMDAO to either explore the design space or to optimize a chosen initial design to find a local optimal solution. The use of the optimizer is to find locally optimal solutions to use as a baseline design for nanosatellites. Future work is required to perform a simultaneous analysis and design to find optimal solutions for an end product, that is beyond the scope of this project. To find a good design compromise with the variety of objectives and constraints, the Kriesselmeier-Steinhauser function will be used to aggregate all constraints into a single criteria which will allow the optimizer to find a local compromise from possible solutions [31, 36]. This function is shown in equation 4.8.

$$KS(x) = f_{i,max}(x) + \frac{1}{\rho} ln \sum_{i=1}^{n} e^{\rho(f_i(x) - f_{i,max}(x))} \tag{4.8}$$

4.3 Simulation Models

The simulation models are used to take analyze designs from the design space exploration and optimization routines and evaluate their feasibility for normal operations. They are not full detailed modes as the level of fidelity is not something that can be done by one person in the time constraints imposed currently. Each section lists the model levels needed to perform at the very basic level a data and power draw test for analyzing some mission requirements. Note that due to time constraints some of these modules may not be implemented.

4.3.1 Orbit Model

The orbit model must provide the position, sunlight line of sigh, target line of sight, and environmental disturbances to the spacecraft.

4.3.2 Attitude Determination and Control System

The ADCS module will only saturate momentum and then dump it via magnetic torque rods. This will be used to exercise the battery modules.

4.3.3 Electrical Power Subsystem

The electrical subsystem must include the solar arrays, batteries, and estimated mass of power regulators and connections. The model must be able to estimate the state of charge based on system draw. The temperature of the solar array also reduces the efficiency of power generation by about 0.5% per degree above 28 $^\circ$ C.

4.3.4 Communications

The communications systems will be able to measure the ideal data rate and power draw on the system when over specified locations in orbit relative the the ground targets and stations.

4.3.5 Payload

The payload must be able to warm up, operate, and gather data based on the LOS of the target. It is expected that the data will simply be randomly generated binary data as it is too difficult to replicate performance data for the payload system.

4.4 Benchmarks for Design of Experiment analysis and Optimization

Due to the large amount of space required to perform and demonstrate DoE techniques it is recommended that the reader understand the relevant sections from the NIST e-handbook of Statistical Methods [32]. An optimization benchmark for a multi-objective constrained problem can be found in the following section. It is important to note that using the K-S optimization routine provided by the pyOpt package that it is possible to initiate a optimization problem outside of the design region and it will automatically move towards to closest locally optimal design point. The benchmark case will showcase the ability of the technique in an easy to view two dimensional problem.

4.4.1 Optimization with Combined Constraints

The following benchmark case comes from the Multi-Objective Optimization Using Evolutionary Algorithms book by K. Deb [37]. The simple optimization problem is only in the $\mathbf{R}^2$ domain which allows it to be easily visualized for both the feasible search space. Similarly with only two objective functions it is possible to graphically show the intersection of f_1 and f_2.

$$\text{Minimize} = \begin{cases} f_1(x, y) = x \\ f_2(x, y) = \frac{1+y}{x} \end{cases} \tag{4.9}$$

In the region $0.1 \leq x \leq 1$ and $0 \leq y \leq 5$ with the following objective constraints

$$\text{s.t} = \begin{cases} g_1(x, y) = y + 9x \geq 6 \\ g_2(x, y) = -y + 9x \geq 1 \end{cases} \tag{4.10}$$

From the constraint functions g_1 and g_2 we can see that there is only a subset of the design space has been allowed for a feasible output. The design space is shown in figure C.7 in the appendix. Because of the objective functions are designed to be minimized it helps to view the response surface so that we can see how the design variables x and y change each objective. The responses for each function can be seen in figures 4.1and 4.2. From both the f_1 and the output response, we see than the lowest possible value of x is desired. However from f_2 we see that a larger value of x is desired. This requires a tradeoff between x and y design variables in order to find compromise between these functions. The relationship between the two output functions can be seen in the Pareto efficiency graph in figure 4.4. In this case we can see that the minimum value of f_1 occurs when f_2 is almost nine. On the other hand we see that f_2 is minimum when f_1 is at its peak. Any combination of design variables which results in a solution of response variables which land on this Pareto front are considered to be optimal solutions as there is no longer a point where you can decrease the output of one response without increasing the response of another. The relation between the two objective functions can be see in figure 4.3 This is a useful tool as it minimizes the possible Pareto-optimal options for a set of given

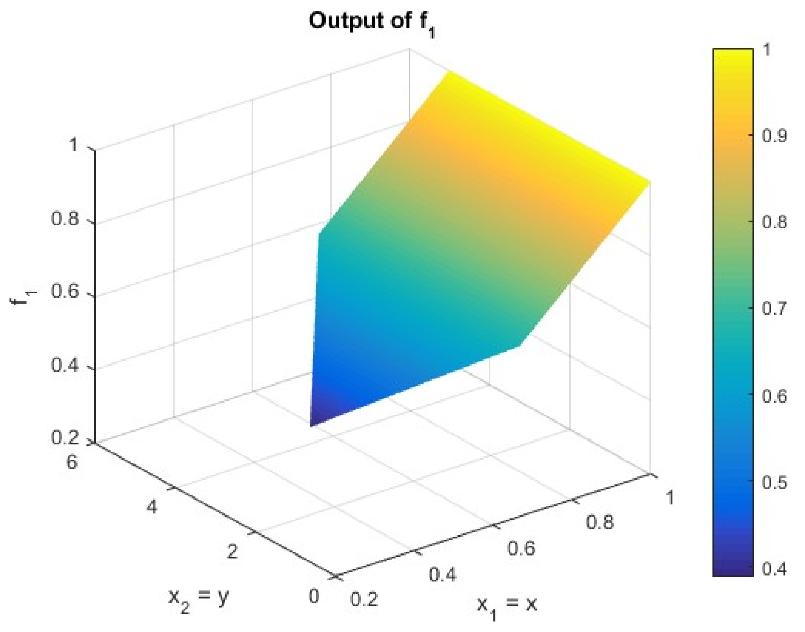


Figure 4.1: Response of output variable f_1 based on design space.

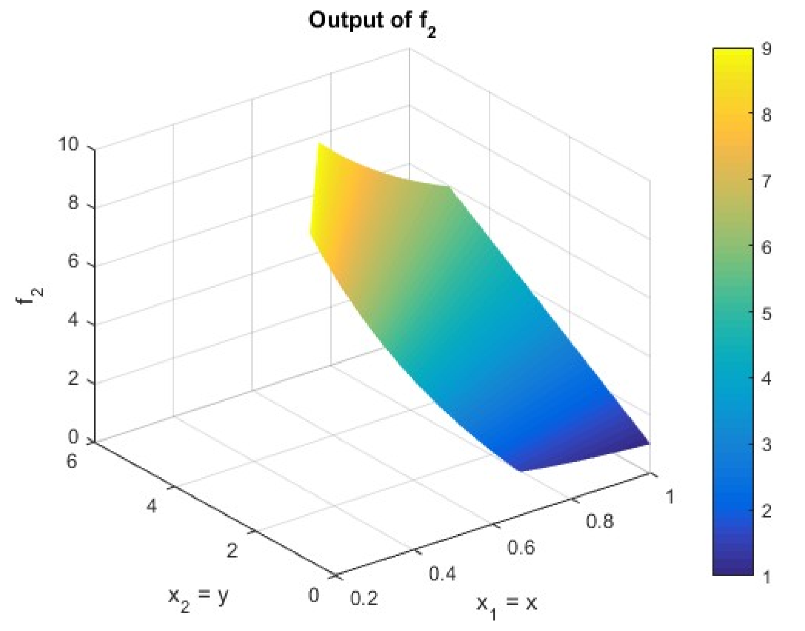


Figure 4.2: Output of the secondary objective function f_2.

constraints and objective functions. When running the optimizer it is expected to get a local solution that is on or very close to the Pareto front.

For the optimization in choice the KS function is used to consolidate the objectives and constraints into a singular unconstrained minimization function which we can then solve for. For this example we need to rewrite the constraints in the form $Ax + B \leq 0$ and scale the objective functions f_1 and f_2. This can be shown below in the equations 4.13, 4.14, 4.11 and 4.12. The scaled objective function is simply the value of the objective function divided by the value of the objective functionat the start of each KS iteration, subtracting the maximum constraint value at the start of an iteration plus one. The initial value for the starting iteration is shown below for the initialization values of $x = 0.35$ and $y = 2.5$.

$$F_1 = \frac{f_1(x)}{f_1(x_0)} - 1 - max(g_{i,o}) = \frac{0.35}{0.35} - 1 - max(11.25, 1.65) = -11.25 \tag{4.11}$$

$$F_2 = \frac{f_2(x)}{f_2(x_0)} - 1 - max(g_{i,o}) = \frac{10}{10} - 1 - max(11.25, 1.65) = -11.25 \tag{4.12}$$

$$F_3 = -9x - y + 6 \leq 0 \tag{4.13}$$

$$F_4 = -9x + y + 1 \leq 0 \tag{4.14}$$

This can then be combined to create the KS function which will be used in the KS optimization routine. This function is evaluated and its optimization envelope minimized while the optimization routine searches the design space. This composite KS function for this benchmark problem is shown below in equation 4.15.

$$KS(x) = max(F_1, F_2, F_3, F_4) + \frac{1}{\rho} ln \sum_{i=1}^{4} e^{\rho F_i - max(F_1, F_2, F_3, F_4)} \tag{4.15}$$

From the From equations 4.11 and 4.12 it is immediately apparent that the initialization is actually outside of the design space based on the constraints, however this does not pose a problem for the KS optimization algorithm. While the function may not converge to a global minimum, it will provide a solution which is a local compromise between the objective functions and within the feasible design space. This is illustrated in figure C.8, located in the appendix, where the found minimum value of the constructed KS function is on the Pareto front and is a feasible solution. Due to the simplicity of this problem only a small portion of the OpenMDAO framework was needed. The benchmark example was ran through the pyOpt package utilizing the builtin KSOPT functionality. This routine was originally shown by Wrenn but has been added to the pyOpt package by Perez [36, 34]. From this optimization benchmark we can see that it is possible to find solutions which provide compromise from multiple objectives. The process used in this section will be performed for each individual subsystem and mission design for the nanosatellite mission in order to look for a valid design in the satellites design space.

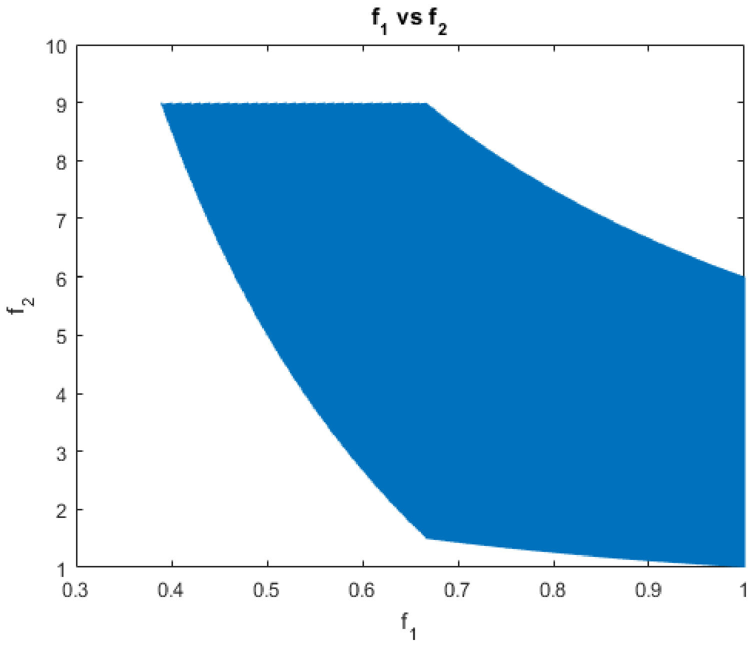


Figure 4.3: Output of f_1 and f_2 for the same input variables x and y.

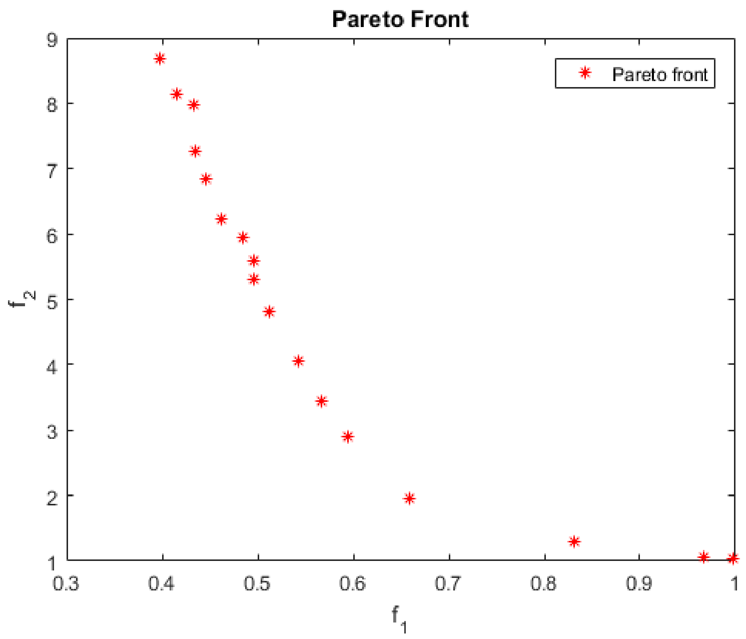


Figure 4.4: Pareto front of benchmark response variables f_1 and f_2.

Chapter 5

DESIGN OF EXPERIMENT AND OPTIMIZATION ANALYSIS

The following subsection give an overview of the DoE analysis as well as the results from the optimization codes.

5.1 Payload

By using the design equations for a passive optical payload, a design space exploration was done using reasonable extreme values to see the interaction of factors at their most extreme. In this case a 5^8 full factorial DoE was performed in order to see the extremes as well as the median values. All DoE plots are scaled to their orthogonal coded factor level. For all plots along the diagonal, -1 represents the lowest value in the design space and +1 represents the highest value in the design space. For all interactions plots (j ¿ i) the far left value of negative one consists of factors at their extreme opposite ends (i.e. the highest from i and the lowest from j), while the values on the far left (i.e. the highest i and highest j) represent the extreme ends on the upper or lower bounds. This can be understood more clearly in the table below. The outside bounds (top row and first column) are the factor levels. The interaction effect ($i \cdot j$) is plotted against the response variable in question. For this case only the aperture diameter interaction is shown here. The rest of the DoE scatter plots may be found in Appendix C. The optimization technique has provided the following system sizes. 0.136x0.477x0.0613 m dimensions in X,Y,Z respectively. An estimated power use of 0.18W and a mass of 0.15 kg. The geometric diffraction limited ground sample distance at nadir = 0° is 88m between

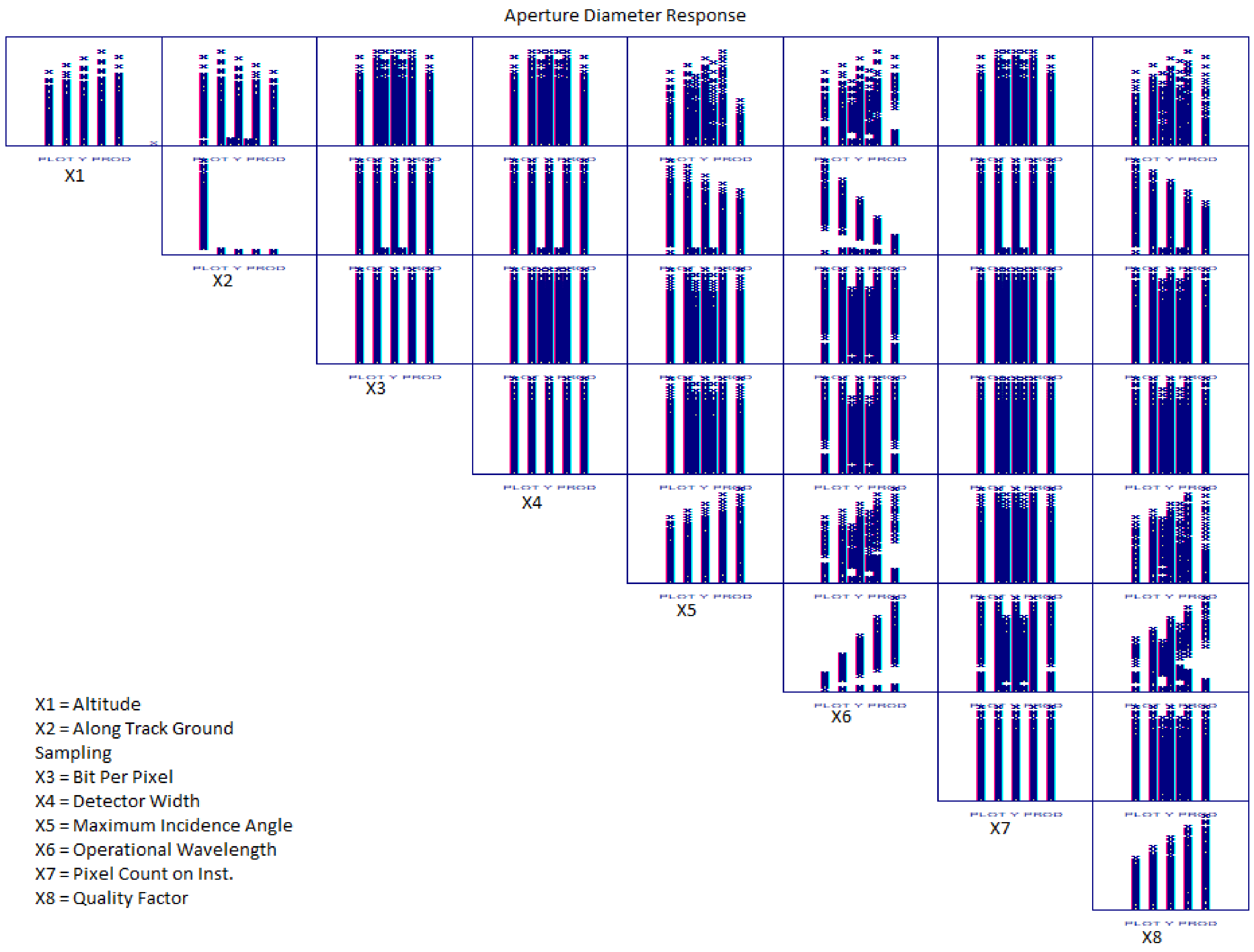


Figure 5.1: DoE scatter plot of Aperture Diameter

Table 5.1: Factor Interaction Chart

Factor Level	-1	-0.5	0	0.5	1
-1	1	0.5	0	-0.5	1
-0.5	0.5	0.25	0	-0.25	-0.5
0	0	0	0	0	0
0.5	0.5	-0.25	0	0.25	0.5
1	-1	-0.5	0	0.5	1

center pixels. The estimated data rate is 2.4 Mbps and the field of view is 0.5 degrees.

5.2 Bus

The following subsections describe the design of experiment and optimizations. DOE scatter plots are provided in Appendix C.

5.2.1 ADCS

The nano-satellite control system can be easily oversized due to the small disturbance torques seen near the launch altitude. In this scenario an altitude of 250 km was selected as a way to determine the sizing for a satellite near the end of its life. The aerodynamic drag and magnetic fields produced the largest amounts of torque which was compensated by small reaction wheels (on the order of 10mm in the z axis and covering the face of 1U. The sizing of reaction wheels and torque rods are dependent on performance requirements for slew rates and momentum saturation and the interaction between the two was not considered. In the case of the star tracker the mass and power were a function of the square of the pointing knowledge, while the physical dimensions scaled linearly. An integrated unit was not considered for this subsystem. The integrated units can provide significantly better results compared to individual products.

Each reaction wheel had an estimated volume of 1180 mm^3. The torque rods were approximately 17 x 17 x 135 mm (overestimated) with a high magnetic dipole required (1.5 $A \cdot m^2$). The Star tracker had the largest volume required at 279560 mm^3, slightly under 1/3U. The overall estimated power was 1.464 watts for steady operations. The peak power was not considered but can reach in the order of 10 watts per wheel based on data sheets from models chosen in the regression model. The total mass is estimated at 0.672 kg.

5.2.2 C&DH

Most flight qualified hardware is more than capable of running the bare minimum software requirements that is estimated. Using the selection of models from the NASA state of the art technologies report the largest form factor, weight, and typical power was chosen. This resulted in a size of 96 x 90 x 12.4 mm and a weight of 0.094 kg. Objective constrained design size:

Chapter 6

MODEL AND SIMULATION ANALYSIS

The current model level is seen in figure 6.1. Only the orbit and line of sight modules have been implemented as of the time of this writing.

6.1 Orbit

Current levels for orbit dynamics include NBodySimulation using the Sun, Earth, Moon, and Juptier. Solar, J2, and atmospheric perturbations have not yet been added. A simulation of the ISS from March 25th 2018 based on data from JPL Horizons has shown good ($\pm 0.0014\%$ difference) agreement for the satellite over the simulation period of one day, a plot of the distances from the center of the earth can be seen in figure 6.2. Solar line of sight and estimated power generation has been recorded but not yet validated.

6.2 Redundancy and Error Detection

The TMR model has been shown to detect discrepancies as long as all three modules have not yet failed. A switching algorithm has been developed to move the failed peripherals from one FPGA to a second FPGA in a reasonable manner however validation of the model on hardware has in progress.

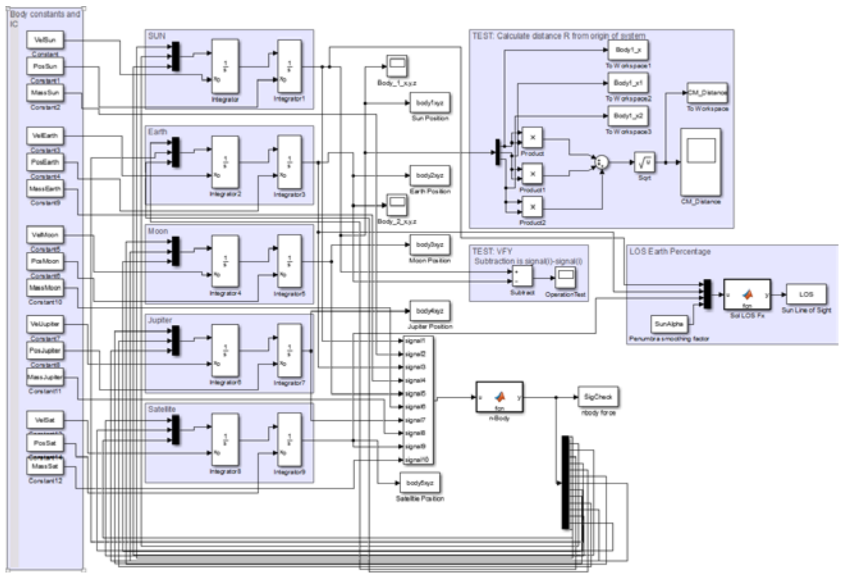


Figure 6.1: Current Simulink Model

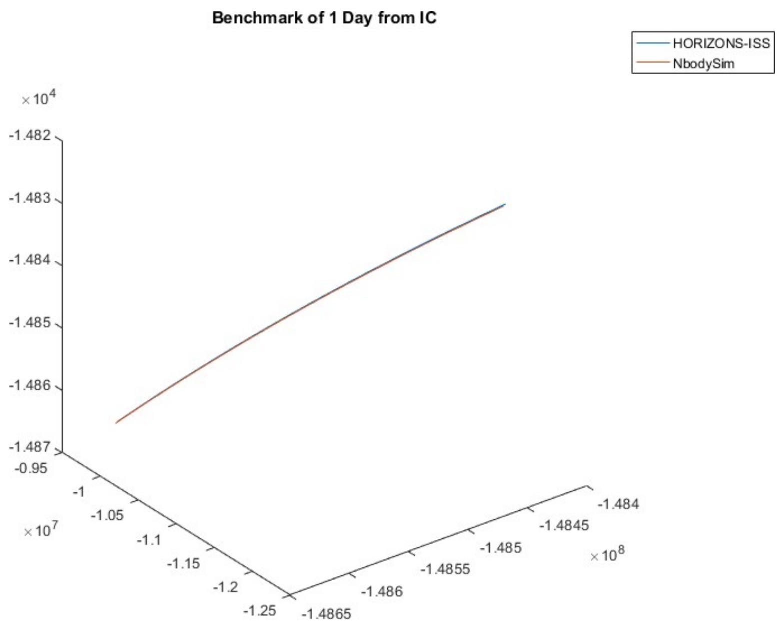


Figure 6.2: ISS Orbit from N-Body sim compared to JPL Horizons log, i.c. March 25, 2018

Chapter 7

PATH FORWARD

High fidelity models of individual subsystems should be integrated into the orbit model. The core of each subsystem model interfaces with the orbit and environmental model and the mission script acts as the OBC controlling the dataflow between the models. The ADCS module will require multiple operating states including detumbling, payload maneuvering, and internal tracking of momentum. The EPS model will consist of power and storage sub modules which will manage electrical states and estimated power draw assuming no faults in the system. The payload simulation model and communication model will require operation when certain objectives are within line of sight. The estimated data (random binary) will need to be stored, fed through the redundancy module too test the fault scheme, and then sent through the communication model to estimate the performance of the communication system. Thermal properties should be monitored in all subsystem modules as small nanosatellites have a small surface area and will be at a higher risk of operating too hot for the stable operating envelope.

ACKNOWLEDGMENTS

I would like to thank Tyler Woods, Reine Ntone, John Hines, Dr. Periklis Papadopoulos & most importantly my family for their help and support throughout this project. The FPGA n-module redundancy algorithm were created as part of a larger project for JH Technology Associates LLC (JHTA) and are the intellectual property of JHTA.

BIBLIOGRAPHY

[1] James R. Wertz and Wiley J. Larson, editors. *Space Mission Analysis and Design*. Springer, third edition, 2010.

[2] Amie C. Stryker and David R. Jacques. Plug-and-play satellite: A modularity assessment. *Journal of Spacecraft and Rockets*, 49(1):91–100, January 2012.

[3] Rapid Spacecraft Development Office. Ssl 1300 spacecraft bus for rsdo applications, 08 2016.

[4] Ronald L. Ticker, Frank Cepollina, and Benjamin B. Reed. Nasa's in-space robotic servicing. In *AIAA SPACE 2015 Conference and Exposition*, AIAA SPACE Forum. American Institute of Aeronautics and Astronautics, August 2015.

[5] James Lyke, Quinn Young, Jacob Christensen, and David Anderson. Lessons learned: Our decade in plug-and-play for spacecraft. In *28th Annual AIAA/SU Conference on Small Satellites*. AIAA/USU, 08 2014.

[6] A. Jacobs, A. D. George, and G. Cieslewski. Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space. In *2009 International Conference on Field Programmable Logic and Applications*, pages 199–204, Aug 2009.

[7] Jennifer S. Hane, Brock J. LaMeres, Todd Kaiser, Raymond Weber, and Todd Buerkle. Increasing radiation tolerance of field-programmable-gate-array-based computers through redundancy and environmental awareness. *Journal of Aerospace Information Systems*, 11(2):68–81, February 2014.

[8] Justin A. Hogan, Raymond J. Weber, and Brock J. LaMeres. Reliability analysis of field-programmable gate-array-based space computer architectures. *Journal of Aerospace Information Systems*, 14(4):247–258, April 2017.

[9] S. N. Chau, L. Alkalai, A. T. Tai, and J. B. Burt. Design of a fault-tolerant cots-based bus architecture. *IEEE Transactions on Reliability*, 48(4):351–359, Dec 1999.

[10] M. Salehi, A. Ejlali, and B. M. Al-Hashimi. Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1497–1510, May 2016.

[11] Ali Nasir, Ella M. Atkins, and Ilya V. Kolmanovsky. Mission-based fault reconfiguration for spacecraft applications. *Journal of Aerospace Information Systems*, 10(11):513–516, November 2013.

[12] Bryan Palmintier, Christopher Kitts, Pascal Stang, and Michael Swartwout. A distributed computing architecture for small satellite and multi-spacecraft missions. *16th Annual AIAA/USU Conference on Small Satellites*, 2002.

[13] Victor Murray, Daniel Llamocca, James Lyke, Keith Avery, Yuebing Jiang, and Marios Pattichis. Cell-based architecture for adaptive wiring panels: A first prototype. *Journal of Aerospace Information Systems*, 10(4):187–208, April 2013.

[14] M. Martin and J. Lyke. Modular open network architecture (monarch): Transitioning plug-and-play to aerospace. In *2013 IEEE Aerospace Conference*, pages 1–10, March 2013.

[15] David McComas. Nasa/gsfcs flight software core flight system community. Flight Software Workshop, 12 2014. Beckman Institute at CalTech, Pasadena, CA.

[16] Edmund Burke. Aerospace vehicle scalable, modular, and reconfigurable technologies to bring innovation and affordability. In *30th Space Symposium*, 2014.

[17] ClydeSpace. Platforms. Website, 2017.

[18] R. R. Bate. *Fundamentals of Astrodynamics*. DOVER PUBLICATIONS, 1971.

[19] NASA Goddard Space Flight Center. *GMAT Mathematical Specifications [DRAFT]*, 2017.

[20] Navigation and Ancillary Information Facility (NASA). *An Overview of Reference Frames and Coordinate Systems in the SPICE Context*, 09 2017.

[21] Eberhard Gill and Oliver Montenbruck. *Satellite Orbits*. Springer, 2013.

[22] Matthew M Berry and Liam M Healy. Implementation of gauss-jackson integration for orbit propagation. *The Journal of the Astronautical Sciences*, 52(03), 01 2002.

[23] J. C. Mendez M.S. Allione, A. L. Blackford and M. M. Whittouck. Nasa contractor report 1005. Technical report, National Aeronautics and Space Administration, 1968.

[24] Ken Fox. Numerical integration of the equations of celestial mechanics. *Celestial Mechanics*, 1984.

[25] Howard Curtis. *Orbital Mechanics for Engineering Students*. Elsevier LTD, Oxford, 2013.

[26] J.M. Picone, A. E. Hedin, D. P. Drob, and A.C. Aikin. Nrlmsise-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *Journal of Geophysical Research*, 107(A12), 2002.

[27] Dennis C. Pak. Linearized equations for j2 perturbed motion relative to an elliptical orbit. Master's thesis, San Jose State University, 2005.

[28] Arnaud Chulliat, Patrick Alken, Manoj Nair, Adam Woods, Stefan Maus, Susan Macmillan, Ciaran Beggan, Brian Hamilton, Victoria Ridley, and Alan Thomson. The us/uk world magnetic model for 2015-2020. *Technical Report, National Geophysical Data Center, NOAA*, 2014.

[29] David McComas. Nasa/gsfcs fligth software core flight system. Flight Software Workshop, 11 2012.

[30] M. Sakovsky, S. Pellegrino, and J. Constantine. Rapid design of deployable antennas for cubesats. *IEEE Antennas & Propagation Magazine*, 2017.

[31] John Hwang. *A modular approach to large-scale design optimization of aerospace systems*. PhD thesis, The University of Michigan, 2015.

[32] National Institute of Standards and Technology. *e-Handbook of Statistical Methods*. NIST/SEMATECH, 2012.

[33] Jerrold H. Zar. *Biostatistical Analysis (5th Edition)*. Pearson, 2009.

[34] Ruben Perez, Peter Jansen, and Joaquim Martins. pyopt: a python-based object-oriented framework for nonlinear constrained optimization. *Structural and Multidisciplinary Optimization*, 45(1):101–118, 01 2012.

[35] John Hwang. Reconfigurable model execution in the openmdao framework. In *18TH AIAA/ISSMO MULTIDISCIPLINARY ANALYSIS AND OPTIMIZATION CONFERENCE*, 2017.

[36] Gregory Wrenn. An indirect method for numerical optimization using the kreisselmeier-steinhauser function. Technical report, Planning Research Corporation - Aerospace Technologies Division, 1989.

[37] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[38] Sandip Tukaram Aghav and Shashikala Achyut Gangal. Simplified orbit determination algorithm for low earth orbit satellites using spaceborne gps navigation sensor. *Artificiall Satellites*, 2014.

Appendix A

ORBITAL MECHANICS, MANEUVERS, AND MISSION DESIGN

A.1 Cartesian to Keplerian

For any spacecraft which is some distance, ${}^{E_0}\mathbf{r}^{E_0}$, and moving at some speed, ${}^{E_0}\mathbf{v}^{E_0}$ from the central body with gravitational parameter μ, the following method can be used to determine its Keplerian elements. The following vector with Keplerian elements can be calculated using only position, velocity, and central body gravitational variables $[a, e, i, \Omega, \omega, \theta]^T = f(X, Y, Z, v_x, v_y, v_z, \mu)$ using the following set of equations.

$$r = \sqrt{\mathbf{r} \cdot \mathbf{r}} = \sqrt{X^2 + Y^2 + Z^2}$$

$$v = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

$$\text{Radial velocity } v_r = \frac{\mathbf{r} \cdot \mathbf{v}}{r}$$

$$\hat{\mathbf{h}} = \mathbf{r} \times \mathbf{v} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ X & Y & Z \\ v_x & v_y & v_z \end{vmatrix}$$

$$h = \sqrt{\hat{\mathbf{h}} \cdot \hat{\mathbf{h}}}$$

$$\hat{\mathbf{N}} = \hat{\mathbf{K}} \times \hat{\mathbf{h}} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 0 & 0 & 1 \\ h_x & h_y & h_z \end{vmatrix}$$

$$N = \sqrt{\hat{\mathbf{N}} \cdot \hat{\mathbf{N}}}$$

$$\mathbf{e} = \frac{1}{\mu}\left[\left(v^2 - \frac{\mu}{r}\right)\mathbf{r} - r v_r \mathbf{v}\right]$$

$$e = \sqrt{\mathbf{e} \cdot \mathbf{e}}$$

$$i = \cos^{-1}\left(\frac{h_z}{h}\right)$$

$$\Omega = \begin{cases} \cos^{-1}\left(\frac{N_x}{N}\right) & (N_Y \geq 0) \\ 360^\circ - \cos^{-1}\left(\frac{N_x}{N}\right) & (N_Y < 0) \end{cases}$$

$$\omega = \begin{cases} \cos^{-1}\left(\frac{\hat{\mathbf{N}} \cdot \mathbf{e}}{Ne}\right) & (e_z \geq 0) \\ 360^\circ - \cos^{-1}\left(\frac{\hat{\mathbf{N}} \cdot \mathbf{e}}{Ne}\right) & (e_z < 0) \end{cases}$$

$$\theta = \begin{cases} \cos^{-1}\left(\frac{\mathbf{e} \cdot \mathbf{r}}{er}\right) & (v \geq 0) \\ 360^\circ - \cos^{-1}\left(\frac{\mathbf{e} \cdot \mathbf{r}}{er}\right) & (v < 0) \end{cases}$$

$$a = \frac{r(1 + e\cos(\theta))}{1 - e^2}$$

A.2 Keplerian to Cartesian

Similarly from the transformation from Cartesian to Keplerian, the reverse is also true. In this case we can also take advantage of the Perifocal frame in order to simplify the transformations. In this case we will go from the planets grid and apply rotation matrices for the Ω, i, and ω using rotational matrices ${}^{EC}\omega^{EC^1}$, ${}^{EC^1}\omega^{EC^{11}}$, and ${}^{EC^{11}}\omega^{P}$ respectively.

$${}^{EC}\omega^{EC^1} = \begin{bmatrix} \cos(\Omega) & \sin(\Omega) & 0 \\ -\sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{EC^1}\omega^{EC^{11}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & \sin(i) \\ 0 & -\sin(i) & \cos(i) \end{bmatrix}$$

$$^{EC^{11}}\omega^P = \begin{bmatrix} \cos(\omega) & \sin(\omega) & 0 \\ -\sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We now multiply through the rotations matrices from ECI to P (Ω, i, ω) in order to obtain the full transform for ECI to P, , $^{EC}\omega^P$. Because all three rotational matrices are orthogonal we can take the transpose of the full transformation matrix to obtain P to ECI.

$$^{EC}\omega^P = \begin{bmatrix} \cos(\Omega)\cos(\omega) - \sin(\Omega)\cos(i)\sin(\omega) & \sin(\Omega)\cos(\omega) + \cos(\Omega)\cos(i)\sin(\omega) & \sin(i)\sin(\omega) \\ -\cos(\Omega)\sin(\omega) - \sin(\Omega)\cos(i)\cos(\omega) & \cos(\Omega)\cos(i)\cos(\omega) - \sin(\Omega)\sin(\omega) & \cos(\omega)\sin(i) \\ \sin(\Omega)\sin(i) & -\cos(\Omega)\sin(i) & \cos(i) \end{bmatrix}$$

$$^{P}\omega^{EC} = \begin{bmatrix} \cos(\Omega)\cos(\omega) - \sin(\Omega)\cos(i)\sin(\omega) & -\cos(\Omega)\sin(\omega) - \sin(\Omega)\cos(i)\cos(\omega) & \sin(\Omega)\sin(i) \\ \sin(\Omega)\cos(\omega) + \cos(\Omega)\cos(i)\sin(\omega) & \cos(\Omega)\cos(i)\cos(\omega) - \sin(\Omega)\sin(\omega) & -\cos(\Omega)\sin(i) \\ \sin(i)\sin(\omega) & \cos(\omega)\sin(i) & \cos(i) \end{bmatrix}$$

At this point it is now necessary to find the $^{P_0}\mathbf{r}^{SC}$ and $^{P_0}\mathbf{v}^{SC}$ vectors in the Perifocal frame, and then we can use the transform matrix $^{P}\omega^{ECI}$ to find the reference system Cartesian coordinates and velocity.

$$\mathbf{r} = \frac{h^2}{\mu}\frac{1}{1 + e\cos(\theta)}\begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix}$$

$$\mathbf{v} = \frac{\mu}{h}\begin{bmatrix} -\sin(\theta) \\ e + \cos(\theta) \\ 0 \end{bmatrix}$$

$$[r_{EC}, v_{EC}]^T = [[^{P}\omega^{EC}][\mathbf{r}]^T, [^{P}\omega^{EC}][\mathbf{v}]^T]^T$$

A.3 Lambert's Problem

Lambert's problem is an important orbital mechanics problem which determines the orbit from two position vectors and a time of flight between points ${}^{EC_0}\mathbf{R}^A$ and ${}^{EC_0}\mathbf{R}^B$. The method for solving the problem is important in mission design, specifically in areas regarding targeting and fuel use optimization, due to finding necessary impulses to change trajectories to the target point in space. This is solved for our problem using the universal variable method.

A.4 Atmospheric Drag - Density

Density calculated from table below from [38]. The expeceted values are a reasonable com compare relatively well with the 1976 standard atmospheric model and provide an estimate for the density. This method was used in the design space exploration due to the ease of implementation, however will be replaced by a more numerically intensive model in the simulation model.The simple atmospheric model can calculate the estimated density by finding the correct base altitude, density and scale height and placing the correct values into equation A.1.

$$\rho_{altitude} = \rho_0 \cdot e^{\frac{h_0 - z}{H}} \tag{A.1}$$

A.5 E_bN_o Derivation

$$\frac{E_b}{N_o} = \frac{P\,L_l G_t L_s L_a G_r}{k T_s R} \tag{A.2}$$

Power flux of a sphere for an isotropic antenna can be found as

$$W_f = \frac{PL_l}{4 \quad r}$$

Table A.1: Simple Atmospheric Model Data from [38]

Altitude Z (km)	Base Altitude h_0 (km)	Density $\rho_0(\frac{kg}{m^3})$	Scale Height H (km)
0	0.00	1.23	7.25
25	25.00	3.899e-2	6.35
30	30.00	1.774e-2	6.68
40	40.00	3.972e-3	7.55
50	50.00	1.057e-3	8.38
60	60.00	3.206e-4	7.71
70	70.00	8.770e-5	6.55
80	80.00	1.905e-5	5.80
90	90.00	3.396e-6	5.38
100	100.00	5.297e-7	5.88
110	110.00	9.661e-8	7.26
120	120.00	2.438e-8	9.47
130	130.00	8.484e-9	12.64
140	140.00	3.845e-9	16.15
150	150.00	2.070e-9	22.52
180	180.00	5.464e-10	29.74
200	200.00	2.789e-10	37.11
250	250.00	7.248e-11	45.55
300	300.00	2.418e-11	53.63
350	350.00	9.518e-12	53.30
400	400.00	3.725e-12	58.52
450	450.00	1.585e-12	60.83
500	500.00	6.967e-13	63.82
600	600.00	1.454e-13	71.84
700	700.00	3.614e-14	88.67
800	800.00	1.170e-14	124.64
900	900.00	5.245e-15	181.05
1000	1000.00	3.019e-15	268.00

The transmitter gain is defined as the ratio of power at the center of the antenna coverage area compared to the theoretical omnidirectional antenna. Effective isotropic radiated power (EIRP) in watts is used to state the effective power of the antenna and is defined as

$$EIRP \overset{\text{def}}{=} P L_l G_t$$

which results in the effective power flux density of the antenna as

$$W_f = \frac{(EIRP)L_a}{4\pi R_r^2}$$

The received power is power flux of the transmitting antenna times the cross sectional area of the receiving antenna and the antenna efficiency.

$$C = W_f \frac{\pi D_r^2}{4} \eta_r = \frac{P L_l G_t L_a D_r^2 \eta_r}{16 R_r^2}$$

The recieving antenna also has a gain for a given wavelength, λ, which can be defined as

$$G_r = \frac{\pi D_r^2 \eta_r}{4} \frac{4\pi}{\lambda^2} = \frac{\pi^2 D_r^2 \eta_r}{\lambda^2}$$

The free space path loss (FSPL) is a loss value which is expressed as the reciprocal of gain and defined as

$$\left(\frac{4\pi R_r^2}{\lambda}\right)^2$$

This allows us to define space loss, L_s, as the inverse of FSPL and integrate it into the received power equation and now we see

$$C = P\, L_l G_t L_s L_a G_r = (EIRP) L_s L_a G_r$$

In dB form the energy per bit to noise density ratio can be written as

$$\frac{E_b}{N_o} = P + L_l + G_t + L_{pt} + L_{pr} + L_s + L_a + G_r + 228.6 - 10 log T_s - 10 log R$$

with P in dBW, T_s in K, R in bps, and 10 log k = $-228.6\ dBW/(Hz \cdot K)$, and the rest of the terms in dB.

Appendix B

REQUIREMENTS DOCUMENT

Generic Satellite
Project Requirements Document
DRAFT 1.1

26-Oct-17

Requirements are grouped by WBS level and contained in separate tab sheets below. The Program Level 1 requirements are also provided in a separate tab for reference.
Trac, allocation and V&V method is shown for each requirement. V&V methods are:
T=Test, D=Demonstration, A=Analysis, I=Inspection

RQMT ID	REQUIREMENT	TRACEABILITY	ALLOCATION	Verification Method				Verification Document(s)	Rationale/Comment
				T	D	A	I		
1.0	GEN SAT MISSION OBJECTIVE								
L1-PTD-01	The purpose of the GenSat is to demonstrate a semi-reconfigurable satelite which can demonstrate new MMR technologies.		Project-wide						

RQMT ID	REQUIREMENT	TRACEABILITY	ALLOCATION	Verification				Verification Document(s)	Rationale/Comment
				T	D	A	I		
2.0	Mission and Technology Requirements								
2.1	Mission Hardware								
2.2	Mission Operations								
2.3	Enviornmental Reqiurements and Launch Provisions								
2.4	Mission and Satellite Reliability								
2.5	Payload and Spacecrat Disposal Orbit Debris, Re-entry Limits								

RQMT ID	REQUIREMENT	TRACEABILITY	ALLOCATION	Verification				Verification Document(s)	Rationale/Comment
				T	D	A	I		
3.0	Payload Requirements								
3.1	Payload Requirements								
3.1.1	The Payload shall not exceed the total volume defined in the Spacecraft to Payload ICD.		Payload			X	X		The exact volume and locations will be negotiated at formation of the ICD.
3.1.2	The Payload developer shall document the measured thrust direction, variability, and location of thrust relative to reference system as documented in the Spacecraft to Payload ICD.		Payload	X		X			Document forces that will be produced by the Payload thrust.
3.1.3	The Payload shall conform to CG and total mass limits specified in the Spacecraft to Payload ICD.		Payload			X	X		Payload mass allocation and CG issues will be negotiated at ICD formation
3.1.4	The Payload shall be capable of operating from spacecraft-supplied power as specified in the Spacecraft to Payload ICD.		Payload	X		X			The exact voltage and whether unregulated or regulated will be negotiated at formation of the ICD.
3.1.5	The Payload shall be configured to communicate signals and data to the Flight System as specified in the Spacecraft to Payload ICD.		Payload	X		X			Most likely RS-422 asynchronous
3.1.6	The Payload shall support a 180 day mission lifetime on orbit.		Payload			X			Operation on orbit could be as long as 180 days. Not meant to infer a 180 day continuous firing capability, but that the payload could be utilized off and on during a 180 day period.
3.1.7	The Payload shall be thermally controlled as specified in the Spacecraft to Payload ICD.		Payload	X		X			The intent of this requirement is to have the Flight system provide some thermal control support for the Payload.
3.1.9	The Payload shall be capable of being powered off for launch and powered on by the Spacecraft at any time during the flight.		Payload			X			Power supply interface from Payload will be switched
3.1.10	The Payload shall provide harnesses and cabling for the Payload System as specified in the Spacecraft to Payload ICD.		Payload	X					The definition of the exact cable/harness interface, as well as cable routing, will be negotiated in the Spacecraft to Payload ICD
3.1.11	The Payload shall be responsible for supplying mounting structures as specified in the Spacecraft to Payload ICD.		Payload				X		Exact interface definition will be negotiated and documented in a Spacecraft to Payload ICD.
3.1.12	The Payload shall provide a simulator of the Payload to support System Integration testing.		Payload		X		X		Support electrical, command and control interfaces to allow for integration testing .
3.1.13	The Payload shall be able to acquire Health and Status information from its instruments.		Payload	X					Necessary to monitor status of subsystem and for diagnostics.
3.1.14	The Payload shall be able to provide Health and Status information for the Payload to the Spacecraft as specified in the Spacecraft to Payload ICD.		Payload, Vendor	X					Necessary to send H&S information to the GDS for analysis and review.
3.1.15	The Payload shall provide engineering telemetry sufficient for MOC to determine the quantity of propellant remaining to within 5% (TBR).		Payload, Vendor, MOC			X			Provide data to determine propellant level, e.g. firing times that can be correlated with lookup tables on the ground.
3.1.16	The Payload shall provide engineering telemetry sufficient for MOC to determine the thruster performance ISP within 5% (TBR).		Payload, Vendor			X			Provide data to model thruster performance to calculate future thrust command durations. The Spacecraft provides engineering telemetry to measure orbit change. Engineering telemetry may be combined with lookup tables or ground analysis to determine actual ISP, the requirement is to make sure whatever data is needed on thruster operation is provided.

