Parametric Study of an Electric Propulsion Spacecraft for Mission Time Optimization

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

Rafael Gutierrez

December 2018

approved by

Dr. Periklis Papadopoulos
Faculty Advisor

San José State
UNIVERSITY

Contents

List of Figures

List of Tables

Abstract

The study varies key parameters and trajectories of a spacecraft mission using an electric propulsion (EP) system. For consistent comparison, the spacecraft is assumed to be on a mission to Mars. The spacecraft is assumed to have started at LEO and should be able to travel to Low Mars Orbit and return to LEO. This would assume that the spacecraft has been transferred to LEO via a chemical propulsion launch vehicle. Optimal trajectories are determined and examined based on previous studies of trajectory optimization and optimal launch windows. The key independent variables of the EP system are the input power, the input current, the mass flow rate, and the exhaust velocity. The key parameters observed for variation are thrust, efficiency, total mass, and total mission time. The variation is examined to determine the effect those parameters have on the mission time. The purpose for examining these parameters is to determine if the issue of mission time can be addressed when using EP. EP is an appealing propulsive system in spacecraft because of the significant weight reduction, but at the cost of an increased mission time in comparison to chemical propulsion. This study is a preliminary determination of the possible time optimization for a space mission to Mars. The total mission time is being set for a time of 150 days in order to provide a decreased amount in mission time. The trajectory optimization is still in progress.

Nomenclature

g	Acceleration due to gravity, m/s^2
m_B	Mass at burnout, kg
m_d	Mass delivered, kg
m_p	Propellant mass, kg
$\dot{m}_p$	Propellant mass flow rate, kg/s
$\vec{r}$	Position Vector, km
q	Electrical Charge, C
v_{ex}	Exhaust Velocity, m/s
$\vec{v}$	Velocity Vector, m/s
$\vec{B}$	Magnetic Field, T
$\vec{E}$	Electric Field, N/m
$\vec{F}$	Force Vector, N
I_b	Beam current, A
I_{sp}	Specific Impulse, s
P_{dis}	Power Dissipated, W
P_{in}	Input Power, W
P_{jet}	Jet Power, W
T	Thrust, Newtons
$\vec{X}$	State Vector, [m,m,m,m/s,m/s,m/s]
V_{an}	Anode Voltage, V
V_b	Beam Voltage, V
η_e	Electrical Efficiency
η_T	Total Efficiency
μ	Standard Gravitational Parameter, m^3/s^2

Δv	Velocity change requirement, m/s
$\Delta \theta$	Change in Angle, rad

1 Introduction

1.1 Motivation

Electric propulsion (EP) systems are an option in spacecraft. The issue with electric propulsion is that it increases the required mission time. It does, however, allow for a lower spacecraft mass. It would be ideal to decrease the mission time and mass simultaneously. This EP system requires a larger power source. In some cases, the idea is to implement nuclear power sources. The issue with nuclear power plants is mainly political in nature so there has be minimal advancement in nuclear technologies in spacecraft. The main goal is to increase the thrust, the total efficiency, minimize mass, and decrease the mission time.

The reduction in mission time will be used to also observe the type of power requirements necessary for a propulsion system of this type. Though not under current consideration, the power requirement would determine the size of the spacecraft as well. With current technology, the more power the system requires, the bigger the power supply tends to be in the spacecraft.

1.2 Literature Review

Robert Hutchings Goddard is one of the earlier mentions who considered EP for use in spacecraft. The initial intent was to electrostatically accelerate electrons to provide a propulsive thrust. Goddard was knowledgeable about canal rays so it was interesting that he had not yet thought of accelerating ions rather than electrons. The idea, eventually, arose and one of the first concepts of an ion thruster began to develop. Ions can be accelerated and ejected to create the propulsive force needed to provide thrust. This means that positively charged particles in equal parts are also ejected.

Goddard would use these ideas to patent the idea of an electrostatic ion accelerator. Figure 1 shows the schematic of the electrostatic ion accelerator [1].

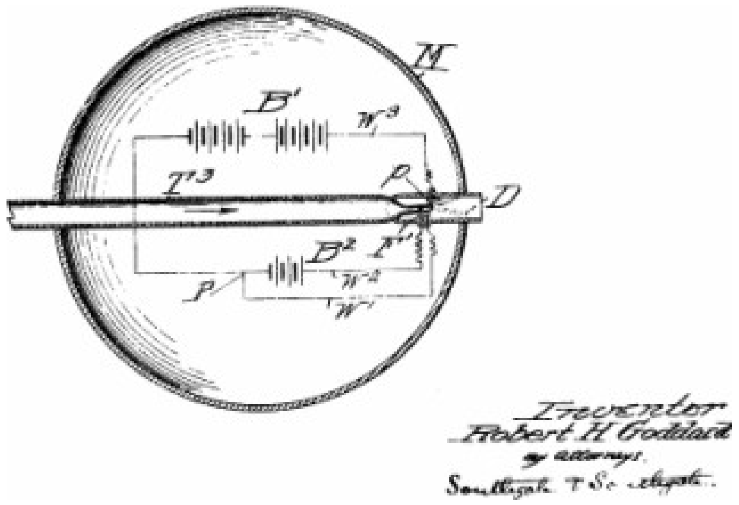


Figure 1: The figure shows the third variation of Goddard's electrostatic ion accelerator from 1917.

The development of EP has been hindered by the high power requirements of the system. Because of the high power requirements, Yuri V. Kondratyuk argued against focusing on EP due to the relationship between the exhaust velocity and the power requirement. The relationship can be seen in Equation 1.

$$\frac{T}{P} = 2\frac{\eta}{v_e} \tag{1}$$

Equation 1 shows that the power and the exhaust velocity are directly proportional. For this reason, Kondratyuk suggested that the focus for propulsion research should be on chemical propulsion rather than electric propulsion. From 1917-1919, the propulsion system of choice was chemical propulsion.

Hermann Julius Oberth was one of the next major influences on electric propulsion. Oberth came along to suggest that the use of electric propulsion would result in significant mass reduction in spacecraft. Oberth's schematics included the use of Goddard's idea of the electrostatic ion accelerator that would later be used to develop the ion thruster. This was the major breakthrough of this era.

This era was followed by an era of scientist using the designs and concepts to create an electric propulsion system. Up until the 1970s, most EP systems were tested experimentally. One of the first ion thrusters was flown in 1994 and since then has remained a popular research area. They have become more popular for commercial use. With EP, it is typically easier to perform station keeping on satellites.

In electric propulsion, systems are separated into 3 general categories of propulsion which go as follows: electrothermal, electrostatic, and electromagnetic.

Electrothermal propulsion is the use of an electrical method to heat up a propellant inducing a thermodynamic expansion in a nozzle. This is the category where resistojets and arc jets typically fall into. Electrostatic propulsion is the when ion are accelerated through an electric field to create a propulsive force. Hall thrusters and ion thruster fall into this category of EP. Electromagnetic propulsion occurs by driving a current through a plasma to create a force. The force created in electromagnetic propulsion is governed by the Lorentz force found in equation 2 [2].

$$\boldsymbol{F} = q\boldsymbol{E} + q\boldsymbol{v} \times \boldsymbol{B} \tag{2}$$

Some examples of electromagnetic propulsion are pulsed plasma thrusters (PPT) and magnetoplasmadynamic thrusters (MPDT).

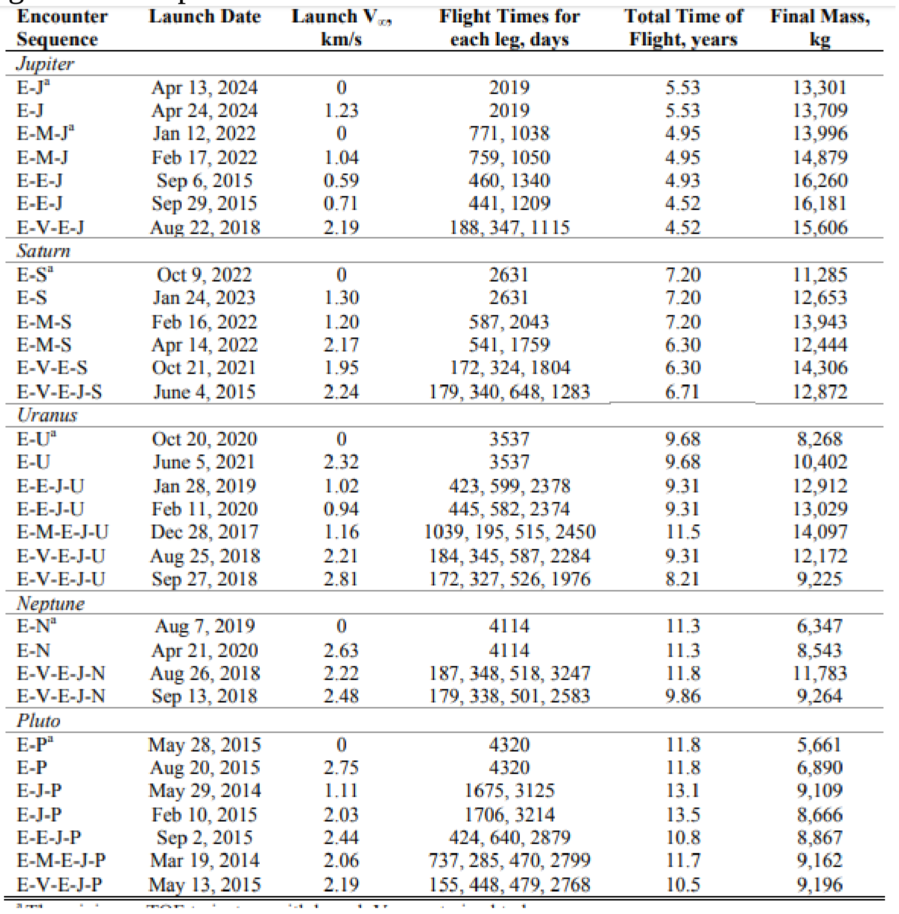
The long mission times have been reduced through research by developing new electric propulsion systems. The development of ion thrusters, hall thrusters, pulsed plasma thrusters and magnetoplasmadynamic thrusters are example of EP systems that have a significant mass reduction. The magnetoplasmadynamic thruster (MPDT) has been considered by some to be the electric propulsion of the future [3]. The problem with this EP system is the high power requirement necessary for most MPDT designs. Some of these systems range from the order of MW to GW requirements in power. Many of these design are experimental and few have gone through a test flight.