3.1.17	The Payload shall provide a safe plug to inhibit unsafe operation on the ground per the electrical interface in the Spacecraft to Payload ICD.		Payload, Vendor	X			X		S&MA safety requirement (and probably also the LV) to physically prohibit a thruster being activated.
3.1.18	Payload bus solution shall provide Payload electrical interfaces as specified in the Spacecraft to Payload ICD.		Payload, Vendor	X		X			
3.1.19	Payload bus solution shall provide Payload mechanical interfaces as specified in the Spacecraft to Payload ICD.		Payload, Vendor	X		X			
3.1.20	Payload bus solution shall provide Payload thermal interfaces as specified in the Spacecraft to Payload ICD.		Payload, Vendor	X		X			
3.1.21	Payload bus solution shall provide Payload communication interfaces as specified in the Spacecraft to Payload ICD.		Payload, Vendor	X		X			
3.1.22	Payload shall provide and EIDP upon delivery containing: •Propulsion System Specs •Propulsion System Drawings (MICD/EICD) •Propulsion Assembly/Handling Procedures •Propulsion System Limits/Constraints •Propulsion System Test Data and Scripts •Propulsion System CMD/TLM Requirements •Propulsion System Acceptance Test Procedures		Payload				X		
3.1.23	The Payload shall provide EMI/EMC test data and analysis (TBD).			X	X				Need to define specifics. Looking for self-compatible, measured output provided to SC vendor, and documenting any particular sensitivities that may disrupt thruster operation.
3.1.24	The Payload shall be shipped to the spacecraft vendor appropriately packaged in a shipping container.		Payload				X		
3.1.25	The Payload developer will provide any required GSE necessary for I&T and/or LV integration of the Payload.		Payload				X		

RQMT	REQUIREMENT	TRACEABILITY	ALLOCATION	Verification				Verification Document(s)	Rationale/Comment
ID				T	D	A	I		
4.0	Bus Functional Requirements								
4.1	Attitude Determination and Control								
4.1.1	GenSat shall be capable of producing tracking and ephemeris solutions based on collected GPS telemetry from the spacecraft.		MOC	X		X			Use of GPS locations and timestamps from sub MEO altitudes in orbit can be used to determine the orbital elements of the satellite.
4.1.2	GenSat shall be capable of selectively transmitting stored telemetry upon command of the MOC.		Vendor	X					
4.2	Propulsion Management and Control								
4.2.1	The GenSat project team shall provide a flatsat demonstration unit with the ability to demonstrate MMR technologies at the hardware, data bus, and software levels		Project-wide		X		X		
4.3	Command and Data Handling								
4.3.1	GenSat bus shall provide bidirectional communication with the Payload		Vendor						
4.3.2	GenSat shall be capable of providing telemetry defining the state of any quantity or function monitored or capable of being updated by the FSW.		Vendor			X			
4.3.3	GenSat shall monitor SC elements necessary for the MOC to assess the state of the spacecraft.		Vendor			X			
4.3.4	GenSat shall be capable of transmitting any or all of its memory contents on command by MOC.		Vendor	X		X			
4.3.5	All clocks shall be synchronized within less than TBD seconds. Time slips and timestamp errors due to CPU interrupts or other sources shall be corrected or prohibited by design.		Vendor	X					
4.4	Condition Sampling and Reporting Functions								
4.4.1	GenSat shall monitor temperatures of critical subsystem components throughout the mission. Temperature sampling and storage shall occur at least once per hour following on-orbit deployment.			X					
4.4.2	Current from the Payload, Main Transponder, and Beacon Transmitter subsystems shall be recorded at least 3 times per orbit			X					
4.4.3	Accuracy current of recording shall be no less than 1/50 the expected peak value.			X					
4.4.4	The voltage of the satellite's batteries shall be recorded at least once per minute following deployment for both sunlight and eclipse phases and during any pre-launch checkout operations.			X					
4.5	Communications								
4.5.1	The Spacecraft shall be capable of radiating telemetry to the ground at a rate and duration able to downlink 250Mbits per day.		Vendor	X		X			

4.5.2	The Spacecraft shall be capable of radiating telemetry to the ground with greater than 3db link margin.	Vendor	x		x		
4.5.3	GenSat shall radiate telemetry on a specified S-band frequency.	Vendor	x				
4.5.4	The Spacecraft shall be capable of maintaining TLM transmission while performing payload system characterization.	Vendor			x		
4.5.5	The Spacecraft shall be capable of operating payload and transmitting data at full rate during eclipses up to 20min.	Vendor	x		x		
4.6	Fault Detection Isolation and Recovery						
4.6.1	After a power outage and restart, the PharmaSat Bus shall be capable of returning to its previous operating state and resuming normal operations without loss of previously stored mission data.	Project-wide		X			
4.6.2	Spacecraft subsy7stem shall be capable of isolating subsystems from controllable critical faults (CCF) in a single subsystem are resetable fauts in orbit and have potentially catastrophic effects on system performance	Project-wide		X			May include Single event latchups, faults in operating program memory, system overtempretures, excessive power utilization
4.6.3	Single Event Upsets of dynamic memories (RAM) shall be detected and corrected. Memory errors shall be removed and any related system function should be fully recoverable.	Project-wide		X			
4.6.4	GenSat shall use an error-correcting memory solution to help ensure memory integrity.	Vendor	x		x		
4.7	Power Management Functions						
4.7.1	The power generation function shall be capable of providing one-time system power surge of up to 5 watts during the eclipse time of the first orbit.	Vendor		X		X	
4.7.2	The GenSat shall provide protection, battery charging and conditioning, and temperature control for the batteries to ensure acceptable battery charging and discharging cycle life throughout the mission timeline.	Vendor		X		X	
4.8	Thermal Control						
4.8.1	The GenSat project team shall provide a flatsat demonstration unit with the ability to demonstrate MMR technologies at the hardware, data bus, and software levels	Project-wide		X		X	

Appendix C

BACKUP GRAPHS, CHARTS AND IMAGES, AND MATLAB PUBLISHED CODE

Contents

```
clc; clear all; close all;
%RXWHEEL DATA: Mass     Power   Xdim    Ydim    Zdim    Peak Torque     Momentum Capacity       Peak Power
RxWheel = [0.96 0.65    109     109     101     0.011   0.42    10;
2.6     1.2     131     131     120     0.11    1.5     113;
0.185   0.3     50      50      40      0.002   0.04    1.8;
0.13    0.6     42      42      19      0.004   0.015   1;
0.24    0.5     58      58      25      0.007   0.05    1;
0.35    0.5     70      70      7       0.007   0.1     1;
0.226   0.9     77      65      38      0.02    0.18    23.4];
```

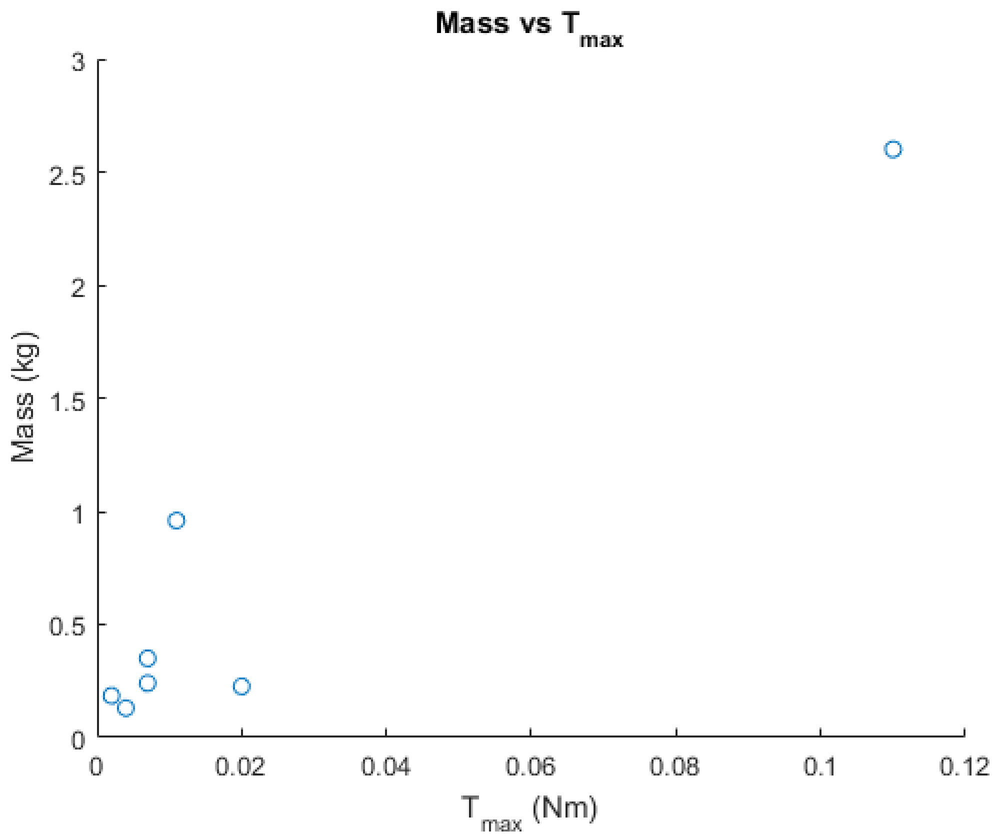
Define Custom Fit Functions

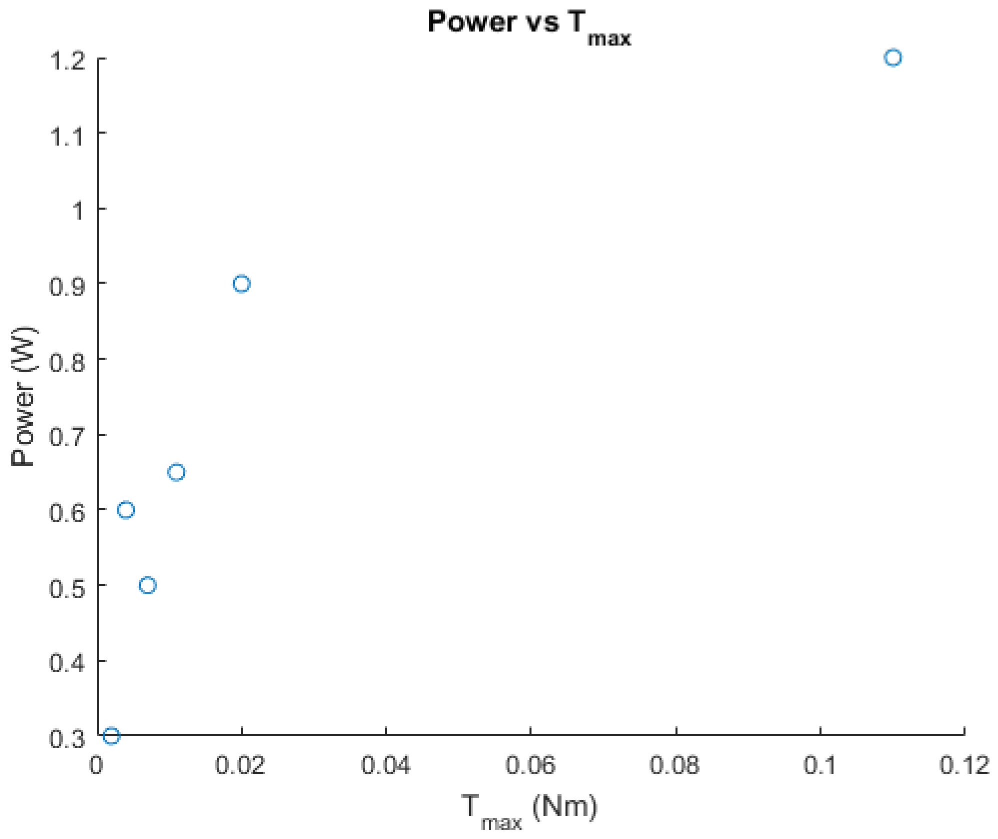
```
LogFit = fittype('a*log(x) + b','dependent',{'y'},'independent',{'x'},...
    'coefficients',{'a','b'})
ExpFit = fittype('a*exp(b*x)','dependent',{'y'},'independent',{'x'},...
    'coefficients',{'a','b'})
% poly1 = linear
% poly2 = quadratic
```

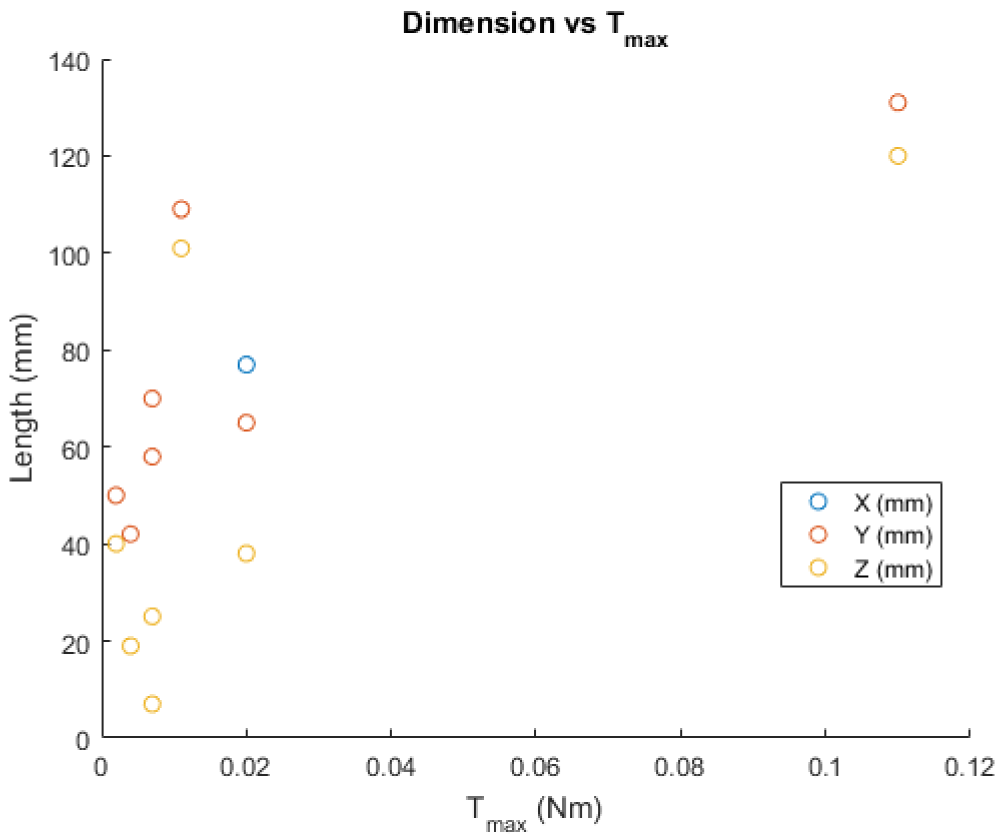
```
LogFit =

     General model:
     LogFit(a,b,x) = a*log(x) + b

ExpFit =

     General model:
     ExpFit(a,b,x) = a*exp(b*x)
```

Max Torque Data Scatter

```
%Plot Response vs Independent Variables
%Mass
figure
scatter(RxWheel(:,6),RxWheel(:,1))
title('Mass vs T_{max}')
xlabel('T_{max} (Nm)')
ylabel('Mass (kg)')
%Power
figure
scatter(RxWheel(:,6),RxWheel(:,2))
title('Power vs T_{max}')
xlabel('T_{max} (Nm)')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(RxWheel(:,6),RxWheel(:,3))
scatter(RxWheel(:,6),RxWheel(:,4))
scatter(RxWheel(:,6),RxWheel(:,5))
title('Dimension vs T_{max}')
xlabel('T_{max} (Nm)')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```

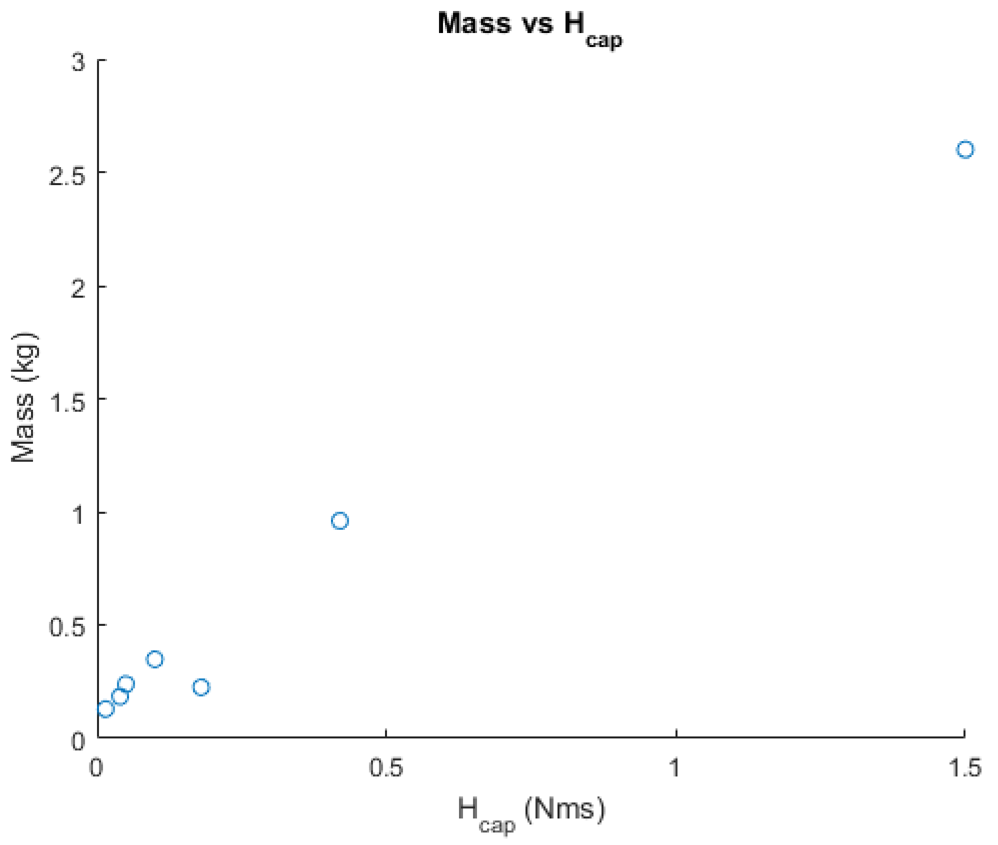

Mass vs T_{max}
Mass (kg)
0
0.5
1
1.5
2
2.5
3
0
0.02
0.04
0.06
0.08
0.1
0.12
T_{max} (Nm)

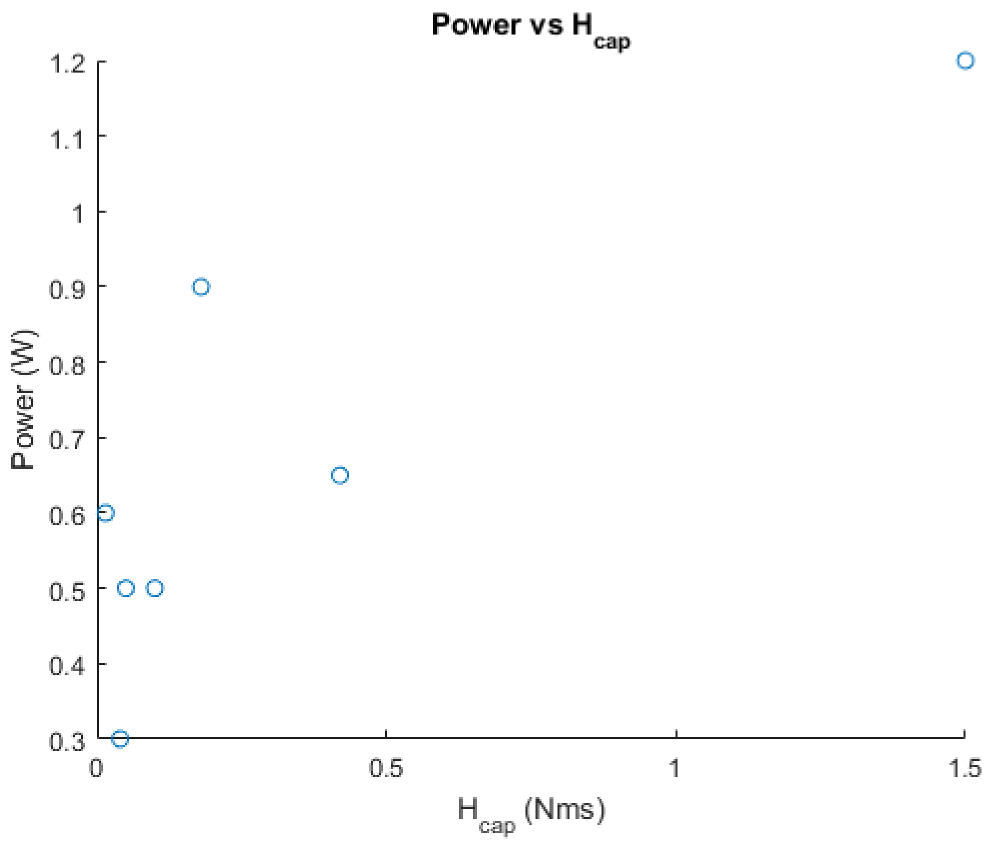

Power vs T_{max}
Power (W)
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
0
0.02
0.04
0.06
0.08
0.1
0.12
T_{max} (Nm)

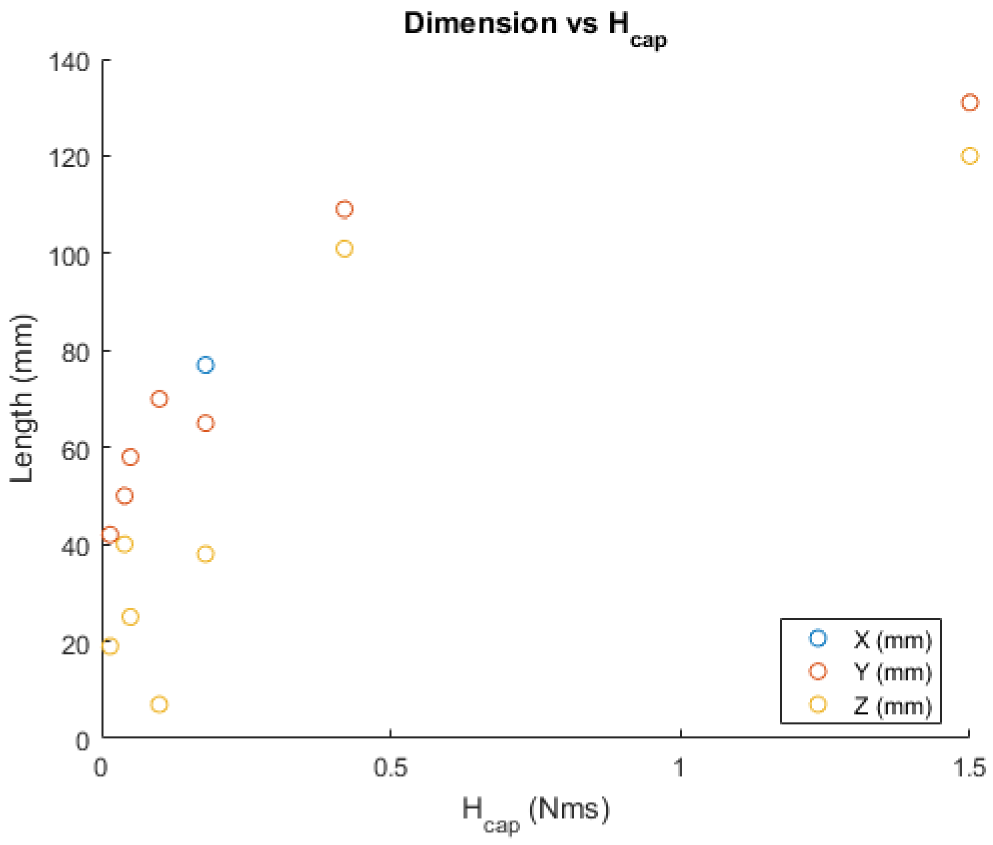


Max Momentum Capacity & Regression

```
%Mass
figure
scatter(RxWheel(:,7),RxWheel(:,1))
title('Mass vs H_{cap}')
xlabel('H_{cap} (Nms)')
ylabel('Mass (kg)')
%Power
figure
scatter(RxWheel(:,7),RxWheel(:,2))
title('Power vs H_{cap}')
xlabel('H_{cap} (Nms)')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(RxWheel(:,7),RxWheel(:,3))
scatter(RxWheel(:,7),RxWheel(:,4))
scatter(RxWheel(:,7),RxWheel(:,5))
title('Dimension vs H_{cap}')
xlabel('H_{cap} (Nms)')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```


Mass vs H_{cap}
Mass (kg)
H_{cap} (Nms)
0
0.5
1
1.5
2
2.5
3
0
0.5
1
1.5


Power vs H_{cap}
Power (W)
H_{cap} (Nms)
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
0
0.5
1
1.5



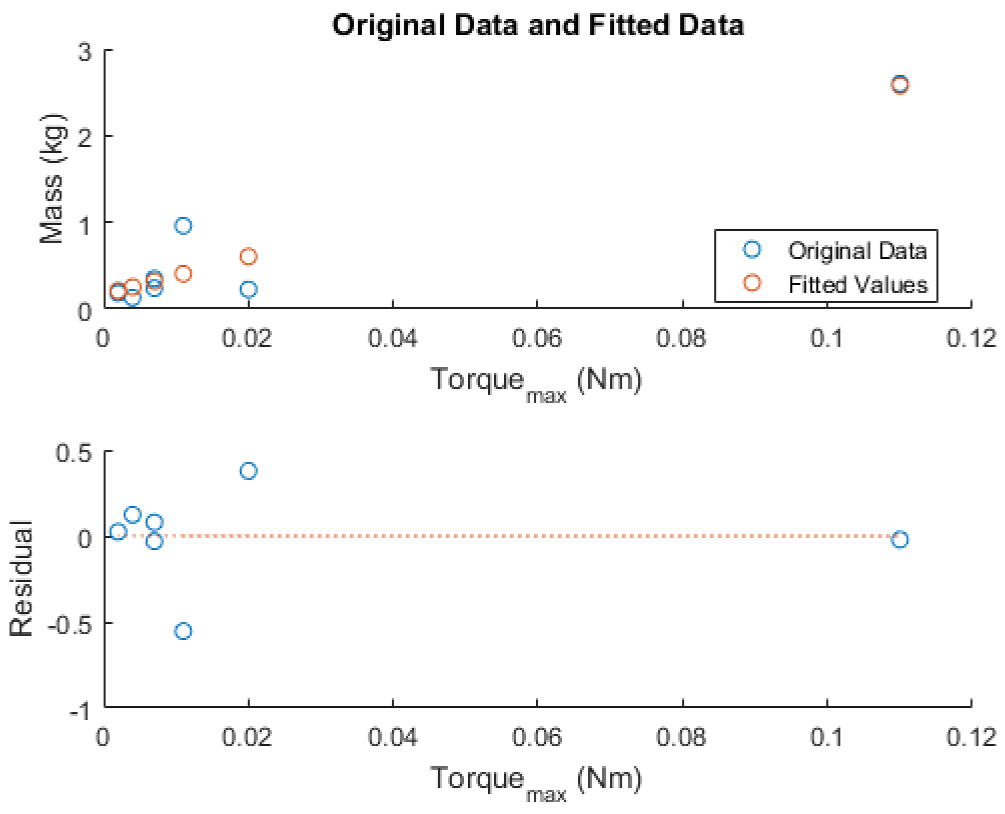
RxWheel Regression

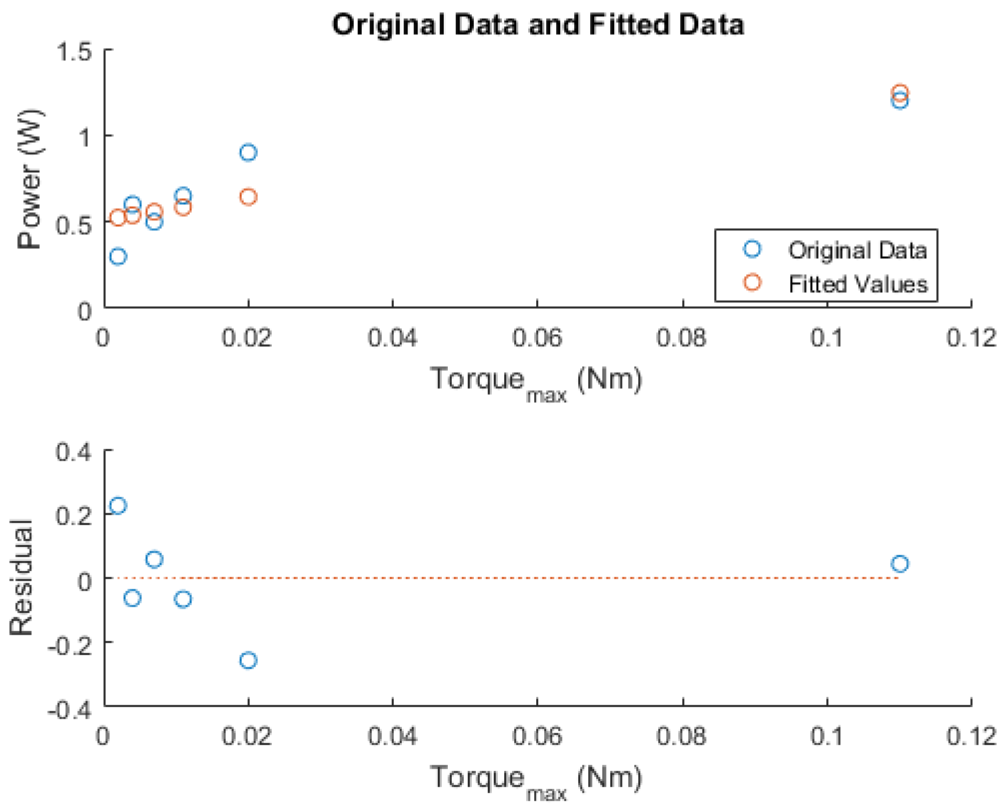
From the above graphs we see that the linear dimensions are logarithmic and the power and mass may or may not be linear or exponential.

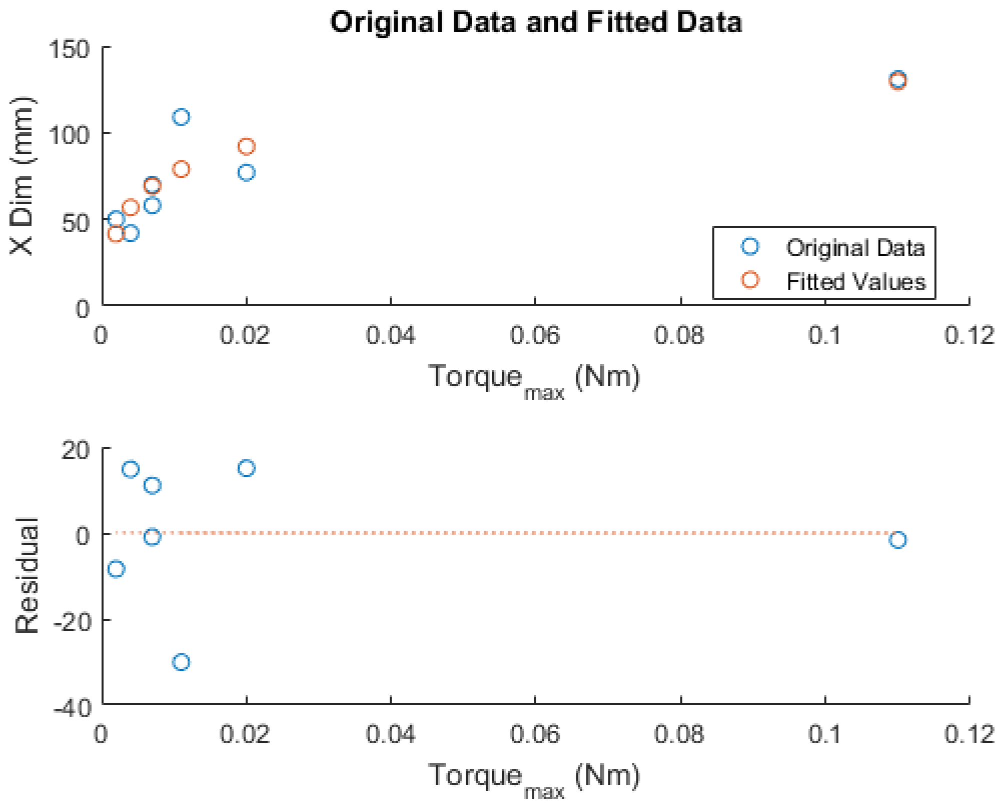
RxWheel Torque Regression Models

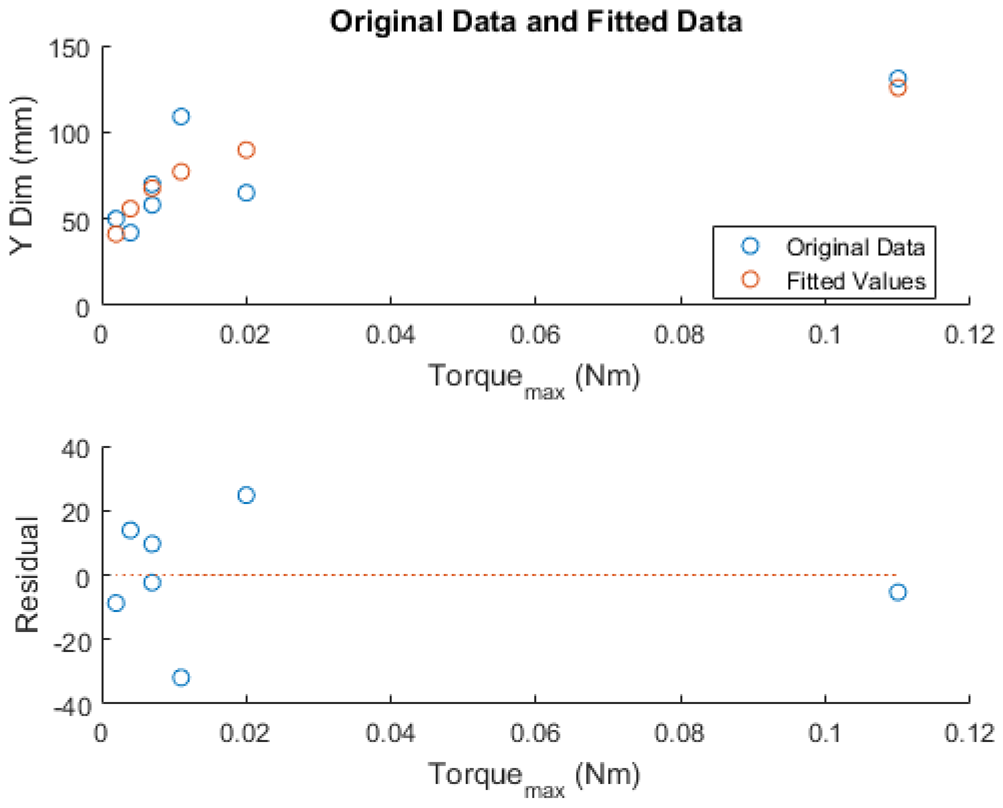
```
[MassT,MassTGOF] = fit(RxWheel(:,6),RxWheel(:,1),'poly1');
[PowerT, PowerTGOF] = fit(RxWheel(:,6),RxWheel(:,2),'poly1');
[XDimFitT, XDimGof] = fit(RxWheel(:,6),RxWheel(:,3),LogFit);
[YDimFitT, YDimGof] = fit(RxWheel(:,6),RxWheel(:,4),LogFit);
[ZDimFitT, ZDimGof] = fit(RxWheel(:,6),RxWheel(:,5),LogFit);
FittedValuesT = [MassT(RxWheel(:,6)), PowerT(RxWheel(:,6)),...
    XDimFitT(RxWheel(:,6)), YDimFitT(RxWheel(:,6)) ZDimFitT(RxWheel(:,6))];
Residuals = FittedValuesT - RxWheel(:,1:5);
FitResiduals(RxWheel(:,6),RxWheel(:,1),FittedValuesT(:,1),Residuals(:,1),...
    'title','Torque_{max} (Nm)','Mass (kg)')
FitResiduals(RxWheel(:,6),RxWheel(:,2),FittedValuesT(:,2),Residuals(:,2),...
    'title','Torque_{max} (Nm)','Power (W)')
FitResiduals(RxWheel(:,6),RxWheel(:,3),FittedValuesT(:,3),Residuals(:,3),...
    'title','Torque_{max} (Nm)','X Dim (mm)')
FitResiduals(RxWheel(:,6),RxWheel(:,4),FittedValuesT(:,4),Residuals(:,4),...
    'title','Torque_{max} (Nm)','Y Dim (mm)')
FitResiduals(RxWheel(:,6),RxWheel(:,5),FittedValuesT(:,5),Residuals(:,5),...
    'title','Torque_{max} (Nm)','Z Dim (mm)')
```

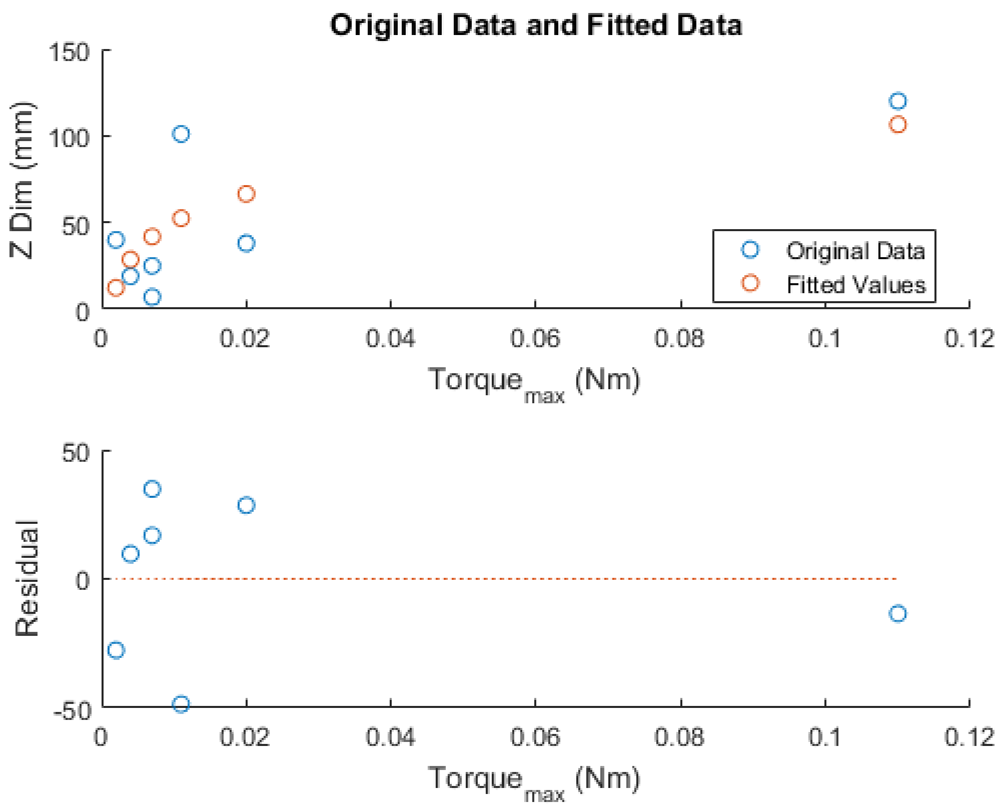
```
Warning: Start point not provided, choosing random start point.
Warning: Start point not provided, choosing random start point.
Warning: Start point not provided, choosing random start point.
```


Original Data and Fitted Data
Mass (kg)
Original Data
Fitted Values
Torque$_{max}$ (Nm)
Residual
Torque$_{max}$ (Nm)


Original Data and Fitted Data
Power (W)
Original Data
Fitted Values
Torque$_{max}$ (Nm)
Residual
Torque$_{max}$ (Nm)


Original Data and Fitted Data
X Dim (mm)
0
50
100
150
Original Data
Fitted Values
Torque$_{max}$ (Nm)
0
0.02
0.04
0.06
0.08
0.1
0.12
Residual
20
0
-20
-40
Torque$_{max}$ (Nm)

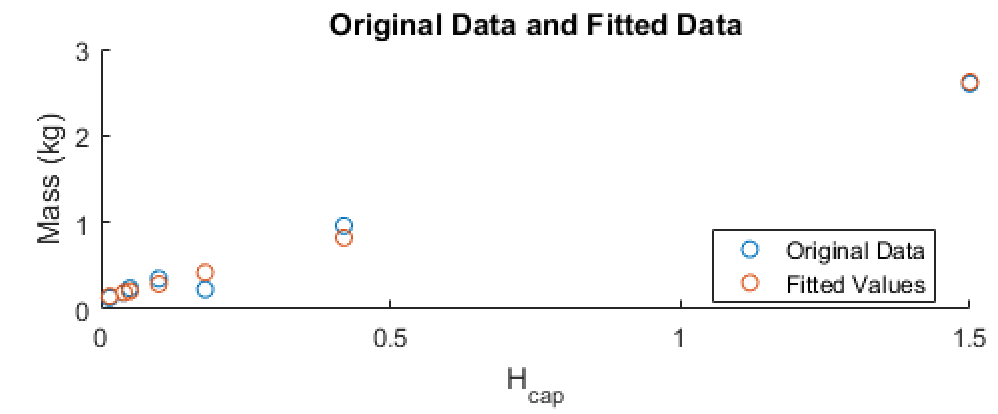

Original Data and Fitted Data
Y Dim (mm)
0
50
100
150
Original Data
Fitted Values
Torque$_{max}$ (Nm)
0
0.02
0.04
0.06
0.08
0.1
0.12
Residual
40
20
0
-20
-40
Torque$_{max}$ (Nm)

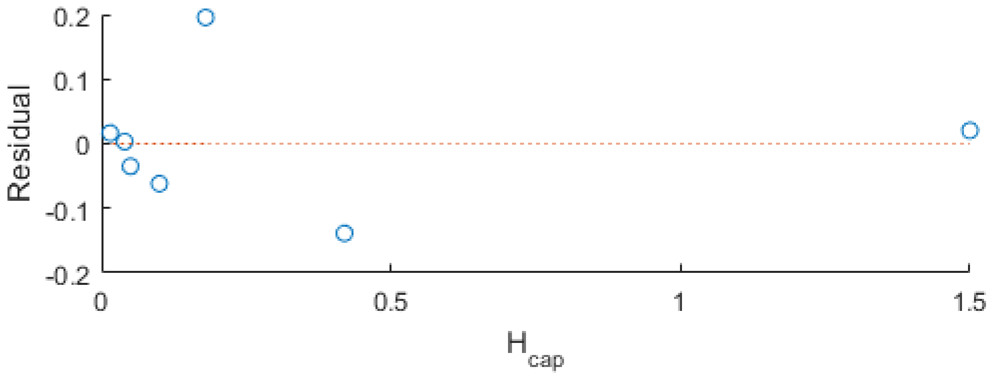


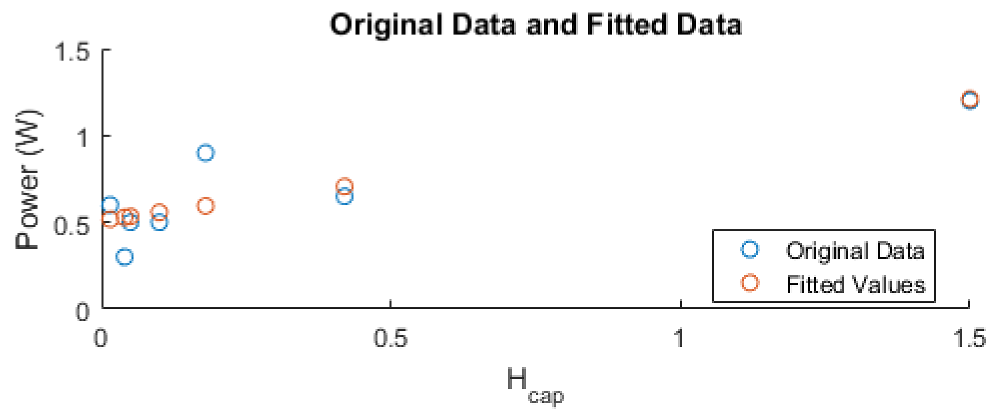
RxWheel Momentum Regression Models

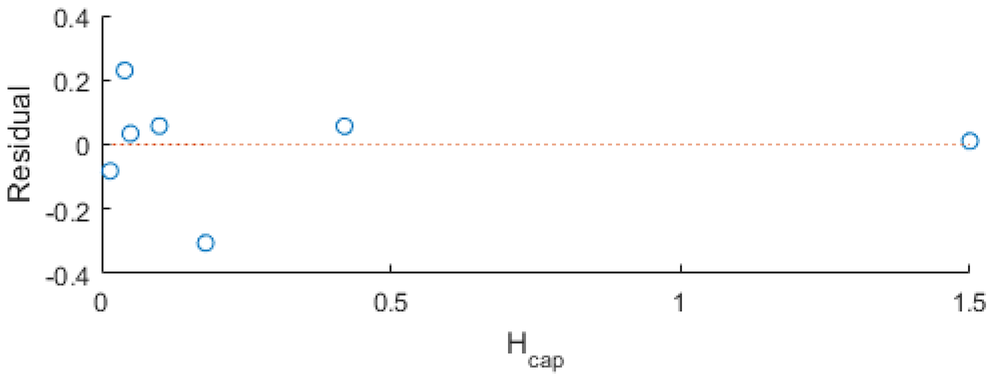
```
[MassH,MassTGOFH] = fit(RxWheel(:,7),RxWheel(:,1),'poly1');
[PowerH, PowerTGOF] = fit(RxWheel(:,7),RxWheel(:,2),'poly1');
[XDimFitH, XDimGOFH] = fit(RxWheel(:,7),RxWheel(:,3),LogFit);
[YDimFitH, YDimGOFH] = fit(RxWheel(:,7),RxWheel(:,4),LogFit);
[ZDimFitH, ZDimGOFH] = fit(RxWheel(:,7),RxWheel(:,5),LogFit);
FittedValuesT = [MassH(RxWheel(:,7)), PowerH(RxWheel(:,7)),...
    XDimFitH(RxWheel(:,7)), YDimFitH(RxWheel(:,7)) ZDimFitH(RxWheel(:,7))];
Residuals = FittedValuesT - RxWheel(:,1:5);
FitResiduals(RxWheel(:,7),RxWheel(:,1),FittedValuesT(:,1),Residuals(:,1),...
    'title','H_{cap}','Mass (kg)')
FitResiduals(RxWheel(:,7),RxWheel(:,2),FittedValuesT(:,2),Residuals(:,2),...
    'title','H_{cap}','Power (W)')
FitResiduals(RxWheel(:,7),RxWheel(:,3),FittedValuesT(:,3),Residuals(:,3),...
    'title','H_{cap}','X Dim (mm)')
FitResiduals(RxWheel(:,7),RxWheel(:,4),FittedValuesT(:,4),Residuals(:,4),...
    'title','H_{cap}','Y Dim (mm)')
FitResiduals(RxWheel(:,7),RxWheel(:,5),FittedValuesT(:,5),Residuals(:,5),...
    'title','H_{cap}','Z Dim (mm)')
```

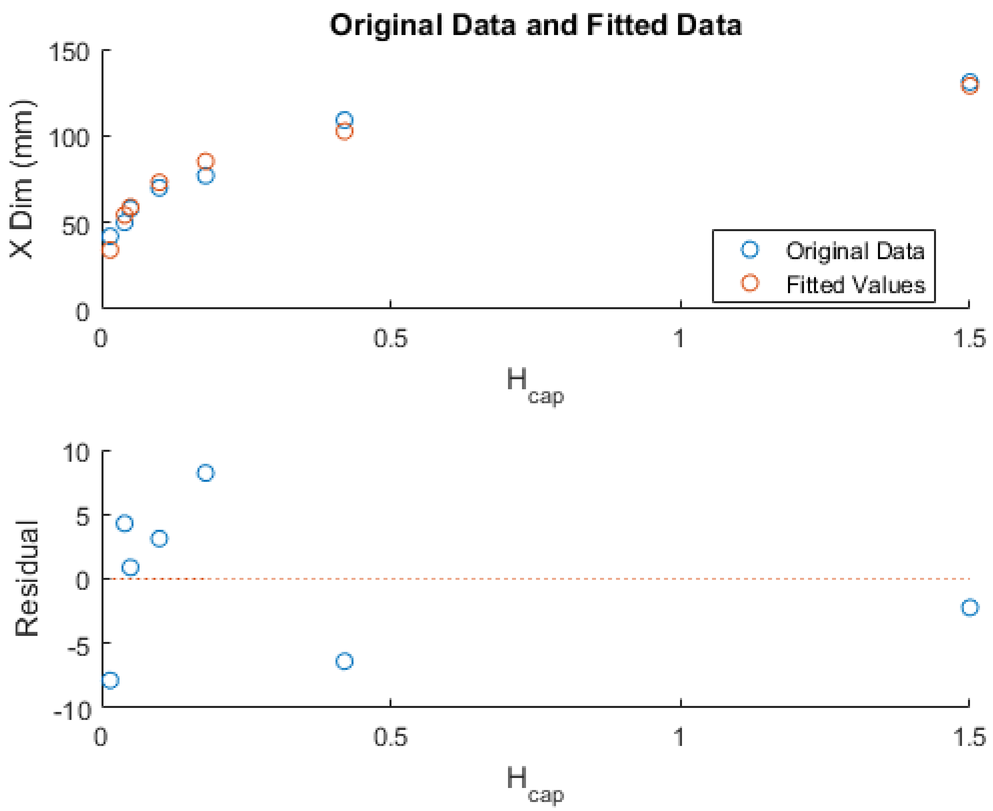
```
Warning: Start point not provided, choosing random start point.
Warning: Start point not provided, choosing random start point.
Warning: Start point not provided, choosing random start point.
```

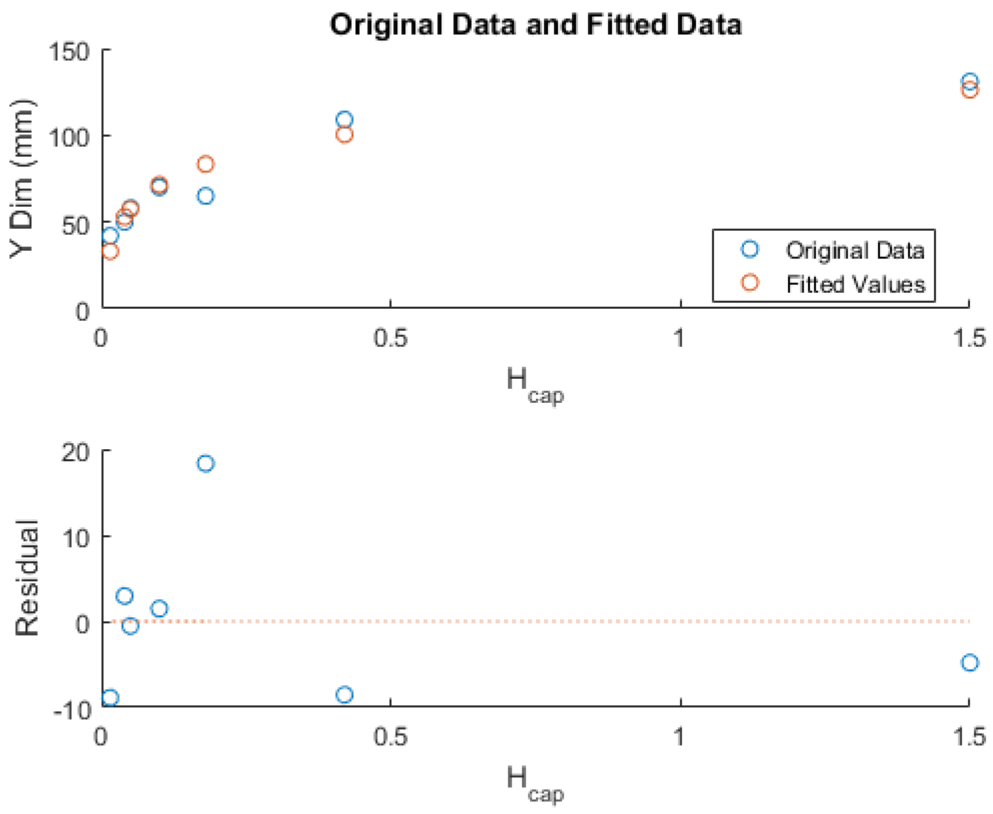

Original Data and Fitted Data
Mass (kg)
H_cap
Original Data
Fitted Values
0
0.5
1
1.5
0
1
2
3

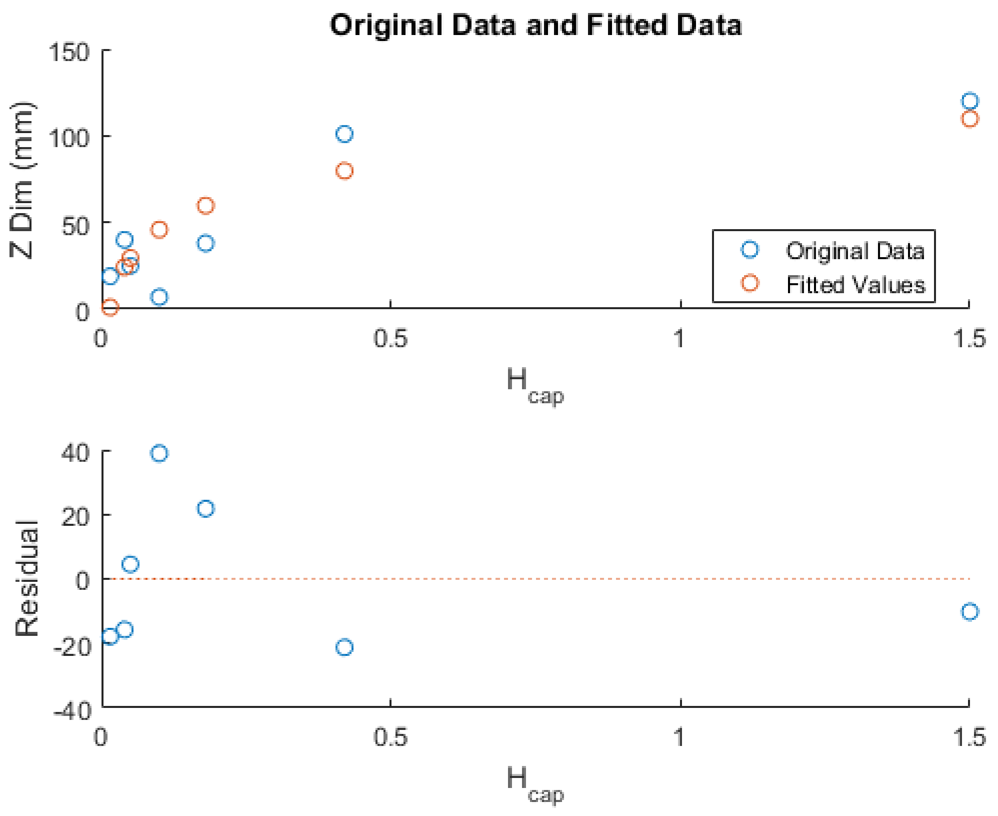

Residual
H_cap
0.2
0.1
0
-0.1
-0.2
0
0.5
1
1.5


Original Data and Fitted Data
Power (W)
H_cap
Original Data
Fitted Values
0
0.5
1
1.5
0
0.5
1
1.5


Residual
H_cap
0.4
0.2
0
-0.2
-0.4
0
0.5
1
1.5


Original Data and Fitted Data
X Dim (mm)
Original Data
Fitted Values
H_{cap}
Residual
H_{cap}


Original Data and Fitted Data
Y Dim (mm)
Original Data
Fitted Values
H_{cap}
Residual
H_{cap}



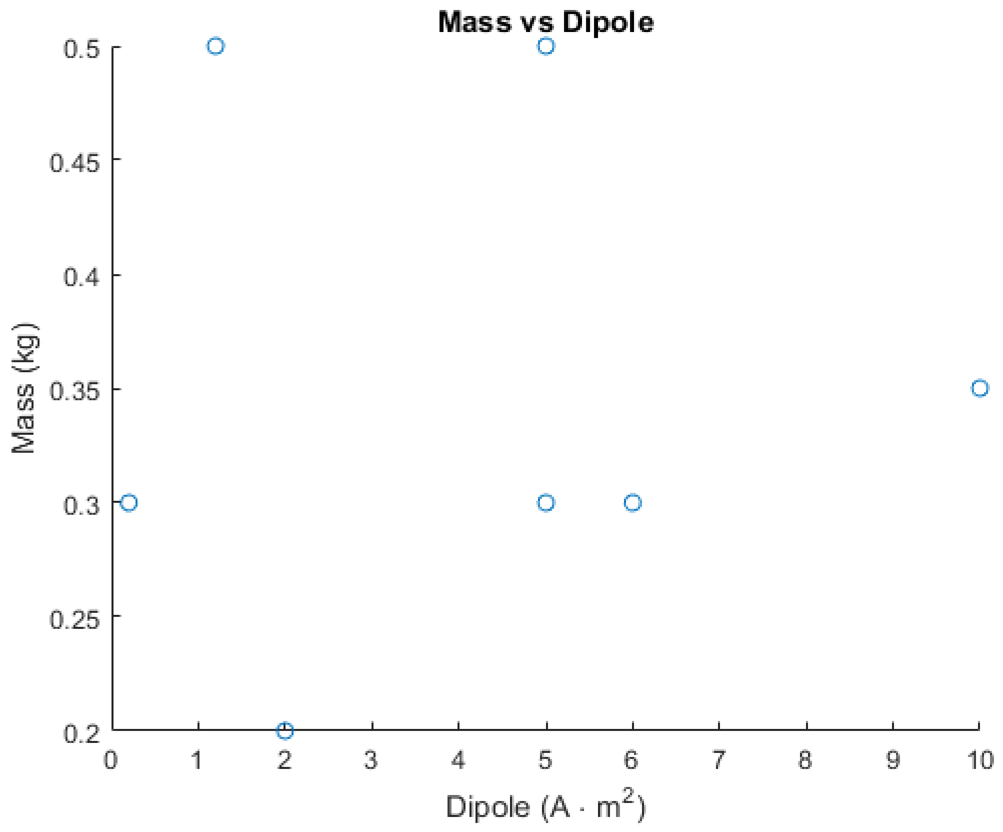
Magnetorquer Data

mass power x y z Dipole (A \cdot m^2)

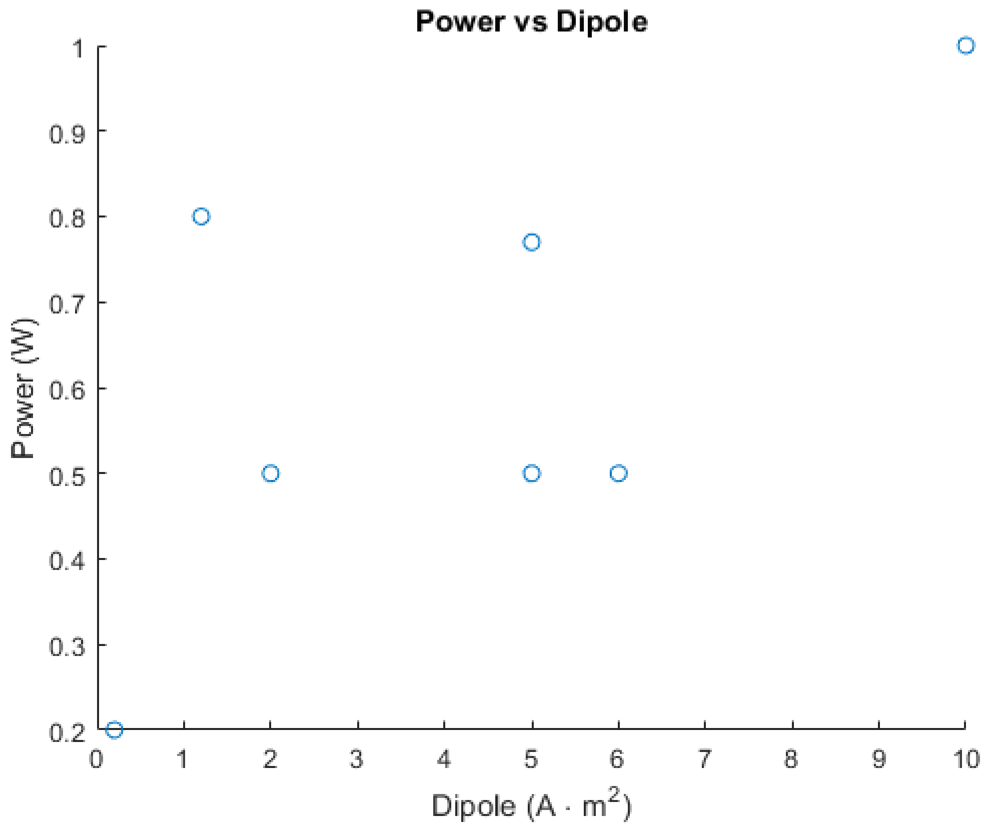
```
MGTQR = [0.5    0.5     66      252     39      5;
0.2     0.5     15      15      157.5   2;
0.3     0.77    18      18      240     5;
0.3     0.5     14.5    14.5    325     6;
0.35    1       17      17      330     10;
0.3     0.2     10 10 70        0.2;
0.5     0.8 15  13      94      1.2];
```

Magnetorquer Graphs (All Linear)

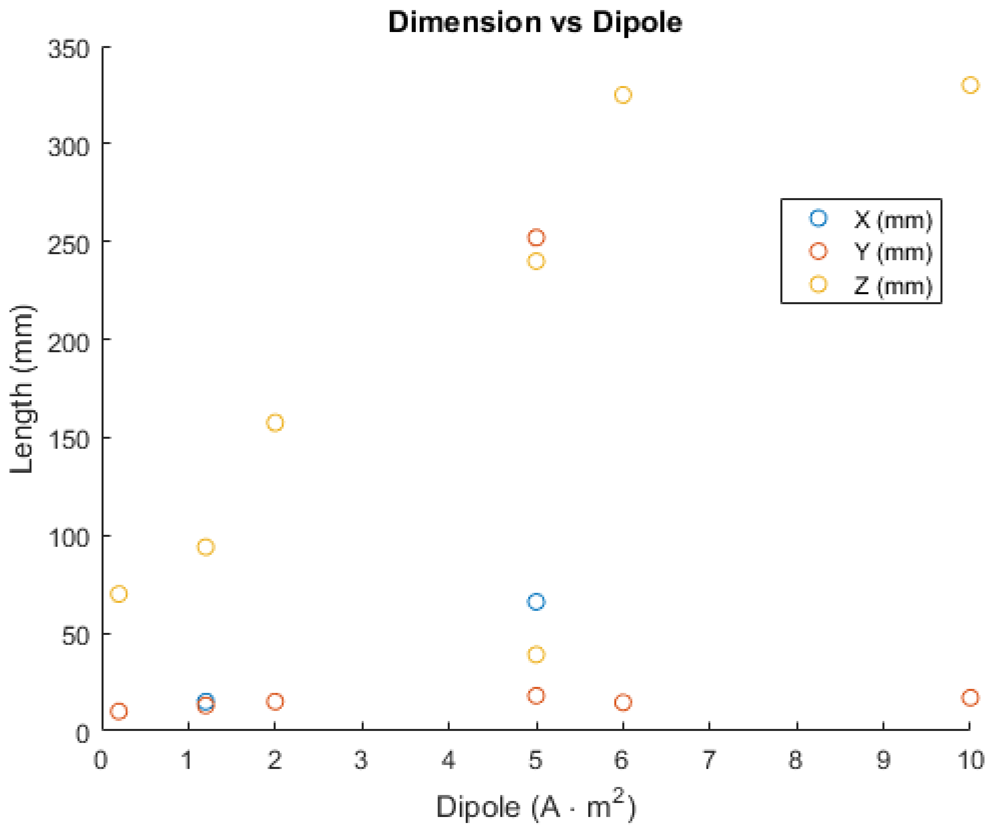
```
%Mass
figure
scatter(MGTQR(:,6),MGTQR(:,1))
title('Mass vs Dipole')
xlabel('Dipole (A \cdot m^2)')
ylabel('Mass (kg)')
%Power
figure
scatter(MGTQR(:,6),MGTQR(:,2))
title('Power vs Dipole')
xlabel('Dipole (A \cdot m^2)')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(MGTQR(:,6),MGTQR(:,3))
scatter(MGTQR(:,6),MGTQR(:,4))
scatter(MGTQR(:,6),MGTQR(:,5))
title('Dimension vs Dipole')
xlabel('Dipole (A \cdot m^2)')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```



Mass vs Dipole
Mass (kg)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
1
2
3
4
5
6
7
8
9
10
Dipole (A · m^2)

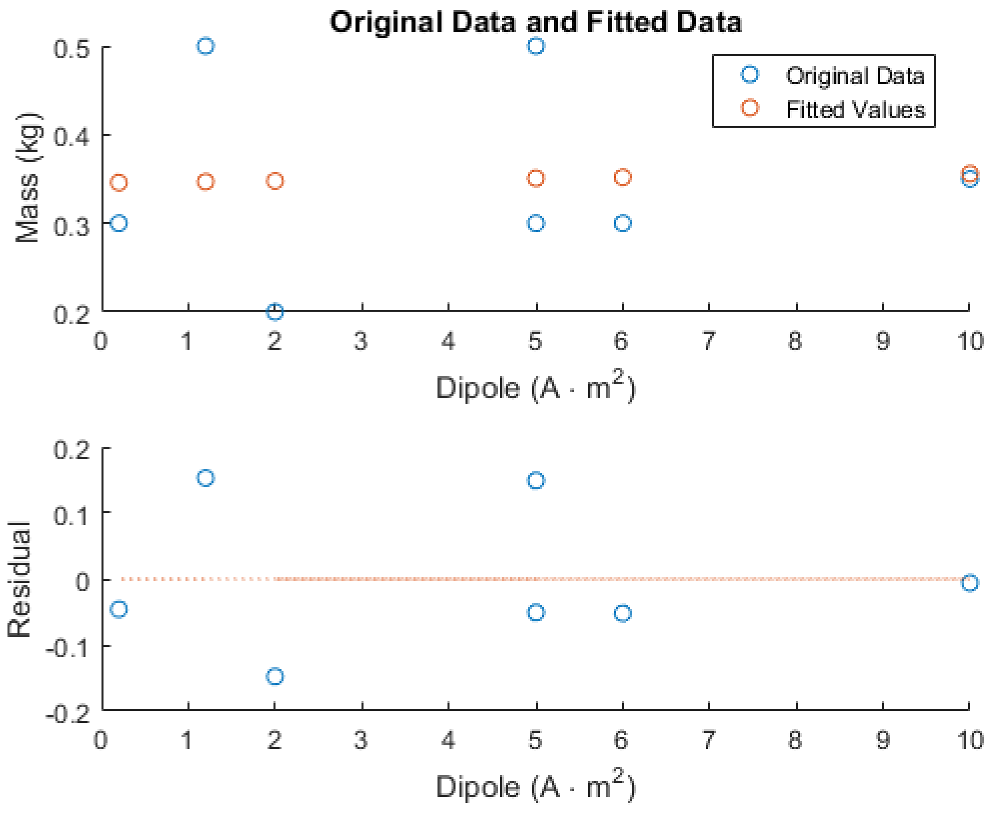


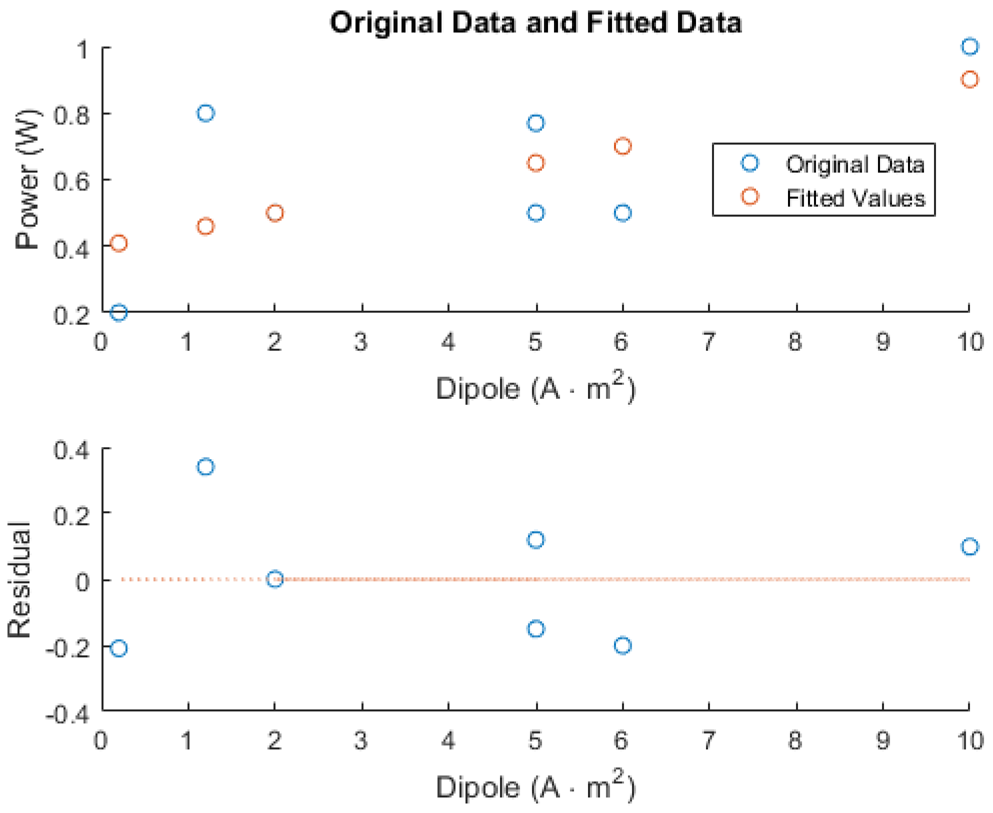
Power vs Dipole
Power (W)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
1
2
3
4
5
6
7
8
9
10
Dipole (A · m^2)

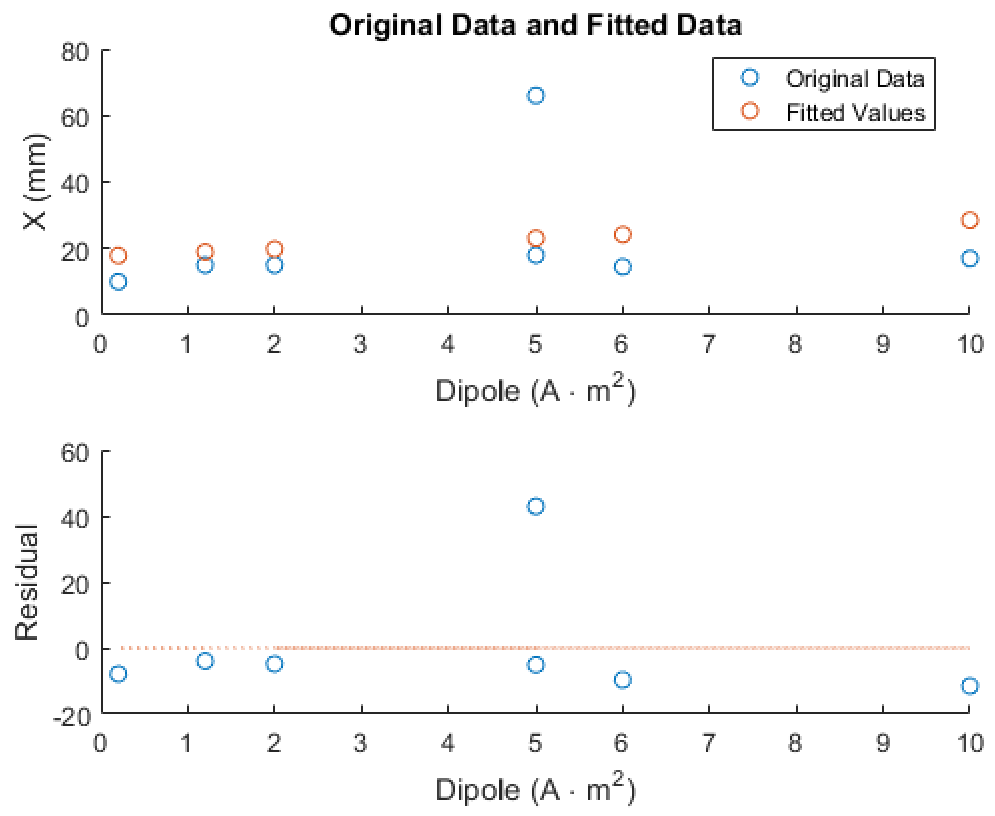


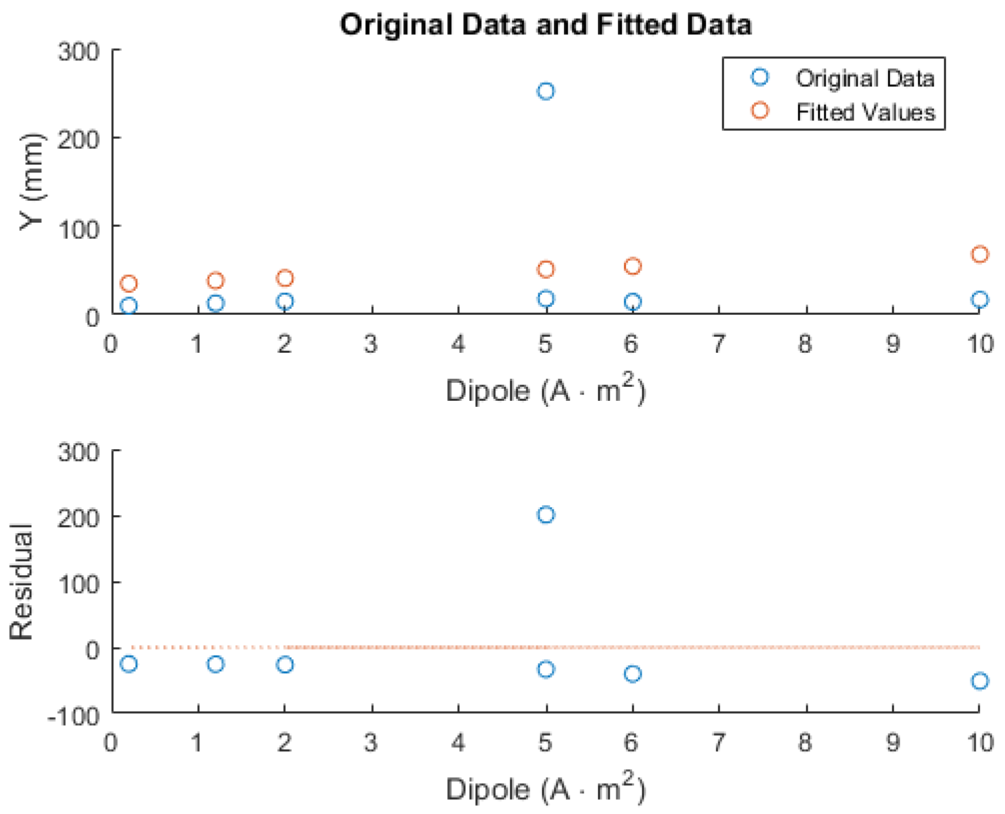
Magnetorquer Regression

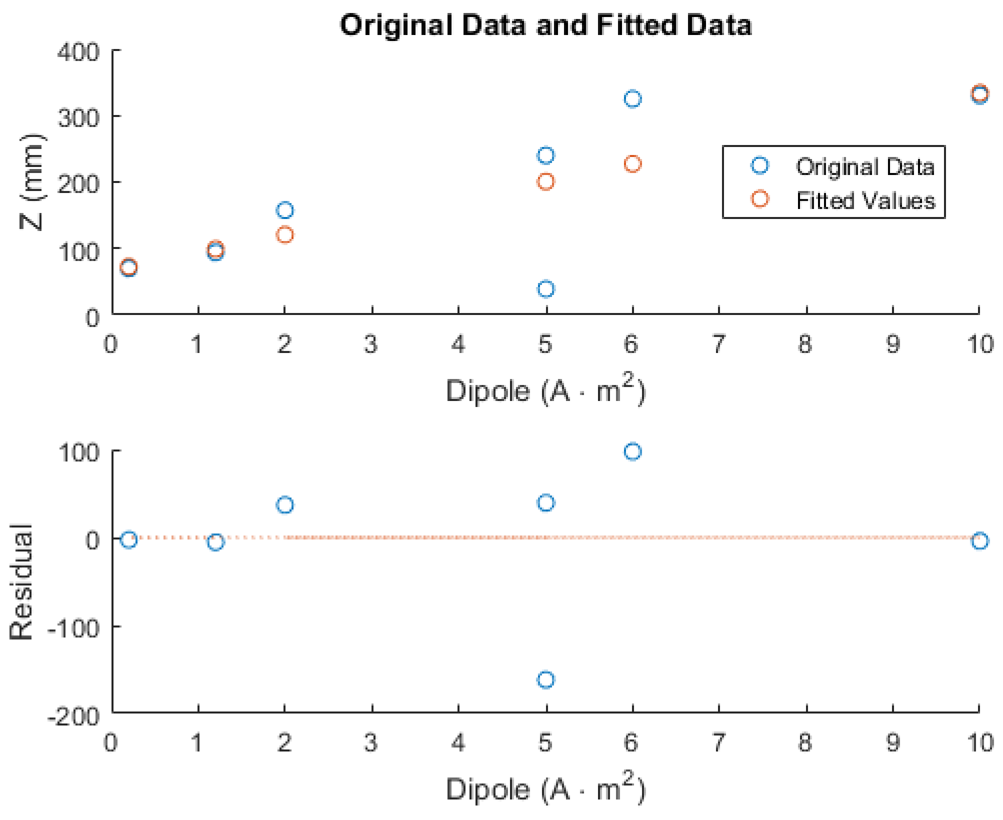
```
[MtqDFit, MtqDGOF] = fit(MGTQR(:,6),MGTQR(:,1),'poly1');
[MtqPFit, MtqPGOF] = fit(MGTQR(:,6),MGTQR(:,2),'poly1');
[MtqXFit, MtqXGOF] = fit(MGTQR(:,6),MGTQR(:,3),'poly1');
[MtqYFit, MtqYGOF] = fit(MGTQR(:,6),MGTQR(:,4),'poly1');
[MtqZFit, MtqZGOF] = fit(MGTQR(:,6),MGTQR(:,5),'poly1');
MDFitted = [MtqDFit(MGTQR(:,6)), MtqPFit(MGTQR(:,6)), MtqXFit(MGTQR(:,6)), ...
    MtqYFit(MGTQR(:,6)),MtqZFit(MGTQR(:,6))];
MDResiduals = MGTQR(:,1:5)-MDFitted;
FitResiduals(MGTQR(:,6),MGTQR(:,1),MDFitted(:,1),MDResiduals(:,1),...
    'title','Dipole (A \cdot m^2)','Mass (kg)')
FitResiduals(MGTQR(:,6),MGTQR(:,2),MDFitted(:,2),MDResiduals(:,2),...
    'title','Dipole (A \cdot m^2)','Power (W)')
FitResiduals(MGTQR(:,6),MGTQR(:,3),MDFitted(:,3),MDResiduals(:,3),...
    'title','Dipole (A \cdot m^2)','X (mm)')
FitResiduals(MGTQR(:,6),MGTQR(:,4),MDFitted(:,4),MDResiduals(:,4),...
    'title','Dipole (A \cdot m^2)','Y (mm)')
FitResiduals(MGTQR(:,6),MGTQR(:,5),MDFitted(:,5),MDResiduals(:,5),...
    'title','Dipole (A \cdot m^2)','Z (mm)')
```


Original Data and Fitted Data
Mass (kg)
Dipole (A · m^2)
Original Data
Fitted Values
Residual
Dipole (A · m^2)


Original Data and Fitted Data
Power (W)
Dipole (A · m^2)
Original Data
Fitted Values
Residual
Dipole (A · m^2)

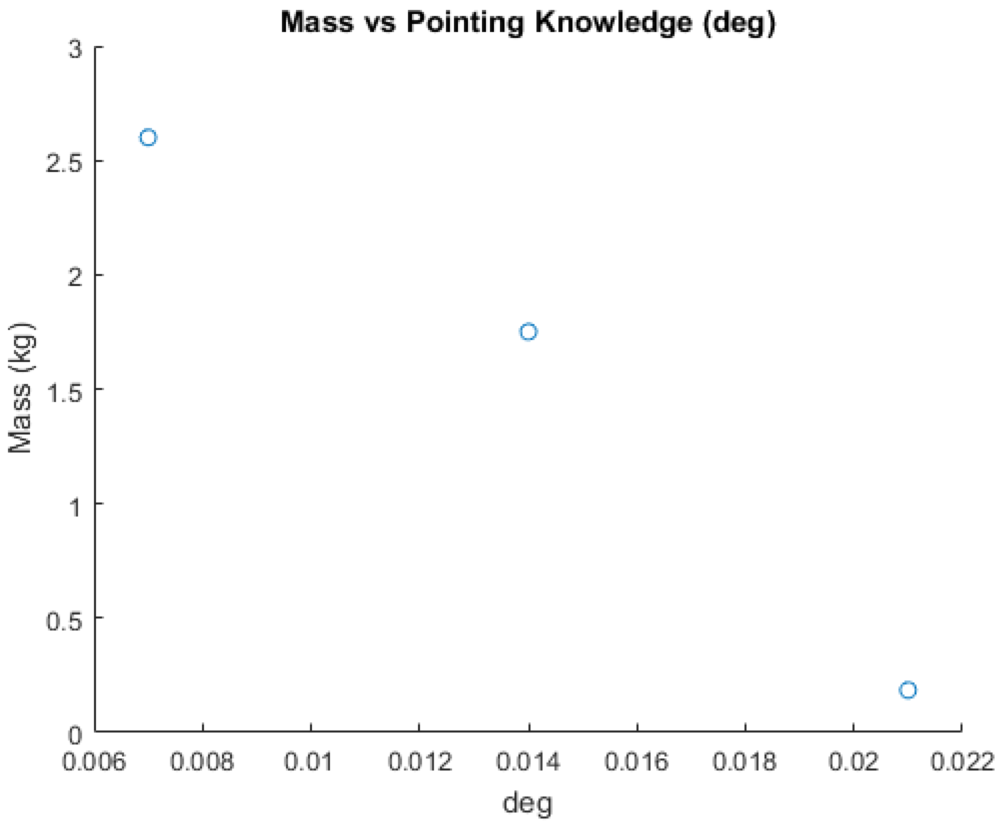

Original Data and Fitted Data
Original Data
Fitted Values
X (mm)
Dipole (A · m^2)
Residual
Dipole (A · m^2)

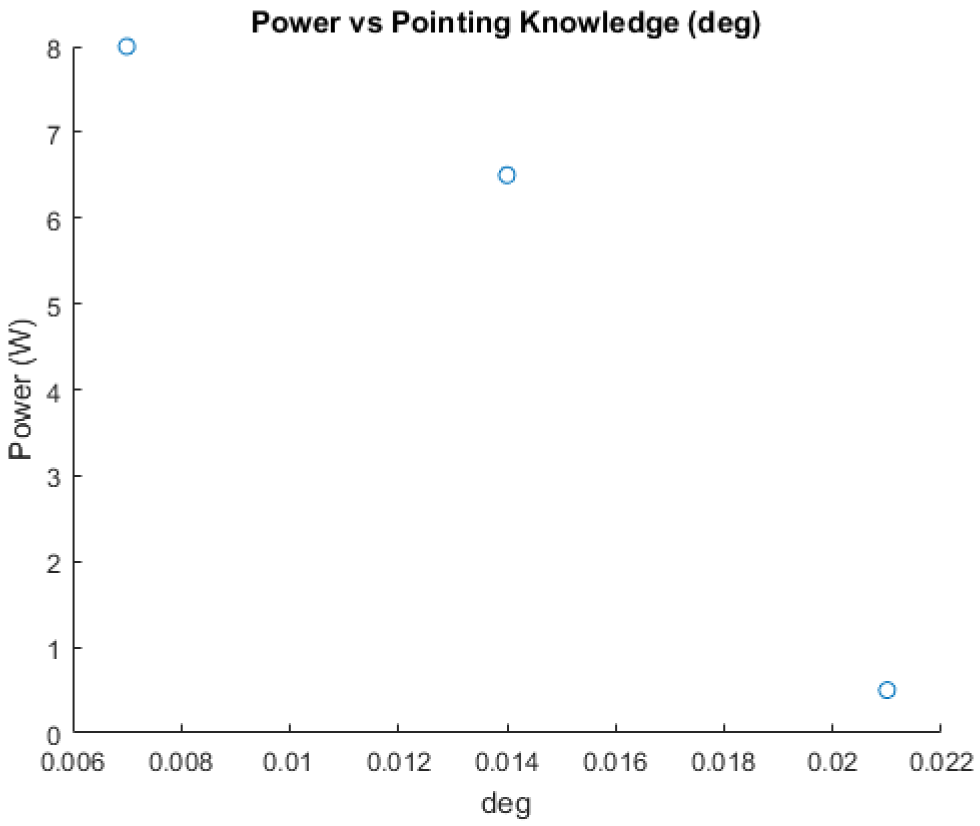

Original Data and Fitted Data
Original Data
Fitted Values
Y (mm)
Dipole (A · m^2)
Residual
Dipole (A · m^2)

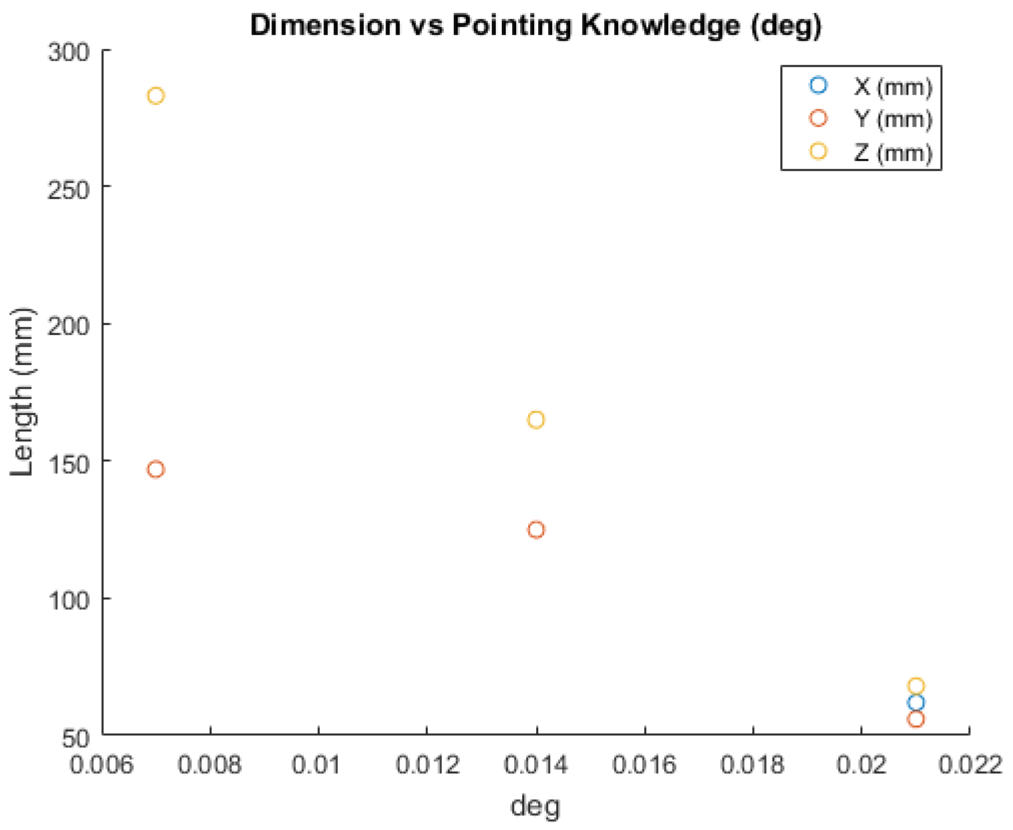


Star Tracker Data

```
STDATA = [2.6   8       147     147     283     0.007;
1.75    6.5     125     125     165     0.014;
0.185   0.5     62      56      68      0.021];
%Mass
figure
scatter(STDATA(:,6),STDATA(:,1))
title('Mass vs Pointing Knowledge (deg)')
xlabel('deg')
ylabel('Mass (kg)')
%Power
figure
scatter(STDATA(:,6),STDATA(:,2))
title('Power vs Pointing Knowledge (deg)')
xlabel('deg')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(STDATA(:,6),STDATA(:,3))
scatter(STDATA(:,6),STDATA(:,4))
scatter(STDATA(:,6),STDATA(:,5))
title('Dimension vs Pointing Knowledge (deg)')
xlabel('deg')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```

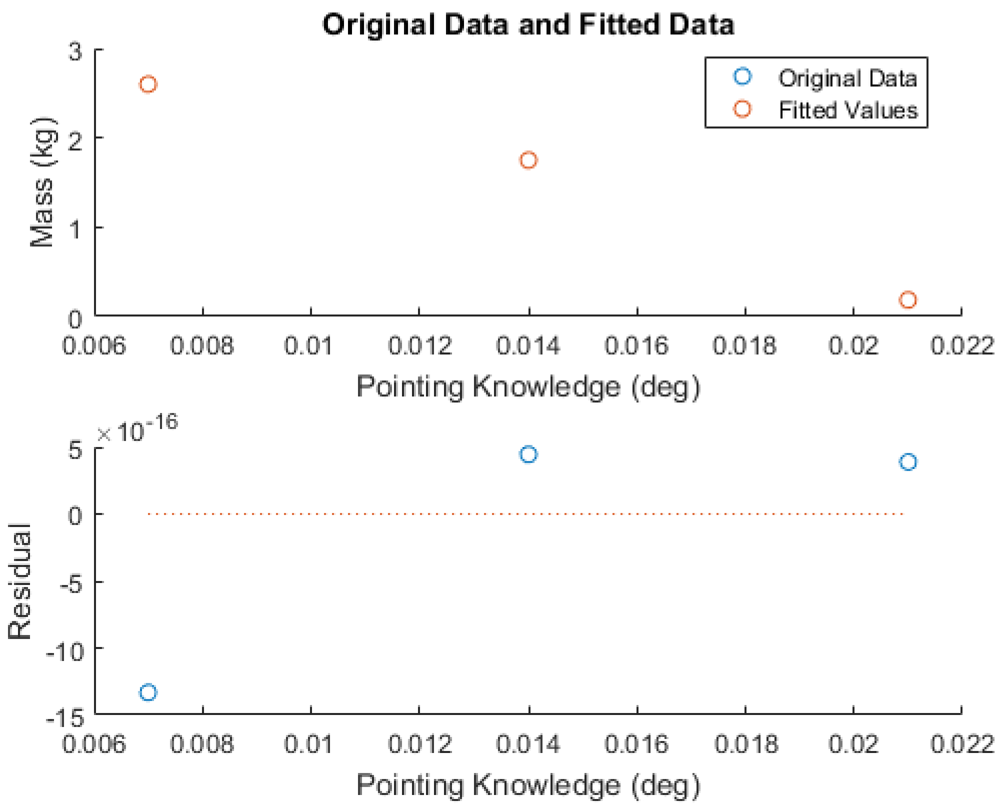

Mass vs Pointing Knowledge (deg)
Mass (kg)
deg
3
2.5
2
1.5
1
0.5
0
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022