Although the MPDT system is considered the future of EP, it would be of use to determine whether or not other forms of EP could also be designed and optimized with respect to the mission time and total efficiency. The main issue in either case would be the power input and the power requirement to be able to significantly decrease the mission time [4].

Decreasing mission time with EP is a current research topic due to the smaller size of EP powered spacecraft. There are several ways being researched to decrease mission time. Several options include trajectory optimization, increasing power levels, and increasing the exhaust velocity. One suggestion is to use nuclear energy to provide larger amounts of energy.

The more feasible of the three options at this time is trajectory optimization[5]. There are ways to efficiently optimize trajectories and they will be taken into account in this analysis[6]. Research on trajectory optimizations has been done for nuclear electric propulsion systems[7]. Typical flight times between Earth to Jupiter for these systems ranges from 4-6 years [8]. The optimization was also done for various destinations between Jupiter and Pluto with a maximum flight time of 14 years from Earth to Pluto.

Table 1: This table was taken from ”Preliminary Design of Nuclear Electric Propulsion Missions to the Outer Planets” [8]]. It shows the summary, including flight times, of traveling to several planets.



Encounter Sequence	Launch Date	Launch V_∞, km/s	Flight Times for each leg, days	Total Time of Flight, years	Final Mass, kg
Jupiter					
E-J[a]	Apr 13, 2024	0	2019	5.53	13,301
E-J	Apr 24, 2024	1.23	2019	5.53	13,709
E-M-J[a]	Jan 12, 2022	0	771, 1038	4.95	13,996
E-M-J	Feb 17, 2022	1.04	759, 1050	4.95	14,879
E-E-J	Sep 6, 2015	0.59	460, 1340	4.93	16,260
E-E-J	Sep 29, 2015	0.71	441, 1209	4.52	16,181
E-V-E-J	Aug 22, 2018	2.19	188, 347, 1115	4.52	15,606
Saturn					
E-S[a]	Oct 9, 2022	0	2631	7.20	11,285
E-S	Jan 24, 2023	1.30	2631	7.20	12,653
E-M-S	Feb 16, 2022	1.20	587, 2043	7.20	13,943
E-M-S	Apr 14, 2022	2.17	541, 1759	6.30	12,444
E-V-E-S	Oct 21, 2021	1.95	172, 324, 1804	6.30	14,306
E-V-E-J-S	June 4, 2015	2.24	179, 340, 648, 1283	6.71	12,872
Uranus					
E-U[a]	Oct 20, 2020	0	3537	9.68	8,268
E-U	June 5, 2021	2.32	3537	9.68	10,402
E-E-J-U	Jan 28, 2019	1.02	423, 599, 2378	9.31	12,912
E-E-J-U	Feb 11, 2020	0.94	445, 582, 2374	9.31	13,029
E-M-E-J-U	Dec 28, 2017	1.16	1039, 195, 515, 2450	11.5	14,097
E-V-E-J-U	Aug 25, 2018	2.21	184, 345, 587, 2284	9.31	12,172
E-V-E-J-U	Sep 27, 2018	2.81	172, 327, 526, 1976	8.21	9,225
Neptune					
E-N[a]	Aug 7, 2019	0	4114	11.3	6,347
E-N	Apr 21, 2020	2.63	4114	11.3	8,543
E-V-E-J-N	Aug 26, 2018	2.22	187, 348, 518, 3247	11.8	11,783
E-V-E-J-N	Sep 13, 2018	2.48	179, 338, 501, 2583	9.86	9,264
Pluto					
E-P[a]	May 28, 2015	0	4320	11.8	5,661
E-P	Aug 20, 2015	2.75	4320	11.8	6,890
E-J-P	May 29, 2014	1.11	1675, 3125	13.1	9,109
E-J-P	Feb 10, 2015	2.03	1706, 3214	13.5	8,666
E-E-J-P	Sep 2, 2015	2.44	424, 640, 2879	10.8	8,867
E-M-E-J-P	Mar 19, 2014	2.06	737, 285, 470, 2799	11.7	9,162
E-V-E-J-P	May 13, 2015	2.19	155, 448, 479, 2768	10.5	9,196

Another method is to increase the exhaust velocity which would have an affect on the payload mass fraction[9]. Changing the payload mass fraction, by extension, can optimize the parameters of the EP system. The payload mass fraction method examines the parameter space to determine viable configurations. This was done by examining the space after trying a range of values for the I_{sp}, the power, and launch energy (km/s^2)[10]. Figure 2 shows an example of the data obtained through variations in the payload mass fraction.

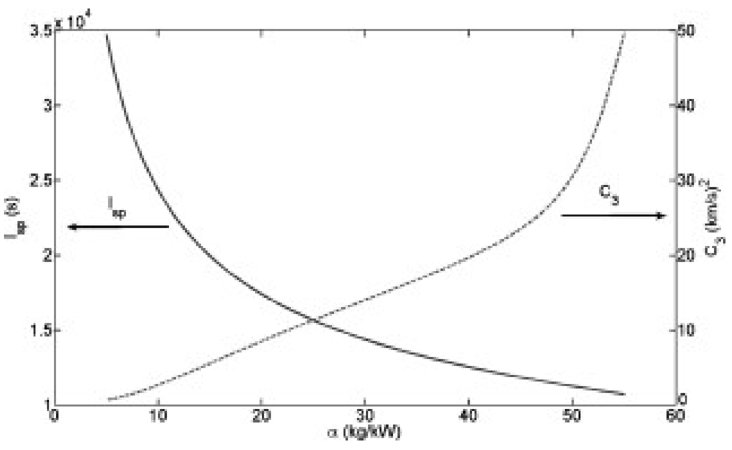


Figure 2: The figure shows variation of I$_{sp}$ and launch energy with the power systems specific mass. The plot is taken from "Maximizing Payload Mass Fractions of Spacecraft for Interplanetary Electric Propulsion Missions [10]."

To begin the optimization process, there was a set of assumptions made to simplify the problem[11]. Because of the inter-dependencies between variables, the list of variable to set as independent variable requires though in order to solve a system of equations. Since the optimization is highly computational and numerical, the set of boundaries and constraints are an important starting point for this process. Therefore, only values that are representative of current EP designs will be considered regarding the random generation of the independent values. The process is iterative and must be done for a different set of values. The set of important dependent variables for which the optimization is done must also be defined. This allows the analysis of any parameter that are considered crucial.

It is important to look at the performance parameters and the trajectory [12]. Trajectory optimization has been done using electric propulsion by setting some reasonable values for the performance parameters in an EP system. The key for this mission and optimization is to reduce the mass of the system.

1.2.1 Monte Carlo Method

The Monte Carlo Method is a statistical computation method used to solve analytical problems. An optimization problem can take advantage of statistical information to select an optimal set of condition for a problem. This method will allow for the statistical optimization of an electric propulsion system.

The way the Monte Carlo method is initiated is as follows:

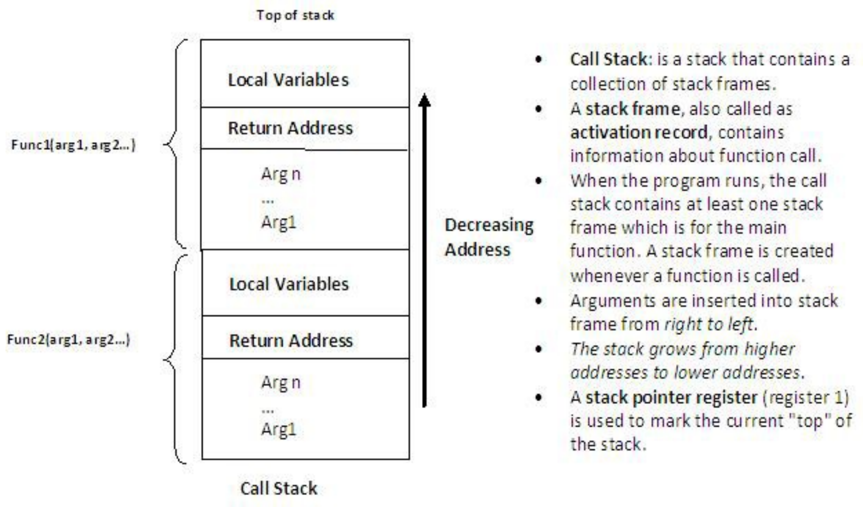
1. Decide on a group of values (inputs) to randomly generate
2. Based on those values, calculate a set of observational values
3. Run N number of trials
4. Use statistical methods to measure the observational data

The simulation has to be run in a set number of trials in order to provide for enough sampling information. The essence of the N number of trials is to create a large enough sample size to work with while also following a random sampling method. The mean and standard deviation of the set of information can be determined and the observed values can be measured and compared to the rest of the data in the set [13].

1.2.2 Computational Limits and the Call Stack

In most programming languages and implementation, the call stack of the program should be a consideration. The call stack occurs every time the program makes a call to a function. The reason for considering this is due to the limitations this may put onto the processing of the code itself.

A call stack can be broken down to its individual stack frames. Every call to a function would contribute a stack frame to the program as a whole. Each stack frame will contain the inputs and the outputs of said function. In C++, there is at least 1 stack frame in the execution of the program. The main function is considered the initial stack frame. The call to the function which the program is currently on, is considered the innermost stack frame. A backtrace of the program can be found. The backtrace provide a summary of the steps or frames the program took to arrive to the final product. This will provide for a flow for the user to follow into the order in which the program called the functions[14]. Figure blank, contains a visual guide for the flow and direction of a calls stack.