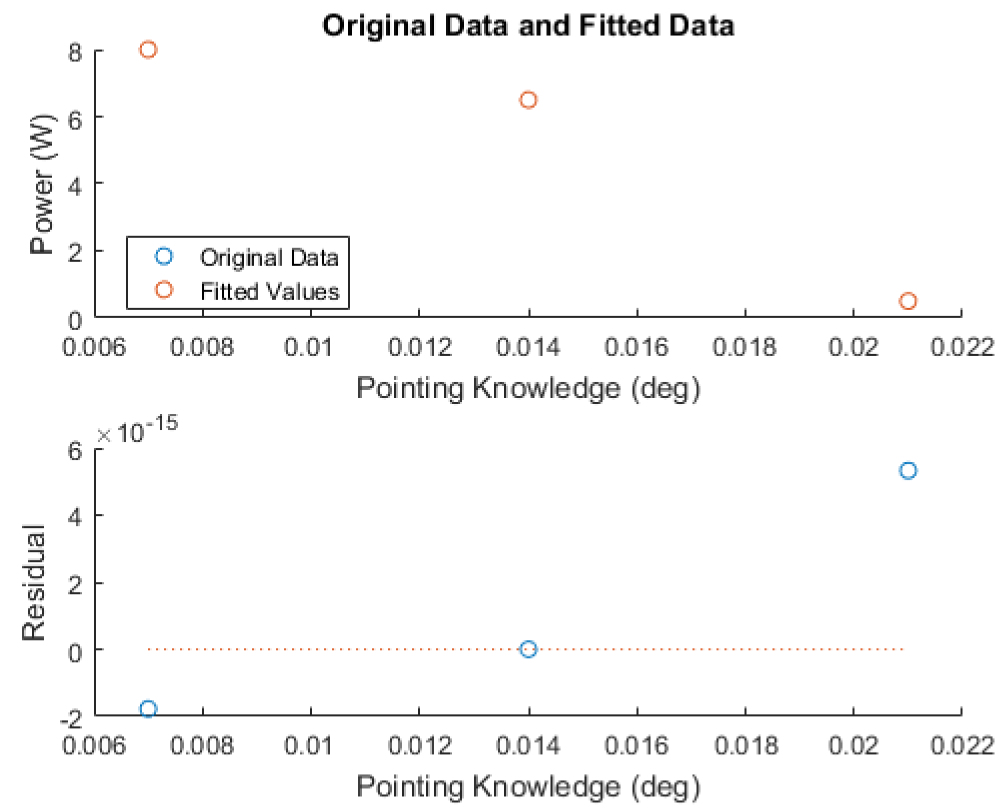

Power vs Pointing Knowledge (deg)
Power (W)
deg
8
7
6
5
4
3
2
1
0
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022

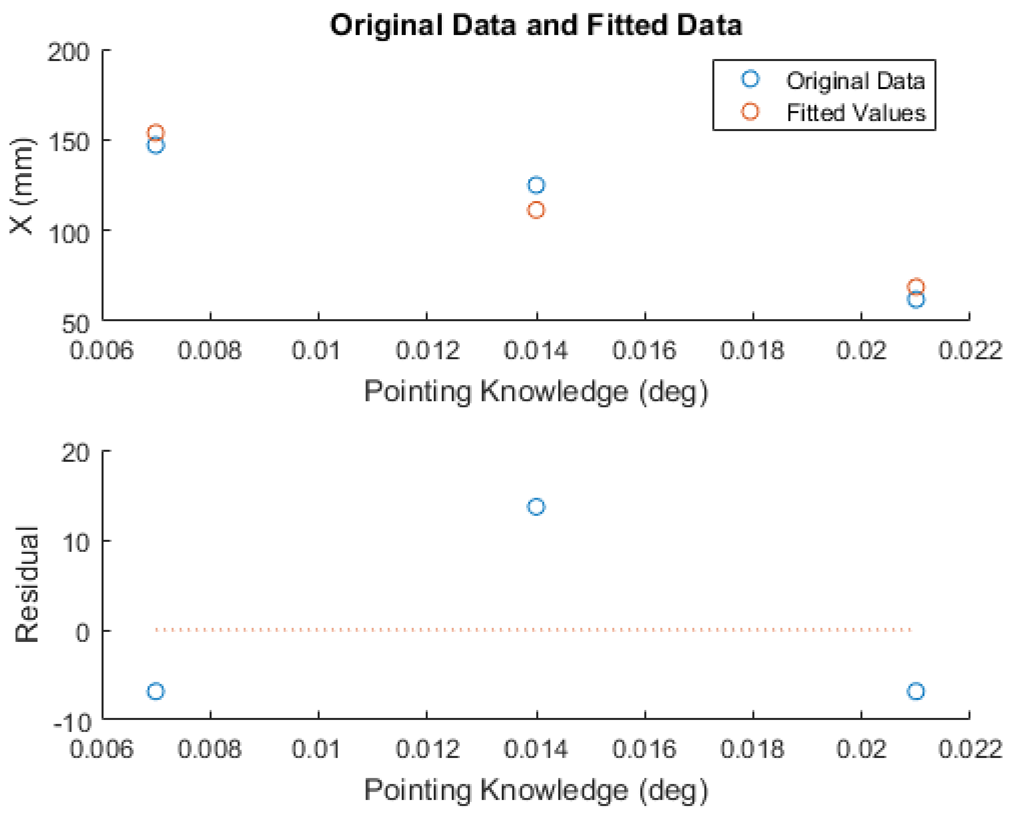


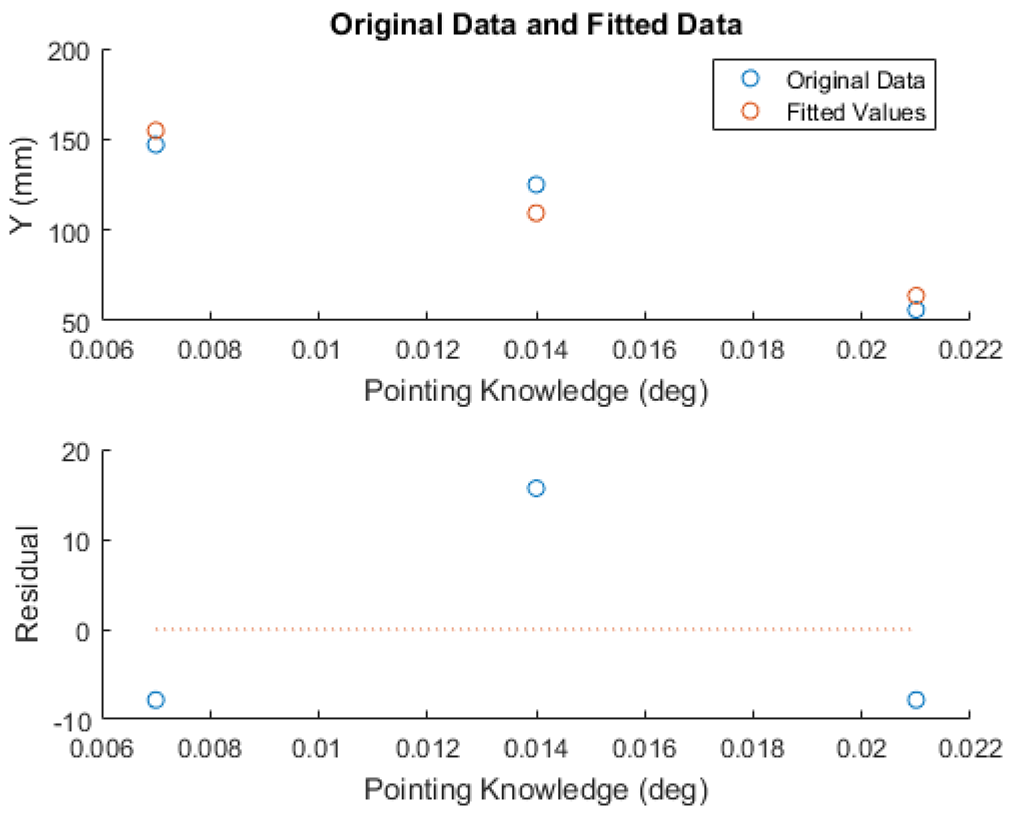
Star Tracker Regression

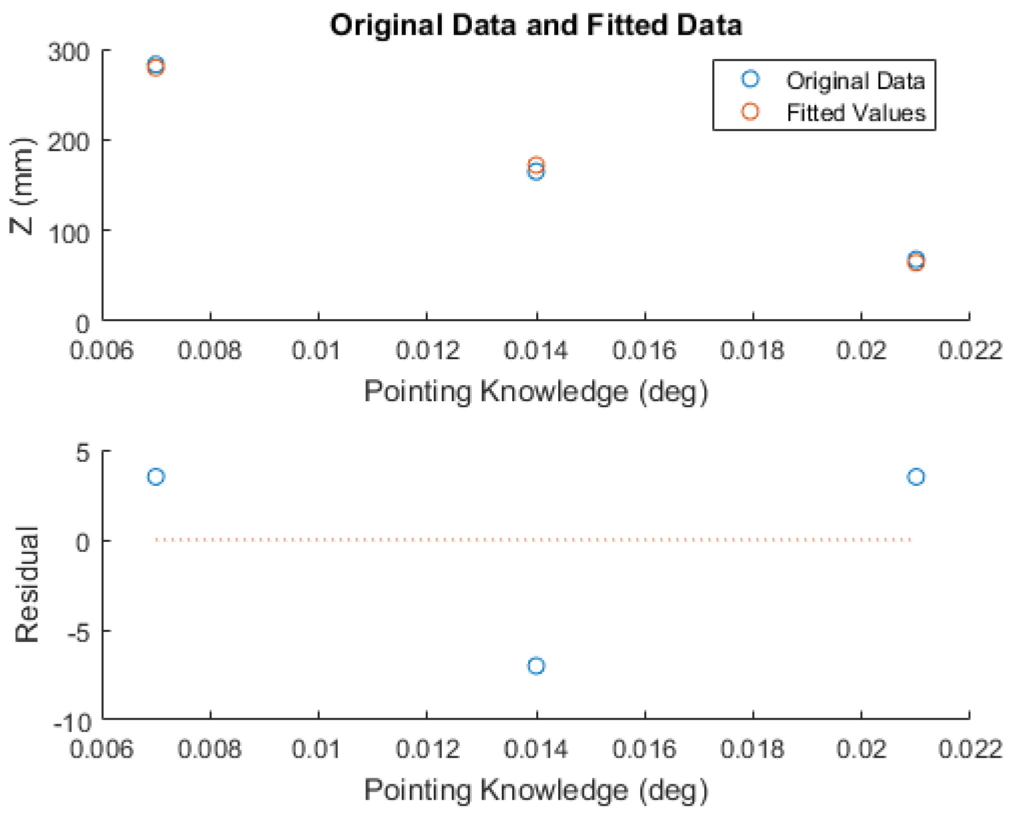
```
[STDFit, STDGOF] = fit(STDATA(:,6),STDATA(:,1),'poly2');
[STPFit, STPGOF] = fit(STDATA(:,6),STDATA(:,2),'poly2');
[STXFit, STXGOF] = fit(STDATA(:,6),STDATA(:,3),'poly1');
[STYFit, STYGOF] = fit(STDATA(:,6),STDATA(:,4),'poly1');
[STZFit, STZGOF] = fit(STDATA(:,6),STDATA(:,5),'poly1');
STFitted = [STDFit(STDATA(:,6)), STPFit(STDATA(:,6)), STXFit(STDATA(:,6)), ...
    STYFit(STDATA(:,6)),STZFit(STDATA(:,6))];
STResiduals = STDATA(:,1:5)-STFitted;
FitResiduals(STDATA(:,6),STDATA(:,1),STFitted(:,1),STResiduals(:,1),...
    'title','Pointing Knowledge (deg)','Mass (kg)')
FitResiduals(STDATA(:,6),STDATA(:,2),STFitted(:,2),STResiduals(:,2),...
    'title','Pointing Knowledge (deg)','Power (W)')
FitResiduals(STDATA(:,6),STDATA(:,3),STFitted(:,3),STResiduals(:,3),...
    'title','Pointing Knowledge (deg)','X (mm)')
FitResiduals(STDATA(:,6),STDATA(:,4),STFitted(:,4),STResiduals(:,4),...
    'title','Pointing Knowledge (deg)','Y (mm)')
FitResiduals(STDATA(:,6),STDATA(:,5),STFitted(:,5),STResiduals(:,5),...
    'title','Pointing Knowledge (deg)','Z (mm)')
```


Original Data and Fitted Data
Mass (kg)
Original Data
Fitted Values
Pointing Knowledge (deg)
×10^-16
Residual
Pointing Knowledge (deg)


Original Data and Fitted Data
Power (W)
Original Data
Fitted Values
Pointing Knowledge (deg)
×10^-15
Residual
Pointing Knowledge (deg)

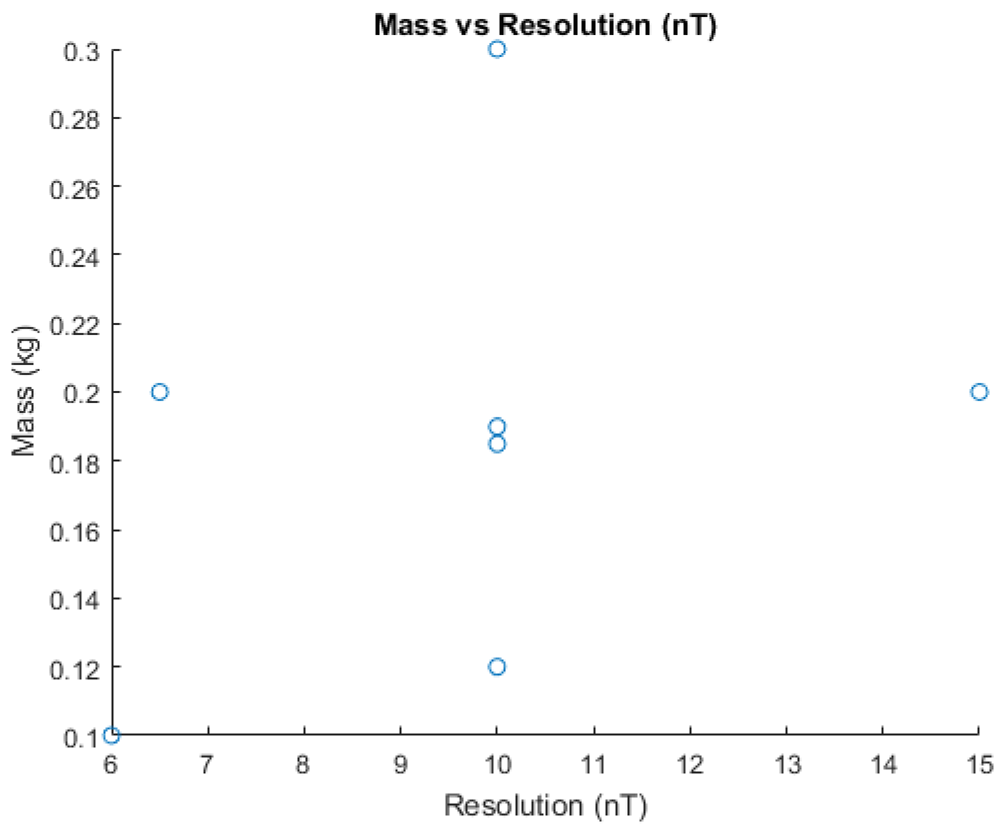

Original Data and Fitted Data
Original Data
Fitted Values
X (mm)
200
150
100
50
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022
Pointing Knowledge (deg)
Residual
20
10
0
-10
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022
Pointing Knowledge (deg)

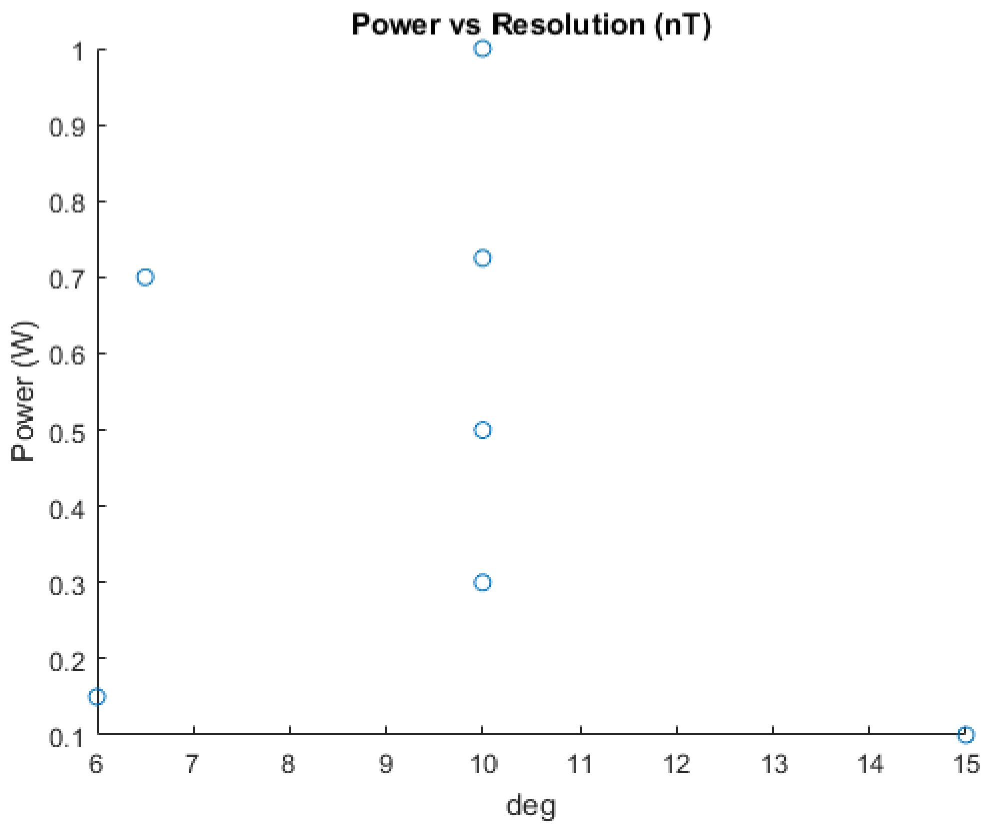

Original Data and Fitted Data
Original Data
Fitted Values
Y (mm)
200
150
100
50
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022
Pointing Knowledge (deg)
Residual
20
10
0
-10
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022
Pointing Knowledge (deg)

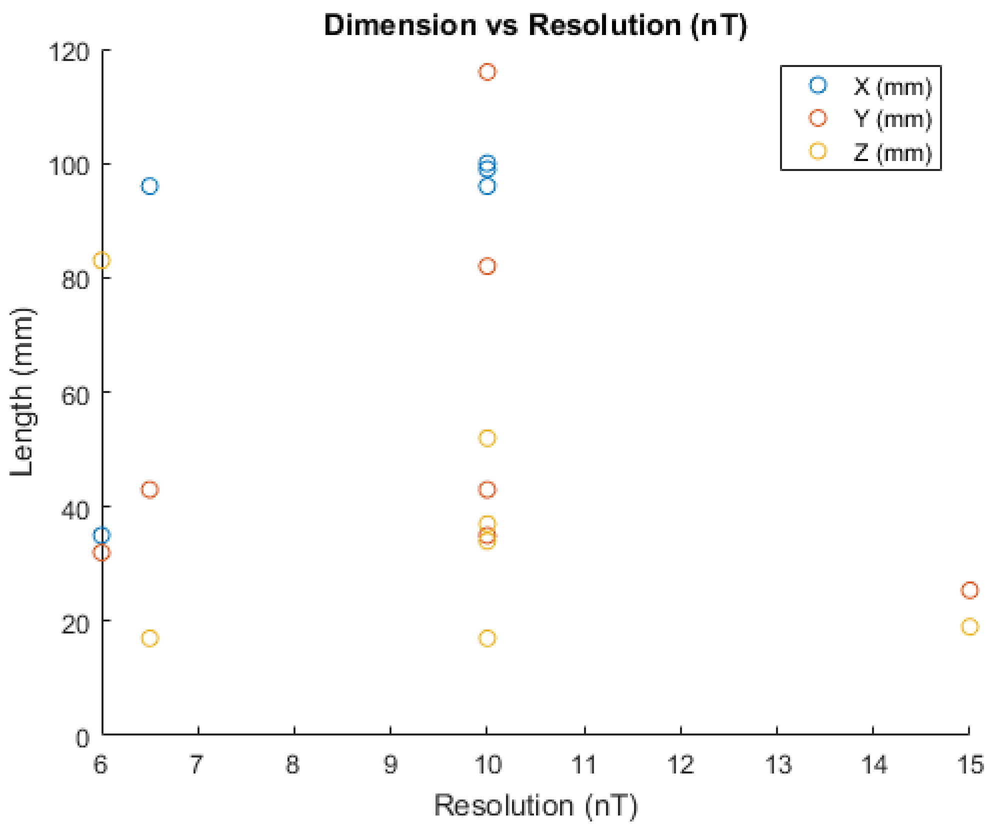


Magnetometer Data

```
%mass power x y z resolution (nT)
MgtmData = [0.12        0.5     116     116     37      10;
0.185   0.725   96      43      17      10;
0.2     0.1     25.4    25.4    19      15;
0.19    0.3     99      35      52      10;
0.3     1       100     82      34      10;
0.2     0.7     96      43      17      6.5;
0.1     0.15    35      32      83      6];
%Mass
figure
scatter(MgtmData(:,6),MgtmData(:,1))
title('Mass vs Resolution (nT)')
xlabel('Resolution (nT)')
ylabel('Mass (kg)')
%Power
figure
scatter(MgtmData(:,6),MgtmData(:,2))
title('Power vs Resolution (nT)')
xlabel('deg')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(MgtmData(:,6),MgtmData(:,3))
scatter(MgtmData(:,6),MgtmData(:,4))
scatter(MgtmData(:,6),MgtmData(:,5))
title('Dimension vs Resolution (nT)')
xlabel('Resolution (nT)')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```

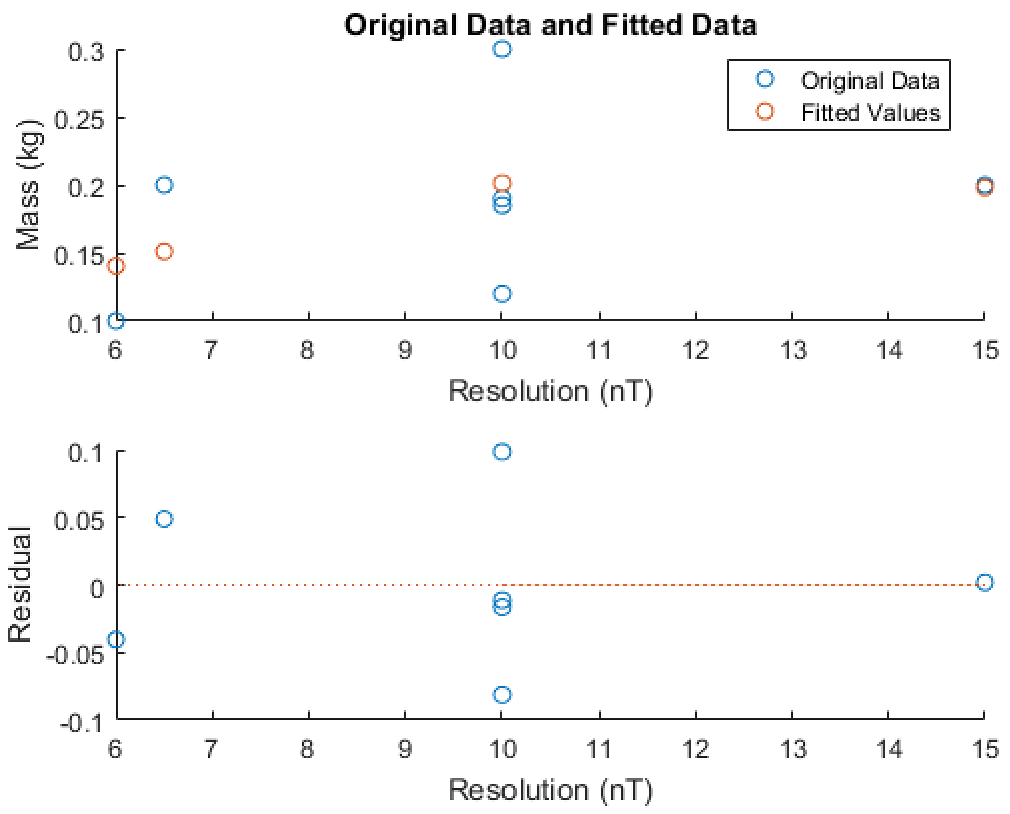

Mass vs Resolution (nT)
Mass (kg)
Resolution (nT)
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0.16
0.14
0.12
0.1
6
7
8
9
10
11
12
13
14
15

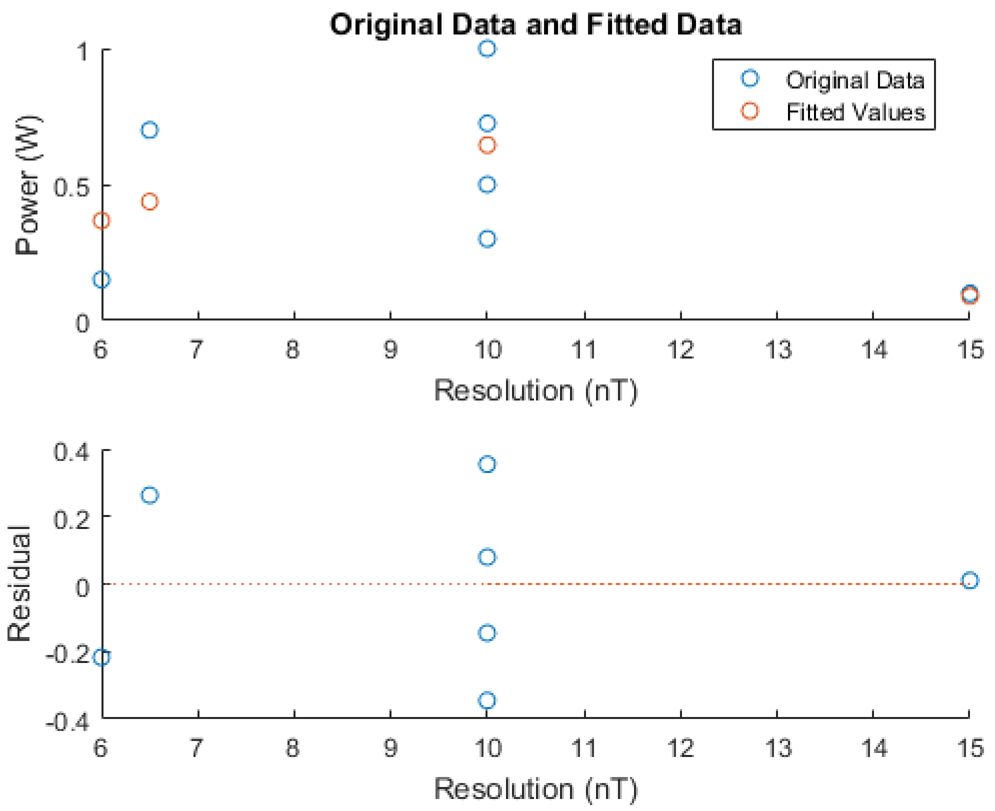

Power vs Resolution (nT)
Power (W)
deg
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
6
7
8
9
10
11
12
13
14
15

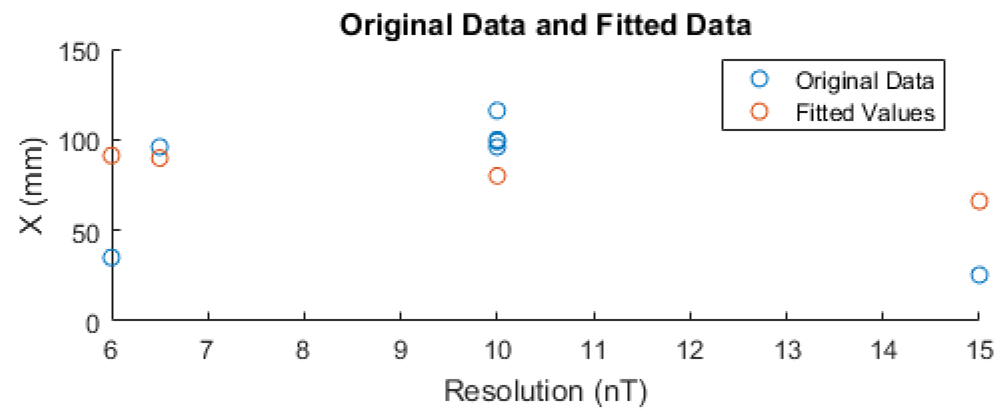


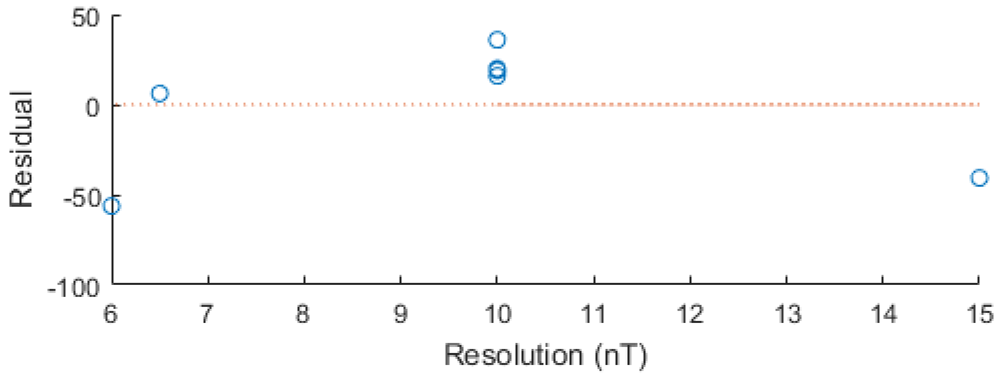
Magnetometer Regression

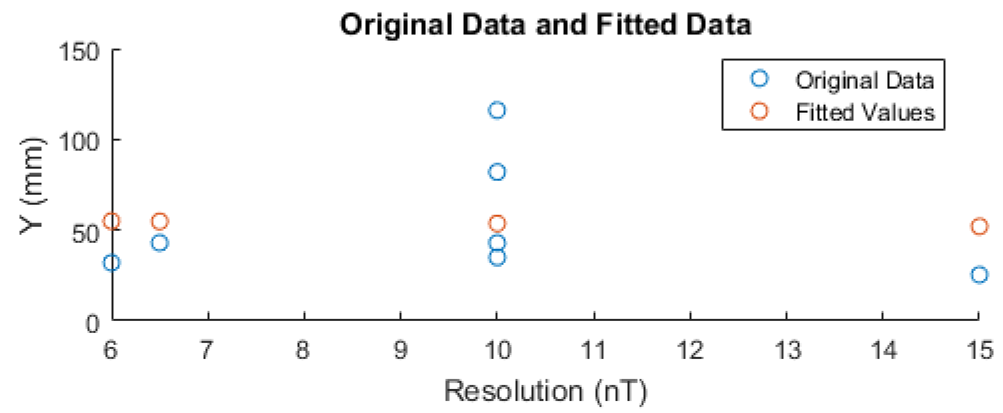
```
[MMDFit, MMDGOF] = fit(MgtmData(:,6),MgtmData(:,1),'poly2');
[MMPFit, MMPGOF] = fit(MgtmData(:,6),MgtmData(:,2),'poly2');
[MMXFit, MMXGOF] = fit(MgtmData(:,6),MgtmData(:,3),'poly1');
[MMYFit, MMYGOF] = fit(MgtmData(:,6),MgtmData(:,4),'poly1');
[MMZFit, MMZGOF] = fit(MgtmData(:,6),MgtmData(:,5),'poly1');
MMFitted = [MMDFit(MgtmData(:,6)), MMPFit(MgtmData(:,6)), MMXFit(MgtmData(:,6)), ...
    MMYFit(MgtmData(:,6)),MMZFit(MgtmData(:,6))];
MMResiduals = MgtmData(:,1:5)-MMFitted;
FitResiduals(MgtmData(:,6),MgtmData(:,1),MMFitted(:,1),MMResiduals(:,1),...
    'title','Resolution (nT)','Mass (kg)')
FitResiduals(MgtmData(:,6),MgtmData(:,2),MMFitted(:,2),MMResiduals(:,2),...
    'title','Resolution (nT)','Power (W)')
FitResiduals(MgtmData(:,6),MgtmData(:,3),MMFitted(:,3),MMResiduals(:,3),...
    'title','Resolution (nT)','X (mm)')
FitResiduals(MgtmData(:,6),MgtmData(:,4),MMFitted(:,4),MMResiduals(:,4),...
    'title','Resolution (nT)','Y (mm)')
FitResiduals(MgtmData(:,6),MgtmData(:,5),MMFitted(:,5),MMResiduals(:,5),...
    'title','Resolution (nT))','Z (mm)')
```

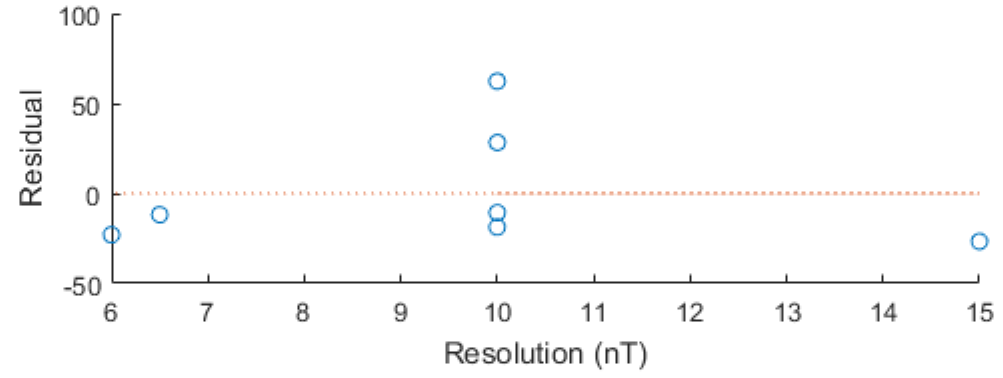

Original Data and Fitted Data
Original Data
Fitted Values
Mass (kg)
0.3
0.25
0.2
0.15
0.1
6
7
8
9
10
11
12
13
14
15
Resolution (nT)
Residual
0.1
0.05
0
-0.05
-0.1
Resolution (nT)

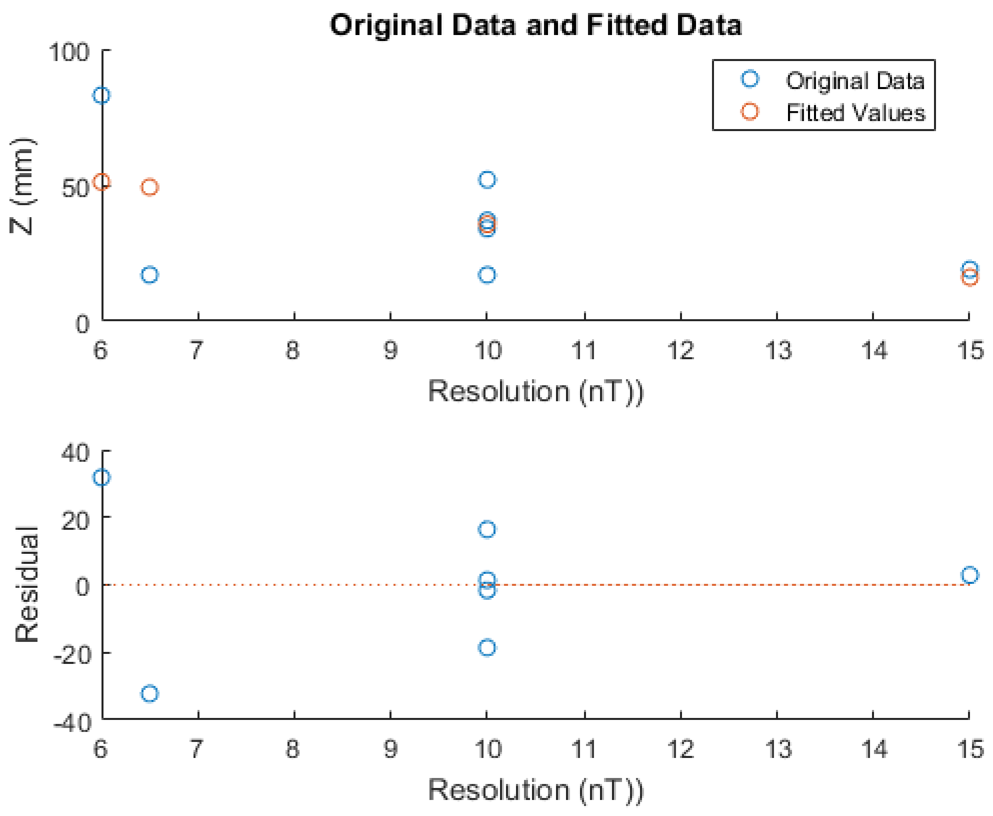

Original Data and Fitted Data
Original Data
Fitted Values
Power (W)
1
0.5
0
6
7
8
9
10
11
12
13
14
15
Resolution (nT)
Residual
0.4
0.2
0
-0.2
-0.4
Resolution (nT)


Original Data and Fitted Data
X (mm)
Resolution (nT)
Original Data
Fitted Values


Residual
Resolution (nT)

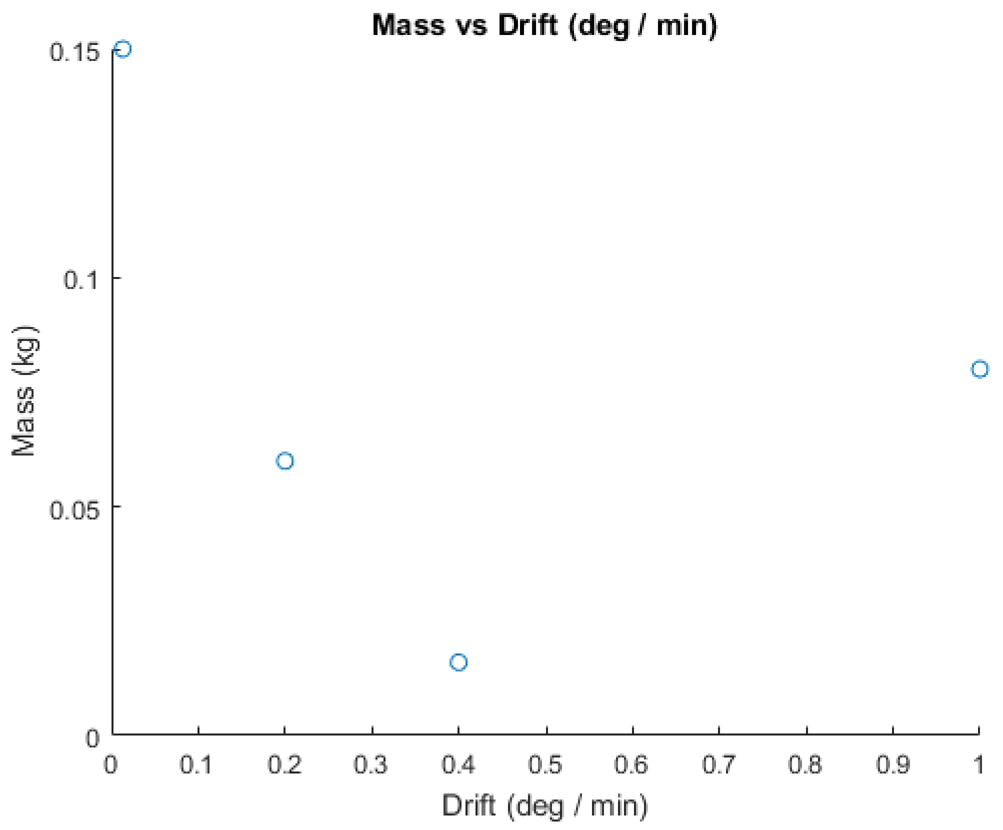

Original Data and Fitted Data
Y (mm)
Resolution (nT)
Original Data
Fitted Values

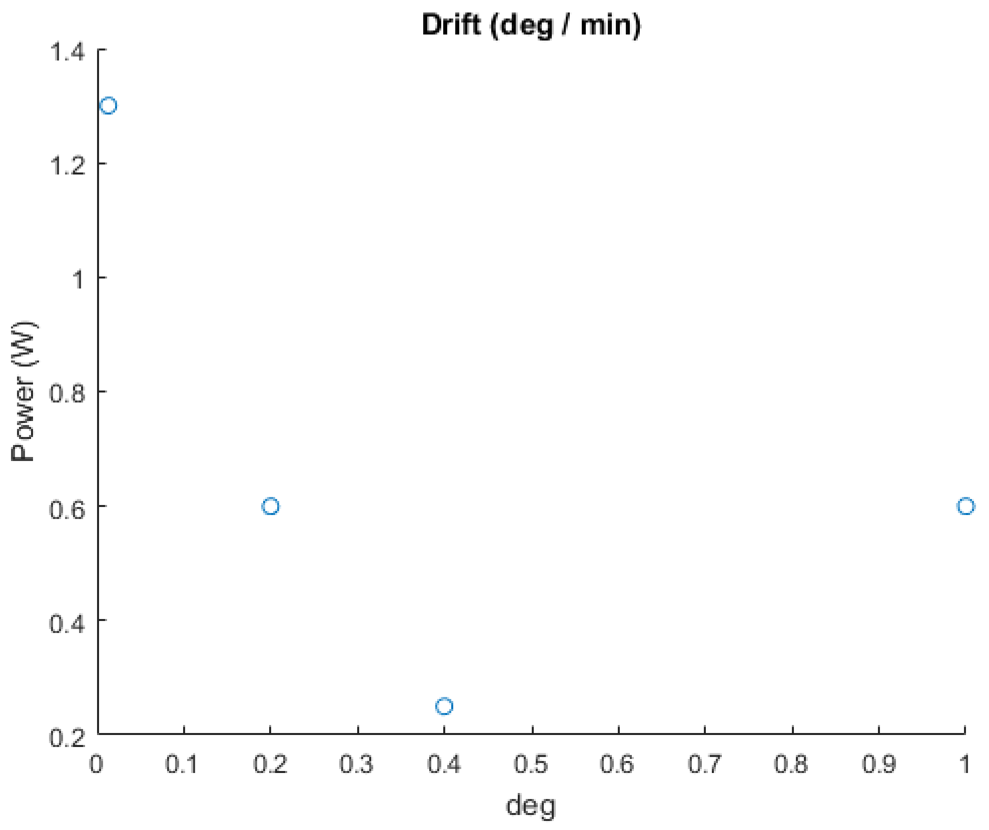

Residual
Resolution (nT)

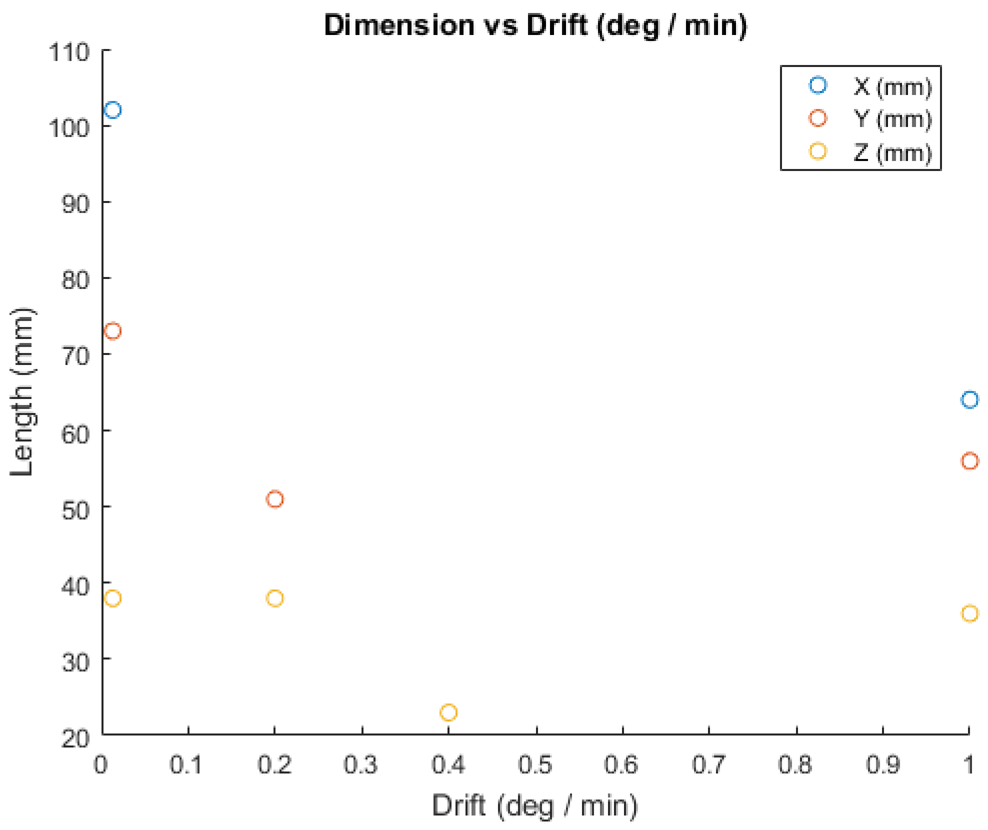


IMU Data