- **Call Stack:** is a stack that contains a collection of stack frames.
- A **stack frame**, also called as **activation record**, contains information about function call.
- When the program runs, the call stack contains at least one stack frame which is for the main function. A stack frame is created whenever a function is called.
- Arguments are inserted into stack frame from *right to left*.
- *The stack grows from higher addresses to lower addresses.*
- A **stack pointer register** (register 1) is used to mark the current "top" of the stack.

Figure 3: The figure shows the flow and direction of the call stack in programming languages such as C++.

This develops a direction and a flow for the way the program will execute the steps to get to a final result. The limitation of the program and the number of iterations or executions it may run may be limited by the amount of memory taken by the call stack.

1.3 Project Proposal

The main objective of this project is to reduce the mission time through key performance parameters of the EP system(i.e. the specific impulse, thrust, electrical efficiency, and total efficiency). These values will be randomly generated to create a several EP system configurations. The parameter will be used to determine the efficiency and effectiveness of the system. Once that is done, the optimal trajectory will be determined. The initial simulation and optimization will be done in MATLab. The trajectory optimization will be based on previous trajectory optimization or may be calculated in MATLab as well. The systems Δv requirement will be based on the idea that the system does not travel to LEO using electric propulsion. The EP system will transfer from LEO to Low Mars Orbit and back to LEO.

1.4 Methodology

The initial MATLab simulations for the main parameter study of the EP system will be based on some of the essential EP equations. The rocket equation can be applied to an EP system but the parameter are slightly different and the values of the exhaust velocity are much higher than chemical propulsion system. Ions will be assumed of single charge, rather than a mixture of single and double charged ions [2]. This allows for simpler calculations. The trajectory optimization will be determined using software or running simulations for optimal orbital transfers [15]. The orbital transfers will assume that there are no perturbations. The EP system will operate in a regime that is near vacuum conditions.

The general scheme of the optimization will follow that of the Monte Carlo simulation scheme. The randomization has been doing using a normal distribution scheme in MATLab. With the difficulty of generating a truly random numbers, the random number generator is dependent on the time itself. Different schemes may be used dependent on the ideal randomization. The optimization will be slightly statistical to select the optimal space system. Based on this, a decision on the type of propulsive system can be made that matches the constraint parameters. Further statistical constraints are applied to narrow down the candidate option down to, at most, 5. This allows for an easier selection process.

2 Approach

2.1 Electric Propulsion System

To perform the parametric study, several assumptions are made regarding the performance parameters of the system. In some cases, the assumption is that there is a power supply large enough to power the propulsive system. The mass being delivered in the mission will stay constant throughout the process at 20kg until a better spacecraft mass is determined. The other parameters that are assumed to be given are η_e, I_{sp}, T, Δv, η_T, m_d. These parameters, aside from the delivery mass (m_d) and Δv will be randomized within a reasonable range of values based on preexisting EP systems [2].

Using the randomized variables, some of the other performance parameters will be determined. The first parameter that can be determined is the exhaust velocity, v_{ex}. The exhaust velocity can be determined through the Equation 1 [2].

$$v_{ex} = I_{sp}g \tag{3}$$

Since the thrust is being randomly generated, the value of the mass flow rate can be determined through Equation 2[2].

$$\dot{m}_p = \frac{T}{v_{ex}} \tag{4}$$

The jet power generated by the propulsive force of the system can be determined from the value of the propellant flow rate and the exhaust velocities. Equation 3 shows the relationship[2].

$$P_{jet} = \frac{1}{2}\dot{m}_p v_{ex}^2 \tag{5}$$

For the initial estimates, the Δv requirement will be assumed to be a constant value. This would suggest that the requirement should stay the same since the mission would be staying the same. The requirement can be determined through previous missions to Mars. The secondary analysis can be done through calculations of orbital determination. The Δv requirement can be used to determine the mass of propellant required for the mission. Equation 4 shows this relationship[2].

$$m_p = m_d \left(e^{-\frac{\Delta v}{v_{ex}}} - 1 \right) \tag{6}$$

This will allow us to determine how large the system will be during deployment.

The input power can be determined by using the total efficiency of the system, η_T . Equation 5 shows the relationship[2].

$$P_{in} = \frac{P_{jet}}{\eta_T} \tag{7}$$

The power dissipated, once P$_{in}$ is determined, is found by equation 6[2].

$$P_{dissipated} = P_{in}(1 - \eta e) \tag{8}$$

These set of equation will be used to determine the effect that the change in parameter has on the mission.

2.2 Requirements

Although the set of equations are known, the requirements will determine which design configurations will move on in the selection process and which design configurations fail the initial constraints. A set of tests are initialized to narrow down possible EP system configurations.

The following initial parameter requirements are as follows: 1)The mass of the propellant should be equal to or less than the delivered mass 2) The electrical efficiency should be above 75 percent 3) The total efficiency of the electric propulsion system should be above 50 percent 4) The total efficiency shall not exceed the electrical efficiency. The last requirement allows for a realistic configuration. Total efficiency is dependent on the efficiency of the components of the system.

To further narrow the options, the average of each of the parameters will be determined along with the standard deviation. The attempt will be to choose the systems above or below two standard deviations depending on the intent of the design. That is, two standard deviation will be chosen if the parameter is supposed to be minimized and two standard deviation above will be chosen if the goal is to maximize the parameter. This is done to determine the optimal configuration. Based on the number or resulting configurations, the requirement may change. Neither of the efficiencies should exceed 100 percent and the total efficiency should not exceed the value of the electrical efficiency.

The trajectory still needs to be determined. Whether doing a gravity-assist transfer or a powered transfer would prove more efficient still needs to be determined. The window of opportunity for orbit transfer was done based on NASA's InSight Lander's launch windows. For the first iterations, a previous orbit window will be used along with the optimal Δv requirement to get to Mars[16]. The optimization of the orbit itself was more difficult to approach due to the number of factor and the number of factors and methods considered in finding an initial orbit and in the Monte-Carlo method. The requirement is driven through a Lambert Method approach of calculating the propellant requirements.

Once an initial set of the requirements are used to eliminate electric propulsion configurations, the next criteria used is elimination through the use of the mean of the data. The initial intent was to use the standard deviation as well but there was too much conflict in the parameter to try to maximize parameters using this method. The requirements are as follows:

1. The propellant mass is less than the average propellant mass

2. The electrical efficiency is greater than the average electrical efficiency

3. The total efficiency is greater than the average total efficiency

2.3 Monte Carlo Method

Following the steps of the Monte Carlo Method, the thrust, specific impulse, and the electrical efficiency are the input parameters that are randomly generated. Though more values were included, the observational parameter of interest are the propellant mass and the input power needed for the system. The input power will suggest the feasibility of an EP system considering the amounts of power one of these systems requires and the existing technology to provide for the power requirement. The propellant mass is, ideally, minimum while still performing the mission in the require amount of time. The number of trials designated for this simulation is not set, but will be above 10000 trials. Once the preliminary designs have been discarded using the requirements mentioned earlier, the standard deviations of the data sets will be used to further eliminate any other designs.

The design and all the pertinent information will be stored in data structure within MATLab. The information for all the designs will be stored in that structure and a new data structure will be created will all the designs that have passed the initial and final requirements. The set of configurations are eliminated until the data structure is left with less than 10 EP designs to ease the decision-making of the ideal configuration.

2.4 Orbit Determination and Δv Requirement

To determine the Δv requirement, the orbit transfer and orbit determinations analysis must be performed. To do so, I chose a launch date, that was current with the analysis of this project. The launch from Earth to Mars would begin on June 8, 2018 with a return date of October 8, 2018. This is the proposed launch window for this project. The position of the planets will be taken at these dates in order to determine the velocity vectors using Lambert's method. The velocity vectors will dictate the Δv requirement for this mission as well as the proposed 4 month duration.

Lambert's method states that the orbit of a particle can be determined through the use of two position vectors as well as the time between the two points. The first step is to find the magnitudes of the position vectors and define the trajectory of the flight as either prograde or retrograde to find $\Delta\theta$.

$$r_1 = \sqrt{\mathbf{r}_1 \cdot \mathbf{r}_1} \tag{9}$$

$$r_2 = \sqrt{\mathbf{r}_2 \cdot \mathbf{r}_2} \tag{10}$$

$$\Delta\theta = \arccos\left(\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1 r_2}\right) \tag{11}$$

The spatial ambiguity is determined by the the z component of the cross product of the first position vector crossed with the second position vector. Once the value of $\Delta\theta$ is found, the value for A can be determined.

$$A = \sin(\Delta\theta)\sqrt{\frac{r_1 r_2}{1 - \cos(\Delta\theta)}} \tag{12}$$

The Stumpff functions $C(z)$ and $S(z)$ are used to determine the value of z to be used for orbit determination. The equations used are as follows:

$$y(z) = r_1 + r_2 + A\frac{zS(z) - 1}{\sqrt{C(z)}} \tag{13}$$

$$\sqrt{\mu}\Delta t = \left(\frac{y(z)}{C(z)}\right)^{\frac{3}{2}} S(z) + A\sqrt{y(z)} \tag{14}$$

$$F(z) = \left(\frac{y(z)}{C(z)}\right)^{\frac{3}{2}} S(z) + A\sqrt{y(z)} - \sqrt{\mu}\Delta t \tag{15}$$

The idea at this point is to find the value of z by iterating through given values of z to find the point where $F(z)$ is about 0. Once the value of z is found, it is plugged into equation 27 to get a value for y. This value will be used to find the Lagrange values in the following equations:

$$f = 1 - \frac{y(z)}{r_1} \tag{16}$$

$$g = A\sqrt{\frac{y(z)}{\mu}} \tag{17}$$

$$\dot{f} = \frac{\sqrt{\mu}}{r_1 r_2}\sqrt{\frac{y(z)}{C(z)}}\,(zS(z) - 1) \tag{18}$$

$$\dot{g} = 1 - \frac{y(z)}{r_2} \tag{19}$$

$$\mathbf{v}_1 = \frac{1}{g}(\mathbf{r}_2 - f\mathbf{r}_1) \tag{20}$$

$$\mathbf{v}_2 = \frac{1}{g}(\dot{g}\mathbf{r}_2 - \mathbf{r}_1) \tag{21}$$

With this information, the Δv requirement can be determined by comparing the orbit velocity with the required velocity of the spacecraft[17].