```
IMUDATA = [0.08 0.6     64      56      36      1;
0.06    0.6     51      51      38      0.2;
0.15    1.3     102     73      38      0.013;
0.016   0.25    23      23      23      0.4];
%Mass
figure
scatter(IMUDATA(:,6),IMUDATA(:,1))
title('Mass vs Drift (deg / min)')
xlabel('Drift (deg / min)')
ylabel('Mass (kg)')
%Power
figure
scatter(IMUDATA(:,6),IMUDATA(:,2))
title('Drift (deg / min)')
xlabel('deg')
ylabel('Power (W)')
%Dimensions
figure
hold on
scatter(IMUDATA(:,6),IMUDATA(:,3))
scatter(IMUDATA(:,6),IMUDATA(:,4))
scatter(IMUDATA(:,6),IMUDATA(:,5))
title('Dimension vs Drift (deg / min)')
xlabel('Drift (deg / min)')
ylabel('Length (mm)')
legend('X (mm)','Y (mm)','Z (mm)','location','best')
```


Mass vs Drift (deg / min)
Mass (kg)
Drift (deg / min)
0.15
0.1
0.05
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


Drift (deg / min)
Power (W)
deg
1.4
1.2
1
0.8
0.6
0.4
0.2
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1



IMU Regression

```
[IMUDFit, IMUDGOF] = fit(IMUDATA(:,6),IMUDATA(:,1),'poly1');
[IMUPFit, IMUPGOF] = fit(IMUDATA(:,6),IMUDATA(:,2),'poly1');
[IMUXFit, IMUXGOF] = fit(IMUDATA(:,6),IMUDATA(:,3),'poly1');
[IMUYFit, IMUYGOF] = fit(IMUDATA(:,6),IMUDATA(:,4),'poly1');
[IMUZFit, IMUZGOF] = fit(IMUDATA(:,6),IMUDATA(:,5),'poly1');
```

Regression Functions and Goodness of Fit

```
disp('Reaction Wheel Functions')
disp('Maximum Torque Regression')
MassT,MassTGOF
PowerT, PowerTGOF
XDimFitT, XDimGof
YDimFitT, YDimGof
ZDimFitT, ZDimGof
MassH,MassTGOFH
disp('Maximum Momentum Capacity')
PowerH, PowerTGOF
XDimFitH, XDimGOFH
YDimFitH, YDimGOFH
ZDimFitH, ZDimGOFH
disp('Maxnetorquer Regression')
MtqDFit, MtqDGOF
MtqPFit, MtqPGOF
MtqXFit, MtqXGOF
MtqYFit, MtqYGOF
MtqZFit, MtqZGOF
disp('Star Tracker Regression')
STDFit, STDGOF
```

```
STPFit, STPGOF
STXFit, STXGOF
STYFit, STYGOF
STZFit, STZGOF
disp('Magnetometer Regression')
MMDFit, MMDGOF
MMPFit, MMPGOF
MMXFit, MMXGOF
MMYFit, MMYGOF
MMZFit, MMZGOF
disp('IMU Regression')
IMUDFit, IMUDGOF
IMUPFit, IMUPGOF
IMUXFit, IMUXGOF
IMUYFit, IMUYGOF
IMUZFit, IMUZGOF
```

```
Reaction Wheel Functions
Maximum Torque Regression

MassT =

     Linear model Poly1:
     MassT(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       21.94 (13.62, 30.25)
       p2 =      0.1656 (-0.189, 0.5203)

MassTGOF =

           sse: 0.4725
       rsquare: 0.9020
           dfe: 5
    adjrsquare: 0.8824
          rmse: 0.3074


PowerT =

     Linear model Poly1:
     PowerT(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       6.662 (2.26, 11.06)
       p2 =      0.5111 (0.3233, 0.6989)

PowerTGOF =

           sse: 0.1604
       rsquare: 0.6994
           dfe: 5
    adjrsquare: 0.6393
          rmse: 0.1791


XDimFitT =

     General model:
     XDimFitT(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =        21.9 (7.559, 36.24)
       b =       177.7 (109.4, 246)

XDimGof = 

           sse: 1.5442e+03
       rsquare: 0.7550
           dfe: 5
    adjrsquare: 0.7060
          rmse: 17.5736


YDimFitT = 

     General model:
     YDimFitT(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =       21.05 (4.643, 37.46)
       b =       172.1  (93.94, 250.3)

YDimGof = 

           sse: 2.0219e+03
       rsquare: 0.6851
           dfe: 5
    adjrsquare: 0.6221
          rmse: 20.1091


ZDimFitT = 

     General model:
     ZDimFitT(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =       23.49 (-4.062, 51.05)
       b =       158.4 (27.1, 289.7)

ZDimGof = 

           sse: 5.7022e+03
       rsquare: 0.4900
           dfe: 5
    adjrsquare: 0.3880
          rmse: 33.7703


MassH = 

     Linear model Poly1:
     MassH(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       1.666 (1.445, 1.887)
       p2 =      0.1216 (-0.009627, 0.2528)

MassTGOFH = 

           sse: 0.0632
       rsquare: 0.9869
           dfe: 5
    adjrsquare: 0.9843
          rmse: 0.1124
```

```
Maximum Momentum Capacity

PowerH =

     Linear model Poly1:
     PowerH(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      0.4666 (0.115, 0.8182)
       p2 =      0.5106 (0.3016, 0.7196)

PowerTGOF =

           sse: 0.1604
       rsquare: 0.6994
           dfe: 5
    adjrsquare: 0.6393
          rmse: 0.1791


XDimFitH =

     General model:
     XDimFitH(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =       20.55 (16.23, 24.87)
       b =       120.4 (109.4, 131.5)

XDimGOFH =

           sse: 203.7929
       rsquare: 0.9677
           dfe: 5
    adjrsquare: 0.9612
          rmse: 6.3842


YDimFitH =

     General model:
     YDimFitH(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =       20.21 (13.29, 27.12)
       b =         118 (100.2, 135.7)

YDimGOFH =

           sse: 522.2982
       rsquare: 0.9186
           dfe: 5
    adjrsquare: 0.9024
          rmse: 10.2205


ZDimFitH =

     General model:
     ZDimFitH(x) = a*log(x) + b
     Coefficients (with 95% confidence bounds):
       a =       23.61 (6.689, 40.53)
       b =       100.2  (56.79, 143.7)
```

```
ZDimGOFH =

           sse: 3.1291e+03
       rsquare: 0.7201
           dfe: 5
    adjrsquare: 0.6641
          rmse: 25.0166

Maxnetorquer Regression

MtqDFit =

     Linear model Poly1:
     MtqDFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =    0.001029 (-0.03713, 0.03919)
       p2 =      0.3457 (0.1461, 0.5453)

MtqDGOF =

           sse: 0.0749
       rsquare: 9.6078e-04
           dfe: 5
    adjrsquare: -0.1988
          rmse: 0.1224


MtqPFit =

     Linear model Poly1:
     MtqPFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =     0.05024 (-0.01899, 0.1195)
       p2 =       0.399 (0.03694, 0.7611)

MtqPGOF =

           sse: 0.2466
       rsquare: 0.4103
           dfe: 5
    adjrsquare: 0.2924
          rmse: 0.2221


MtqXFit =

     Linear model Poly1:
     MtqXFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       1.087 (-5.444, 7.618)
       p2 =       17.65 (-16.51, 51.81)

MtqXGOF =

           sse: 2.1946e+03
       rsquare: 0.0353
           dfe: 5
    adjrsquare: -0.1576
          rmse: 20.9505
```

```
MtqYFit =

     Linear model Poly1:
     MtqYFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       3.363 (-27.05, 33.77)
       p2 =       34.37 (-124.7, 193.4)

MtqYGOF =

           sse: 4.7586e+04
       rsquare: 0.0159
           dfe: 5
    adjrsquare: -0.1809
          rmse: 97.5565


MtqZFit =

     Linear model Poly1:
     MtqZFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       26.67 (-0.7283, 54.07)
       p2 =       67.33 (-75.97, 210.6)

MtqZGOF =

           sse: 3.8630e+04
       rsquare: 0.5560
           dfe: 5
    adjrsquare: 0.4672
          rmse: 87.8979

Star Tracker Regression

STDFit =

     Linear model Poly2:
     STDFit(x) = p1*x^2 + p2*x + p3
     Coefficients:
       p1 =       -7296
       p2 =       31.79
       p3 =       2.735

STDGOF =

           sse: 2.2218e-30
       rsquare: 1
           dfe: 0
    adjrsquare: NaN
          rmse: NaN


STPFit =

     Linear model Poly2:
     STPFit(x) = p1*x^2 + p2*x + p3
     Coefficients:
       p1 =  -4.592e+04
       p2 =         750
```

```
        p3 =           5

STPGOF =

           sse: 2.3666e-29
       rsquare: 1
           dfe: 0
    adjrsquare: NaN
          rmse: NaN


STXFit =

     Linear model Poly1:
     STXFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       -6071 (-2.756e+04, 1.541e+04)
       p2 =       196.3 (-128.5, 521.2)

STXGOF =

           sse: 280.1667
       rsquare: 0.9280
           dfe: 1
    adjrsquare: 0.8561
          rmse: 16.7382


STYFit =

     Linear model Poly1:
     STYFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       -6500 (-3.113e+04, 1.813e+04)
       p2 =       200.3 (-172.1, 572.7)

STYGOF =

           sse: 368.1667
       rsquare: 0.9183
           dfe: 1
    adjrsquare: 0.8367
          rmse: 19.1877


STZFit =

     Linear model Poly1:
     STZFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 = -1.536e+04 (-2.636e+04, -4353)
       p2 =         387 (220.6, 553.4)

STZGOF =

           sse: 73.5000
       rsquare: 0.9968
           dfe: 1
    adjrsquare: 0.9937
          rmse: 8.5732
```

```
Magnetometer Regression

MMDFit =

     Linear model Poly2:
     MMDFit(x) = p1*x^2 + p2*x + p3
     Coefficients (with 95% confidence bounds):
       p1 =   -0.001765  (-0.009936, 0.006406)
       p2 =     0.04348  (-0.128, 0.2149)
       p3 =    -0.05692  (-0.9138, 0.7999)

MMDGOF =

           sse: 0.0208
       rsquare: 0.1741
           dfe: 4
    adjrsquare: -0.2389
          rmse: 0.0721


MMPFit =

     Linear model Poly2:
     MMPFit(x) = p1*x^2 + p2*x + p3
     Coefficients (with 95% confidence bounds):
       p1 =    -0.02006  (-0.05542, 0.01529)
       p2 =      0.3905  (-0.3514, 1.132)
       p3 =      -1.254  (-4.961, 2.454)

MMPGOF =

           sse: 0.3889
       rsquare: 0.4134
           dfe: 4
    adjrsquare: 0.1202
          rmse: 0.3118


MMXFit =

     Linear model Poly1:
     MMXFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      -2.795 (-16.24, 10.65)
       p2 =         108 (-26.72, 242.7)

MMXGOF =

           sse: 7.1582e+03
       rsquare: 0.0541
           dfe: 5
    adjrsquare: -0.1351
          rmse: 37.8371


MMYFit =

     Linear model Poly1:
     MMYFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =     -0.3261 (-13.16, 12.51)
       p2 =       56.92 (-71.74, 185.6)

MMYGOF = 

           sse: 6.5270e+03
       rsquare: 8.5208e-04
           dfe: 5
    adjrsquare: -0.1990
          rmse: 36.1304


MMZFit = 

     Linear model Poly1:
     MMZFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      -3.896 (-12.12, 4.327)
       p2 =       74.57 (-7.854, 157)

MMZGOF = 

           sse: 2.6792e+03
       rsquare: 0.2288
           dfe: 5
    adjrsquare: 0.0746
          rmse: 23.1480

IMU Regression

IMUDFit = 

     Linear model Poly1:
     IMUDFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =    -0.04192 (-0.4176, 0.3337)
       p2 =      0.0934 (-0.1124, 0.2992)

IMUDGOF = 

           sse: 0.0084
       rsquare: 0.1034
           dfe: 2
    adjrsquare: -0.3450
          rmse: 0.0647


IMUPFit = 

     Linear model Poly1:
     IMUPFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =     -0.4949 (-3.239, 2.249)
       p2 =      0.8871 (-0.6161, 2.39)

IMUPGOF = 

           sse: 0.4473
       rsquare: 0.2314
           dfe: 2
    adjrsquare: -0.1530
          rmse: 0.4729
```

```
IMUXFit =

     Linear model Poly1:
     IMUXFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      -21.93 (-245.4, 201.5)
       p2 =       68.84 (-53.56, 191.2)

IMUXGOF =

           sse: 2.9657e+03
       rsquare: 0.0818
           dfe: 2
    adjrsquare: -0.3773
          rmse: 38.5077


IMUYFit =

     Linear model Poly1:
     IMUYFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      -10.02 (-154.4, 134.3)
       p2 =       54.79 (-24.28, 133.9)

IMUYGOF =

           sse: 1.2375e+03
       rsquare: 0.0427
           dfe: 2
    adjrsquare: -0.4359
          rmse: 24.8748


IMUZFit =

     Linear model Poly1:
     IMUZFit(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      -2.082 (-53.07, 48.9)
       p2 =       34.59 (6.663, 62.52)

IMUZGOF =

           sse: 154.3662
       rsquare: 0.0152
           dfe: 2
    adjrsquare: -0.4772
          rmse: 8.7854
```

Published with MATLAB® R2016a

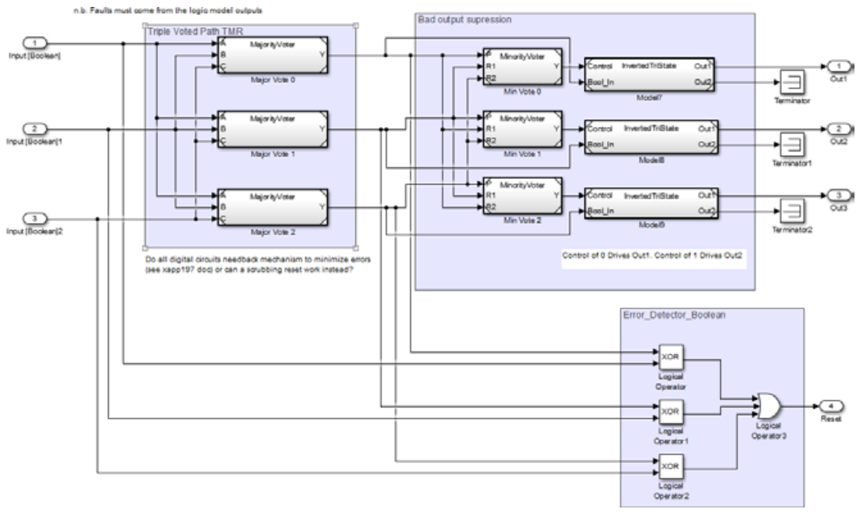


Figure C.1: TMR Model

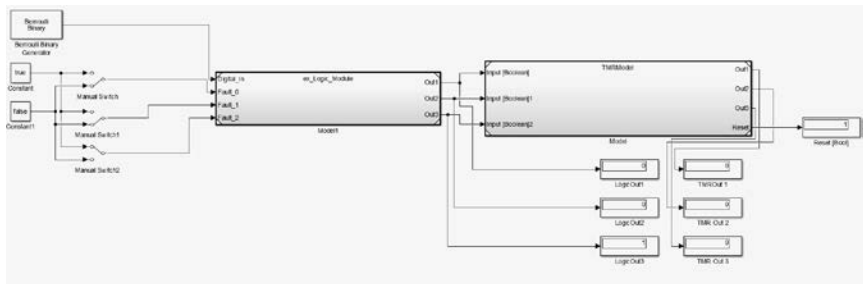


Figure C.2: TMR test detecting error

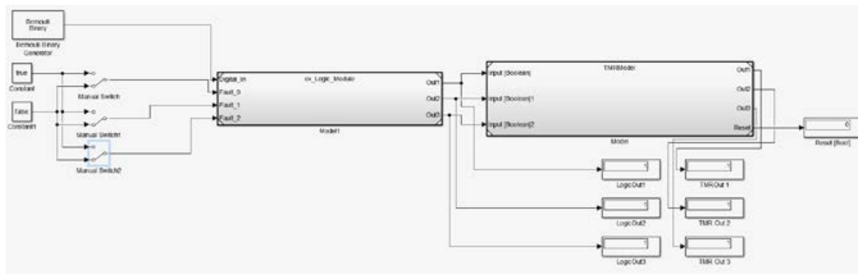


Figure C.3: TMR test detecting no failures

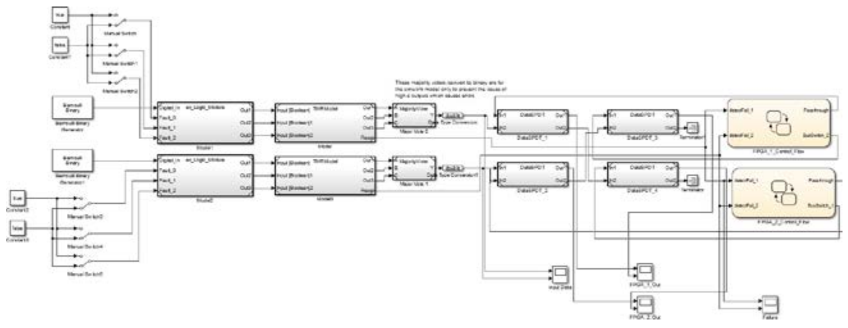


Figure C.4: Setup to test a novel switching algorithm

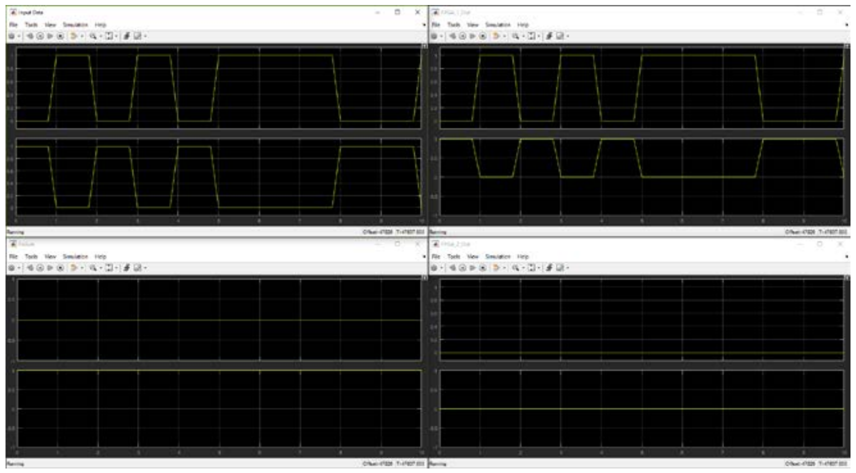


Figure C.5: Detection of a failed fpga.

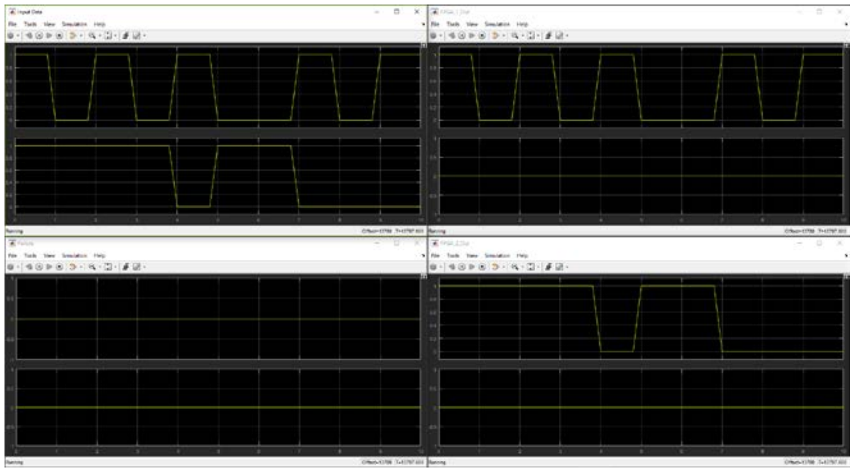


Figure C.6: No failures detected in FPGAs

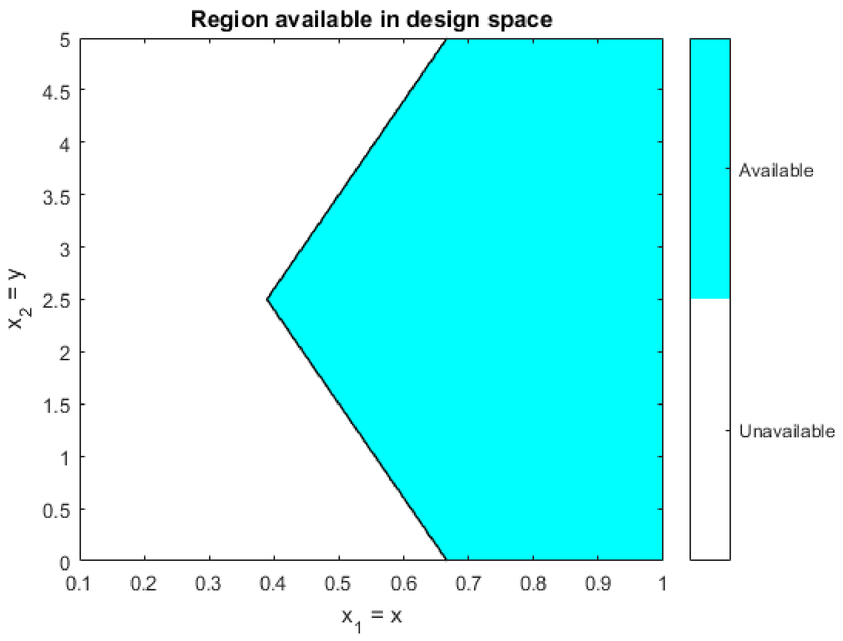


Figure C.7: Design space of benchmark optimization problem

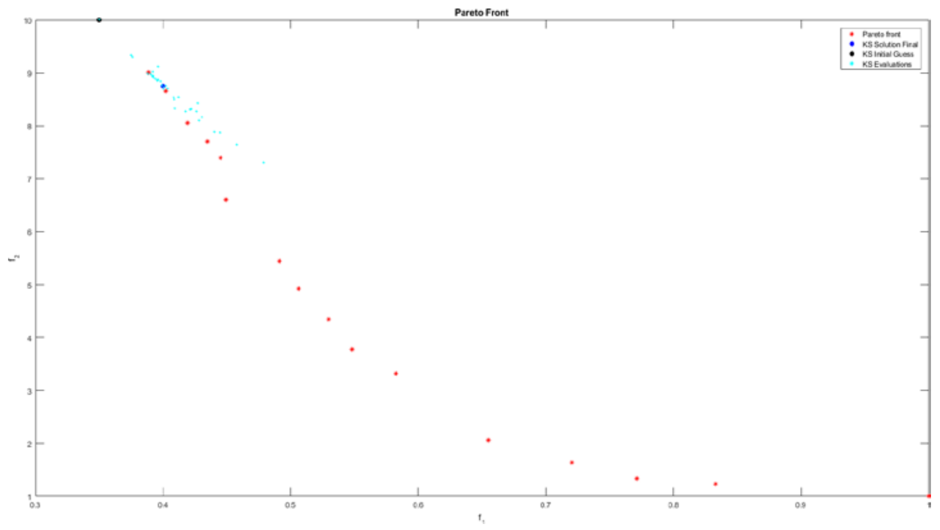
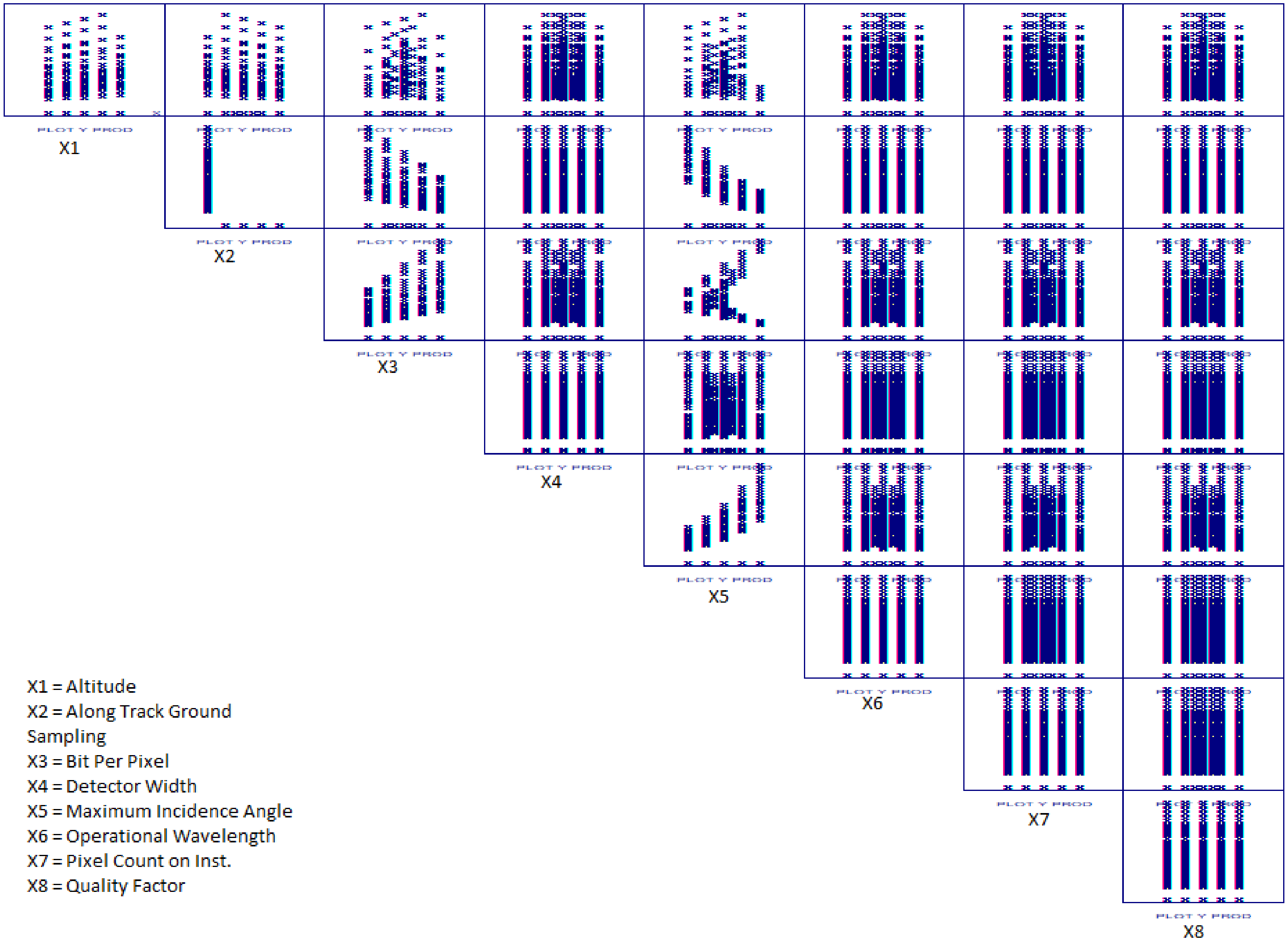


Figure C.8: Evaluation from the initialization point of the KS function. The initial function is outside of the feasible design space while the final solution is feasible and located within the Pareto optimized region.

Data Rate Response



X1 = Altitude
X2 = Along Track Ground Sampling
X3 = Bit Per Pixel
X4 = Detector Width
X5 = Maximum Incidence Angle
X6 = Operational Wavelength
X7 = Pixel Count on Inst.
X8 = Quality Factor

Figure C.9: DoE scatter plot of Aperture Diameter

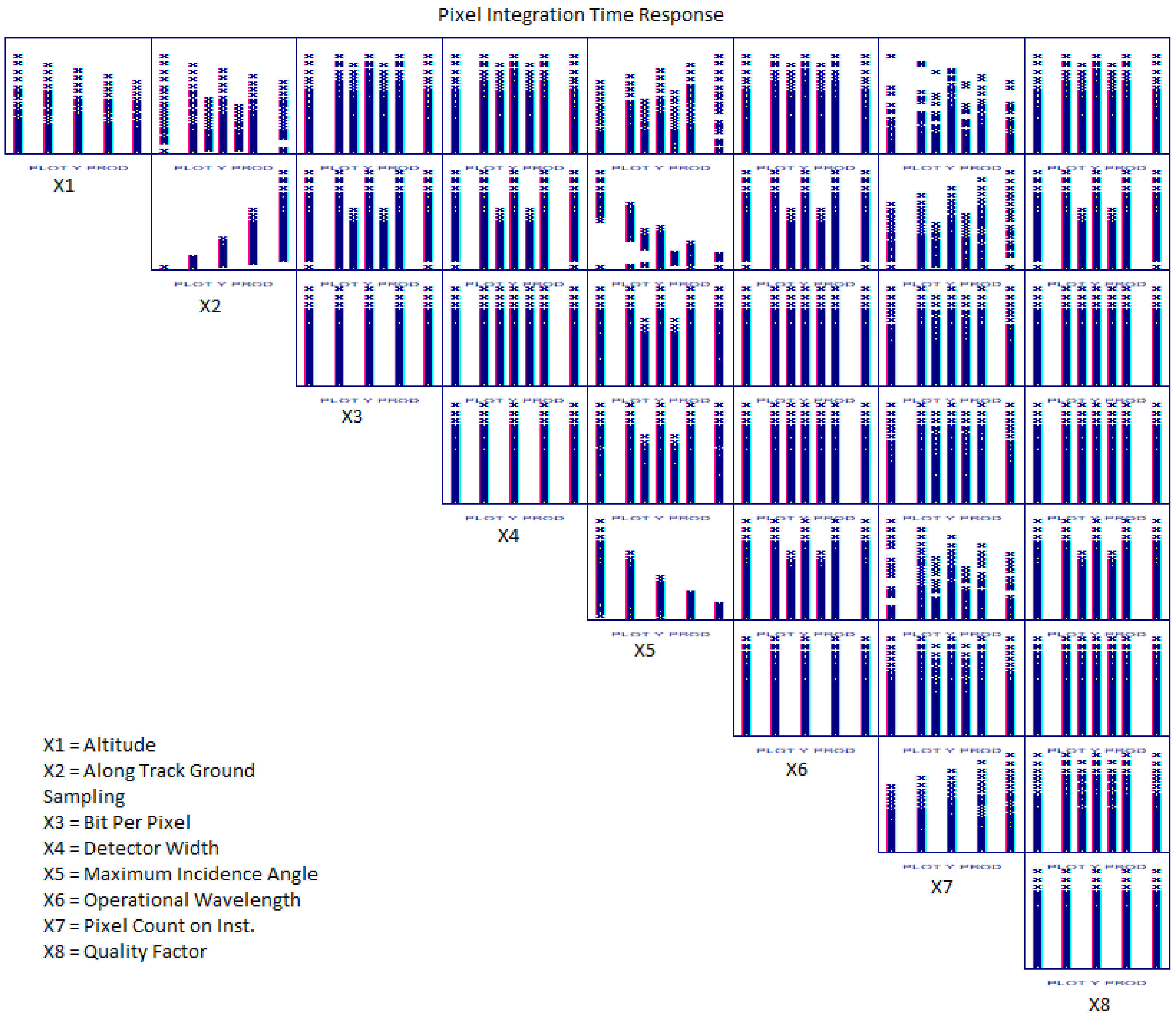


Figure C.10: DoE scatter plot of Aperture Diameter

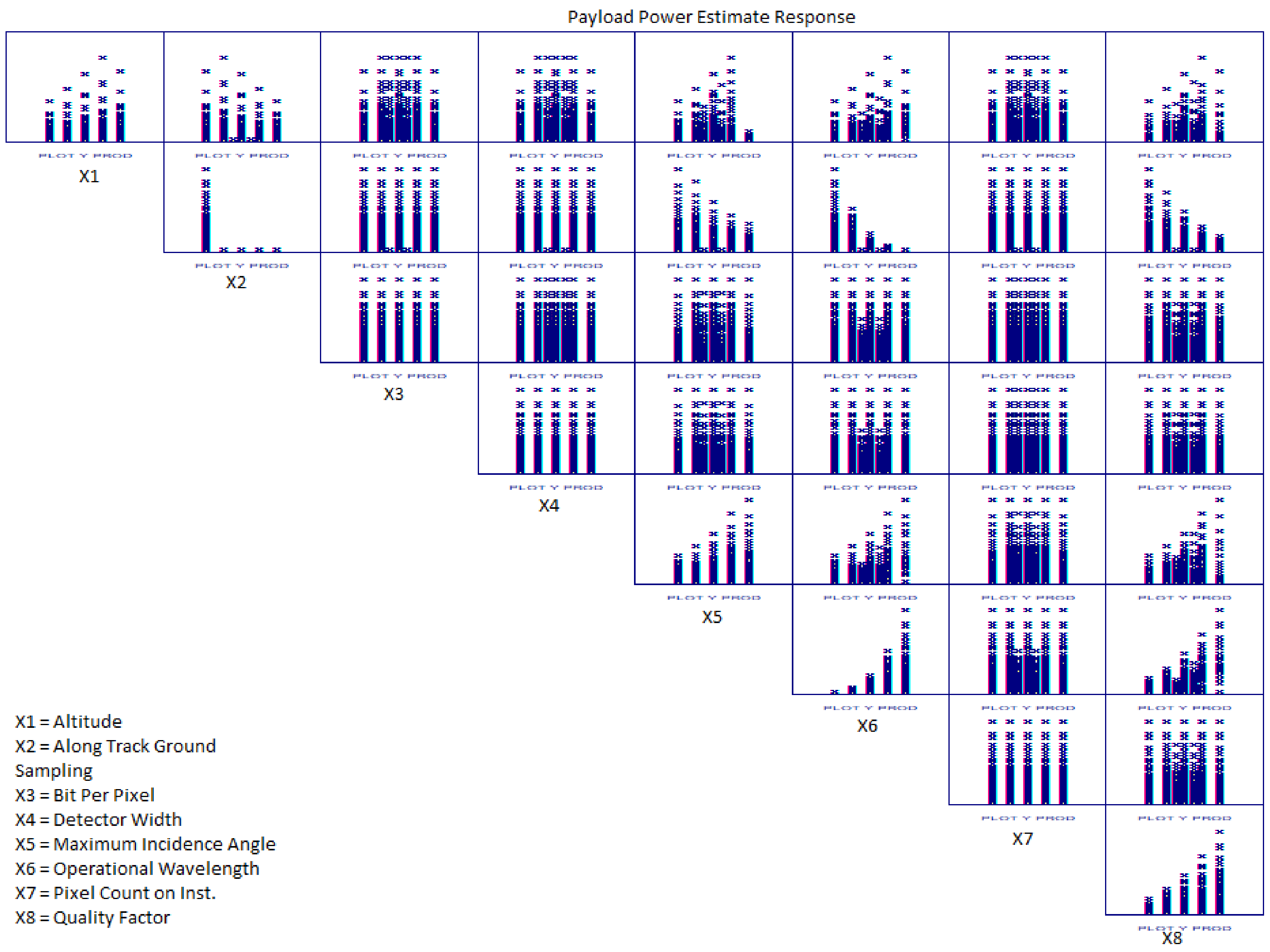


Figure C.11: DoE scatter plot of Aperture Diameter

Contents

Data Analysis for DOEs

```
clc; clear all; close all;
```

READ Payload Data file

```
if exist('ADCSDOEData.mat', 'file') == 2
    load('ADCSDOEData.mat') %Saves a few minutes if the data has been parsed and saved and analyzed
else
```

```
    fid = fopen('DOE_ACDS','r');
    text = textscan(fid,'%s','Delimiter','','endofline','');
    text = text{1}{1};
    fid = fclose(fid);
    AerodynamicTorque = regexp(text,'AerodynamicTorque:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    CenterGravity = regexp(text,'CenterGravity:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    CenterPressure = regexp(text,'CenterPressure:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    CenterSolarPressure = regexp(text,'CenterSolarPressure:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    CoeffDrag = regexp(text,'CoeffDrag:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    Density = regexp(text,'Density:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    DisturbanceTorque = regexp(text,'DisturbanceTorque:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens'); %modified to nt contain IFOV
    GravityGradient = regexp(text,'GravityGradient:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    IncidenceAngle = regexp(text,'IncidenceAngle:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    Iy  = regexp(text,'Iy:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    Iz  = regexp(text,'Iz:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MagDipole = regexp(text,'MagDipole:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MagneticField = regexp(text,'MagneticField:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MomentumStorageRx = regexp(text,'MomentumStorageRx:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MtxMass = regexp(text,'MtxMass:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MtxPWR = regexp(text,'MtxPWR:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MtxX  = regexp(text,'MtxX:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MtxY  = regexp(text,'MtxY:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    MtxZ  = regexp(text,'MtxZ:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    OrbPer = regexp(text,'OrbPer:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    PointKnowledge = regexp(text,'PointKnowledge:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    Radius = regexp(text,'Radius:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    ReflectanceFactor = regexp(text,'ReflectanceFactor:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    ResidualDipole = regexp(text,'ResidualDipole:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    RxMass = regexp(text,'RxMass:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    RxPWR = regexp(text,'RxPWR:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    RxX  = regexp(text,'RxX:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    RxY  = regexp(text,'RxY:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    RxZ  = regexp(text,'RxZ:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    STMass = regexp(text,'STMass:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    STPWR = regexp(text,'STPWR:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
    STX = regexp(text,'STX:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
```

```
STY  = regexp(text,'STY:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
STZ  = regexp(text,'STZ:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
SolarRadiation = regexp(text,'SolarRadiation:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
SunIA = regexp(text,'SunIA:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
SurfaceArea = regexp(text,'SurfaceArea:\s+(\d*(.)?(\d*)?(e)?[+-]?(\d*))','tokens');
clear text fid%Remove file to clear up memory
```

Convert to Usable Doubles

```
AerodynamicTorque = str2double([AerodynamicTorque{:}]');
CenterGravity = str2double([CenterGravity{:}]');
CenterPressure = str2double([CenterPressure{:}]');
CenterSolarPressure = str2double([CenterSolarPressure{:}]');
CoeffDrag = str2double([CoeffDrag{:}]');
Density = str2double([Density{:}]');
DisturbanceTorque = str2double([DisturbanceTorque{:}]');
GravityGradient = str2double([GravityGradient{:}]');
IncidenceAngle = str2double([IncidenceAngle{:}]');
Iy = str2double([Iy{:}]');
Iz = str2double([Iz{:}]');
MagDipole = str2double([MagDipole{:}]');
MagneticField = str2double([MagneticField{:}]');
MomentumStorageRx = str2double([MomentumStorageRx{:}]');
MtxMass = str2double([MtxMass{:}]');
MtxPWR = str2double([MtxPWR{:}]');
MtxX = str2double([MtxX{:}]');
MtxY = str2double([MtxY{:}]');
MtxZ = str2double([MtxZ{:}]');
OrbPer = str2double([OrbPer{:}]');
PointKnowledge = str2double([PointKnowledge{:}]');
Radius = str2double([Radius{:}]');
ReflectanceFactor = str2double([ReflectanceFactor{:}]');
ResidualDipole = str2double([ResidualDipole{:}]');
RxMass = str2double([RxMass{:}]');
RxPWR = str2double([RxPWR{:}]');
RxX = str2double([RxX{:}]');
RxY = str2double([RxY{:}]');
RxZ = str2double([RxZ{:}]');
STMass = str2double([STMass{:}]');
STPWR = str2double([STPWR{:}]');
STX = str2double([STX{:}]');
STY = str2double([STY{:}]');
STZ = str2double([STZ{:}]');
SolarRadiation = str2double([SolarRadiation{:}]');
SunIA = str2double([SunIA{:}]');
SurfaceArea = str2double([SurfaceArea{:}]');
```

Remove NaN caused by metadata and parameter saving

```
AerodynamicTorque = AerodynamicTorque(~isnan(AerodynamicTorque));
CenterGravity = CenterGravity(~isnan(CenterGravity));
CenterPressure = CenterPressure(~isnan(CenterPressure));
CenterSolarPressure = CenterSolarPressure(~isnan(CenterSolarPressure));
Density = Density(~isnan(Density));
CoeffDrag = CoeffDrag(~isnan(CoeffDrag));
DisturbanceTorque = DisturbanceTorque(~isnan(DisturbanceTorque));
GravityGradient = GravityGradient(~isnan(GravityGradient));
IncidenceAngle = IncidenceAngle(~isnan(IncidenceAngle));
Iy = Iy(~isnan(Iy));
Iz = Iz(~isnan(Iz));
MagDipole = MagDipole(~isnan(MagDipole));
MagneticField = MagneticField(~isnan(MagneticField));
MomentumStorageRx = MomentumStorageRx(~isnan(MomentumStorageRx));
MtxMass = MtxMass(~isnan(MtxMass));
MtxPWR = MtxPWR(~isnan(MtxPWR));
MtxX = MtxX(~isnan(MtxX));
MtxY = MtxY(~isnan(MtxY));
MtxZ = MtxZ(~isnan(MtxZ));
OrbPer = OrbPer(~isnan(OrbPer));
```

```
    PointKnowledge = PointKnowledge(~isnan(PointKnowledge));
    Radius = Radius(~isnan(Radius));
    ReflectanceFactor = ReflectanceFactor(~isnan(ReflectanceFactor));
    ResidualDipole = ResidualDipole(~isnan(ResidualDipole));
    RxMass = RxMass(~isnan(RxMass));
    RxPWR = RxPWR(~isnan(RxPWR));
    RxX = RxX(~isnan(RxX));
    RxY = RxY(~isnan(RxY));
    RxZ = RxZ(~isnan(RxZ));
    STMass = STMass(~isnan(STMass));
    STPWR = STPWR(~isnan(STPWR));
    STX = STX(~isnan(STX));
    STY = STY(~isnan(STY));
    STZ = STZ(~isnan(STZ));
    SolarRadiation = SolarRadiation(~isnan(SolarRadiation));
    SunIA = SunIA(~isnan(SunIA));
    SurfaceArea = SurfaceArea(~isnan(SurfaceArea));
    save('ADCSDOEData.mat')
```

```
end
```

Design Variable Orthogonolization

Design variables include radius, Iy, Iz, Reflectance Factor, SunIA, Cd,
Pt Knowledge

```
[Radius, RadiusA, RadiusB] = CodeFactorLevel(Radius);
[Iy, IyA, IyB] = CodeFactorLevel(Iy);
[Iz, IzA, IzB] = CodeFactorLevel(Iz);
[ReflectanceFactor, RFA, RFB] = CodeFactorLevel(ReflectanceFactor);
[SunIA, SIAA, SIAB] = CodeFactorLevel(SunIA);
[CoeffDrag, CDA, CDB] = CodeFactorLevel(CoeffDrag);
[PointKnowledge, PKA, PKB] = CodeFactorLevel(PointKnowledge);
[ResidualDipole, RDA, RDB] =  CodeFactorLevel(ResidualDipole);
```

Response Variable Test of Normality

```
[H, PVal, WStatistic] = kstest(RxX);
if H == 0
    fprintf('RxX / RxY is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('RxX/RxY is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(RxZ);
if H == 0
    fprintf('RxZ is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('RxZ is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(RxMass);
if H == 0
    fprintf('RxMass is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('RxMass is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(RxPWR);
if H == 0
    fprintf('RxPWR is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('RxPWR is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(MtxX);
if H == 0
```

```
    fprintf('MtX is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('Mtx is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(MtxZ);
if H == 0
    fprintf('MtxZ is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('MtxZ is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(MtxMass);
if H == 0
    fprintf('MtxMass is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('MtxMass is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(MtxPWR);
if H == 0
    fprintf('MtxPWR is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('MtxPWR is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(STX);
if H == 0
    fprintf('STX is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('STX is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(STZ);
if H == 0
    fprintf('STZ is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('STZ is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(STMass);
if H == 0
    fprintf('STMass is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('STMass is not a normal distribution.\n')
end
[H, PVal, WStatistic] = kstest(STPWR);
if H == 0
    fprintf('STPWR is a normal distribution with PVal %d and Wstat %d.\n',...
        PVal, WStatistic)
else
    fprintf('STPWR is not a normal distribution.\n')
end
```

```
RxX/RxY is not a normal distribution.
RxZ is not a normal distribution.
RxMass is not a normal distribution.
RxPWR is not a normal distribution.
Mtx is not a normal distribution.
MtxZ is not a normal distribution.
MtxMass is not a normal distribution.
MtxPWR is not a normal distribution.
STX is not a normal distribution.
STZ is not a normal distribution.
STMass is not a normal distribution.
STPWR is not a normal distribution.
```

ANOVA Test

```
Can not be run due to non-normal response distribution
```

Data Files for NIST DATAPlot

For a 8^5 factorial analysis these end up around 42MB per response variable

```
DesVarData = [Radius, Iy, Iz, ReflectanceFactor, SunIA, CoeffDrag, PointKnowledge, ResidualDipole];
%Reaction Wheel
Data = [3*RxX.*RxY.*RxZ/1000/1000/1000, DesVarData]';
fileID = fopen('ADCSRxVol.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);

Data = [3*RxMass, DesVarData]';
fileID =  fopen('ADCSRxMass.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);
Data = [3*RxPWR, DesVarData]';
fileID = fopen('ADCSRxPower.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);
%Star Tracker
Data = [STX.*STY.*STZ/1000/1000/1000, DesVarData]';
fileID = fopen('ADCSSTVol.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);

Data = [STMass, DesVarData]';
fileID = fopen('ADCSSTMass.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);

Data = [STPWR, DesVarData]';
fileID = fopen('ADCSSTPower.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);
%MagneticTorqueRod
Data = [2*MtxX.*MtxY.*MtxZ/1000/1000/1000, DesVarData]';
fileID = fopen('ADCSMagTorqueVol.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);

Data = [2*MtxMass, DesVarData]';
fileID = fopen('ADCSMagTorqueMass.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);

Data = [2*MtxPWR, DesVarData]';
fileID = fopen('ADCSMagTorquePower.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s\r\n','Y','X1','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f\r\n',Data);
fclose(fileID);
```

Orthogonality Verification