2.4.1 Numerical Solution Method for F(z)

There are several methods to determine the 0 of the function of F(z). In this analysis, the solution to F(z) will be found using the bisection method. The bisection method is typically used for finding roots of a nonlinear function. The bisection method works by searching for a change of sign between two points. If there is no such change in sign, the root of the function in that interval is inconclusive and either a new interval is tested or the search may also be concluded. The purpose of the bisection method is to hone in on two points opposite in sign to determine the possibility of a root in the function. The ideal case is that the function has 1 root and the sign change is found. This would be the case in Figure 3.

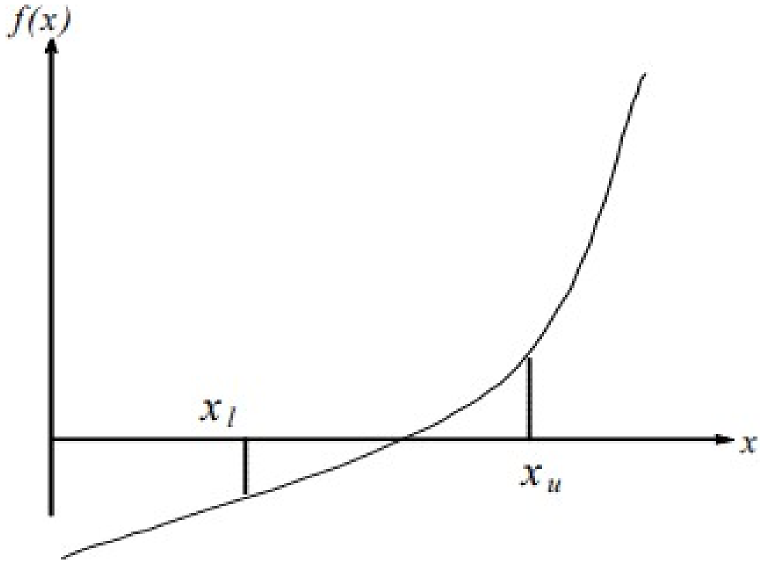


Figure 4: The figure shows the ideal case when running the bisection method to find a root of the function.

In the case of Figure 3, the bisection method would work to find the zero ofthe function. Typically a tolerance can be set in order to get a pretty accurate value of the root while decreasing the number of iterations the program may need to find the root. When the initial interval is found the next interval is shrunk to begin honing in on the root of the function.

There are cases where the bisection method has its downfall when trying to find the root of the function. There are cases when the sign is different and there exists no root. One example is the case of rational functions. The graph of 1/x has a change in signs if you pick a point less than 0 and a point greater than 0, but there is no root to that function.

The next case is when you do have a root but there is no interval in the domain where the values of the function have different signs. If the function is of the form $(x-a))^2$, where a is a constant, then there the bisection method of root finding would fail even though there exists a root to the function[18].

The last case to mention is when there is more than one root to the function. This would cause the bisection method run into the issue of having no change in sign even though there is a root within such interval. This scenario would result into the inspection of a different interval or, possibly, ending the search for the root. The issue is in this case is, also, determining which root to use to satisfy the function if a new interval is examined to determine the roots in this case.

2.4.2 Propagator for Orbit Visualization

To generate visualization for the spacecraft mission, a propagator was developed to plot the points in the trajectory. The Runge-Kutta (i.e. RK4) method was used for the propagator. The method would be implemented to develop a function in MATLab taking in the initial state vector as an input. The initial state vector would be read as follows: $[\vec{x} : \vec{v}]$ where $\vec{x}$ =[x,y,z] and $\vec{v}$ =[v_x, v_y, v_z]. For the purpose of this propagator, 6 coefficients need to be determined.

$$b(1) = v_x \tag{22}$$

$$b(2) = v_y \tag{23}$$

$$b(3) = v_z \tag{24}$$

$$b(4) = -\frac{\mu x}{|\vec{x}|^3} \tag{25}$$

$$b(5) = -\frac{\mu y}{|\vec{x}|^3} \tag{26}$$

$$b(6) = -\frac{\mu z}{|\vec{x}|^3} \tag{27}$$

These values, let's call it $\vec{b}$, are used to determine the value of the position and velocity in the next time step, Δt. The value for $\vec{b}$ is a function of μ and the state vector

X. The calculations for the next values in the position and velocity are calculated as follows:

$$k_1 = b(X_n, \mu)\Delta t \tag{28}$$

$$k_2 = b(X_n + \frac{k_1}{2}, \mu)\Delta t \tag{29}$$

$$k_3 = b(X_n + \frac{k_2}{2}, \mu)\Delta t \tag{30}$$

$$k_4 = b(X_n + k_3, \mu)\Delta t \tag{31}$$

$$X_{n+1} = X_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{32}$$

This method runs under the assumption that there are no perturbations in the system. The orbit is running under ideal conditions. The name of this numerical scheme suggest a 4th order accurate numerical scheme. For the purpose of a visualization and mission trajectory, this numerical scheme is sufficient.

2.5 Design of Experiments

To further test some of the values that were determined in the optimization process, a design of experiments can be done. In this case, based on the variable and parameter that were randomized, three of those value are examined and a response surface will be generated for each scenario. The 3 values in question are the thrust, specific impulse, and electrical efficiency. One of the 3 parameter is varied while the other time are kept constant in order to test the sensitivity of each input parameter. One the data is gather, a response surface is generated to aid in visualizing the sensitivity of the system.

The data, at this point, should be fairly optimized so the response surface plots should not need to have a wide range of values but it serves to help visualize how much more optimal the system can get and the sensitivity of the system after the optimization. This could be used later on to generate more trials and can serve to further optimize the design configurations.

2.6 C++ Implementation of the Code

The C++ code can be generated by mimicking the MATLab script and following the same set of instruction using the C++ syntax. Since MATLab has some of the built-in math function already defined, these function may need to be designed in C++. Such functions include the function to perform dot products and cross products. The arrays and the structure for the EP systems have to be designed to develop the functions to calculate for the required velocity using Lambert's Problem.

The purpose of the implementation of this code is to create a comparison between the run-time of the MATLab script and the C++ code. To provide for a fair comparison, the plots and the figures generated in MATLab will not be generated for the comparison. The initial development of the C++ code will not include the graphics created using MATLab. If time permits, the entirety of the C++ code will be written. Otherwise, the sections of the code can be compared.

3 Results

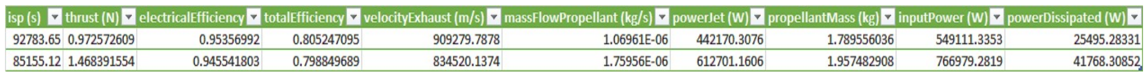
3.1 Parametric Optimization

To arrive to the optimal electric propulsive system, the values are randomly generated through the use of a normal distribution via the Monte Carlo simulation. The randomly generated values are used to determine the remaining parameters of the propulsive system. The requirements mentioned before are implemented in a MAT-

Lab script in order to organize the EP systems. To provide for a fair amount of options, 100000 iterations were run to find propulsion system parameters matching the minimum scope. By doing so, there were approximately 3000 options left to choose from once the simulation was run. From those results, the next criteria needs to be implemented. The Δv requirements and the delivered mass were kept constant for the first step in this process. The following step, the orbit determination, is to determine the most efficient Δv trajectory for the mission. This will still need to be determined.

The current trajectory is based on the launch windows that were provided for NASA's InSight Lander. There may be a better way to optimize the trajectory of the flight. The methods of trajectory optimization still need to be examined to provide for a more efficient spacecraft flight.

From those 100000 iterations, the EP system configurations have been narrowed down to 5 systems. The following table gives a table of the 2 configurations found in MATLab.



isp (s)	thrust (N)	electricalEfficiency	totalEfficiency	velocityExhaust (m/s)	massFlowPropellant (kg/s)	powerJet (W)	propellantMass (kg)	inputPower (W)	powerDissipated (W)
92783.65	0.972572609	0.95356992	0.805247095	909279.7878	1.06961E-06	442170.3076	1.789556036	549111.3353	25495.28331
85155.12	1.468391554	0.945541803	0.798849689	834520.1374	1.75956E-06	612701.1606	1.957482908	766979.2819	41768.30852

Table 2: The table shows the initials set of optimized EP configurations.

The MATLab code is set to have a maximum of 5 EP configurations so the code eliminated values until it had 2 EP configurations left. Based on the table of values the same Monte Carlo process can be followed to further optimize the system. This provides for a baseline of values to use for the sensitivity analysis done in the DOE portion.

3.2 Orbital Mechanics

The launch window for this flight was chosen to start on June 8, 2018 and the mission was to last until November 15, 2018. From these values, the Δv requirement can be determined. The assumption was made that the time it took from June 8th to July 18 to get from Earth to Mars. The spacecraft stayed in Mars orbit from July 18 to August 27. Then it goes from August 27 to November 5 on its return trip to Earth from Mars. The problem was solved using Lambert's Problem. The mapping seen in the following figure was found using the propagator.

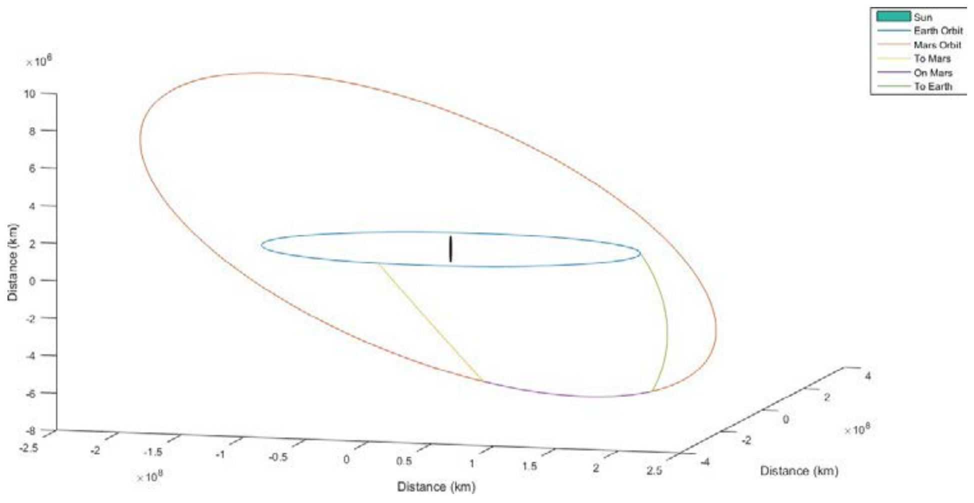


Figure 5: The figure shows a plot of the trajectory of the mission where the sun in considered the origin of the plot.

Lambert's Problem is used when the spacecraft is going from Earth orbit into Mars orbit. This helped provide the parameters needed as input for the propagator to provide the visual of the orbit transfer. The Δv requirement based on this analysis was found to be 34.64 km/s. The goal is to minimize the cost of the mission, but this comes at the cost of increasing the Δv requirement of the mission.

3.2.1 Challenges with Lambert's Problem and Propagator

The initial issue with the propagator occurred with the bisection method when solving for the root of the function F(z). Originally, there were issue with determining a root of the function and MATLab script would throw and error stopping the code at Lambert's Problem. After inspection of the value of the function, there were complex root to consider in this solution set. To adjust for this scenario, only the real portion of each solution was considered. This allowed for the correct trajectory visualization.

The trajectory was compared to the existing Lambert code shared on an open source website. Once the prior code was debugged for this issue, the two trajectories were compared to ensure that the solutions were similar.

3.3 Electric Propulsion Parameters

The requirements are used to narrow down the possible options for an electric propulsion system. The systems that are left can be used to determine an ideal electric propulsion system to launch. The system will determine the mission time needed to achieve the goal of orbiting to Mars and back to Earth. With the script finished for the initial step in the optimization, there are around 3000 viable configurations for the EP system after the initial requirements were applied.

The drawback is the power requirement for a system to fulfill this mission. The initial input power requirement distribution can be seen in Figure 5.

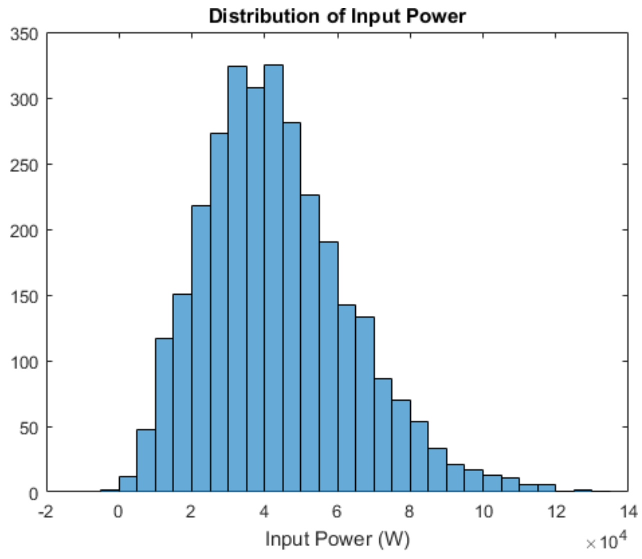


Figure 6: The figure shows the distribution in power requirements for the remaining configuration after the initial constraints.

Based on the mission requirements, the amount of power required for this system is in the range of tens to hundreds of kW for a delivery mass of 20 kilograms. It has not been determined yet whether solar panels would provide enough energy for any of the EP configurations found. For a larger payload mass, the required power would increase. It is typical for larger spacecraft to require power on the order of MW if they are using an EP system. Some of the EP parameters were refined due to the occurrence of negative values. A further process of elimination will be invoked to check for these extraneous solutions.

The initial distribution for the other parameter of interest can be found in figures 7, 8, and 9. The MATLab script is designed to check for negative values in the thrust and in the input power, as this was a previous issue.

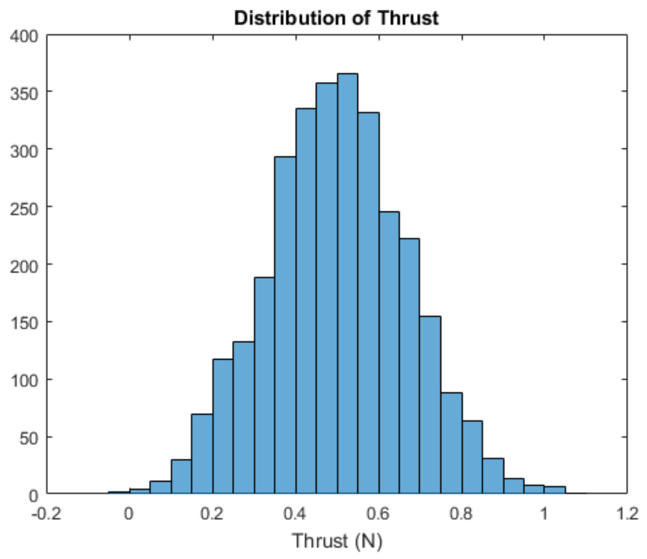


Figure 7: The figure shows the distribution in thrust for the remaining configuration after the initial constraints.

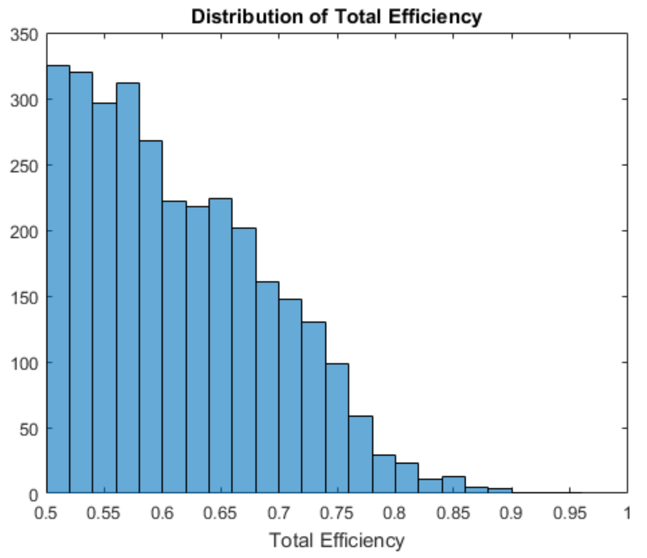


Figure 8: The figure shows the distribution fulfilling the initial total efficiency requirements.

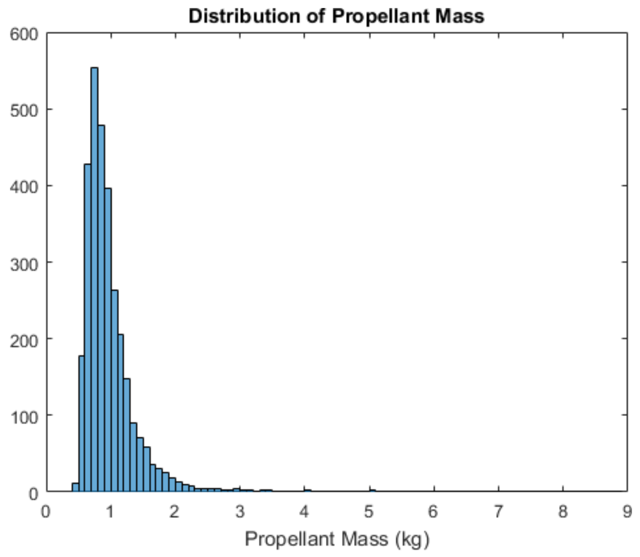


Figure 9: The figure shows the distribution of initial propellant masses fulfilling the requirements.

3.4 Design of Experiments Results

The response surfaces generated do not supply a wide range of values and this would, in part, be a result of optimizing the system before the sensitivity check. In effect, it seems the least variable combination of parameter was the interaction space between the power, thrust and electrical efficiency, but this would require further analysis. The outlier in the data must first be examined and fixed to reach this conclusion.

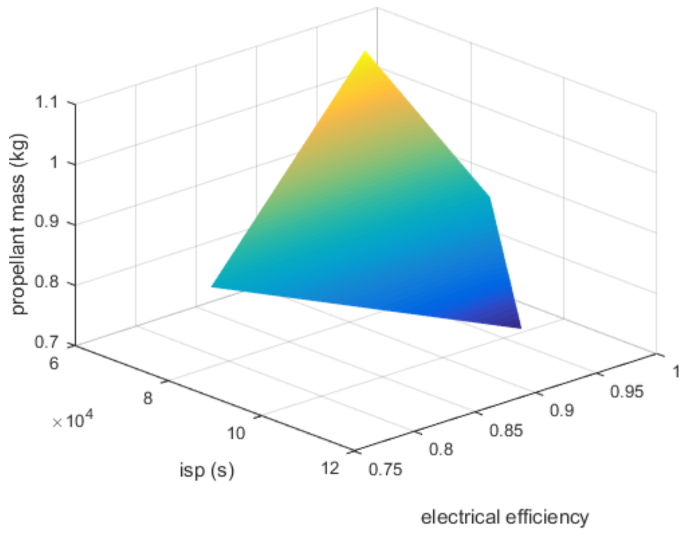


Figure 10: The figure shows the response surface of the interaction between the electrical efficiency and the I$_{sp}$ and their implication on the propellant mass.