```
DesVars = {Radius,Iy, Iz, ReflectanceFactor, SunIA, CoeffDrag, PointKnowledge, ResidualDipole};
NUMFAC = length(DesVars);
orthocheck = zeros(NUMFAC);
for i = 1:NUMFAC
    for j = 1:NUMFAC
        orthocheck(i,j) = sum(Data(i+1,:).*Data(j+1,:));
        if i == j
            orthocheck(i,j) = 0;
        end

    end
end
disp(orthocheck)
```

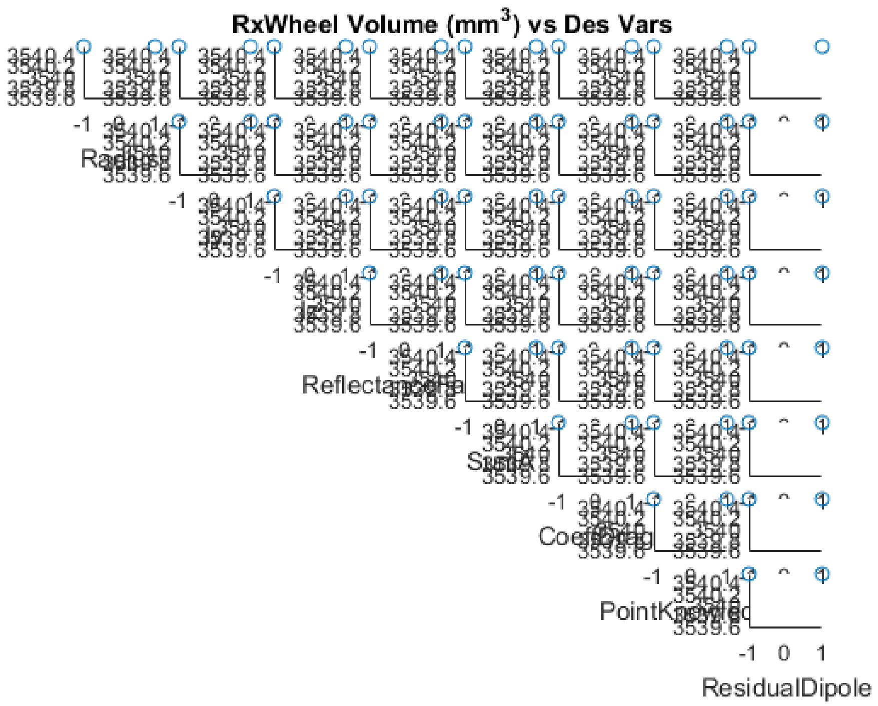
```
   1.0e-12 *

  Columns 1 through 7

         0    0.3182    0.0893         0         0         0   -0.2274
    0.3182         0   -0.2056    0.0056   -0.0016         0   -0.4545
    0.0893   -0.2056         0    0.0018    0.0018         0    0.3411
         0    0.0056    0.0018         0         0         0   -0.2274
         0   -0.0016    0.0018         0         0         0   -0.2274
         0         0         0         0         0         0         0
   -0.2274   -0.4545    0.3411   -0.2274   -0.2274         0         0
    0.0002    0.2269    0.2218    0.0002    0.0002         0   -0.4545

  Column 8

    0.0002
    0.2269
    0.2218
    0.0002
    0.0002
         0
   -0.4545
         0
```

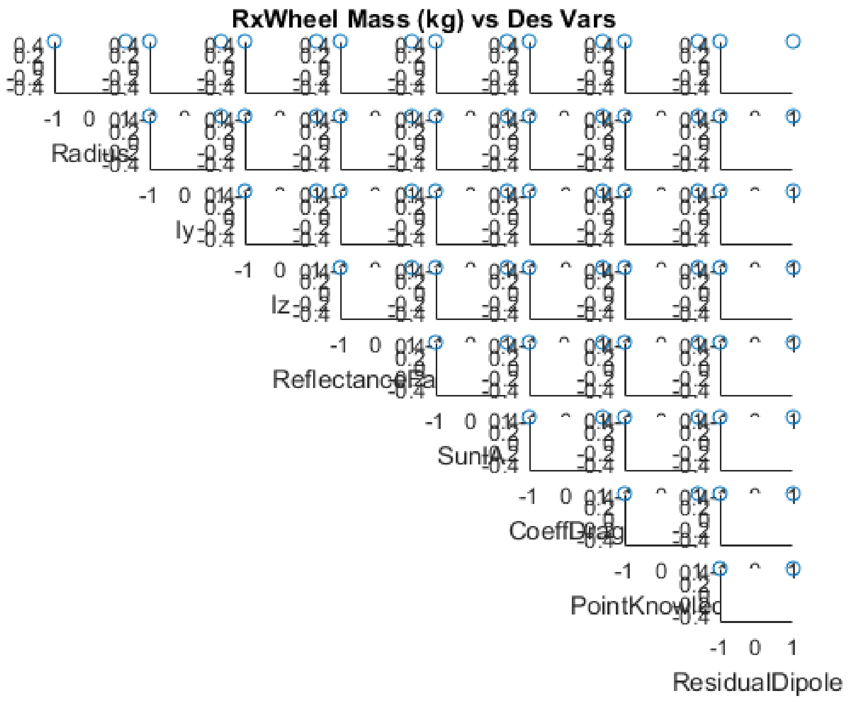
DOE Interaction Plot VERIFICATION FOR NIST

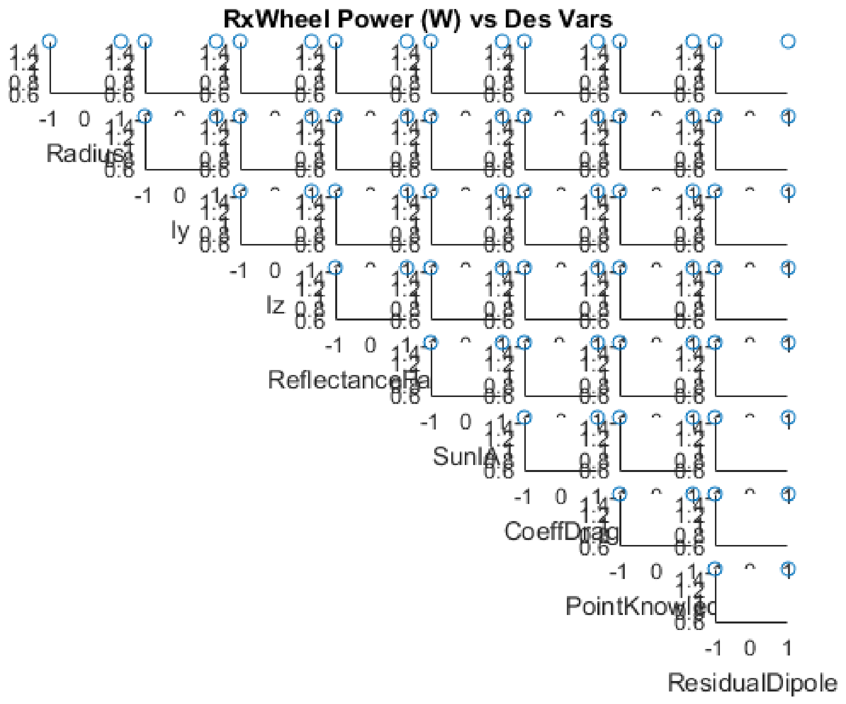
Reaction Wheel

```
Data(1,:) = 3*RxX.*RxY.*RxZ;
fRxVol = figure;
NUMFAC = length(DesVars);
DesVarNames = {'Radius','Iy', 'Iz','ReflectanceFactor','SunIA','CoeffDrag',...
    'PointKnowledge', 'ResidualDipole'};
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('RxWheel Volume (mm^3) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*RxMass;
fRxMass = figure;

for i = 1:NUMFAC
    for j = 1:NUMFAC
```

```
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('RxWheel Mass (kg) vs Des Vars');
a.Visible = 'off'; %  set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*RxPWR;
fRxPower = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('RxWheel Power (W) vs Des Vars');
a.Visible = 'off'; %  set(a,'Visible','off');
t1.Visible = 'on'; %  set(t1,'Visible','on');
```

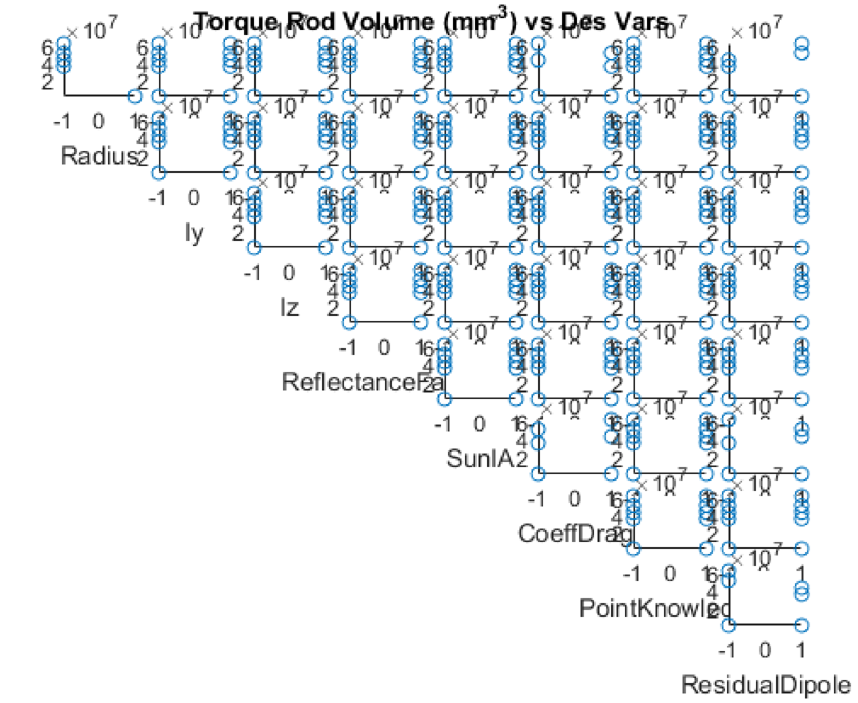

RxWheel Volume (mm^3) vs Des Vars
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole
3540.4
3540.2
3540
3539.8
3539.6
-1
0
1

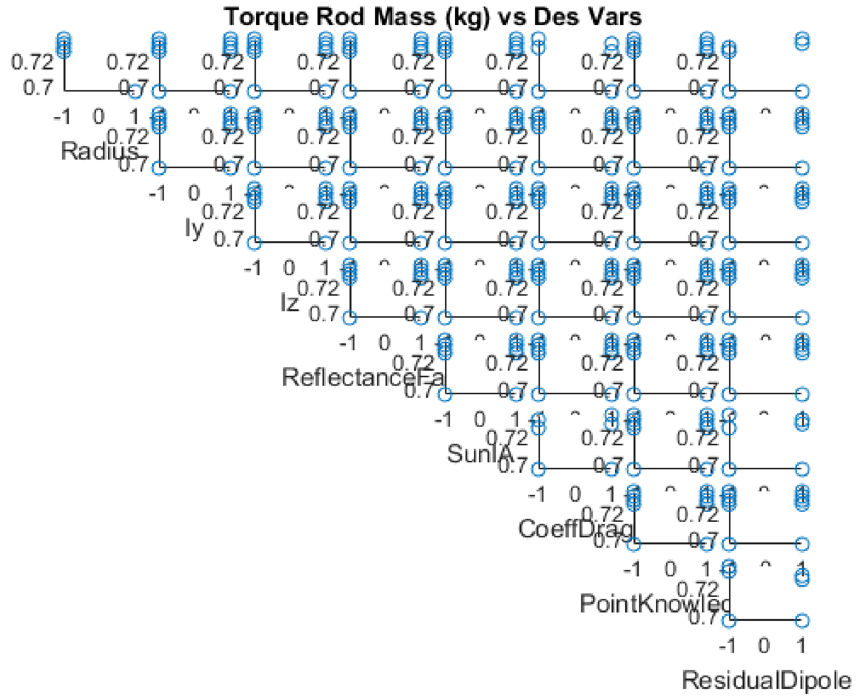

RxWheel Mass (kg) vs Des Vars
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole
0.4
0.2
0
-0.2
-0.4
-1
0
1

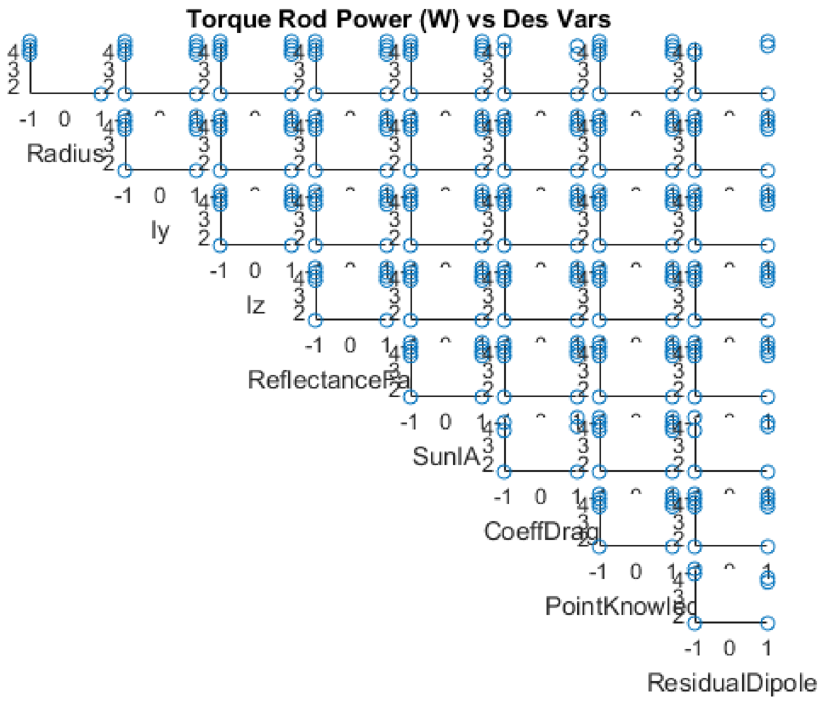


Magnetorquers

```
Data(1,:) = 2*MtxX.*MtxY.*MtxZ;
fRxVol = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Torque Rod Volume (mm^3) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 2*MtxMass;
fRxMass = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Torque Rod Mass (kg) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*MtxPWR;
fRxPower = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Torque Rod Power (W) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; %  set(t1,'Visible','on');
```

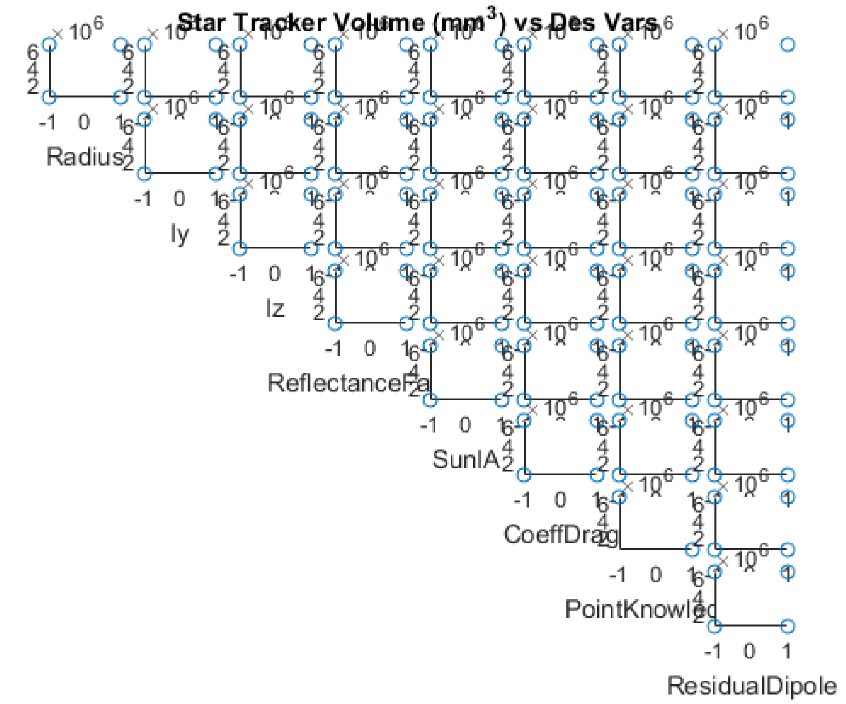

Torque Rod Volume (mm^3) vs Des Vars
× 10^7
6
4
2
-1 0 1
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole

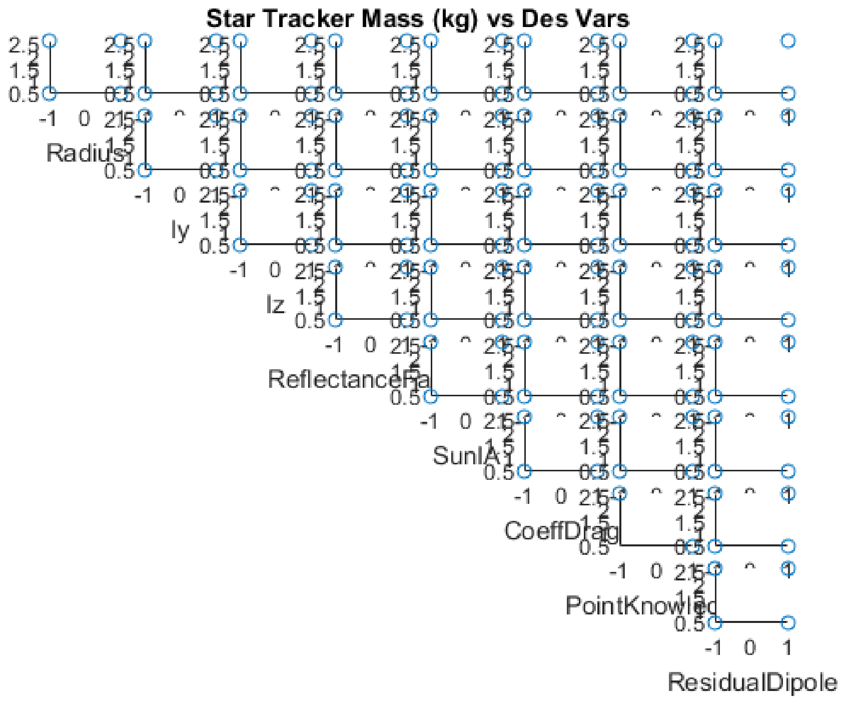

Torque Rod Mass (kg) vs Des Vars
0.72
0.7
-1 0 1
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole

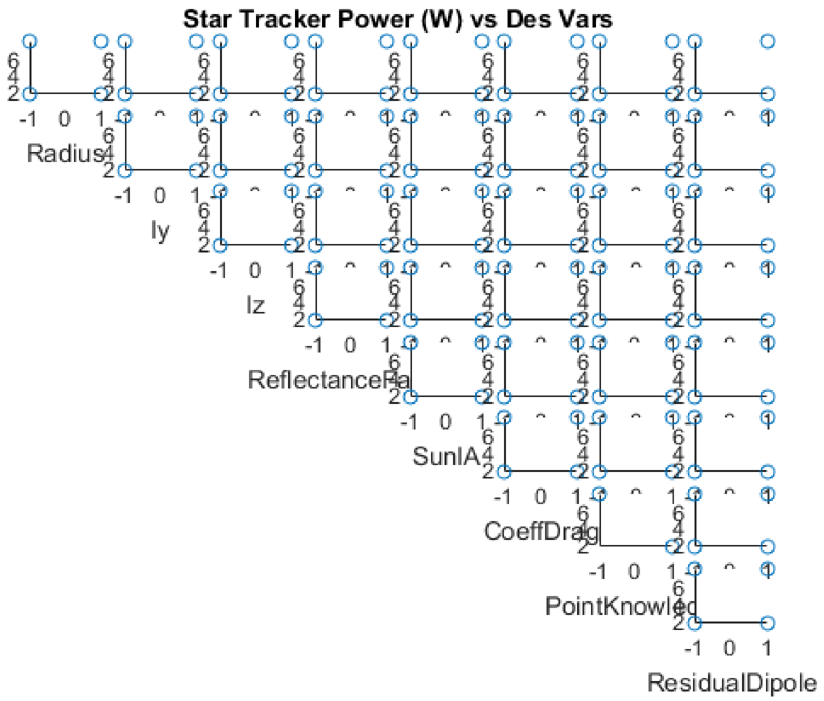


StarTracker

```
Data(1,:) = STX.*STY.*STZ;
fSTVol = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Star Tracker Volume (mm^3) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = STMass;
fRxMass = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Star Tracker Mass (kg) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = STPWR;
fRxPower = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(Data(i+1,:),Data(1,:))
            xlabel(varname)
            axis([-1 1 -inf inf])
        elseif j > i
            KVec = Data(i+1,:).*Data(j+1,:);
            %disp(min(KVec));
            %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
            scatter(KVec,Data(1,:))
            axis([-1 1 -inf inf])
        end
    end
end
a = axes;
t1 = title('Star Tracker Power (W) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; %  set(t1,'Visible','on');
```


Star Tracker Volume (mm^3) vs Des Vars
× 10^6
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole


Star Tracker Mass (kg) vs Des Vars
Radius
Iy
Iz
ReflectanceFa
SunIA
CoeffDrag
PointKnowled
ResidualDipole



Published with MATLAB® R2016a

Appendix D

CODES

D.1 Design of Experiments

D.1.1 Code Factor Level

```
function [Y,a,b] = CodeFactorLevel(U)
    a = (max(U)+min(U))/2;
        b = (max(U)−min(U))/2;
    Y = (U−a)/b;
end
```

D.1.2 Inverse Code Factor Level

```
function Y = InvCodeFactorLevel(U,a,b)
    %U is the coded values from CodeFactorLevel()
    %a is the constant provided by CodeFactorLevel()
    %b is the constant provided by CodeFactorLevel()
    Y = b*U+a;
end
```

D.1.3 DOE Payload

```
import numpy as np
import math

from openmdao.api import Component, IndepVarComp, Group, Component
    , Problem, ScipyOptimizer, ExecComp, DumpRecorder
```

```
from openmdao.drivers.fullfactorial_driver import
    FullFactorialDriver

class OrbitPeriod(Component):
    """ Evaluates the period for a circular orbit in min:1
        .658669e-04*(6378.14+Altitude)**(3/2) """

    def __init__(self):
        super(OrbitPeriod, self).__init__()
        self.add_param('Altitude', val=0.0)
        self.add_output('OrbPer',val=1.0)

    def solve_nonlinear(self, params, unknowns, resids):
        h1= params['Altitude']
        unknowns['OrbPer'] = 1.658669e-04*(6378.14+h1)**1.5

    def linearize(self, params, unknowns, resids):
        J = {}
        J['OrbPer', 'Altitude'] = 0.0002488*(6378.14+params['
            Altitude'])**.5
        return J

class GroundVelocity(Component):
    """ Evaluates the equation f(x,y) = 2*pi*Re/P"""
    def __init__(self):
        super(GroundVelocity, self).__init__()
        self.add_param('OrbPer',val=0.0)
        self.add_output('GroundVelocity',val=1.)

    def solve_nonlinear(self, params, unknowns, resids):
        OrbPer1 = params['OrbPer']
        unknowns['GroundVelocity'] = 2*math.pi*6378.14/OrbPer1/60

    def linearize(self, params, unknowns, resids):
        J = {}
        J['GroundVelocity', 'OrbPer'] = 40074.2496/params['OrbPer
            ']**2
        return J

class AngularRadius(Component):
    """Angular Radius as seen from spacecraft"""
    def __init__(self):
        super(AngularRadius, self).__init__()
        self.add_param('Altitude',val=0.0)
        self.add_output('AngRadius',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        h1 = params['Altitude']
        unknowns['AngRadius'] = math.asin(6378.14/(6378+h1))*180/
            math.pi #returns in degrees

    def linearize(self, params, unknowns, resids):
        J = {}
        J['AngRadius','Altitude'] = -(6378.14*((params['Altitude'
            ]+6378.14)**2)**.5)/((6378.14+params['Altitude'])**2*(
            params['Altitude']**2+12756.28*params['Altitude'])
            **.5)

class LNot(Component):
    """Angular Radius measured from the center of earth of the
       region seen by the spacecraft"""
    def __init__(self):
        super(LNot, self).__init__()
        self.add_param('AngRadius',val=0.0)
        self.add_output('LNot',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        AngRadius = params['AngRadius']
        unknowns['LNot'] = 90-AngRadius #returns in degrees

    def linearize(self, params, unknowns, resids):
        J = {}
        J['LNot','AngRadius'] = -1
        return J


class DMax(Component):
    """Solves for the distance to the horizon from the satellite
       """
    def __init__(self):
        super(DMax, self).__init__()
        self.add_param('LNot',val=0.0)
        self.add_output('DMax',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        LNot = params['LNot']
        unknowns['DMax'] = math.tan(LNot*math.pi/180)*6378.14 #
            returns km

    def linearize(self, params, unknowns, resids):
        J = {}
        J['DMax','LNot'] = 6378.14/(tan(params['LNot']*math.pi
            /180)**2)
        return J

class EtaLook(Component):
    """Nadir Angle Range"""
    def __init__(self):
        super(EtaLook, self).__init__()
        self.add_param('IAMax',val=0.0)
        self.add_param('AngRadius',val=0.0)
        self.add_output('EtaLook',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        IAMax = params['IAMax']
        AngRadius = params['AngRadius']
        unknowns['EtaLook'] = math.asin(math.cos((90-IAMax)*math.
            pi/180)*math.sin(AngRadius*math.pi/180))*180/math.pi #
            returns deg

    def linearize(self, params, unknowns, resids):
        J = {}
        J['EtaLook','IAMax'] = -(math.sin(params['IAMax']*math.pi
            /180) * math.sin(params['AngRadius']*math.pi/180))
            /((1-math.cos(params['IAMax']*math.pi/180)**2*math.sin
            (params['AngRadius']*math.pi/180)**2))**0.5
        J['EtaLook','AngRadius'] = math.cos(params['IAMax']*math.
            pi/180)*math.cos(params['AngRadius']*math.pi/180)/(1-
            math.cos(params['IAMax']*math.pi/180)**2*math.sin(
            params['AngRadius']*math.pi/180)**2)**0.5
        return J

class ECAMax(Component):
    def __init__(self):
        super(ECAMax, self).__init__()
        self.add_param('EtaLook',val=0.0)
        self.add_param('IAMax',val=0.0)
        self.add_output('ECAMax',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        EtaLook = params['EtaLook']
        IAMax = params['IAMax']
        unknowns['ECAMax'] = 90-(90-IAMax)-EtaLook #returns deg

    def linearize(self, params, unknowns, resids):
        J = {}
        J['ECAMax','IAMax'] = -1
        J['ECAMax','EtaLook'] =-1
        return J

class SlantRange(Component):
    def __init__(self):
        super(SlantRange, self).__init__()
        self.add_param('ECAMax',val=0.0)
        self.add_param('EtaLook',val=0.0)
        self.add_output('SlantRange',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        ECAMax= params['ECAMax']
        EtaLook = params['EtaLook']
        unknowns['SlantRange'] = 6378.14*math.sin(ECAMax*math.pi
            /180)/math.sin(EtaLook*math.pi/180) #returnskm

    def linearize(self, params, unknowns, resids):
        J = {}
        ECAMax= params['ECAMax']
        EtaLook = params['EtaLook']
        J['SlantRange','ECAMax'] = 6378.14*math.cos(ECAMax*math.
            pi/180)/math.sin(EtaLook*math.pi/180)
        J['SlantRange','EtaLook']= -6378.14*math.cos(ECAMax*math
            .pi/180)/math.sin(EtaLook*math.pi/180)/math.tan(
            EtaLook*math.pi/180)
        return J

class SwathWidth(Component):
    def __init__(self):
        super(SwathWidth, self).__init__()
        self.add_param('ECAMax',val=0.0)
        self.add_output('SwathWidth',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        ECAMax= params['ECAMax']
        unknowns['SwathWidth']= 2*ECAMax

    def linearize(self, params, unknowns, resids):
        J = {}
        J['SwathWidth','ECAMax']= 2
        return J

class IFOV(Component):
    def __init__(self):
        super(IFOV, self).__init__()
        self.add_param('YMax',val=0.0)
        self.add_param('SlantRange',val=0.0)
        self.add_output('IFOV',val=0.0)

    def solve_nonlinear(self,params,unknowns, resids):
        YMax = params['YMax']
        SlantRange = params['SlantRange']
        unknowns['IFOV'] = YMax/1000/SlantRange*180/math.pi #
            returns degrees

    def linearize(self, params, unknowns, resids):
        J = {}
        YMax= params['YMax']
        SlantRange = params['SlantRange']
        J['IFOV','YMax'] = 180*math.pi/SlantRange
        J['IFOV','SlantRange'] = -YMax*180/math.pi/(SlantRange
            **2)
        return J

class XMax(Component):
    def __init__(self):
        super(XMax, self).__init__()
        self.add_param('YMax',val=0.0)
        self.add_param('IAMax',val=0.0)
        self.add_output('XMax',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        YMax= params['YMax']
        IAMax = params['IAMax']
        unknowns['XMax'] = YMax/math.cos(IAMax*math.pi/180) #
            returns m

    def linearize(self, params, unknowns, resids):
        J = {}
        YMax = params['YMax']
        IAMax = params['IAMax']
        J['XMax','YMax'] = 1/math.cos(IAMax)
        J['XMax','IAMax'] = YMax*math.tan(IAMax)/math.cos(IAMax)
        return J

class CrossTrackPixelResolution(Component):
    """X in SMAD equation table p288 @nadir"""
    def __init__(self):
        super(CrossTrackPixelResolution, self).__init__()
        self.add_param('IFOV', val=0.0)
        self.add_param('Altitude', val=0.0)
        self.add_output('CrossTrackPixelResolution', val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        IFOV = params['IFOV']
        Altitude = params['Altitude']
        unknowns['CrossTrackPixelResolution'] = IFOV*Altitude*
            math.pi/180 #returns m

    def linearize(self, params, unknowns, resids):
        J = {}
        IFOV = params['IFOV']
        Altitude = params['Altitude']
        J['CrossTrackPixelResolution','IFOV'] = Altitude*math.pi
            /180
        J['CrossTrackPixelResolution','Altitude'] = IFOC*math.pi
```

```
            /180
        return J

class AlongTrackPixelResolution(Component):
    """Y in SMAD equation table p288 @ nadir"""
    def __init__(self):
        super(AlongTrackPixelResolution, self).__init__()
        self.add_param('IFOV',val=0.0)
        self.add_param('Altitude',val=0.0)
        self.add_output('AlongTrackPixelResolution',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        IFOV = params['IFOV']
        Altitude = params['Altitude']
        unknowns['AlongTrackPixelResolution'] = IFOV*Altitude*
            math.pi/180 #returns m

    def linearize(self, params, unknowns, resids):
        J = {}
        IFOV = params['IFOV']
        Altitude = params['Altitude']
        J['AlongTrackPixelResolution','IFOV'] = Altitude*math.pi
            /180
        J['AlongTrackPixelResolution','h'] = IFOC*math.pi/180
        return J

class CrossTrackPixelCount(Component):
    """ZC in SMAD equatin table p288"""
    def __init__(self):
        super(CrossTrackPixelCount, self).__init__()
        self.add_param('EtaLook',val=0.0)
        self.add_param('IFOV',val=0.0)
        self.add_output('CrossTrackPixelCount',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        IFOV = params['IFOV']
        EtaLook = params['EtaLook']
        unknowns['CrossTrackPixelCount'] = 2*EtaLook/IFOV #
            returns m

    def linearize(self, params, unknowns, resids):
        J = {}
        IFOV = params['IFOV']
        EtaLook = params['EtaLook']
        J['CrossTrackPixelCount','IFOV'] = -2*EtaLook/(IFOV**2)
        J['CrossTrackPixelCount','EtaLook'] = 2/IFOV
        return J

class SwathCount(Component):
    """ZA @ NADIR in SMAD Equation table p288"""
    def __init__(self):
        super(SwathCount, self).__init__()
        self.add_param('GroundVelocity',val=0.0)
        self.add_param('AlongTrackPixelResolution',val=0.0)
        self.add_output('SwathCount',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        GroundVelocity=params['GroundVelocity']
        AlongTrackPixelResolution= params['
            AlongTrackPixelResolution']
        unknowns['SwathCount'] = GroundVelocity/
            AlongTrackPixelResolution #Returns swaths at nadir per
             second

    def linearize(self, params, unknowns, resids):
        J = {}
        GroundVelocity=params['GroundVelocity']
        AlongTrackPixelResolution=params['
            AlongTrackPixelResolution']
        J['SwathCount','GroundVelocity']= 1/
            AlongTrackPixelResolution
        J['SwathCount','AlongTrackPixelResolution'] = -
            GroundVelocity/(AlongTrackPixelResolution**2)
        return J

class PixelRate(Component):
    def __init__(self):
        super(PixelRate, self).__init__()
        self.add_param('SwathCount',val=0.0)
        self.add_param('CrossTrackPixelCount',val=0.0)
        self.add_output('PixelRate',val=0.0)

    def solve_nonlinear(self,params,unknowns, resids):
        SwathCount = params['SwathCount']
        CrossTrackPixelCount = params['CrossTrackPixelCount']
        unknowns['PixelRate'] = SwathCount*CrossTrackPixelCount

    def linearize(self, params, unknowns, resids):
        J = {}
        SwathCount = params['SwathCount']
        AlongTrackPixelCount = params['CrossTrackPixelCount']
        J['PixelRate','SwathCount'] = CrossTrackPixelCount
        J['PixelRate','CrossTrackPixelCount'] = SwathCount
        return J

class DataRate(Component):
    def __init__(self):
        super(DataRate, self).__init__()
        self.add_param('PixelRate',val=0.0)
        self.add_param('BitPerPixel',val=0.0)
        self.add_output('DataRate',val=0.0)

    def solve_nonlinear(self,params,unknowns, resids):
        PixelRate = params['PixelRate']
        BitPerPixel = params['BitPerPixel']
        unknowns['DataRate'] = PixelRate*BitPerPixel

    def linearize(self, params, unknowns, resids):
        J = {}
        PixelRate= params['PixelRate']
        BitPerPixel= params['BitPerPixel']
        J['DataRate','BitPerPixel']= PixelRate
        J['DataRate','PixelRate']= BitPerPixel
        return J

class PixelIntegrationTime(Component):
    """ This must be higher than the time constant for the
        detector; used in optimizer"""
    def __init__(self):
        super(PixelIntegrationTime, self).__init__()
        self.add_param('AlongTrackPixelResolution', val=0.0)
        self.add_param('GroundVelocity', val=0.0)
        self.add_param('CrossTrackPixelCount', val=0.0)
        self.add_param('PixelInstrumentCount', val=0.0)
        self.add_output('PixelIntegrationTime', val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        AlongTrackPixelResolution = params['
            AlongTrackPixelResolution']
        GroundVelocity = params['GroundVelocity']
        CrossTrackPixelCount = params['CrossTrackPixelCount']
        PixelInstrumentCount = params['PixelInstrumentCount']
        unknowns['PixelIntegrationTime'] =
            AlongTrackPixelResolution*PixelInstrumentCount/
            GroundVelocity/CrossTrackPixelCount

    def linearize(self, params, unknowns, resids):
        J = {}
        AlongTrackPixelResolution = params['
            AlongTrackPixelResolution']
        GroundVelocity = params['GroundVelocity']
        CrossTrackPixelCount = params['CrossTrackPixelCount']
        PixelInstrumentCount = params['PixelInstrumentCount']
        J['PixelIntegrationTime','AlongTrackPixelResolution'] =
            PixelInstrumentCount/CrossTrackPixelCount/
            GroundVelocity
        J['PixelIntegrationTime','GroundVelocity'] = -
            AlongTrackPixelResolution*PixelInstrumentCount/(
            GroundVelocity**2)/CrossTrackPixelCount
        J['PixelIntegrationTime','CrossTrackPixelCount'] = -
            AlongTrackPixelResolution*PixelInstrumentCount/
            GroundVelocity/(CrossTrackPixelCount**2)
        J['PixelIntegrationTime','PixelInstrumentCount'] =
            AlongTrackPixelResolution/CrossTrackPixelCount/
            GroundVelocity
        return J

class FocalLength(Component):
    def __init__(self):
        super(FocalLength, self).__init__()
        self.add_param('Altitude', val=0.0)
        self.add_param('CrossTrackPixelResolution',val=0.0)
        self.add_param('DetWidth',val=0.0)
        self.add_output('FocalLength',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        Altitude = params['Altitude']
        DetWidth = params['DetWidth']
        CrossTrackPixelResolution = params['
            CrossTrackPixelResolution']
        unknowns['FocalLength'] = Altitude*DetWidth/
            CrossTrackPixelResolution

    def linearize(self, params, unknowns, resides):
        J = {}
        Altitude= params['Altitude']
        DetWidth = params['DetWidth']
        CrossTrackPixelResolution= params['
            CrossTrackPixelResolution']
        J['FocalLength','Altitude'] = DetWidth/
            CrossTrackPixelResolution
        J['FocalLength','DetWidth']= Altitude/
            CrossTrackPixelResolution
        J['FocalLength', 'CrossTrackPixelResolution'] = -Altitude
            *DetWidth/(CrossTrackPixelResolution**2)
        return J

class ApertureDiameter(Component):
    def __init__(self):
        super(ApertureDiameter, self).__init__()
        self.add_param('DetWidth',val=0.0)
        self.add_param('FocalLength',val=0.0)
        self.add_param('QualFactor',val=0.0)
        self.add_param('OpWavelength',val=0.0)
        self.add_output('ApertureDiameter',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        DetWidth = params['DetWidth']
        OpWavelength = params['OpWavelength']
        FocalLength = params['FocalLength']
        QualFactor = params['QualFactor']
        unknowns['ApertureDiameter'] = 2.44*OpWavelength*
            FocalLength*QualFactor/DetWidth

    def linearize(self, params, unknowns, resids):
        J = {}
        DetWidth = params['DetWidth']
        OpWavelength = params['OpWavelength']
        FocalLength = params['FocalLength']
        QualFactor = params['QualFactor']
        J['ApertureDiameter','DetWidth'] = -2.44*OpWavelength*
            FocalLength*QualFactor/(DetWidth**2)
        J['ApertureDiameter','OpWavelength'] = 2.44*FocalLength*
            QualFactor/DetWidth
        J['ApertureDiameter','FocalLength'] = 2.44*OpWavelength*
            QualFactor/DetWidth
        J['ApertureDiameter','QualFactor'] = 2.44*OpWavelength*
            FocalLength/DetWidth
        return J

class FOV(Component):
    def __init__(self):
        super(FOV, self).__init__()
        self.add_param('IFOV',val=0.0)
        self.add_param('PixelInstrumentCount',val=0.0)
        self.add_output('FOV',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        IFOV = params['IFOV']
        PixelInstrumentCount = params['PixelInstrumentCount']
        unknowns['FOV'] = IFOV*PixelInstrumentCount

    def linearize(self, params, unknowns, resids):
        J = {}
        J['FOV','IFOV'] = params['PixelInstrumentCount']
        J['FOV','PixelInstrumentCount'] = params['IFOV']
        return J

class PhysParams(Component):
    """Based of ThematicMapper scaling equations from SMAD ch 9
        """
    def __init__(self):
        super(PhysParams, self).__init__()
        self.add_param('ApertureDiameter',val=0.0)
        self.add_output('ApRat',val=0.0)
        self.add_output('XDim',val=0.0)
        self.add_output('YDim',val=0.0)
        self.add_output('ZDim',val=0.0)
        self.add_output('PwrEst',val=0.0)
        self.add_output('MassEst',val=0.0)

    def solve_nonlinear(self, params, unknowns, resids):
        ApertureDiameter= params['ApertureDiameter']
        Ratio= ApertureDiameter/0.015
        if Ratio <=0.5:
            K = 2
        else:
            K = 1
        unknowns['XDim'] = Ratio*0.045
        unknowns['YDim'] = Ratio*0.050
        unknowns['ZDim'] = Ratio*0.080
        unknowns['ApRat']=K
        unknowns['PwrEst'] = K*(Ratio**3)*1.26
        unknowns['MassEst'] = K*(Ratio**3)*0.23

class PayloadDesign(Group):
    def __init__(self):
        super(PayloadDesign, self).__init__()
        self.add('Altitude', IndepVarComp('Altitude',450.),
            promotes=['Altitude'])
        self.add('IAMax', IndepVarComp('IAMax',70.),promotes=['
            IAMax'])
        self.add('YMax', IndepVarComp('YMax',0.1),promotes=['
            YMax'])
        self.add('BitPerPixel',IndepVarComp('BitPerPixel',8.),
            promotes=['BitPerPixel'])
        self.add('PixelInstrumentCount', IndepVarComp('
            PixelInstrumentCount',200.),promotes=['
            PixelInstrumentCount'])
        self.add('DetWidth',IndepVarComp('DetWidth',30.),
            promotes=['DetWidth'])
        self.add('QualFactor', IndepVarComp('QualFactor',1.1),
            promotes=['QualFactor'])
        self.add('OpWavelength', IndepVarComp('OpWavelength', 4.2
            e-06),promotes=['OpWavelength'])

        self.add('d01',OrbitPeriod(),promotes=['Altitude','
            OrbPer'])
        self.add('d02', GroundVelocity(), promotes=['OrbPer', '
            GroundVelocity'])
        self.add('d03', AngularRadius(),promotes=['Altitude','
            AngRadius'])
        self.add('d04', LNot(),promotes=['LNot','AngRadius'])
        self.add('d05', DMax(),promotes=['LNot','DMax'])
        self.add('d06', EtaLook(),promotes=['EtaLook','IAMax','
            AngRadius'])
        self.add('d07', ECAMax(), promotes=['ECAMax', 'EtaLook','
            IAMax'])
        self.add('d08', SlantRange(), promotes=['ECAMax','EtaLook
            ','SlantRange'])
        self.add('d09', SwathWidth(),promotes=['SwathWidth','
            ECAMax'])
        self.add('d10', IFOV(),promotes=['YMax','SlantRange','
            IFOV'])
        self.add('d11', XMax(),promotes=['YMax','IAMax','XMax'])
        self.add('d12', CrossTrackPixelResolution(),promotes=['
            IFOV','Altitude','CrossTrackPixelResolution'])
        self.add('d13',AlongTrackPixelResolution(),promotes=['
            IFOV','Altitude','AlongTrackPixelResolution'])
        self.add('d14',CrossTrackPixelCount(),promotes=['
            EtaLook','IFOV','CrossTrackPixelCount'])
        self.add('d15', SwathCount(), promotes=['GroundVelocity',
            'AlongTrackPixelResolution','SwathCount'])
        self.add('d16', PixelRate(), promotes=['PixelRate','
            CrossTrackPixelCount','SwathCount'])
        self.add('d17', DataRate(), promotes=['PixelRate','
            BitPerPixel','DataRate'])
        self.add('d18',PixelIntegrationTime(), promotes=['
            AlongTrackPixelResolution','GroundVelocity','
            CrossTrackPixelCount','PixelInstrumentCount','
            PixelIntegrationTime'])
        self.add('d19',FocalLength(),promotes=['Altitude','
            DetWidth','CrossTrackPixelResolution','FocalLength'])
        self.add('d20',ApertureDiameter(),promotes=['
            OpWavelength','FocalLength','QualFactor','DetWidth','
            ApertureDiameter'])
        self.add('d21',FOV(), promotes=['IFOV','
            PixelInstrumentCount','FOV'])
        self.add('d22', PhysParams(), promotes=['ApertureDiameter
            ','XDim','YDim','ZDim','PwrEst','MassEst','ApRat'])

        self.add('obj_cmp',ExecComp('obj = ApertureDiameter',
            ApertureDiameter=0.0), promotes=['obj', '
            ApertureDiameter'])

top = Problem()
root = top.root = PayloadDesign()

#FIRESAT EXAMPLE
#top.driver = FullFactorialDriver(num_levels=2, num_par_doe=1,
    load_balance=False)
#top.driver.add_desvar('h',lower=700.0,upper=710)
#top.driver.add_desvar('IAMax', lower= 68., upper= 70.)
#top.driver.add_desvar('YMax', lower= 67,upper = 68.0)
#top.driver.add_desvar('BitPerPixel',lower=8.,upper = 16.)