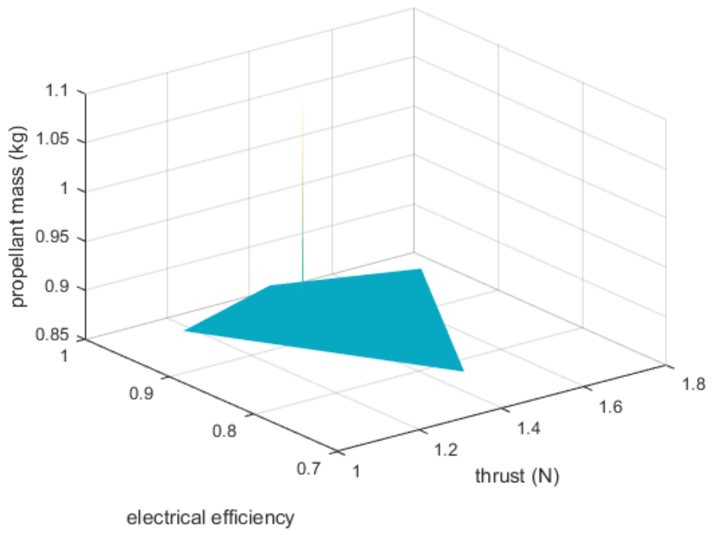


Figure 11: The figure shows the response surface of the interaction between the electrical efficiency and the thrust and their implication on the propellant mass.

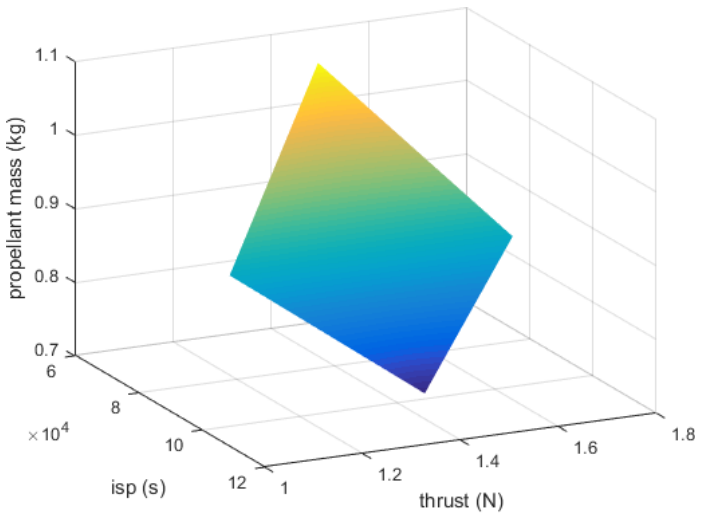


Figure 12: The figure shows the response surface of the interaction between the I$_{sp}$ and the thrust and their implication on the propellant mass.

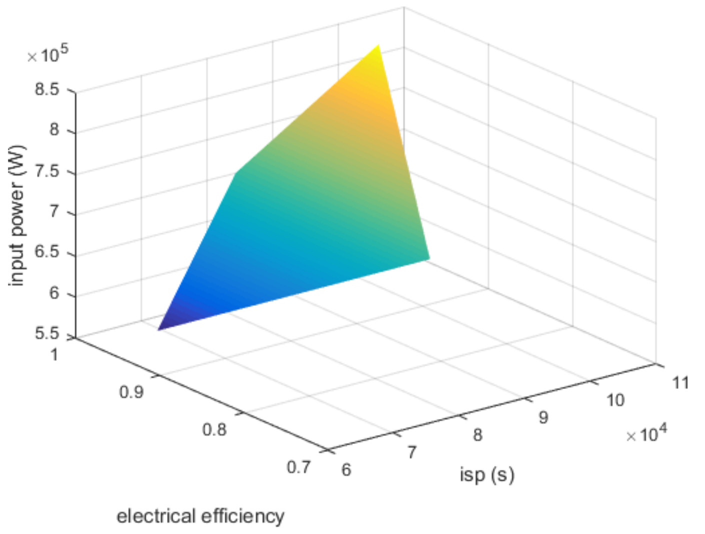


Figure 13: The figure shows the response surface of the interaction between the I$_{sp}$ and the electrical efficiency and their implication on the input power.

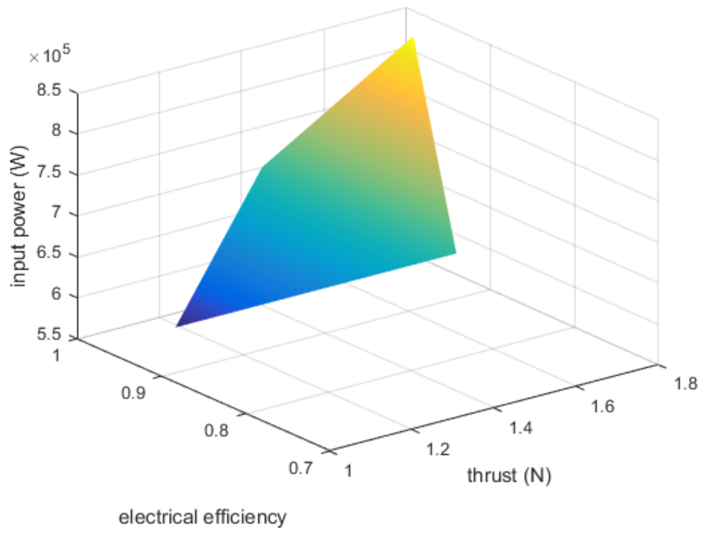


Figure 14: The figure shows the response surface of the interaction between the thrust and the electrical efficiency and their implication on the input power.

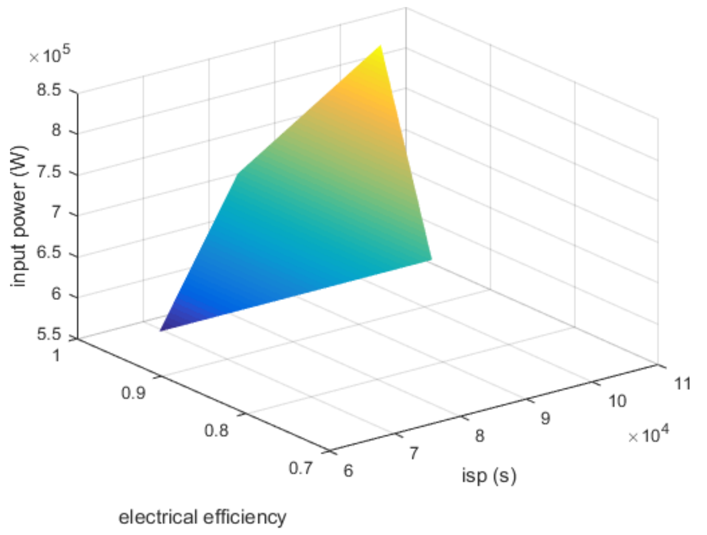


Figure 15: The figure shows the response surface of the interaction between the I_{sp} and the thrust and their implication on the input power.

The response surface found in Figure 10 makes complete sense, the higher the value of the I_{sp} and the electrical efficiency the lower the mass of the propellant required. The value of I_{sp} is known to be indicative of the amount of propellant required regardless of it being a chemical, solid, or electric propulsion system. In this case the darker the blue the more optimal the system.

Figure 11 needs further analysis. There is an outlier in the response surface that should be determined and omitted from the response surface. The response surface with this scheme would suggest that propellant mass will not change drastically with the change in the propulsive thrust.

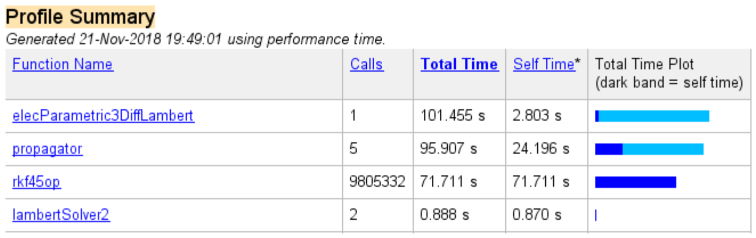
Figure 12 suggest that there is an optimal range or value for the thrust that would result in the decrease of the propellant mass. The value is between 1.5 and 1.6 based on the response surface. This visual would allow us to run another set of simulations based on these values to further optimize the system.

In contrast to Figure 10, Figure 13 shows that the lower the I_{sp} the lower the input requirement. The higher the value of the I_{sp} the more energy that is required for the EP system. This would imply an need for compromise between the input power and the propellant mass because of conflicting influence of the I_{sp} on both of those values.

Figure 14 shows that the lower the thrust of the EP system, the lower the required input power. Because of the direct correlation, the higher the electrical efficiency, the lower the required input power.

Figure 15 shows that the higher the I_{sp}, the higher the input power requirement and the higher the electrical efficiency, the lower the input power requirement. The EP configuration will have to balance the I_{sp} in order to compromise a decrease in both the input power and the propellant mass. Running more possibilities and configurations will increase the run time for the MATLab script but testing the difference in the result can prove useful.

The current run time for the MATLab script is nearly 110 seconds. This is due to the process of the Monte Carlo Method along with the generation of 100000 different configurations. It would be useful to compare the results to determine whether it is efficient to generate this number of configurations. This could also help determine whether or not the number of configurations should be increased. This test of convergence is limited by the amount of processing power needed and the amount of processing power available. The following table shows the current distribution of the run time for the simulations done with 100000 generated configurations.



Profile Summary

Generated 21-Nov-2018 19:49:01 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
elecParametric3DiffLambert	1	101.455 s	2.803 s	
propagator	5	95.907 s	24.196 s	
rkf45op	9805332	71.711 s	71.711 s	
lambertSolver2	2	0.888 s	0.870 s	

Table 3: The figure shows the major components of the run time of the Monte Carlo simulation. The total time is roughly 110 seconds.

Based on this result, it would be interesting to attempt to examine the difference in run-time if the simulation were done in a programming language. This resulted in an attempt to begin writing the coded in C++ but under the time given the C++ code is incomplete. This will be turned into future work, but it would be interesting to begin and observe some of the differences between MATLab and C++ as well as any differences in writing the functions needed to run the simulation.

3.5 Implementation of C++ Code/MATLab Script

The development of the code is still in progress. So far, the functions for the cross product, the dot product, and the magnitude have been developed. The random generator has been tested so that the numbers are randomized based on the run-time of the code execution. It is desired that the code develop a random number between 0 and 1 so the cstdlib library is being used to create a function to do this task. The random generator, as mentioned earlier, takes the time-based random number and divides it by the library defined RAND MAX. Upon testing the random number generation, the issue of the hardware limitations became apparent. This along with the large call stack may have caused the program to compile the code but prevent the code from generating the required output. The program was asked to generate 300000 random numbers and this resulted in the lack of an output. Once the desired array size of the randomly generated numbers was brought down to 200000 random numbers, the program worked fluidly through the execution.

This would lead to the testing of the MATLab script to examine if the program would run under the desired conditions of increasing the number of elements in the array. Upon testing an array of this magnitude, the MATLab script began to take too long or even cause MATLab to stop responding. To work around this there is either a need for better hardware to generate a larger number of random EP configurations or continue to iterate through the same number of configurations more than one time around. Multiple iterations of the same process could cause the solution set to converge towards a certain set of EP parameters. Generating more configurations than 200000 would cause the program to fail to execute properly.

The call stack takes up a place in the computers memory so adding more function calls could result in the inability of a program to complete the task. The number of configurations will be kept under 200000 to conform to the hardware limitations at hand.

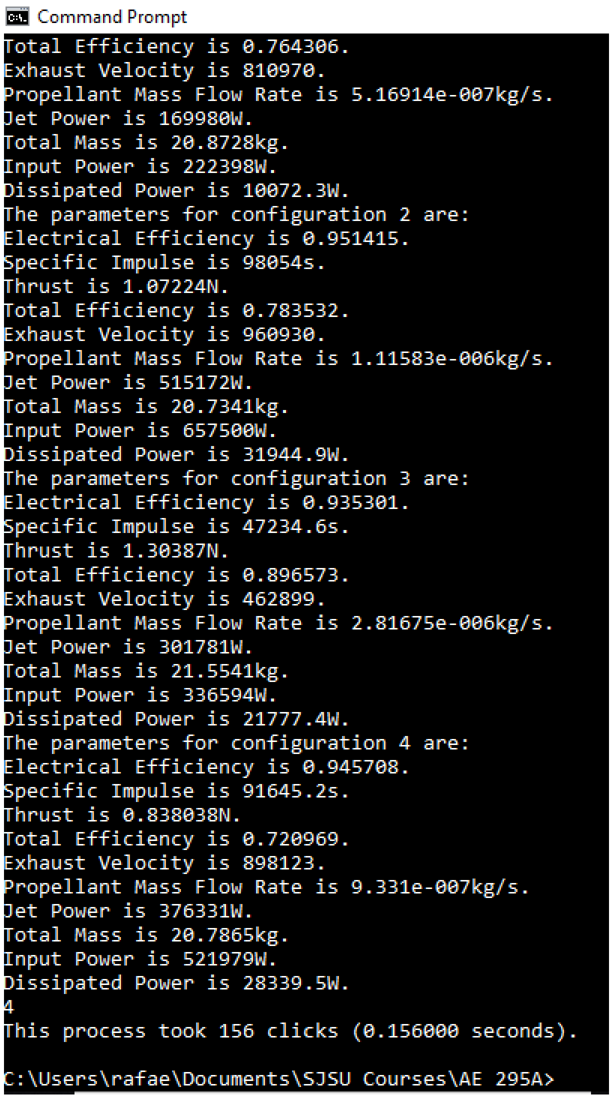
The first challenge in creating the C++ code thus far was creating a function to allow for an array of values as the output. There are many solutions to this problem but both manipulate and use pointers to solve this issue. The following pseudocode shows the approach taken to achieve the solution to this issue.

```
outputType * myFunction(inputType varName1, inputType varName2...){
  static double returnVariable;
  some lines of code here
  ...
  return returnVariable;
}
```

This would force the use of pointer to this problem in order to access the value of the desired array.

The next step in this process would be to work on the Lambert Solver function as well as the necessary function to complete the Lambert Solver. The initial plan was to create a structure for the Lambert Solver, but a similar solution to the one mentioned above can be used. A pointer to the array can be used to extract all the values needed. The array would contain the initial velocity needed, the final velocity needed, and the solution to the set of Stumpff functions. The current state of the C++ code can be found in the Appendix but it has not been completed. There was not enough time to turn the MATLab script into C++ code and that portion of the project is considered to be possible future work. There were several issues when beginning the C++ implementation, such as the understanding of returning the pointer to an array, that may have come up and taken even longer for the implementation of the C++ code. As it stands, the functions work correctly and the main script was simply used to debug or test some of the functions.

To serve as a point of comparison in C++, the Δv requirement was considered constant and the number of configurations developed was set to 10000. The visual plots were also taken out of the MATLab script and this would allow for a fair comparison between the script and the C++ code. Using the clock functions C++ and using the ”Run and Time” debugger in MATLab, the times of the two implementations could be determined. The following figure shows the output given from the C++ script:



```
Command Prompt
Total Efficiency is 0.764306.
Exhaust Velocity is 810970.
Propellant Mass Flow Rate is 5.16914e-007kg/s.
Jet Power is 169980W.
Total Mass is 20.8728kg.
Input Power is 222398W.
Dissipated Power is 10072.3W.
The parameters for configuration 2 are:
Electrical Efficiency is 0.951415.
Specific Impulse is 98054s.
Thrust is 1.07224N.
Total Efficiency is 0.783532.
Exhaust Velocity is 960930.
Propellant Mass Flow Rate is 1.11583e-006kg/s.
Jet Power is 515172W.
Total Mass is 20.7341kg.
Input Power is 657500W.
Dissipated Power is 31944.9W.
The parameters for configuration 3 are:
Electrical Efficiency is 0.935301.
Specific Impulse is 47234.6s.
Thrust is 1.30387N.
Total Efficiency is 0.896573.
Exhaust Velocity is 462899.
Propellant Mass Flow Rate is 2.81675e-006kg/s.
Jet Power is 301781W.
Total Mass is 21.5541kg.
Input Power is 336594W.
Dissipated Power is 21777.4W.
The parameters for configuration 4 are:
Electrical Efficiency is 0.945708.
Specific Impulse is 91645.2s.
Thrust is 0.838038N.
Total Efficiency is 0.720969.
Exhaust Velocity is 898123.
Propellant Mass Flow Rate is 9.331e-007kg/s.
Jet Power is 376331W.
Total Mass is 20.7865kg.
Input Power is 521979W.
Dissipated Power is 28339.5W.
4
This process took 156 clicks (0.156000 seconds).

C:\Users\rafae\Documents\SJSU Courses\AE 295A>
```

Figure 16: The figure shows the output of the C++ code with the time the code took in the process.

As can be seen in the figure, the time it took for the C++ code to run was about half as much as the time it took for the MATLab script to run. MATLab took 0.346 seconds where as the C++ code took 0.156 seconds. This is minimal in this case but this could make a bigger difference if the inclusion of visuals was also done in C++.

4 Conclusion

4.1 Discussion of Results

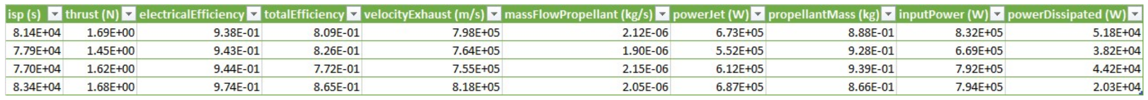
The input power required is in the magnitude of 100's of kW. The value for this would make sense given the designed mission time. There is a need for a greater Δv in this mission. The next step in this process would be to determine a power source capable of providing the amount of power needed for this spacecraft. It would be interesting to examine the number of existing viable EP systems for this mission, assuming a power source is within the feasible spacecraft weight limits.

After an observation of some of the available power supplies that can produce 200kW of power, the size of the spacecraft desired, in this case, would not be feasible. It would need to undergo a revision in order to make the sizing of the system more realistic. The purpose of this study was to examine the amount of power needed to run a mission to Mars under the assumption that the power supply was not much of a concern. This would suggest that the amount of power needed is the largest factor in the weight of the spacecraft. It would be useful to perform future work and run another iteration of simulations under the assumption that the weight of the power supply is known within a certain margin.

An initial set of design parameters has been established and a set number of configurations has been determined based on the use of the initial requirements and the use of statistics to narrow down the results to desirable parameters. The intent is to determine a method to optimize the orbital trajectory, as well, to decrease the ΔV requirement for this mission.

The result show that the propellant mass and the specific impulse are indirectly proportional while the input power and the specific impulse are directly proportional. This would direct the system to a compromise between the two values and the use of statistics can attempt to quantify a system that is better suited for the mission. This information is confirmed in the response surface shown earlier in this report. This relationship can also be seen in the equations used to determine the EP parameters.

The following list shows the final list of optimal configurations. Many of the configurations would revolve around these values because of the inverse relationship between the power required and the specific impulse.



isp (s)	thrust (N)	electricalEfficiency	totalEfficiency	velocityExhaust (m/s)	massFlowPropellant (kg/s)	powerJet (W)	propellantMass (kg)	inputPower (W)	powerDissipated (W)
8.14E+04	1.69E+00	9.38E-01	8.09E-01	7.98E+05	2.12E-06	6.73E+05	8.88E-01	8.32E+05	5.18E+04
7.79E+04	1.45E+00	9.43E-01	8.26E-01	7.64E+05	1.90E-06	5.52E+05	9.28E-01	6.69E+05	3.82E+04
7.70E+04	1.62E+00	9.44E-01	7.72E-01	7.55E+05	2.15E-06	6.12E+05	9.39E-01	7.92E+05	4.42E+04
8.34E+04	1.68E+00	9.74E-01	8.65E-01	8.18E+05	2.05E-06	6.87E+05	8.66E-01	7.94E+05	2.03E+04

Table 4: The table shows a list of the configurations developed where each row represents a configuration.