#top.driver = FullFactorialDriver(num_levels=5, num_par_doe=1,
    load_balance=False) #For use in full DOE
top.driver = FullFactorialDriver(num_levels=5, num_par_doe=1,l
    oad_balance=False) #for use to find DOE factor generation
top.driver.add_desvar('Altitude',lower=300,upper=450)
top.driver.add_desvar('IAMax', lower = 50., upper = 77.)
top.driver.add_desvar('YMax', lower = 100, upper = 1000)
top.driver.add_desvar('BitPerPixel',lower=8.,upper= 16.)
top.driver.add_desvar('PixelInstrumentCount', lower=200., upper =
    300.)
top.driver.add_desvar('DetWidth',lower= 2.0e-6,upper = 40.0e
    -6)
top.driver.add_desvar('QualFactor',lower=1.1,upper =2.0)
top.driver.add_desvar('OpWavelength', lower = 3.0e-06, upper =
    17.0e-06)

top.driver.add_objective('obj')
#recorder = DumpRecorder('DOEPayload')#For use in full DOE
recorder = DumpRecorder('DOEPayload')#increase I to see changes
    between DOE desvar iter
recorder.options['record_params'] = False
recorder.options['record_unknowns'] = True
recorder.options['record_resids'] = False
recorder.options['excludes']= ['OrbPer','GroundVelocity','LNot',
    'DMax','SlantRange','AngRadius','ECAMax','EtaLook','SwathWidth
    ','XMax','CrossTrackPixelResolution','
    AlongTrackPixelResolution','CrossTrackPixelCount','SwathCount'
    ,'PixelRate','FocalLength']
top.driver.add_recorder(recorder)

top.setup()
top.run()

top.cleanup()
```

D.1.4 Payload DOE Analysis

```
%%Data Analysis for DOEs
clc; clear all; close all;
%% REGEX TEST
% PARABOLOID TEST
%fxy = regexp(text,'fxy:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
    tokens'); %value of function
%fxy= str2double([fxy{:}]');
%xparams = regexp(text,'comp¥.x:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))'
    ,'tokens')';
%xparams = str2double([xparams{:}]');
%yparams = regexp(text,'comp¥.y:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))'
    ,'tokens')';
%yparams = str2double([yparams{:}]');
%%Payload Parsing RegExpress
%VarName = regexp(text,'VarName:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))
    ','tokens');
%VarName= str2double([VarName{:}]');
% Design Vars: Altitude, BitPerPixel, DetWidth, IAMax,
    OpWavelength,
% PixelInstrumentCount, QualFactor
%% READ Payload Data file
if exist('PayloadDOEData.mat', 'file')==2
    load('PayloadDOEData.mat') %Saves a few minutes if the data
```

```
        has been parsed and saved and analyzed
else
    fid  = fopen('DOEPayload','r');
    text = textscan(fid,'%s','Delimiter','','endofline','');
    text = text{1}{1};
    fid  = fclose(fid);
    Altitude = regexp(text,'Altitude:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥
        d*))','tokens');
    AlongTrackGroundSampling = regexp(text,'YMax:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    ApertureDiameter = regexp(text,'ApertureDiameter:¥s+(¥d*(.)
        ?(¥d*)?(e)?[+-]?(¥d*))','tokens');
    BitPerPixel = regexp(text,'BitPerPixel:¥s+(¥d*(.)?(¥d*)?(e)
        ?[+-]?(¥d*))','tokens');
    DataRate = regexp(text,'DataRate:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥
        d*))','tokens');
    DetWidth = regexp(text,'DetWidth:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥
        d*))','tokens');
    FOV = regexp(text,'[^I]+FOV:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))'
        ,'tokens');%modified to nt contain IFOV
    IAMax = regexp(text,'IAMax:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))',
        'tokens');
    IFOV = regexp(text,'IFOV:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    MassEst = regexp(text,'MassEst:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d
        *))','tokens');
    OpWavelength = regexp(text,'OpWavelength:¥s+(¥d*(.)?(¥d*)?(e)
        ?[+ −]?(¥d ∗))','tokens');
    PixelInstrumentCount = regexp(text,'PixelInstrumentCount:¥s
        +(¥d ∗ (.)?(¥d ∗)?(e)?[+ −]?(¥d ∗))','tokens');
    PixelIntegrationTime = regexp(text,'PixelIntegrationTime:¥s
        +(¥d ∗ (.)?(¥d ∗)?(e)?[+ −]?(¥d ∗))','tokens');
    PwrEst =regexp(text,'PwrEst:¥s+(¥d ∗(.)?(¥d∗)?(e)?[+−]?(¥d∗))'
        ,'tokens');
    QualFactor = regexp(text,'QualFactor:¥s+(¥d ∗(.)?(¥d∗)?(e)
        ?[+ −]?(¥d ∗))','tokens');
    XDim = regexp(text,'XDim:¥s+(¥d ∗(.)?(¥d∗)?(e)?[+ −]?(¥d ∗))','
        tokens');
    YDim = regexp(text,'YDim:¥s+(¥d ∗(.)?(¥d∗)?(e)?[+ −]?(¥d ∗))','
        tokens');
    ZDim = regexp(text,'ZDim:¥s+(¥d ∗(.)?(¥d∗)?(e)?[+ −]?(¥d ∗))','
        tokens');
    KRatio = regexp(text,'ApRat:¥s+(¥d ∗(.)?(¥d∗)?(e)?[+ −]?(¥d ∗))'
        ,'tokens');
    clear text fid%Remove file to clear up memory
    %%Convert to Usable Doubles
    Altitude = str2double([Altitude{:}]');
    AlongTrackGroundSampling = str2double([
        AlongTrackGroundSampling{:}]');
    ApertureDiameter = str2double([ApertureDiameter{:}]');
    BitPerPixel = str2double([BitPerPixel{:}]');
    DataRate = str2double([DataRate{:}]');
    DetWidth = str2double([DetWidth{:}]');
    FOV = str2double([FOV{:}]');
```

```
IAMax = str2double([IAMax{:}]');
IFOV = str2double([IFOV{:}]');
MassEst = str2double([MassEst{:}]');
OpWavelength = str2double([OpWavelength{:}]');
PixelInstrumentCount = str2double([PixelInstrumentCount{:}]')
    ;
PixelIntegrationTime = str2double([PixelIntegrationTime{:}]')
    ;
PwrEst = str2double([PwrEst{:}]');
QualFactor = str2double([QualFactor{:}]');
XDim = str2double([XDim{:}]');
YDim = str2double([YDim{:}]');
ZDim = str2double([ZDim{:}]');
KRatio = str2double([KRatio{:}]');
%%Remove NaN caused by metadata and parameter saving
Altitude = Altitude(~isnan(Altitude));
[Altitude, AltitudeA, AltituudeB] = CodeFactorLevel(Altitude)
    ;
AlongTrackGroundSampling = AlongTrackGroundSampling(~isnan(
    AlongTrackGroundSampling));
[AlongTrackGroundSampling, AtgsA, AtgsB] = CodeFactorLevel(
    AlongTrackGroundSampling);
ApertureDiameter = ApertureDiameter(~isnan(ApertureDiameter))
    ;
%[ApertureDiameter, ApDiamA, ApDiamB] = CodeFactorLevel(
    ApertureDiameter);
BitPerPixel = BitPerPixel(~isnan(BitPerPixel));
```

```
    [BitPerPixel, BitPerPixelA, BitPerPixelB] = CodeFactorLevel(
        BitPerPixel);
    DetWidth = DetWidth(~isnan(DetWidth));
    [DetWidth, DetWidthA, DetWidthB] = CodeFactorLevel(DetWidth);
    DataRate = DataRate(~isnan(DataRate));
    %[DataRate, DataRateA, DataRateB] = CodeFactorLevel(DataRate)
        ;
    FOV = FOV(~isnan(FOV));
    %[FOV, FovA, FovB] = CodeFactorLevel(FOV);
    IAMax = IAMax(~isnan(IAMax));
    [IAMax, IAMaxA, IAMaxB] = CodeFactorLevel(IAMax);
    IFOV = IFOV(~isnan(IFOV));
    %[IFOV, IfovA, ifovB] = CodeFactorLevel(IFOV);
    MassEst = MassEst(~isnan(MassEst));
    %[MassEst, MassEstA, MassEstB] = CodeFactorLevel(MassEst);
    OpWavelength = OpWavelength(~isnan(OpWavelength));
    [OpWavelength, OpwavelengthA, OpWavelengthB] =
        CodeFactorLevel(OpWavelength);
    PixelInstrumentCount = PixelInstrumentCount(~isnan(
        PixelInstrumentCount));
    [PixelInstrumentCount, PicA, PicB] = CodeFactorLevel(
        PixelInstrumentCount);
    PixelIntegrationTime= PixelIntegrationTime(~isnan(
        PixelIntegrationTime));
    %[PixelIntegrationTime, PitA, PitB] = CodeFactorLevel(
        PixelIntegrationTime);
    PwrEst = PwrEst(~isnan(PwrEst));
    %[PwrEst, PwrEstA, PwrEstB] = CodeFactorLevel(PwrEst);
    QualFactor = QualFactor(~isnan(QualFactor));
    [QualFactor, QualFactorA, QualFactorB] = CodeFactorLevel(Q
        ualFactor);
    XDim = XDim(~isnan(XDim));
    %[XDim, XdA, XdB] = CodeFactorLevel(XDim);
    YDim = YDim(~isnan(YDim));
    %[YDim, YdA, YdB] = CodeFactorLevel(YDim);
    ZDim = ZDim(~isnan(ZDim));
    %[ZDim, ZdA, ZdB] = CodeFactorLevel(ZDim);
    KRatio=KRatio(~isnan(KRatio));
    save('PayloadDOEData.mat')
    %%ANOVA Test
    DesVars = {Altitude, AlongTrackGroundSampling, BitPerPixel,
        DetWidth, IAMax, ...
         OpWavelength, PixelInstrumentCount, QualFactor};
    DesVarNames = {'Altitude','AlongTrackGroundSampling', '
        BitPerPixel',...
         'DetWidth', 'IAMax','OpWavelength', 'PixelInstrumentCount
             ', 'QualFactor'};
    [DRAnovaP, DRAnovaTbl, DRAnovaStat]  = anovan(DataRate,
        DesVars,'model'...
         ,'interaction','varnames',DesVarNames);
    [ADAnovaP, ADAnovaTbl, ADAnovaStat] = anovan(ApertureDiameter
        ,DesVars,...
         'model','interaction','varnames',DesVarNames);
    [PWRAnovaP, PWRAnovaTbl, PWRAnovaStat] = anovan(PwrEst,
        DesVars,'model',...
        'interaction','varnames',DesVarNames);
    [PITAnovaP,PITAnovaTbl,PITAnovaStat] = anovan(
        PixelIntegrationTime,...
        DesVars,'model','interaction','varnames',DesVarNames);
    save('PayloadDOEData.mat')
end
%% Data Test
%For a 8^5 factorial analysis these end up around 42MB per
    response
%variable
Data = [PixelIntegrationTime,Altitude,AlongTrackGroundSampling,
    BitPerPixel,DetWidth,IAMax,OpWavelength,PixelInstrumentCount,
    QualFactor]';%Only for Data2
fileID= fopen('PayloadPIT.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s¥r¥n','Y','X1
    ','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
     %9.8f¥r¥n',Data);
fclose(fileID);
%% DOE Scatter Plots (for comparing to DATAPLOT from NIST)
% figure %POWER ESTIMATE
%scatter(Altitude, PwrEst)
%title('Altitude Response')
%
%figure
% scatter(AlongTrackGroundSampling, PwrEst)
% title('YMax Response')
%
% figure
% scatter(BitPerPixel, PwrEst)
% title('BitPerPixel Response')
%
% figure
% scatter(DetWidth, PwrEst)
% title('DetWidth Response')
%
% figure
% scatter(IAMax, PwrEst)
% title('IAMax Response')
%
% figure
% scatter(OpWavelength, PwrEst)
% title('OpWavelength Response')
%
% figure
% scatter(PixelInstrumentCount, PwrEst)
% title('PixelQtyInstResponse')
%
% figure
% scatter(QualFactor, PwrEst)
% title('QualFactorResponse')
%%Normality Tests
%
```

```
% Uses the Shaprio Wilk to test normality of variables from DOE.
% Coded vectors are all [−1 −0.5 0 0.5 1]'
%Altitude
testvec = [−1 −0.5 0 0.5 1]'; %This is the same as the coded
    levels for all sets
testvec= InvCodeFactorLevel(testvec, AltitudeA, AltituudeB);
[H, PVal, WStatistic]=swtest(testvec);
if H== 0
    fprintf('Altitude is a normal distribution with PVal %d and
       Wstat %d.¥n',...
        PVal, WStatistic)
else
    fprintf('Altitude is not a normal distribution.¥n')
end
% Along Track Ground Sample
testvec=[−1 −0.5 0 0.5 1]';
testvec= InvCodeFactorLevel(testvec, AtgsA, AtgsB);
[H, PVal, WStatistic]=swtest(testvec);
if H== 0
      fprintf('ATGS is a normal distribution with PVal %d and Wstat
        %d.¥n',...
        PVal, WStatistic)
else
      fprintf('ATGS is not a ncrmal distribution.¥n')
end
% Bit Per Pixel
  testvec=[−1 −0.5 0 0.5 1]';
```

```
testvec = InvCodeFactorLevel(testvec ,BitPerPixelA , BitPerPixelB);
[H, PVal, WStatistic] = swtest(testvec);
if H==0
      fprintf('BPP is a normal distribution with PVal %d and Wstat
          %d.¥n',...
            PVal, WStatistic)
else
      fprintf('BPP is not a normal distribution.¥n')
end
% DetWidth
testvec=[−1 −0.5 0 0.5 1]';
testvec= InvCodeFactorLevel(testvec ,DetWidthA, DetWidthB);
[H, PVal, WStatistic]=swtest(testvec);
if H== 0
      fprintf('DetWidth is a normal distribution with PVal %d     and
          Wstat %d.¥n',...
            PVal, WStatistic)
else
      fprintf('DetWidth is not a normal distribution.¥n')
end
% IAMax
testvec=[−1 −0.5 0 0.5 1]';
testvec=InvCodeFactorLevel(testvec,IAMaxA, IAMaxB);
[H, PVal, WStatistic]=swtest(testvec);;
if H== 0
      fprintf('IAMax is a normal distribution with PVal %d and
          Wstat %d.¥n',...
```

```
            PVal, WStatistic)
else
    fprintf('IAMax is not a normal distribution.¥n')
end
%Operational Wavelength
testvec=[-1 -0.5 0 0.5 1]';
testvec= InvCodeFactorLevel(testvec,OpwavelengthA, OpWavelengthB
    );
[H, PVal, WStatistic] = swtest(testvec);
if H==0
    fprintf('Op Wavelength is a normal distribution with PVal %d
        and Wstat %d.¥n',...
            PVal, WStatistic)
else
    fprintf('Op Wavelength is not a normal distribution.¥n')
end
% PixelInstrument Count
testvec=[-1 -0.5 0 0.5 1]';
testvec= InvCodeFactorLevel(testvec,PicA,PicB);
[H, PVal,WStatistic]=swtest(testvec);
if H == 0
    fprintf('PixelInst Ct is a normal distribution with PVal %d
        and Wstat %d.¥n',...
            PVal, WStatistic)
else
    fprintf('PixelInst Ct is not a normal distribution.¥n')
end
```

```
% Quality Factor
testvec= [−1 −0.5 0 0.5 1]';
testvec= InvCodeFactorLevel(testvec,QualFactorA, QualFactorB);
[H, PVal, WStatistic]= swtest(testvec);
if H == 0
      fprintf('QualFactor is a normal distribution with PVal %d and
            Wstat %d.¥n',...
            PVal, WStatistic)
else
      fprintf('QualFactor is not a normal distribution.¥n')
end
% DataRate
[H, PVal, KSSTAT, cv] = kstest(DataRate,'alpha',0.1);
if H == 0
         fprintf('DataRate is a normal distribution with PVal %d and
          Wstat %d.¥n',...
            PVal, WStatistic)
else
      fprintf('DataRate is not a normal distribution.¥n')
end

%%Residual Plots vs Design Var (DATA RATE)
figure
scatter(Altitude,DRAnovaStat.resid)
title('Altitude vs Data Rate Residuals')
xlabel('Altitude')
ylabel('DR Residual')
```

```
figure
scatter(AlongTrackGroundSampling,DRAnovaStat.resid)
title('ATGS vs Data Rate Residuals')
xlabel(' Along Track Ground Sampling ')
ylabel('DR Residual')
figure
scatter(BitPerPixel,DRAnovaStat.resid)
title('BPP vs Data Rate Residuals')
xlabel('Bit Per Pixel')
ylabel('DR Residual')
figure
scatter(DetWidth,DRAnovaStat.resid)
title('Detector Width vs Data Rate Residuals')
xlabel('Detector Width')
ylabel('DR Residual')
figure
scatter(IAMax,DRAnovaStat.resid)
title('Max Incidence Angle vs Data Rate Residuals')
xlabel('IA_{max}')
ylabel('DR Residual')
figure
scatter(OpWavelength,DRAnovaStat.resid)
title('Operational Wavelength vs Data Rate Residuals')
xlabel('OpWavelength')
ylabel('DR Residual')
figure
scatter(PixelInstrumentCount,DRAnovaStat.resid)
```

```
title('PIC vs Data Rate Residuals')
xlabel('PixelInstrument Count')
ylabel('DR Residual')
figure
scatter(QualFactor,DRAnovaStat.resid)
title('QualFactor vs Data Rate Residuals')
xlabel('QualFactor')
ylabel('DR Residual')
%% ResidualPlots
figure
plot([1:1:length(DRAnovaStat.resid)],DRAnovaStat.resid)
title('Data Rate Residual')
xlabel('Observation#')
ylabel('DR Residual')
figure
LagPlotVec = zeros(length(DRAnovaStat.resid),1);
for i=2:length(DRAnovaStat.resid)
    LagPlotVec(i)=DRAnovaStat.resid(i-1);
end
scatter(LagPlotVec,DRAnovaStat.resid)
title('Data Rate Residual Lag Plot')
xlabel('DR Residual(Lag{1})')
ylabel('DR Residual')
%%
figure
histogram(DRAnovaStat.resid)
title('Histogram of Data Rate Residuals')
```

```
xlabel('Residual')
ylabel('Count')
ResidualVec=sort(DRAnovaStat.resid);
n = length(DRAnovaStat.resid);
NormOrdStatMed=[1:1:n]';
NormOrdStatMed(end)= 0.5^(1/n);
NormOrdStatMed(1)=1- 0.5^(1/n);
for i=2:n
    NormOrdStatMed(i) =(i-0.3175)/(n+0.365);
end
figure
plot(NormOrdStatMed,ResidualVec)
title('DistributionofResiduals')
xlabel('CDF')
ylabel('Residual')
%%
figure
probplot('exponential',DataRate)
figure
histfit(DataRate)
kstest(boxcox(DataRate))
%% OrthogonalityVerification
NUMFAC= length(DesVars);
orthocheck=zeros(NUMFAC);
for i=1:NUMFAC
    for j=1:NUMFAC
        orthocheck(i,j) = sum(Data(i+1,:).*Data(j+1,:));
```

```
            if  i == j
                orthocheck(i,j)= 0;
            end

        end
end
disp(orthocheck)
%% DOE Interaction Plot VERIFICATION FOR NIST (PIT)
% fPIT = figure;
% NUMFAC= length(DesVars);
% DesVarNames = {'Altitude','ATGS', 'Bit/Pixel',...
%       'DetWidth', 'IAMax','OpWave', 'PixelCount', 'QualFac'};
% for i = 1:NUMFAC
%       for j = 1:NUMFAC
%           if  i == j
%               varname = cellstr(DesVarNames{i});
%               subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
%               scatter(Data(i+1,:),Data(1,:))
%               xlabel(varname)
%               axis([-1 1 -inf inf])
%           elseif j > i
%               KVec = Data(i+1,:).*Data(j+1,:);
%               %disp(min(KVec));
%               %disp(max(KVec));
%               subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
%               scatter(KVec,Data(1,:))
%               axis([-1 1 -inf inf])
%               end
%         end
% end
% a = axes;
% t1 = title('Pixel Integration Time vs Des Vars');
% a.Visible='off';%set(a,'Visible','off');
% t1.Visible='on';%set(t1,'Visible','on');
%% Verify ifsatatements work in DOE. [It does]
% ApTest = ApertureDiameter;
% K= zeros(length(ApTest),1);
% for i=1:length(ApTest)
%         if ApTest(i) < 0.5
%               K(i)=2;
%         else
%               K(i)=1;
%         end
% end
```

D.1.5 DOE ADCS

```
import numpy as np
import math

from openmdao.api import Component, IndepVarComp, Group, Component
    , Problem, ScipyOptimizer, ExecComp, DumpRecorder
from openmdao.drivers.fullfactorial_driver import
    FullFactorialDriver

```

```
## Estimate denviornmentaleffects
 class GravGradient(Component):
     """ Evaluates gravity gradient p366"""

     def __init__(self):
         super(GravGradient, self).__init__()
         self.add_param('Radius', val=0.0) #Orbit Radius in meters
             (Re+ Alt)
         self.add_param('Iz', val= 0.0) #Moment of inertia about
            Z axis
         self.add_param('Iy', val = 0.0) #Moment of inertia about
            y axis
         self.add_param('IncidenceAngle', val= 0.0) #max
            deviation of z axis from local vertical in radians
         self.addoutput('Gravity Gradient',val=1.0)

     def solve_nonlinear(self, params, unknowns, resids):
         mu = 3.986e14 #m^3/s^2
         R = params['Radius']
         Iz = params['Iz']
         Iy = params['Iy']
         Theta = params['IncidenceAngle']
         unknowns['GravityGradient'] = 3*mu/2/(R*R*R)*abs(Iz-Iy)*
            math.sin(2*Theta)

 class SolarRadiation(Component):
     def __init__(self):
         super(SolarRadiation,self).__init__()
         self.add_param('CenterSolarPressure',val=0.0)
         self.add_param('CenterGravity',val=0.0)
         self.add_param('SurfaceArea',val=0.0)
         self.add_param('ReflectanceFactor',val=0.0)
         self.add_param('SunIA',val=0.0)
         self.add_output('SolarRadiation',val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         CPS= params['CenterSolarPressure']
         CG= params['CenterGravity']
         SA = params['SurfaceArea']
         q= params['ReflectanceFactor']
         i=params ['SunIA']
         Fs = 1367 #W/m^2
         c = 3e8
         unknowns['SolarRadiation']= (Fs/c*SA*(1+q)*math.cos(i))
            *(CPS-CG)

 class MagneticField(Component):
     def __init__(self):
         super(MagneticField,self).__init__()
         self.add_param('Radius',val=0.0) #mfrom center of earth
         self.add_param('ResidualDipole', val= 0.0) #Am^2
         self.add_output('MagneticField', val= 0.0) #Nm

     def solve_nonlinear(self, params, unknowns, resids):
         M = 7.96e15 #Tesla m^3
         D = params['ResidualDipole']
         R = params['Radius']
         unknowns['MagneticField'] = (2*M)/(R*R*R)*D

 class Density(Component):
     def __init__(self):
         super(Density, self).__init__()
         self.add_param('Radius', val = 0.0) #Radius from center
            of earth
         self.add_output('Density', val=0.0) #Outputs density

     def solve_nonlinear(Self, params, unknowns, resids):
         R = params['Radius']/1000 - 6378#convert km
         atmos = np.array([[0,0.00,1.23,7.25],
                         [25, 25.00, 3.899e-2, 6.35],
                         [30, 30.00, 1.774e-2, 6.68],
                         [40, 40.00, 3.972e-3, 7.55],
                         [50, 50.00, 1.057e-3, 8.38],
                         [60, 60.00, 3.206e-4, 7.71],
                         [70, 70.00, 8.770e-5, 6.55],
                         [80, 80.00, 1.905e-5, 5.80],
                         [90, 90.00, 3.396e-6, 5.38],
                         [100, 100.00,5.297e-7, 5.88],
                         [110, 110.00,9.661e-8, 7.26],
                         [120, 120.00,2.438e-8, 9.47],
                         [130, 130.00,8.484e-9, 12.64],
                               [140, 140.00, 3.845e−9, 16.15],
                               [150, 150.00, 2.070e−9, 22.52],
                               [180, 180.00, 5.464e−10, 29.74],
                               [200, 200.00, 2.789e−10, 37.11],
                               [250, 250.00, 7.248e−11, 45.55],
                               [300, 300.00, 2.418e−11, 53.63],
                               [350, 350.00, 9.518e−12, 53.30],
                               [400, 400.00, 3.725e−12, 58.52],
                               [450, 450.00, 1.585e−12, 60.83],
                               [500, 500.00, 6.967e−13, 63.82],
                               [600, 600.00, 1.454e−13, 71.84],
                               [700, 700.00, 3.614e−14, 88.67],
                               [800, 800.00, 1.170e−14, 124.64],
                               [900, 900.00, 5.245e−15, 181.05],
                               [1000, 1000.00, 3.019e−15, 268.00]])
         for i in range(27):
            if R >= atmos[i,0] and R < atmos[i+1,0]:
                H = atmos[i,3]
                rhon = atmos[i,2]
                base = atmos[i,1]
            elif R >= atmos[i+1,0]:
                H = atmos[i+1,3]
                rhon = atmos[i+1,2]
                base = atmos[i+1,1]
         unknowns['Density'] = rhon*math.exp(−(R−base)/H)

 class AerodynamicTorque(Component):
     def __init__(self):
         super(AerodynamicTorque,self).__init__()
         self.add_param('Radius',val=0.0)
         self.add_param('Density',val=0.0)
         self.add_param('CoeffDrag',val=0.0)
         self.add_param('SurfaceArea',val=0.0)
         self.add_param('CenterGravity',val=0.0)
         self.add_param('CenterPressure',val=0.0)
         self.add_output('AerodynamicTorque',val=0.04)

     def solve_nonlinear(self, params, unknowns, resids):
         R = params['Radius']
         rho = params['Density']
         Cd = params['CoeffDrag']
         SA = params['SurfaceArea']
         vel = math.sqrt(3.986e14/R) #Assumes circ orbit for
             initial
         F = 0.5*rho*Cd*SA*vel*vel
         CenterGravity = params['CenterGravity']
         CenterPressure = params['CenterPressure']
         unknowns['AerodynamicTorque'] = F*(CenterPressure-
             CenterGravity)

 class DistubranceTorque(Component):
     def __init__(self):
         super(DistubranceTorque, self).__init__()
         self.add_param('AerodynamicTorque',val=0.0)
         self.add_param('GravGradient', val= 0.0)
         self.add_param('MagneticField', val=0.0)
         self.add_param('SolarRadiation', val=0.0)
         self.add_output('DisturbanceTorque', val= 0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         A=params['AerodynamicTorque']
         G=params['GravGradient']
         M= params['MagneticField']
         S= params['SolarRadiation']
         unknowns['DisturbanceTorque']= A + G + M + S

 class OrbitPeriod(Component):
     """ Evaluates the period for a circular orbit in min:1
         .658669e-04*(6378.14+Altitude)**(3/2)"""

     def __init__(self):
         super(OrbitPeriod, self).__init__()
         self.add_param('Radius', val=0.0)
         self.add_output('OrbPer', val=1.0)

     def solve_nonlinear(self, params, unknowns, resids):
         r=params['Radius']
         unknowns['OrbPer']= 1.658669e-04*(r/1000)**1.5


 class SlewTorque(Component):
     def __init__(self):
         super(SlewTorque,self).__init__()
         self.add_param('Iz',val=0.0) # kg-m2
         self.add_param('SlewMaxDeg',val=0.0) #deg from slew rate
         self.add_param('SlewMaxTime', val=0.0) #time (sec) from s
             lew rate
         self.add_output('SlewTorque', val=0.0)

     def solve_nonlinear(self, params,  unknowns, resids):
         I = params['Iz']
         Theta = params['SlewMaxDeg']
         tau = params['SlewMaxTime']
         unknowns['SlewTorque'] = 4*Theta*math.pi/180*I/tau/tau

 class MomentumStorageRx(Component):
     def __init__(self):
         super(MomentumStorageRx, self).__init__()
         self.add_param('DisturbanceTorque', val=0.0)
         self.add_param('OrbPer',val=0.0)
         self.add_output('MomentumStorageRx', val=0.0)

     def solve_nonlinear(self, params, unknowns,  resids):
         TD= params['DisturbanceTorque']
         P = params['OrbPer']
         unknowns['MomentumStorageRx'] = TD * P/4 * 0.707

 class MomentumStorageMW(Component):
     def __init__(self):
         super(MomentumStorageMW, self).__init__()
         self.add_param('DisturbanceTorque', val=0.0)
         self.add_param('OrbPer', val=0.0)
         self.add_param('YawAcc', val=0.0)
         self.add_output('MomentumStorageMW', val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         TD= params['DisturbanceTorque']
         P = params['OrbPer']
         ThetaA = params['YawAcc']
         unknowns['MomentumStorageMW']= TD * P / 4 / ThetaA

 class MomentumSpinnerOmega(Component):
     def __init__(self):
         super(MomentumSpinnerOmega,self).__init__()
         self.add_param('MomentumStorageMW', val=0.0)
         self.add_param('Iz',val=0.0)
         self.add_output('MomentumSpinnerOmega',val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         h = params['MomentumStorageMW']
         I = params['Iz']
         unknowns['MomentumSpinnerOmega']= h / I

 class MagDipole(Component):
     def __init__(self):
         super(MagDipole, self).__init__()
         self.add_param('DisturbanceTorque', val=0.0)
         self.add_param('MagneticField', val=0.0)
         self.add_output('MagDipole', val = 0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         T = params['DisturbanceTorque']
         B = params['MagneticField']
         unknowns['MagDipole']=1.5*T/B

 class RxPhysParams(Component):
     """based on ADCS regression"""
     def __init__(self):
         super(RxPhysParams,self).__init__()
         self.add_param('MomentumStorageRx', val=0.0)
         self.add_output('RxX',val=0.0)
         self.add_output('RxY',val=0.0)
         self.add_output('RxZ',val=0.0)
         self.add_output('RxPWR',val=0.0)
         self.add_output('RxMass',val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         """ One Rx Wheel Dim"""
         H = params['MomentumStorageRx']
         if H <= 0.015: #Smallest found reaction wheel
             H = 0.015
         unknowns['RxX'] = 20.55*math.log(H)+120.4
         unknowns['RxY'] = 20.21*math.log(H)+118.0
         unknowns['RxZ'] = 23.61*math.log(H)+100.2
         unknowns['RxPWR'] = 0.466*H + .5106
         unknowns['RxMass'] = 1.666*H+.1216

 class MGTQRPhysParams(Component):
     """ One Magnetic Torque Rod Dim"""
     def __init__(self):
         super(MGTQRPhysParams, self).__init__()
         self.add_param('MagDipole',val=0.0)
         self.add_output('MtxX',val=0.0)
         self.add_output('MtxY',val=0.0)
         self.add_output('MtxZ',val=0.0)
         self.add_output('MtxPWR',val=0.0)
         self.add_output('MtxMass',val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         """ Total Volume Dimensions not the  physical
             configuration"""
         D = params['MagDipole']
         unknowns['MtxX'] = 0.1216 + 10.87*D
         unknowns['MtxY'] = 118.0 + 3.363*D
         unknowns['MtxZ'] = 100.2 + 26.67*D
         unknowns['MtxPWR']=.0502*D+.399
         unknowns['MtxMass'] = .001029*D+.3457

 class STPhysParams(Component):
     """based on ADCS regression"""
     def __init__(self):
         super(STPhysParams,self).__init__()
         self.add_param('PointKnowledge',val=0.0)
         self.add_output('STX',val=0.0)
         self.add_output('STY',val=0.0)
         self.add_output('STZ',val=0.0)
         self.add_output('STPWR',val=0.0)
         self.add_output('STMass',val=0.0)

     def solve_nonlinear(self, params, unknowns, resids):
         """ Total Volume Dimensions not the physical
             configuration"""
         o=params['PointKnowledge']
         unknowns['STX'] = -6071* o +196.3
         unknowns['STY'] = -6500* o +200.3
         unknowns['STZ']= -1.536e4 * o+387
         unknowns['STPWR']= -4.592e4 * o * o+750* o+5
         unknowns['STMass'] = -7296*o*o+ 31.79*o+ 2.735

 class ADCSDesign(Group):
     def __init__(self):
         super(ADCSDesign, self).__init__()
         #Input Variables based on previous systems
         self.add('Radius', IndepVarComp('Radius',7078.0),
             promotes=['Radius'])
         self.add('Iz', IndepVarComp('Iz',100.),promotes=['Iz'])
         self.add('Iy', IndepVarComp('Iy', 100.), promotes=['Iy'])
         self.add('IncidenceAngle', IndepVarComp('IncidenceAngle',
             0.0), promotes=['IncidenceAngle'])
         self.add('CenterSolarPressure', IndepVarComp('
             CenterSolarPressure', 0.0), promotes=['
             CenterSolarPressure'])
         self.add('CenterGravity', IndepVarComp('CenterGravity',
             0.0), promotes=['CenterGravity'])
         self.add('ReflectanceFactor', IndepVarComp('
             ReflectanceFactor',0.0), promotes=['ReflectanceFactor'
             ])
         self.add('SunIA', IndepVarComp('SunIA',0.0), promotes=['
             SunIA'])
     self.add('ResidualDipole',IndepVarComp('ResidualDipole'
         ,0.0), promotes=['ResidualDipole'])
     self.add('CenterPressure',IndepVarComp('CenterPressure',
         0.0), promotes=['CenterPressure'])
         self.add('SurfaceArea', IndepVarComp('SurfaceArea',0.0),
             promotes=['SurfaceArea'])
         self.add('CoeffDrag', IndepVarComp('CoeffDrag',0.0),
             promotes=['CoeffDrag'])
         self.add('PointKnowledge', IndepVarComp('PointKnowledge'
             ,0.0), promotes=['PointKnowledge'])
         #Design equations
         self.add('d01', GravGradient(), promotes=['Radius','Iz','
             Iy','IncidenceAngle','GravityGradient'])
         self.add('d02',SolarRadiation(), promotes=['
             CenterSolarPressure','CenterGravity','
             ReflectanceFactor','SunIA','SolarRadiation'])
         self.add('d03',MagneticField(), promotes=['Radius','R
             esidualDipole','MagneticField'])
         self.add('d04',Density(),promotes=['Radius','Density'])
         self.add('d05', AerodynamicTorque(), promotes=['Radius','
             Density','CoeffDrag','SurfaceArea','CenterGravity','
             CenterPressure','AerodynamicTorque'])
         self.add('d06',DistubranceTorque(),promotes=['
             DisturbanceTorque','AerodynamicTorque','GravGradient',
             'SolarRadiation','MagneticField'])
         self.add('d07', OrbitPeriod(),  promotes=['Radius','OrbPer
             '])
         self.add('d08', MomentumStorageRx(), promotes=['OrbPer','
             DisturbanceTorque','MomentumStorageRx']) #dont  use
             spinners or momentum wheels for this ADCS
         self.add('do9', MagDipole(),  promotes=['DisturbanceTorque
             ','MagneticField','MagDipole'])
         self.add('d10',RxPhysParams(),promotes =['RxX','RxY','
             RxZ','RxPWR','RxMass','MomentumStorageRx'])
         self.add('d11', MGTQRPhysParams(), promotes =['MtxX','MtxY
             ','MtxZ','MtxPWR','MtxMass','MagDipole'])
         self.add('d12',STPhysParams(),promotes =['STX','STY','
             STZ','STPWR','STMass','PointKnowledge'])
         self.add('obj_cmp',ExecComp('obj = DisturbanceTorque',
             DisturbanceTorque=0.0), promotes=['obj', '
             DisturbanceTorque'])
```

```
top = Problem ( )
 root = top . root = ADCSDesign()

#FIRESAT EXAMPLE
top . driver = FullFactorialDriver(num_levels=2, num_par_doe=1,l
    oad_balance=False)
top . driver . add_desvar('Radius', lower=6500.e3, upper=7200.e3) #
    meters
top . driver . add_desvar('Iz',lower=1.7e−3,upper =1.8e−3) #kg^
    m2, see http://www.leodium.ulg.ac.be/cmsms/uploads/08−09
    _Pierlot.pdf
top . driver . add_desvar('Iy',lower= 1.9e−3,upper = 2.1e−3) #kg m
    ^2 see http://www.leodium.ulg.ac.be/cmsms/uploads/08−09
    _Pierlot.pdf
top . driver . add_desvar('IncidenceAngle', lower=0., upper = 0.) #
    rad about z axis
top . driver . add_desvar('CenterSolarPressure',lower=.03, upper =
    .03) #meter
top . driver . add_desvar('CenterGravity', lower= 0., upper = 0.) #
    meter
top . driver . add_desvar('ReflectanceFactor',lower=0.5,upper =
    0.8) #0−1 typ o.6
top . driver . add_desvar('SunIA',lower= 0., upper= 10.) #rad
top . driver . add_desvar('ResidualDipole',lower= 1.,upper = 1.2)
    #Am^2
top . driver . add_desvar('CenterPressure',lower= 0.02,upper =
    0.02) #m from center see
top.driver.add_desvar('SurfaceArea', lower= .0294, upper =
    .0294) #m^2 see https://digitalcommons.usu.edu/cgi/viewcontent
    .cgi?article=1074&context=smallsat
top.driver.add_desvar('CoeffDrag', lower = 2.0, upper = 2.2) #
    dimensionless
top.driver.add_desvar('PointKnowledge', lower= .007, upper =
    .02) #degrees

top.driver.add_objective('obj')
recorder = DumpRecorder('DOE_ACDS')#
recorder.options['record_params'] = False
recorder.options['record_unknowns'] = True
recorder.options['record_resids'] = False
#recorder.options['excludes'] = ['OrbPer', 'Lnot']
top.driver.add_recorder(recorder)

top.setup()
top.run()

top.cleanup()
```

D.1.6 ADCS DOE Analysis

```
%%Data Analysis for DOEs
clc; clear all; close all;
%% READ Payload Data file
if exist('ADCSDOEData.mat', 'file')==2
```

```
    load('ADCSDOEData.mat') %Saves a few minutes if the data has
        been parsed and saved and analyzed
else
    fid  = fopen('DOE.ACDS','r');
    text = textscan(fid,'%s','Delimiter','','endofline','');
    text = text{1}{1};
    fid  =fclose(fid);
    AerodynamicTorque = regexp(text,'AerodynamicTorque:¥s+(¥d*(.)
        ?(¥d*)?(e)?[+-]?(¥d*))','tokens');
    CenterGravity = regexp(text,'CenterGravity:¥s+(¥d*(.)?(¥d*)?(
        e)?[+-]?(¥d*))','tokens');
    CenterPressure = regexp(text,'CenterPressure:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    CenterSolarPressure = regexp(text,'CenterSolarPressure:¥s+(¥d
        *(.)?(¥d*)?(e)?[+-]?(¥d*))','tokens');
    CoeffDrag = regexp(text,'CoeffDrag:¥s+(¥d*(.)?(¥d*)?(e)
        ?[+-]?(¥d*))','tokens');
    Density = regexp(text,'Density:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d
        *))','tokens');
    DisturbanceTorque = regexp(text,'DisturbanceTorque:¥s+(¥d*(.)
        ?(¥d*)?(e)?[+-]?(¥d*))','tokens'); %modified to nt contain
         IFOV
    GravityGradient = regexp(text,'GravityGradient:¥s+(¥d*(.)?(¥d
        *)?(e)?[+-]?(¥d*))','tokens');
    IncidenceAngle = regexp(text,'IncidenceAngle:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    Iy = regexp(text,'Iy:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    Iz = regexp(text,'Iz:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    MagDipole = regexp(text,'MagDipole:¥s+(¥d*(.)?(¥d*)?(e)
        ?[+-]?(¥d*))','tokens');
    MagneticField = regexp(text,'MagneticField:¥s+(¥d*(.)?(¥d*)?(
        e)?[+-]?(¥d*))','tokens');
    MomentumStorageRx = regexp(text,'MomentumStorageRx:¥s+(¥d*(.)
        ?(¥d*)?(e)?[+-]?(¥d*))','tokens');
    MtxMass = regexp(text,'MtxMass:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d
        *))','tokens');
    MtxPWR = regexp(text,'MtxPWR:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))
        ','tokens');
    MtxX = regexp(text,'MtxX:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    MtxY = regexp(text,'MtxY:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    MtxZ = regexp(text,'MtxZ:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    OrbPer =regexp(text,'OrbPer:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))'
        ,'tokens');
    PointKnowledge = regexp(text,'PointKnowledge:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    Radius = regexp(text,'Radius:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))
        ','tokens');
    ReflectanceFactor = regexp(text,'ReflectanceFactor:¥s+(¥d*(.)
        ?(¥d*)?(e)?[+-]?(¥d*))','tokens');
    ResidualDipole = regexp(text,'ResidualDipole:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    RxMass = regexp(text,'RxMass:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))
        ','tokens');
    RxPWR = regexp(text,'RxPWR:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))',
        'tokens');
    RxX = regexp(text,'RxX:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    RxY = regexp(text,'RxY:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    RxZ = regexp(text,'RxZ:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    STMass = regexp(text,'STMass:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))
        ','tokens');
    STPWR = regexp(text,'STPWR:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))',
        'tokens');
    STX = regexp(text,'STX:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    STY = regexp(text,'STY:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    STZ = regexp(text,'STZ:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))','
        tokens');
    SolarRadiation = regexp(text,'SolarRadiation:¥s+(¥d*(.)?(¥d*)
        ?(e)?[+-]?(¥d*))','tokens');
    SunIA = regexp(text,'SunIA:¥s+(¥d*(.)?(¥d*)?(e)?[+-]?(¥d*))',
        'tokens');
    SurfaceArea = regexp(text,'SurfaceArea:¥s+(¥d*(.)?(¥d*)?(e)
        ?[+ −]?(¥d ∗))','tokens');
    clear text fid%Remove file to clear up memory
    %%Convert to Usable Doubles
    AerodynamicTorque = str2double([AerodynamicTorque{:}]');
    CenterGravity = str2double([CenterGravity{:}]');
    CenterPressure = str2double([CenterPressure{:}]');
    CenterSolarPressure = str2double([CenterSolarPressure{:}]');
    CoeffDrag = str2double([CoeffDrag{:}]');
    Density = str2double([Density{:}]');
    DisturbanceTorque = str2double([DisturbanceTorque{:}]');
    GravityGradient = str2double([GravityGradient{:}]');
    IncidenceAngle = str2double([IncidenceAngle{:}]');
    Iy = str2double([Iy{:}]');
    Iz = str2double([Iz{:}]');
    MagDipole = str2double([MagDipole{:}]');
    MagneticField = str2double([MagneticField{:}]');
    MomentumStorageRx = str2double([MomentumStorageRx{:}]');
    MtxMass = str2double([MtxMass{:}]');
    MtxPWR = str2double([MtxPWR{:}]');
    MtxX = str2double([MtxX{:}]');
    MtxY = str2double([MtxY{:}]');
    MtxZ = str2double([MtxZ{:}]');
    OrbPer = str2double([OrbPer{:}]');
    PointKnowledge = str2double([PointKnowledge{:}]');
    Radius = str2double([Radius{:}]');
    ReflectanceFactor = str2double([ReflectanceFactor{:}]');
    ResidualDipole = str2double([ResidualDipole{:}]');
```

```
    RxMass =str2double([RxMass{:}]');
    RxPWR= str2double([RxPWR{:}]');
    RxX = str2double([RxX{:}]');
    RxY = str2double([RxY{:}]');
    RxZ = str2double([RxZ{:}]');
    STMass= str2double([STMass{:}]');
    STPWR= str2double([STPWR{:}]');
    STX = str2double([STX{:}]');
    STY = str2double([STY{:}]');
    STZ = str2double([STZ{:}]');
    SolarRadiation= str2double([SolarRadiation{:}]');
    SunIA = str2double([SunIA{:}]');
    SurfaceArea= str2double([SurfaceArea{:}]');
    %%Remove NaN caused by metadata and parameter saving
    AerodynamicTorque=AerodynamicTorque(~isnan(
        AerodynamicTorque));
    CenterGravity= CenterGravity(~isnan(CenterGravity));
    CenterPressure= CenterPressure(~isnan(CenterPressure));
    CenterSolarPressure= CenterSolarPressure(~isnan(
        CenterSolarPressure));
    Density = Density(~isnan(Density));
    CoeffDrag= CoeffDrag(~isnan(CoeffDrag));
    DisturbanceTorque= DisturbanceTorque(~isnan(
        DisturbanceTorque));
    GravityGradient= GravityGradient(~isnan(GravityGradient));
    IncidenceAngle= IncidenceAngle(~isnan(IncidenceAngle));
    Iy= Iy(~isnan(Iy));
    Iz= Iz(~isnan(Iz));
    MagDipole = MagDipole (~isnan(MagDipole ));
    MagneticField= MagneticField(~isnan(MagneticField));
    MomentumStorageRx = MomentumStorageRx (~isnan(
        MomentumStorageRx ));
    MtxMass = MtxMass (~isnan( MtxMass ));
    MtxPWR = MtxPWR(~isnan(MtxPWR));
    MtxX = MtxX(~isnan(MtxX));
    MtxY = MtxY(~isnan(MtxY));
    MtxZ = MtxZ (~isnan(MtxZ ));
    OrbPer = OrbPer (~isnan( OrbPer ));
    PointKnowledge = PointKnowledge(~isnan(PointKnowledge));
    Radius  = Radius (~isnan(Radius ));
    ReflectanceFactor= ReflectanceFactor(~isnan(
        ReflectanceFactor));
    ResidualDipole= ResidualDipole(~isnan(ResidualDipole));
    RxMass = RxMass (~isnan( RxMass ));
    RxPWR = RxPWR(~isnan(RxPWR));
    RxX = RxX(~isnan(RxX));
    RxY = RxY(~isnan(RxY));
    RxZ = RxZ(~isnan(RxZ));
    STMass = STMass (~isnan( STMass ));
    STPWR = STPWR(~isnan(STPWR));
    STX = STX(~isnan(STX));
    STY = STY(~isnan(STY));
    STZ = STZ(~isnan(STZ));
    SolarRadiation= SolarRadiation(~isnan(SolarRadiation));
```

```
        SunIA = SunIA (~isnan(SunIA));
        SurfaceArea = SurfaceArea(~isnan(SurfaceArea));
        save('ADCSDOEData.mat')
end
%% Design Variable Orthogonolization
%
%   Design variables include radius, Iy, Iz, Reflectance            Factor,
    SunIA, Cd,
%   Pt Knowledge
%
%
[Radius, RadiusA, RadiusB] = CodeFactorLevel(Radius);
[Iy,  IyA,  IyB]  = CodeFactorLevel(Iy);
[Iz,  IzA,  IzB]  = CodeFactorLevel(Iz);
[ReflectanceFactor, RFA, RFB] = CodeFactorLevel(ReflectanceFactor
    );
[SunIA, SIAA, SIAB] = CodeFactorLevel(SunIA);
[CoeffDrag, CDA, CDB]= CodeFactorLevel(CoeffDrag);
[PointKnowledge, PKA, PKB]= CodeFactorLevel(PointKnowledge);
[ResidualDipole, RDA, RDB]= CodeFactorLevel(ResidualDipole);
%% Response Variable Test of Normality
[H, PVal, WStatistic]=kstest(RxX);
if H == 0
        fprintf('RxX/RxY is a normal distribution with PVal %d and
            Wstat %d.¥n',...
              PVal, WStatistic)
else
```

```
    fprintf('RxX/RxY is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(RxZ);
if H==0
    fprintf('RxZ is a normal distribution with PVal %d and Wstat
        %d.¥n',...
        PVal, WStatistic)
else
    fprintf('RxZ is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(RxMass);
if H==0
    fprintf('RxMass is a normal distribution with PVal %d and
        Wstat %d.¥n',...
        PVal, WStatistic)
else
    fprintf('RxMass is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(RxPWR);
if H==0
    fprintf('RxPWR is a normal distribution with PVal %d and
        Wstat %d.¥n',...
        PVal, WStatistic)
else
    fprintf('RxPWR is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(MtxX);
```

```
if H==0
      fprintf('MtX is a normal distribution with PVal %d and Wstat
          %d.¥n',...
            PVal, WStatistic)
else
      fprintf('Mtx is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(MtxZ);
if H==0
      fprintf('MtxZ is a normal distribution with PVal %d and Wstat
          %d.¥n',...
            PVal, WStatistic)
else
      fprintf('MtxZ is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(MtxMass);
if H==0
      fprintf('MtxMass is a normal distribution with PVal %d and
          Wstat %d.¥n',...
            PVal, WStatistic)
else
      fprintf('MtxMass is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(MtxPWR);
if H==0
      fprintf('MtxPWR is a normal distribution with PVal %d and
          Wstat %d.¥n',...
```

```
            PVal, WStatistic)
else
      fprintf('MtxPWR is not a normal distribution.¥n')
end
[H, PVal, WStatistic]=kstest(STX);
if H == 0
      fprintf('STX is a normal distribution with PVal %d      and  Wstat
         %d.¥n',...
            PVal, WStatistic)
else
       fprintf('STX is not a normal distribution.¥n')
end
[H, PVal, WStatistic]=kstest(STZ);
if H == 0
         fprintf('STZ is a normal distribution with PVal  %d  and  Wstat
         %d.¥n',...
            PVal, WStatistic)
else
      fprintf('STZ is not a normal distribution.¥n')
end
[H, PVal, WStatistic] = kstest(STMass);
if H == 0
      fprintf('STMass is a normal distribution with PVal %d and
         Wstat %d.¥n',...
            PVal, WStatistic)
else
      fprintf('STMass is not a normal distribution.¥n')
```

```
end
[H, PVal, WStatistic] = kstest(STPWR);
if H == 0
    fprintf('STPWR is a normal distribution with PVal %d and
        Wstat %d.¥n',...
        PVal, WStatistic)
else
    fprintf('STPWR is not a normal distribution.¥n')
end
%% ANOVA Test
%
%  Can not be run due to non-normal response distribution
%

%% Data Files for NIST DATAPlot
%
% For a 8^5 factorial analysis these end up around 42MB per
    response
% variable
%
DesVarData = [Radius, Iy, Iz, ReflectanceFactor, SunIA, CoeffDrag
    , PointKnowledge, ResidualDipole];
%Reaction Wheel
Data = [3*RxX.*RxY.*RxZ/1000/1000/1000, DesVarData]';
fileID= fopen('ADCSRxVol.dat','w');
fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s¥r¥n','Y','X1
    ','X2','X3','X4','X5','X6','X7','X8');
```

```
fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
    %9.8f¥r¥n',Data);
fclose(fileID);

Data = [3 * RxMass, DesVarData]';
fileID = fopen('ADCSRxMass.dat','w');
fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
    ','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
    %9.8f¥r¥n',Data);
fclose(fileID);
Data = [3 *RxPWR, DesVarData]';
fileID = fopen('ADCSRxPower.dat','w');
fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
    ','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
    %9.8f¥r¥n',Data);
fclose(fileID);
%StarTracker
Data  =[STX.*STY.*STZ/1000/1000/1000,DesVarData]';
fileID= fopen('ADCSSTVol.dat','w');
fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
    ','X2','X3','X4','X5','X6','X7','X8');
fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
    %9.8f¥r¥n',Data);
fclose(fileID);

```

```
Data = [ STMass , DesVarData ]';
 fileID = fopen('ADCSSTMass.dat','w');
 fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
     ','X2','X3','X4','X5','X6','X7','X8');
 fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
       %9.8f¥r¥n',Data);
 fclose(fileID);

Data = [STPWR, DesVarData ]';
 fileID = fopen('ADCSSTPower.dat','w');
 fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
     ','X2','X3','X4','X5','X6','X7','X8');
 fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
       %9.8f¥r¥n',Data);
 fclose(fileID);
%MagneticTorqueRod
 Data = [2*MtxX.*MtxY.* MtxZ/1000/1000/1000,DesVarData]';
 fileID= fopen('ADCSMagTorqueVol.dat','w');
 fprintf(fileID,'%12s %9s  %9s  %9s  %9s  %9s  %9s  %9s  %9s¥r¥n','Y','X1
     ','X2','X3','X4','X5','X6','X7','X8');
 fprintf(fileID ,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
       %9.8f¥r¥n',Data);
 fclose(fileID);

Data = [2 * MtxMass, DesVarData]';
 fileID = fopen('ADCSMagTorqueMass.dat','w');
 fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s¥r¥n','Y','X1
     ','X2','X3','X4','X5','X6','X7','X8');
 fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
       %9.8f¥r¥n',Data);
 fclose(fileID);

Data = [2*MtxPWR, DesVarData]';
 fileID = fopen('ADCSMagTorquePower.dat','w');
 fprintf(fileID,'%12s %9s %9s %9s %9s %9s %9s %9s %9s¥r¥n','Y','X1
     ','X2','X3','X4','X5','X6','X7','X8');
 fprintf(fileID,'%12.12f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f %9.8f
       %9.8f¥r¥n',Data);
 fclose(fileID);

%% Orthogonality Verification
 DesVars = {Radius,Iy,Iz,ReflectanceFactor,SunIA,CoeffDrag,
     PointKnowledge,ResidualDipole};
NUMFAC= length(DesVars);
 orthocheck=zeros(NUMFAC);
 for i=1:NUMFAC
       for j=1:NUMFAC
           orthocheck(i,j)=sum(Data(i+1,:).*Data(j+1,:));
           if i==j
               orthocheck(i,j)= 0;
           end

       end
end
```

```
disp(orthocheck)
%% DOE Interaction Plot VERIFICATION FOR NIST
%%Reaction Wheel
Data(1,:)= 3*RxX.*RxY.*RxZ;
fRxVol=figure;
NUMFAC= length(DesVars);
DesVarNames ={'Radius','Iy', 'Iz','ReflectanceFactor','SunIA','
    CoeffDrag',...
     'PointKnowledge', 'ResidualDipole'};
for i=1:NUMFAC
     for j=1:NUMFAC
          if i==j
               varname =cellstr(DesVarNames{i});
          subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
               scatter(Data(i+1,:),Data(1,:))
               xlabel(varname)
               axis([-1 1 -inf inf])
          elseif j>i
               KVec=Data(i+1,:).*Data(j+1,:);
               %disp(min(KVec));
               %disp(max(KVec));
          subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
               scatter(KVec,Data(1,:))
               axis([-1 1 -inf inf])
          end
     end
end
```

```
a = axes;
 t1 = title('RxWheel Volume (mm^3) vs Des Vars');
 a.Visible = 'off'; % set(a,'Visible','off');
 t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*RxMass;
fRxMass = figure;

 for i = 1:NUMFAC
     for j = 1:NUMFAC
         if i == j
             varname = cellstr(DesVarNames{i});
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(Data(i+1,:),Data(1,:))
             xlabel(varname)
             axis([-1 1 -inf inf])
         elseif j > i
             KVec = Data(i+1,:).*Data(j+1,:);
             %disp(min(KVec));
             %disp(max(KVec));
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(KVec,Data(1,:))
             axis([-1 1 -inf inf])
         end
     end
end
a = axes;
 t1 = title('RxWheel Mass (kg) vs Des Vars');
```

```
a.Visible = 'off'; % set(a,'Visible','off');
 t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*RxPWR;
fRxPower = figure;
 for i = 1:NUMFAC
     for j = 1:NUMFAC
         if i == j
             varname = cellstr(DesVarNames{i});
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(Data(i+1,:),Data(1,:))
             xlabel(varname)
             axis([-1 1 -inf inf])
         elseif j > i
             KVec = Data(i+1,:).*Data(j+1,:);
             %disp(min(KVec));
             %disp(max(KVec));
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(KVec,Data(1,:))
             axis([-1 1 -inf inf])
         end
     end
 end
 a = axes;
 t1 = title('RxWheel Power (W) vs Des Vars');
 a.Visible='off';%set(a,'Visible','off');
 t1.Visible= 'on';%set(t1,'Visible','on');
 %% Magnetorquers
```

```
Data(1,:) = 2*MtxX.*MtxY.*MtxZ;
fRxVol = figure;
 for i = 1:NUMFAC
     for j = 1:NUMFAC
         if i == j
             varname = cellstr(DesVarNames{i});
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(Data(i+1,:),Data(1,:))
             xlabel(varname)
             axis([-1 1 -inf inf])
         elseif j > i
             KVec = Data(i+1,:).*Data(j+1,:);
             %disp(min(KVec));
             %disp(max(KVec));
             subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
             scatter(KVec,Data(1,:))
             axis([-1 1 -inf inf])
         end
     end
 end
 a = axes;
 t1 = title('Torque  Rod Volume (mm^3) vs Des Vars');
 a.Visible='off'; %set(a,'Visible','off');
 t1.Visible= 'on'; %set(t1,'Visible','on');
Data(1,:) = 2* MtxMass;
fRxMass = figure;
 for i = 1:NUMFAC
```

```
        for j = 1:NUMFAC
            if i == j
                varname = cellstr(DesVarNames{i});
                subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(Data(i+1,:),Data(1,:))
                xlabel(varname)
                axis([-1 1 -inf inf])
            elseif j > i
                KVec = Data(i+1,:).*Data(j+1,:);
                %disp(min(KVec));
                %disp(max(KVec));
                subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(KVec,Data(1,:))
                axis([-1 1 -inf inf])
            end
        end
end
a = axes;
t1 = title('Torque Rod Mass (kg) vs Des Vars');
a.Visible = 'off'; % set(a,'Visible','off');
t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = 3*MtxPWR;
fRxPower = figure;
for i = 1:NUMFAC
        for j = 1:NUMFAC
            if i == j
                varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(Data(i+1,:),Data(1,:))
                xlabel(varname)
                axis([-1 1 -inf inf])
            elseif j > i
                KVec = Data(i+1,:).*Data(j+1,:);
                %disp(min(KVec));
                %disp(max(KVec));
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(KVec,Data(1,:))
                axis([-1 1 -inf inf])
            end
        end
end
a=axes;
t1=title('Torque Rod Power (W) vs Des Vars');
a.Visible='off';%set(a,'Visible','off');
t1.Visible='on';%set(t1,'Visible','on');
%% StarTracker
Data(1,:) = STX.*STY.*STZ;
fSTVol = figure;
for i = 1:NUMFAC
    for j = 1:NUMFAC
        if i == j
            varname = cellstr(DesVarNames{i});
            subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(Data(i+1,:),Data(1,:))
                xlabel(varname)
                axis([-1 1 -inf inf])
            elseif j > i
                KVec = Data(i+1,:).*Data(j+1,:);
                %disp(min(KVec));
                %disp(max(KVec));
                subplot(NUMFAC,NUMFAC,(NUMFAC*i-NUMFAC)+j)
                scatter(KVec,Data(1,:))
                axis([-1 1 -inf inf])
            end
        end
end
a = axes;
 t1 = title('Star Tracker Volume (mm^3) vs Des Vars');
 a.Visible = 'off'; % set(a,'Visible','off');
 t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:) = STMass;
fRxMass = figure;
 for i = 1:NUMFAC
        for j = 1:NUMFAC
            if i == j
                varname = cellstr(DesVarNames{i});
                subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(Data(i+1,:),Data(1,:))
                xlabel(varname)
                axis([-1 1 -inf inf])
            elseif j > i
```

```
                KVec = Data(i+1,:).*Data(j+1,:);
                %disp(min(KVec));
                %disp(max(KVec));
                subplot(NUMFAC,NUMFAC,(NUMFAC*i-NUMFAC)+j)
                scatter(KVec,Data(1,:))
                axis([-1 1 -inf inf])
            end
        end
end
a=axes;
 t1 = title('Star Tracker Mass (kg) vs Des Vars');
 a.Visible = 'off'; % set(a,'Visible','off');
 t1.Visible = 'on'; % set(t1,'Visible','on');
Data(1,:)=STPWR;
fRxPower=figure;
 for i=1:NUMFAC
        for j=1:NUMFAC
            if i==j
                varname = cellstr(DesVarNames{i});
                subplot(NUMFAC,NUMFAC,(NUMFAC*i-NUMFAC)+j)
                scatter(Data(i+1,:),Data(1,:))
                xlabel(varname)
                axis([-1 1 -inf inf])
            elseif j > i
                KVec = Data(i+1,:).*Data(j+1,:);
                %disp(min(KVec));
                %disp(max(KVec));
                subplot(NUMFAC, NUMFAC, (NUMFAC*i-NUMFAC)+j)
                scatter(KVec,Data(1,:))
                axis([-1 1 -inf inf])
            end
        end
end
a = axes;
 t1 = title('Star Tracker Power (W) vs Des Vars');
 a.Visible = 'off'; % set(a,'Visible','off');
 t1.Visible = 'on'; % set(t1,'Visible','on');
```

D.2 Optimization Codes

D.2.1 Payload Optimization

```

#!/usr/bin/env python
# ======================================
# Standard Python modules
# ======================================
import os, sys, time
import pdb
import numpy as np
import math
# Run with Python 2.7 distribution on laptop else crashy crash
# ======================================
# Extension modules
# ======================================
#from pyOpt import *
```

```
from pyOpt import Optimization
from pyOpt import KSOPT
# ====================================
# VariableMapping
# ====================================
#    x[0] = Altitude
#    x[1]=IAMax
#    x[2]= YMax
#    x[3]= BitPer Pixel
#    x[4] = Pixel Instrument Count
#    x[5] = Detector Width
#    x[6] = Quality Factor
#    x[7] = Operational Wavelength
# ====================================
# DefinitionsforObj Function
# ====================================
def objfunc(x):
    #Design Equations for Optical Payload
    OrbPer = 1.658669e-04*(6378.14+x[0])**1.5
    GroundVelocity = 2*math.pi*6378.14/OrbPer/60
    AngRadius = math.asin(6378.14/(6378+x[0]))*180/math.pi
    LNot = 90-AngRadius
    DMax = math.tan(LNot*math.pi/180)*6378.14 #returns km
    EtaLook = math.asin(math.cos((90-x[1])*math.pi/180)*math.sin(
        AngRadius*math.pi/180))*180/math.pi
    ECAMax= 90-(90-x[1])-EtaLook
    SlantRange = 6378.14*math.sin(ECAMax*math.pi/180)/math.sin(
        EtaLook*math.pi/180)
    SwathWidth = 2*ECAMax
    IFOV = x[2]/1000/SlantRange*180/math.pi
    XMax = x[2]/math.cos(x[1]*math.pi/180)
    CrossTrackPixelResolution = IFOV*x[0]*math.pi/180
    AlongTrackPixelResolution = IFOV*x[0]*math.pi/180
    CrossTrackPixelCount = 2*EtaLook/IFOV
    SwathCount = GroundVelocity/AlongTrackPixelResolution
    PixelRate = SwathCount*CrossTrackPixelCount
    DataRate = PixelRate*x[3]
    PixelIntegrationTime = AlongTrackPixelResolution*x[4]/
        GroundVelocity/CrossTrackPixelCount
    FocalLength = x[0]*x[5]/CrossTrackPixelResolution
    ApertureDiameter = 2.44*x[7]*FocalLength*x[6]/x[5]
    FOV = IFOV*x[4]
    Ratio = ApertureDiameter/0.015
    if Ratio <=0.5:
        K = 2
    else:
        K = 1
    XDim = Ratio*0.045
    YDim = Ratio*0.050
    ZDim = Ratio*0.080
    PwrEst = K*(Ratio**3)*1.26
    MassEst = K*(Ratio**3)*0.230
    GSD = math.tan(IFOV/2*math.pi/180)*2*x[0]
    f=[0.0]*4
    f[0] = ApertureDiameter #FOr Size
    f[1] = GSD #For Data quality
    f[2] = -PixelIntegrationTime
    f[3] = ZDim
    g = [0.0]*1
   g[0] = ZDim-0.1
    fail =0
    return f,g, fail
# ===================================
# InitializeOptimizationProblem
# ===================================
opt_prob = Optimization('Passive Optic Payload Optimization C
    onstrained',objfunc)
#opt_prob.addVar('x1','c',lower=0.1,upper=1.0,value=0.35)# 0.1
    <=x <= 1
#opt_prob.addVar('x2','c',lower=0.0,upper=5.0,value=2.5)# 0 <=y
    <=5
opt_prob.addVar('Alt',lower=300.0,upper=450.0,value=400.0)
opt_prob.addVar('IA Max', lower = 30.0, upper = 77.0, value =
    50.)
opt_prob.addVar('YMax',lower=0.1,upper =150.0,value=20.)
opt_prob.addVar('BPP',lower=8.,upper =16.,value=8.)
opt_prob.addVar('PIC',lower=100, upper = 1000,value= 300)
opt_prob.addVar('DetWidth', lower = 1.1e-6, upper = 30.0e-6,
    value=2.0e-6)
opt_prob.addVar('QualFac', lower= 1.1, upper = 2.0, value= 1.5)
opt_prob.addVar('OpWavelength',lower= 3.0e-06, upper = 17.0e
    −06,value=5e−06)
opt_prob.addObj('f')
opt_prob.addCon('g',type='i',lower= −1e21,upper =0.0)
print opt_prob #VFY
# ======================================
# Initialize  Solver and Solve/Record
# ======================================
ksopt = KSOPT()
ksopt.setOption('IPRINT',2)
ksopt(opt_prob,sens_type='FD',store_hst=True)
opt_prob.write2file('FileTest.txt')
print opt_prob.solution(0)
# ======================================
#    x[0] =  Altitude
#    x[1] =IAMax
#    x[2] = YMax
#    x[3] = BitPerPixel
#    x[4] =  Pixel Instrument Count
#    x[5]  = Detector  Width
#    x[6]  = Quality  Factor
#    x[7] =  Operational Wavelength
```

D.2.2 Payload Optimization Verification

```
%% Quick Check
clc; clear all; close all;
%%
Alt = 400000 %
```

```
DetWidth = 1.19e-6; %sq pixel size
 f = 4.02e-3; %physical focal
IFOV = 2*atan2d(DetWidth,(2*f)) %deg
GSD = 2*Alt*tand(IFOV/2) %cnovert to deg and perform tangent
%%Verify the payload optimization results
%x = [700 70 68 8 256 30e-6 1.1 4.2e-6] %FIRESAT
x = [300 30 100 8 30 0.4e-4 1.1.3e-5]
OrbPer = 1.658669e-04*(6378.14+x(1))^1.5
 GroundVelocity = 2*pi*6378.14/OrbPer/60
AngRadius = asin(6378.14/(6378+x(1)))*180/pi
LNot = 90-AngRadius
DMax = tan(LNot*pi/180)*6378.14 %returns km
 EtaLook = asin(cos((90-x(2))*pi/180)*sin(AngRadius*pi/180))*180/
    pi
ECAMax = 90-(90-x(2))-EtaLook
SlantRange = 6378.14*sin(ECAMax*pi/180)/sin(EtaLook*pi/180)
SwathWidth = 2*ECAMax
IFOV = x(3)/1000/SlantRange*180/pi
XMax = x(3)/cos(x(2)*pi/180)
 CrossTrackPixelResolution = IFOV*x(1)*pi/180 %in km
 AlongTrackPixelResolution = IFOV*x(1)*pi/180 %in km
 CrossTrackPixelCount = 2*EtaLook/IFOV
SwathCount = GroundVelocity/AlongTrackPixelResolution
 PixelRate = SwathCount*CrossTrackPixelCount
DataRate = PixelRate*x(4)
 PixelIntegrationTime = AlongTrackPixelResolution*x(5)/
    GroundVelocity/CrossTrackPixelCount
```

```
FocalLength = x(1)*x(6)/CrossTrackPixelResolution
ApertureDiameter = 2.44*x(8)*FocalLength*x(7)/x(6)
FOV = IFOV*x(5)
Ratio = ApertureDiameter/0.406
if Ratio <= 0.5
    K = 2
else
    K = 1
end
XDim = Ratio*2.0
YDim = Ratio*0.7
ZDim = Ratio*0.9
PwrEst = K*(Ratio^3)*280
MassEst = K*(Ratio^3)*239
GSDPayload = 2*x(1)*1000*tand(IFOV/2)
```

D.2.3 ADCS Optimization

```

#!/usr/bin/env python
# ======================================
# Standard Python modules
# ======================================
import os, sys, time
import pdb
import numpy as np
import math
# Run with Python 2.7 distribution on laptop else crashy crash
# =====================================
# Extension modules
# =====================================
#from pyOpt import *
from pyOpt import Optimization
from pyOpt import KSOPT
# =====================================
#  Definitions for Obj Function
# =====================================
 def objfunc(x):
        #Design Equations for Optical Payload
        arc=x[13]*180/math.pi
        GravGradient = 3*3986e14/2/(x[0]**3)*abs(x[2]-x[1])*math.sin
            (2*x[3])
        SolarRadiation = (1367/3e8*x[6]*(1+x[7])*math.cos(x[8]))*(x
            [4]-x[5])
        MagneticField = (2*7.96e15)/(x[0]**3)*x[9]
        atmos = np.array([[0, 0.00, 1.23, 7.25],
                          [25, 25.00,3.899e-2,6.35],
                          [30, 30.00,1.774e-2,6.68],
                          [40, 40.00,3.972e-3,7.55],
                          [50, 50.00,1.057e-3,8.38],
                          [60, 60.00,3.206e-4,7.71],
                          [70, 70.00,8.770e-5,6.55],
                          [80, 80.00,1.905e-5,5.80],
                          [90, 90.00,3.396e-6,5.38],
                          [100, 100.00, 5.297e-7, 5.88],
                    [110, 110.00, 9.661e-8,  7.26],
                    [120, 120.00, 2.438e-8,  9.47],
                    [130, 130.00, 8.484e-9,  12.64],
                    [140, 140.00, 3.845e-9,  16.15],
                    [150, 150.00, 2.070e-9,  22.52],
                    [180, 180.00, 5.464e-10,  29.74],
                    [200, 200.00, 2.789e-10,  37.11],
                    [250, 250.00, 7.248e-11,  45.55],
                    [300, 300.00, 2.418e-11,  53.63],
                    [350, 350.00, 9.518e-12,  53.30],
                    [400, 400.00, 3.725e-12,  58.52],
                    [450, 450.00, 1.585e-12,  60.83],
                    [500, 500.00, 6.967e-13,  63.82],
                    [600, 600.00, 1.454e-13,  71.84],
                    [700, 700.00, 3.614e-14,  88.67],
                    [800, 800.00, 1.170e-14, 124.64],
                    [900, 900.00, 5.245e-15, 181.05],
                    [1000, 1000.00, 3.019e-15, 268.00]])
    for i in range(27):
        if x[0] >= atmos[i,0] and x[0] <  atmos[i+1,0]:
            H = atmos[i,3]
            rhon = atmos[i,2]
            base = atmos[i,1]
        elif x[0] >= atmos[i+1,0]:
            H = atmos[i+1,3]
            rhon = atmos[i+1,2]
            base =  atmos[i+1,1]
    Density =rhon *math.exp(-(x[0]/1000 - base)/H)
    vel=math.sqrt(3.986e14/x[0])
    AerodynamicTorque = (0.5*Density*x[10]*x[6]*vel*vel)*(x[14] -
        x[5])
    DistubranceTorque= AerodynamicTorque + GravGradient +
        MagneticField + SolarRadiation
    OrbitPeriod = 1.658669e-04*(x[0]/1000)**1.5
    SlewTorque = 4*x[11]*math.pi/180*x[2]/(x[12]**2)
    H = DistubranceTorque * OrbitPeriod / 4 * 0.707
    if H <= 0.015: #Smallest found reaction wheel
        H=0.015
    MagDipole = 1.5*DistubranceTorque/MagneticField
    RxVol = (20.55*math.log(H)+120.4)*(20.21*math.log(H)+118.0)
        *(23.61* math.log(H)+100.2)
    Mtx = (0.1216 + 10.87*MagDipole)
    Mty = (118.0 + 3.363*MagDipole)
    Mtz = (100.2 + 26.67*MagDipole)
    STVol= (-6071*arc + 196.3)*(-6500*arc + 200.3)*(-1.536e4*arc
        + 387)
    PwrST= -4.592e4*arc**2 + 750*arc + 5
    Pwr = -4.592e4*arc**2 + 750*arc + 5+ 0.466*H + .5106 +
        .0502* MagDipole + .399
    Mass = -7296*arc**2 + 31.79*arc + 2.735 + 1.666*H+.1216 +
        .001029* MagDipole +.3457
    f=[0.0]*5
    f[0]= -x[13]
    f[1]= Pwr
```

```
        f[2]=x[12]
        f[3]=Mass
        f[4]=MagDipole
        g= [0.0]*3
        g[0] =Mtx - 100
        g[1] =Mty - 100
        g[2] =Mtz - 100
        fail=0
        return f,g, fail
# ========================================
# Variable Mapping and Constants
# ========================================
 #     x[0]= R Radius  (m)
 #     x[1] = Iy
 #     x[2] = Iz
 #     x[3] = IncidenceAngle(Theta)
 #     x[4] = CenterSolarPressure
 #     x[5] = CenterGravity
 #     x[6] = SurfaceArea
 #     x[7] = ReflectionFactor
 #     x[8]=SunIA
 #     x[9] = Residual Dipole (D)
 #     x[10] = CoeffDrag
 #     x[11]=SlewMaxDeg
 #     x[12]=SlewMaxTime
 #     x[13]= PointKnowledge (o)
 #     x[14] = CenterPressure
 # ========================================
 # InitializeOptimizationProblem
 # ========================================
 opt_prob= Optimization('ADCSOptimization',objfunc)
 #opt_prob.addVar('x1','c',lower=0.1,upper=1.0,value=0.35) # 0.1
     <=x <= 1
#opt_prob.addVar('x2','c',lower=0.0,upper=5.0,value=2.5) # 0 <=y
     <= 5
 opt_prob.addVar('Radius', lower=6621e3, upper=6621e3, value=6621
     e3)
 opt_prob.addVar('Iy',lower= 1.9e-3, upper = 2.1e-3, value= 2e
     -3)
 opt_prob.addVar('Iz', lower = 1.7e-3, upper = 1.8e-3, value =
     1.75e-3)
 opt_prob.addVar('IA',lower=0.,upper =5., value=0.)
 opt_prob.addVar('CSP',lower=.02,upper =.02, value=.02)
 opt_prob.addVar('CG',lower=0.0,upper =0.0,value=0.0)
 opt_prob.addVar('SA',lower=.0292,upper =.0298,value=
     .0294)
 opt_prob.addVar('Reflect', lower = 0.5, upper = 0.8, value =
     0.65)
 opt_prob.addVar('SunIA', lower= 0.0, upper= 30*math.pi/180,
     value= 0.0)
 opt_prob.addVar('RD', lower = 1.0, upper = 1.2, value = 1.1)
 opt_prob.addVar('Cd', lower = 2.0, upper = 2.2, value = 2.1)
 opt_prob.addVar('SlewAng', lower = 10.0*math.pi/180, upper =
     35.0*math.pi/180, value= 30.0*math.pi/180)
```

```
opt_prob.addVar('SlewTime', lower = 4.0, upper = 60.0, value =
    10.0)
opt_prob.addVar('PointKnwledge',lower= .007*math.pi/180, upper
    = .021*math.pi/180, value= .015*math.pi/180)
opt_prob.addVar('Cp',lower=.01,upper =.03,value=.02)
opt_prob.addObj('f')
opt_prob.addCon('g',type='i',lower= −1e21,upper =0.0)
print opt_prob #VFY
# =====================================
# Initialize Solver and Solve/Record
# =====================================
ksopt = KSOPT()
ksopt.setOption('IPRINT',2)
ksopt.setOption('IFILE','ADCS_KS_Soln.txt')
ksopt(opt_prob,sens_type='FD',store_hst=True)
opt_prob.write2file('ADCSOpt.txt')
print opt_prob.solution(0)
```

D.3 Codes for Benchmarks

D.3.1 Benchmark Visualization

```
%%Example Search Space for Constrained Optimization Problems
% Justin Ancheta
% March 2nd 2018
% Code for visualing the benchmark optimization problem for
    masters proj.
close all; clear all; clc;
%%Objective Functions,Constraints,Search Region
```

```
fone = @(x) x;
ftwo = @(x,y) (1+y)/x;
gone =@(x,y) y + 9* x − 6;   %gone must be greater than or equal
    to 0
gtwo = @(x,y) −y + 9* x − 1; %gtwo must be greater than or equal
    to 0
%%Visual Search Region
figure
[x1,x2] = meshgrid([0.1:0.001:1],[0.0:0.01:5]);
zed = ones(size(x1));
bound1 = x2 + 9* x1 −6 >= 0;
bound2 = −x2 +9*x1 −1 >= 0;
zed(~bound1)=0;
zed(~bound2)=0;
contourf(x1,x2,zed)
title('Region available in design space')
xlabel('x_1 = x')
ylabel('x_2 = y')
cmap=jet(2);
cmap(1,:)=[1,1,1];
colormap(cmap);
colorbar('Ticks',[0.25 0.75],'TickLabels',{'Unavailable','
    Available'})
%%Finding the values of f1 and f2      _
%Creating the design space
data x1 = x1.* zed;
data x2 = x2.* zed;
```

```
size_data = size(x1);
lengthx = size_data(1);
lengthy = size_data(2);
outf1 = zeros(lengthx,lengthy);
outf2 = outf1;
for i = 1:lengthx
    for j = 1:lengthy
        outf1(i,j) = fone(datax1(i,j));
        outf2(i,j) = ftwo(datax1(i,j),datax2(i,j));
    end
end
outf1(outf1==0) = NaN;
%% f1 v f2 vars
out2f1 = reshape(outf1,[lengthx*lengthy 1]);
out2f2 = reshape(outf2,[lengthx*lengthy 1]);
%%Viewing f1 and f2
%Function 1
figure
h = surf(x1,x2,outf1);
set(h,'LineStyle','none')
title('Output of f1')
xlabel('x_1 = x')
ylabel('x_2 = y')
zlabel('f1')
colorbar
%Function 2
figure
```

```
h = surf(x1,x2,outf2);
 set(h,'LineStyle','none')
 title('Output of f2')
 xlabel('x_1 = x')
 ylabel('x_2 = y')
 zlabel('f2')
 colorbar
%f1 v f2
 figure
 plot(out2f1, out2f2)
 xlabel('f1')
 ylabel('f2')
 title('f1vsf2')
%%Pareto Frontier
 fitfunction = @(x)[x(1),(1+x(2))/x(1)];
 nvars = 2;
a= [-9,-1;-9,1];%same as above bounds but meant for leq not geq
 b=[-6,- 1];
 lb =[0.1  0];
 ub = [1,  5];
 [ParFront,fval]= gamultiobj(fitfunction,nvars,a,b,[],[],lb,ub);
%%View Pareto Frontier
 figure
 plot(fval(:,1),fval(:,2),'r*')
 hold on
 xlabel('f1')
 ylabel('f2')
```

```
title('Pareto Front')
%% KS from pyOpt
fileID = 'BenchOptData.csv';
dataKSOPT = csvread(fileID,1,0);
iters=size(dataKSOPT)
for i=1:iters(1)
        dataKSOPT(i,4) = fone(dataKSOPT(i,1));
        dataKSOPT(i,5) = ftwo(dataKSOPT(i,1),dataKSOPT(i,2));
end
sz=5;
scatter(dataKSOPT(1,4),dataKSOPT(1,5),'ko','filled')
scatter(dataKSOPT(:,4),dataKSOPT(:,5),sz,'c*')
legend('Pareto  front','KS  Solution  Final','KS  Initial  Guess','KS
    Evaluations')
%%Visuallizing KS
figure
plot3(dataKSOPT(:,1),dataKSOPT(:,2),dataKSOPT(:,3))
hold on
plot3(dataKSOPT(1,1),dataKSOPT(1,2),dataKSOPT(1,3),'ro')
plot3(dataKSOPT(end,1),dataKSOPT(end,2),dataKSOPT(end,3),'ko')
xlabel('x1')
ylabel('x2')
zlabel('KS')
legend('KS','KS_o','KS_f')
title('Evaluation of KS')
```

D.3.2 Benchmark Optimization

```
#!/usr/bin/env python
# ======================================
# Standard Python modules
# ======================================
import os, sys, time
import pdb
# Run with Python 2.7 distribution on laptop else crashy crash
# ======================================
# Extension modules
# ======================================
#from pyOpt import *
from pyOpt import Optimization
from pyOpt import KSOPT
# ======================================
#  Definitions for Obj Function
# ======================================
 def objfunc(x):
        c = x[0]
        d = (x[1]+1)/x[0]
        f = [0.0] * 2
        f[0]  = c
        f[1]  = d
        g = [0.0]*2
        g[0] = -9.* x[0] - x[1] + 6
        g[1] = -9.* x[0] + x[1] + 1

        fail = 0
        return f,g, fail
# ======================================
# Initialize Optimization Problem
# ======================================
opt_prob = Optimization('Benchmark ConstEx Constrained Problem',o
    bjfunc)
opt_prob.addVar('x1','c',lower=0.1,upper=1.0,value=0.35)
opt_prob.addVar('x2','c',lower=0.0,upper=5.0,value=2.5)
opt_prob.addObj('f')
opt_prob.addCon('g1','i')
opt_prob.addCon('g2','i')
print opt_prob #VFY
# ======================================
# Initialize Solver and Solve/Record
# ======================================
ksopt = KSOPT()
ksopt.setOption('IPRINT',2)
ksopt(opt_prob,sens_type='FD',store_hst=True)
opt_prob.write2file('FileTest.txt')
print opt_prob.solution(0)
```

D.4 MBSE Supporting Functions

D.4.1 NBody Function

```
%%N Body Function
% Justin Ancheta
% December 18 2017 - updated 12/18/2017
% rev00.00
```

```
%%Revision log
% 12/18/17 - Base Code Started
%
%%Purpose and Use
%
 % This function file will generate the set of functions for the
     integrator.
 % nbody=f(ic_i,m_i) where ic_i is a[7*n+1,1] set of positions
     and velocities
% given in the form of [r1,r2,r3,...,rn,v1,v2,v3,...,vn,m1,...,mn
     ,n]
%
%%Construction of set of functions for Integrators
% The set of equations needed based on ic are
% [v1x,v1y,v1z,v2x,v2y,v2z,...,vnx,vny,vnz,a1x,a1y,a1z,...,anx,any
     ,anz]
% as y=[rx ry rz vx vy vz] and y'=[vx vy vz ax ay az]
 function dx = nbody(t,y,m)
G= 6.67408e-20; %km^3/(kg s^2)
n=length(m);
 endval = 6*n;
dx= zeros(6*n,1);
 xpos=y(1:3:endval/2);
 ypos=y(2:3:endval/2);
 zpos = y(3:3:endval/2);
tx  = bsxfun(@minus,xpos,xpos'); %difference between x of i and j
ty  = bsxfun(@minus,ypos,ypos'); %see above for y
```

```
tz= bsxfun(@minus,zpos,zpos');%see above for z
r= sqrt(tx.^2+ ty.^2+ tz.^2).^3;%cubed distance between I and J
%% changing y'(1:end/2) to previous y'' values
for i=1:3*n
        dx(i)=y(i+3*n);
end
%%Calculating x y and z accelerations
r= r+ eye(n);%to prevent singularties
mt = repmat(m,n,1)';%repeats m for forces
fx=mt./r.*tx;
fx(1:(n+1):n*n)=0;
fx_i= sum(fx,1);
fy=mt./r.*ty;
fy(1:(n+1):n*n)=0;
fy_i= sum(fy,1);
fz=mt./r.*tz;
fz(1:(n+1):n*n)=0;
fz_i= sum(fz,1);
for i=1:n
        dx(i*3+n*3 -2) = G* fx_i(i);
        dx(i*3+n*3 -1) = G* fy_i(i);
        dx(i*3+n*3) = G* fz_i(i);
 end
 end
```

D.4.2 Solar Line of Sight with Penumbra

```
function y= fcn(u)
```

```
% BASED ON HWANGS DISSERTATION SMOOTHING FUNCTION FOR UMBRA
y = 0;
Re = 6371.01;
PosSun = [u(1);u(2);u(3)];
PosEarth = [u(4);u(5);u(6)];
PosSat = [u(7); u(8); u(9)];
 alp = u(10);
 SatEarth = PosSat-PosEarth;
 EarthSun = PosEarth-PosSun;
EarthSun = EarthSun/norm(EarthSun);
ds = norm(cross(SatEarth,EarthSun));
 eta = (ds-alp*Re)/(Re-alp*Re);
 check = dot(SatEarth,EarthSun);
 if check >= 0
        y = 1;
 elseif ds > Re
        y =1;
 elseif ds < alp*Re
        y = 0;
 else
        y = 3*eta^2-2*eta^3;
end
```

D.5 Simulink Run and Test Code

```
%%N-Body Simulation Simulink Analysis
 clc; clear all; close all;
 format long g
```

```
%% Rotatations
%
% ICRF/BCRF−GCRF TBD; GCRF ICRF BCRF ARE ESSENTIALLY ALLIGNED FOR
    MOST
% APPLICATIONS . THIS WILL REQUIRE ACTUAL WORK AND RESEARCH TO DO
    RIGHT.
% ASSUME EFFECTS ARE SMALL FOR THE TIME BEING .
% IAU , SOFA Tools for Earth Attitude ]
%
%%Various Constants and Inputs
SimTime = (60*60*24)*5; %1 day sim time
DistAUtoKM = 1.496e+8;
TimeDaytoSec = 60*60*24;
SunAlpha = 0.9;
% Solar Array Design Options
 PowerType = 'DET'; %Case sensitive: Options include DET (direct
    energy transfer) or PPT (Peak Power Tracking)
  Peclipse = 100; %Watts needed power eclipse
Pday = 100; %Watts needed power during time in sun
 SunTempMax = 100; %Maximum temperture of solar cell array
 EclipseTempMax = −80; %Minimum temperatre of solar cell array
% %% Benchmark against JPL Horizons for Earth Pos
% HORZDATA= csvread('EarthTestXYZ.csv',1);
% % Earth Sun Moon Initial Conditions March 25, 2018 from
    Horizons
% MassSun = 1.988544e30; %kg
% MassEarth = 5.97219e24; %kg
% MassMoon = 734.9e20; %kg
% MassJupiter = 1898.13e24;
% MassSat = 10; %kg
% RadSun = 6.963e5; %km
% RadEarth = 6371.01; %km
% RadMoon = 1737.4; %km
% PosSunAU = [1.293353689013552e-03; 6.522361385771349e
    -03; -1.079851494007183e-04];
% VelSunAUD = [-6.409885860066381e-06;4.440045984244416e
    -06;1.528999606408731e-07];
% PosEarthAU = [-9.933675647884848E-01; -6.380096968050979e-02;
    -9.938491024392269e-05];
% VelEarthAUD = [9.341394242566131e-04; -1.722037211216111e
    -02;3.297113152345686e-07];
% PosMoonAU = [-9.937383674834914e-01; -6.135886608208557e-02;
    -2.263388774617808e-04];
% VelMoonAUD = [3.321828061507152e-04; -1.731613336450174e-02;
    4.558197105418077e-05];
% PosJupiterAU = [-3.852523741724474e0; -3.800676827730020e0;
    1.019361973208450e-01];
% VelJupiterAUD = [5.211670508998708e-03; -5.013451102063293e-03;
     -9.574580960891983e-05];
% PosSatAU = [-9.933738053211382E-01; -6.384356151687003E-02;
    -8.512557260487344E-05];
% VelSatAUD = [3.699397572966039E-03; -1.867888405149421E
    -02; -3.136616450617170E-03];
% PosSun = PosSunAU*DistAUtoKM;
% PosEarth = PosEarthAU*DistAUtoKM ;
% PosMoon = PosMoonAU*DistAUtoKM ;
% PosJupiter = PosJupiterAU *DistAUtoKM ;
% PosSat = PosSatAU*DistAUtoKM ;
% VelSun = VelSunAUD*DistAUtoKM/ TimeDaytoSec ;
% VelEarth = VelEarthAUD*DistAUtoKM/ TimeDaytoSec ;
% VelMoon = VelMoonAUD*DistAUtoKM/ TimeDaytoSec ;
% VelJupiter = VelJupiterAUD *DistAUtoKM/ TimeDaytoSec ;
% VelSat = VelSatAUD*DistAUtoKM/ TimeDaytoSec ;
sim ('NBODYSIMULINK') ;
%% Compare to Benchmark
 % This is for fixed step size of 60s and one day only for testing
     .
% if SimTime == 86400
 %       figure
 %       plot3(HORZDATA(:,2),HORZDATA(:,3),HORZDATA(:,4))
 %       hold on
 %       plot3(body2xyz.data(:,1),body2xyz.data(:,2),body2xyz.data
     (:,3)) %Earth
 %       title('Benchmark of 1 Day from IC')
 %       legend('HORIZONS−ISS','NbodySim')
 %       RelativePosOffsetJPL = 100*abs([body2xyz.data(:,1),body2xyz
     .data(:,2),body2xyz.data(:,3)]−[HORZDATA(:,2),HORZDATA(:,3),
    HORZDATA(:,4)])./[HORZDATA(:,2),HORZDATA(:,3),HORZDATA(:,4)];
 %       MaxErrorXYZ = max(RelativePosOffsetJPL)
 %       MinErrorXYZ = min(RelativePosOffsetJPL)
% end
```

```
%%Sun around Barycenter
%figure
%plot3(body1xyz.data(:,1),body1xyz.data(:,2),body1xyz.data(:,3))
    %Sun
%%Planets around Bareycenter
 figure
 plot3(body1xyz.data(:,1),body1xyz.data(:,2),body1xyz.data(:,3)) %
    Sun
%
% Sun barely moves from barycenter, only viewable around 1e6
    scale
%
 hold on
 plot3(body2xyz.data(:,1),body2xyz.data(:,2),body2xyz.data(:,3))  %
    Earth
 plot3(body3xyz.data(:,1),body3xyz.data(:,2),body3xyz.data(:,3))  %
    Moon
 plot3(body4xyz.data(:,1),body4xyz.data(:,2),body4xyz.data(:,3))  %
    Jupiter
 scatter3(0,0,0)%BARYCENTER
 grid on
 title('Position wrt Barycenter of Solar System')
 legend('Sun','Earth','Moon','Jupiter')
 xlabel('x (km)')
 ylabel('y (km)')
 zlabel('z (km)')
 %% Earth Moon Sat wrt to Sun
```

```
EarthSun = [body2xyz.data(:,1),body2xyz.data(:,2),body2xyz.data
    (:,3)]-[body1xyz.data(:,1),body1xyz.data(:,2),body1xyz.data
    (:,3)];
MoonSun = [body3xyz.data(:,1),body3xyz.data(:,2),body3xyz.data
    (:,3)]-[body1xyz.data(:,1),body1xyz.data(:,2),body1xyz.data
    (:,3)];
SatSun = [body5xyz.data(:,1),body5xyz.data(:,2),body5xyz.data
    (:,3)]-[body1xyz.data(:,1),body1xyz.data(:,2),body1xyz.data
    (:,3)];
 figure
 plot3(EarthSun(:,1),EarthSun(:,2),EarthSun(:,3))
 hold on
 plot3(MoonSun(:,1),MoonSun(:,2),MoonSun(:,3))
 plot3(SatSun(:,1),SatSun(:,2),SatSun(:,3))
 title('Position w.r.t. Sun')
 legend('Earth','Moon','Sat')
 grid on
 xlabel('x (km)')
 ylabel('y (km)')
 zlabel('z (km)')
%% Moon/Sat with respect to Earth
MoonEarth = [body3xyz.data(:,1),body3xyz.data(:,2),body3xyz.data
    (:,3)]-[body2xyz.data(:,1),body2xyz.data(:,2),body2xyz.data
    (:,3)];
SatEarth = [body5xyz.data(:,1),body5xyz.data(:,2),body5xyz.data
    (:,3)]-[body2xyz.data(:,1),body2xyz.data(:,2),body2xyz.data
    (:,3)];
```

```
[Ex,Ey,Ez] = sphere(20);
xEast   = RadEarth * Ex;
yNorth = RadEarth * Ey;
zUp     = RadEarth * Ez;
figure
plot3(MoonEarth(:,1),MoonEarth(:,2),MoonEarth(:,3))
hold on
plot3(SatEarth(:,1),SatEarth(:,2),SatEarth(:,3))
surf(xEast,yNorth,zUp,'FaceColor','blue','FaceAlpha',0.5)
title('Moon/Sat Orbit around Earth')
legend('Moon','Satellite')
axlim = 5e5;
axis([-axlim axlim -axlim axlim -axlim axlim])
grid on
xlabel('x (km)')
ylabel('y (km)')
zlabel('z (km)')
%% LOS over time (ISS on March 25th 2018) - Check against STKS
    till
%
% This is only useful in really small durations (< 1day),
    otherwise you get
% a really solid block which is useless.
%
% figure
% plot(LOS.time(:,1),LOS.data(:,1))
% title('LOS_{Earth} over Time of Sat')
```

```
%xlabel('Time(hour)')
%ylabel('LOS')
%%Finding Solar Array Power Size
switch PowerType
    case 'DET'
        Xe=0.65;
        Xd=0.85;
    case 'PPT'
        Xe=0.6;
        Xd=0.8;
    otherwise%ideal
        Xe=1;
        Xd=1;
end
LOSDayFrac=mean(LOS.data(:,1));
LOSNightFrac=1 - LOSDayFrac;
PsaReqEst=(Peclipse*LOSNightFrac/Xe + Pday*LOSDayFrac/Xd);
%%Altitude over time
Altitude= [1:1:length(SatEarth)];
for i= 1:length(SatEarth)
    Altitude(i)= dot(SatEarth(i,:),SatEarth(i,:)/norm(SatEarth(i
        ,:)))-6371;
end
figure
plot(Altitude)
```