The table was generated by using the workspace and using the following line of code in MATLab:

```
writetable(struct2table(structVar),’ fileName.xlsx’ );
```

The major point made here is the need for higher electrical and total efficiencies for an electric propulsion system. The values for the thrust, specific impulse and the propellant mass revolve around a certain range of values. This would suggest that in, most cases, there is an optimal range for the design on an EP system based on minimizing the weight of the system as a whole. As a future work, it would be interesting to possibly develop a cost function in order to more accurately optimize the problem.

At this point in time, the number of configurations that can be generated by the code is limited by the hardware in use. If more configurations are desired, then the code can run multiple iterations of the Monte Carlo in order to increase the sample space. Each time, it should only save the optimal configurations and at the end of N configurations, compile each of the optimal configurations into one structure. This would allow for a larger design space. As far as the simulation goes, doing further research into increasing the processing power or improving the hardware at hand would be useful if less iterations are desired. The goal of this project was to examine the effects of attempting to shorten the mission time on the design parameters of an EP system. The end goal after the completion of this project is to attempt to design a software that could generate some possible EP configurations that may be within reason. The current constraint to current EP technology is the size and amount of a powering system needed for some of these systems. The MPDT would be a great example of high power requirements. The typical power system needed for a system such as this is typically large in size. The end goal after this project is to develop some software or GUI that allows the user to input specific mission requirements and the program would select some EP system options.

4.2 Future Remarks

Before continuing to optimize the trajectory, a method should be proposed to continue cycling through the optimization. To get a better set of results, the optimal values can be reused to run another iteration of Monte Carlo simulations. This time, the value can be based off of one of the final designs. The comparison process can continue to find even better options. The randomization can be done within a given percentage of said values. The trouble is going to be the balance between I_{sp} and its effect on the input power and propellant mass.

The calculations are currently done for a specified Δv. The value has not yet been optimized and would be the next step in the process of designing the mission using an EP system. A separate set of calculations will be done to determine the most efficient orbit transfers. The Δv requirement may decrease because it will be assumed that the EP system was transferred to LEO via a chemical rocket. The orbit determination remains a work in progress. This will be further explored to provide for better optimization. The time window would ideally be planned and optimized based on previous trajectory optimization algorithms.

In terms of the coding, it would be ideal to keep track of the run-time for the MATLab script. If time permits, the code will be moved over to C++. This would allow for a comparison in run-time in an attempt to optimize the code as well. Each configuration could be defined as a class of its own in C++ so that it would store all the necessary parameter of the system.

Based on some of the results, there may need to be a more strict criteria for the configurations. There have been cases where the total efficiency has exceeded 100 percent. The total efficiency should also be lower than or equal to the electrical efficiency. This method may decrease the number of steps required to narrow down the option for EP configurations. Reducing the number of required steps may also decrease the run-time of the MATLab script because each step would require less data to store.

The next step would be to determine if there are any existing EP systems to satisfy the optimal conditions. This would force us to examine all types of EP systems. The biggest concern is the power source for the system. If there exists a power source for this system that can provide enough energy, the size of that system would, ideally, have to fall within the weight constraint enforced in the optimization. It would be interesting to design the physical system and try to run simulations. Trying to utilize the physics behind electromagnetism or plasma physics, though, would require more research and time to provide for such a simulation.

The C++ code is currently under development before the trajectory optimization. Ideally, the comparison of the run-time of the code can be performed. The current C++ code needs more work and some of the function still need to be generated. If there are workarounds for the call stack limitation, that would be interesting as it would enable for a higher number of iterations to be executed in the program. This would be done before the trajectory optimization is done.

The MATLab script could also be altered in order to run more than the 100000 configurations being generated. By dumping and filling up the data structure, the simulations can be done for a larger amount of different electric propulsion parameters. This would, then, allow for a better optimization of the EP system.

It would be ideal to use this information to begin designing and running simulations an a system that can match the final parameters. It would be interesting to try to run simulations either through development of the plasma physics in MATLab or through the use of ANSYS. This would allow for a more accurate efficiency analysis and this would allow for the observation of the exhaust flume effects on the system.

5 Appendix

Orbit Determination Script:

```
clc; close all; clear all;

%% Orbit Determination
% Assume a start date or launch from LEO to Mars on June 8, 2018 at
    ↪ 7:30AM UTC
% Mission should last between 120 to 150 days, this will determine
    ↪ the
% delta v  requirements
% 1 au = 149,598,000 kilometers
%% Initial launch from Earth to Mars (all values in  km or km/s)
earthInitPos=[-2.244542935534042E-1,-9.898623255903916E
    ↪ -1,4.560701155303384E-5]*149598000;
earthInitVel=[1.649999873155679E-2,-3.875737397365565E
    ↪ -3,4.005872971272586E-7]*149598000/86400;
marsInsPos=[4.000337515036055E-1,-1.366767680356064,-3.845438967555528
    ↪ E-2]*149598000;
marsInsVel=[1.395777605470139E-2,5.132183704069709E
    ↪ -3,-2.349802301687601E-4]*149598000/86400;
% deltaV1=norm(marsInsVel)-norm(earthInitVel);


%% Return trip from Mars to Earth (all value in km or km/s)
marsDepPos=[1.192813351251525,-6.969502737632471E-1,-4.387304975491351
    ↪ E-2]*149598000;
```

```
marsDepVel=[7.591205446206280E-3,1.327897806399470E
    ↪-2,9.197681741173623E-5]*149598000/86400;
earthRetPos=[9.751904220415768E-1,2.205338163864697E
    ↪ -1,-1.485898354345033E-5]*149598000;
earthRetVel=[-4.072134829623000E-3,1.672285385762414E
    ↪ -2,-1.282508017134359E-6]*149598000/86400;
% deltaV2=norm(marsDepVel)-norm(earthRetVel);


%% Visual Aid for Mission
mu=1.32712440042E11;
mSun=1.989E30;
radiusSun=695508;
deltaT1 = 24*(3600*(JulianDay(2018,7,18,7.5)-JulianDay(2018,6,8,7.5)))
    ↪ ;
deltaT2 = 24*(3600*(JulianDay(2018,9,28,7.5)-JulianDay(2018,7,18,7.5))
    ↪ );
[velFromEarth,velAtMars,f,yOfZ, zVal]=lambertSolver2(earthInitPos,
    ↪ marsInsPos,deltaT1,' prograde' , 1.989E30);
[posEarthToMars,velEarthToMars] = propagator(earthInitPos(1),
    ↪ earthInitPos(2),earthInitPos(3),velFromEarth(1),velFromEarth(2)
    ↪ ,velFromEarth(3),60,deltaT1,mSun);
[posMarsMissionOrbit,velMarsMissionOrbit] = propagator(marsInsPos(1),
    ↪ marsInsPos(2),marsInsPos(3),marsInsVel(1),marsInsVel(2),
    ↪ marsInsVel(3),60,deltaT2,mSun);
[earthOrbit,velEarthOrbit] = propagator(earthInitPos(1),earthInitPos
    '→ (2),earthInitPos(3),earthInitVel(1),earthInitVel(2),
    '→  earthInitVel(3),50,366*24*3600,mSun);
[marsOrbit,velMarsOrbit] = propagator(marsInsPos(1),marsInsPos(2),
    '→ marsInsPos(3),marsInsVel(1),marsInsVel(2),marsInsVel(3)
    '→ ,50,80000000,mSun);
[velFromMars,velAtEarth]=lambertSolver2(marsDepPos,earthRetPos,deltaT1
    '→ ,' prograde' , 1.989E30);
[posMarsToEarth,velMarsToEarth]=propagator(marsDepPos(1),marsDepPos(2)
    '→ ,marsDepPos(3),velFromMars(1),velFromMars(2),velFromMars(3),60,
    '→ deltaT1,mSun);
figure(1)
hold on
[x,y,z]=sphere;
x=radiusSun*x;
y=radiusSun*y;
z=radiusSun*z;
surf(x,y,z,' DisplayName' ,' Sun' )
plot3(earthOrbit(1,:),earthOrbit(2,:),earthOrbit(3,:), ' DisplayName' ,
    '→  ' EarthuOrbit' )
plot3(marsOrbit(1,:),marsOrbit(2,:),marsOrbit(3,:),  ' DisplayName' ,  '
    '→MarsuOrbit' )
plot3(posEarthToMars(1,:),posEarthToMars(2,:),posEarthToMars(3,:),     '
    '→ DisplayName' ,' TouMars' )
plot3(posMarsMissionOrbit(1,:),posMarsMissionOrbit(2,:),
    '→ posMarsMissionOrbit(3,:), ' DisplayName' ,' OnuMars' )
```

```
plot3(posMarsToEarth(1,:),posMarsToEarth(2,:),posMarsToEarth(3,:), '
    '→ DisplayName' , ' TouEarth' )
xlabel(' Distanceu(km)' );ylabel(' Distanceu(km)' );zlabel(' Distanceu(km)'
    '→ );
legend
hold off;

%% Calculating Delta V
deltaV=abs(norm(earthInitVel)-norm(velFromEarth))+abs(norm(marsInsVel)
    '→ -norm(velAtMars))...
    +abs(norm(marsDepVel)-norm(velFromMars))+abs(norm(earthRetVel)-
        '→ norm(velAtEarth));
deltaV=deltaV*1000;

n=100000; % number of iterations
%% Developing the Basic Random Generation of EP Systems
rng(' shuffle' );
electricPropulsionParameters.electricalEfficiency=zeros(n,1);
electricPropulsionParameters.isp=zeros(n,1);
electricPropulsionParameters.thrust=zeros(n,1);
electricPropulsionParameters.totalEfficiency=zeros(n,1);
electricPropulsionParameters.velocityExhaust=zeros(n,1);
electricPropulsionParameters.massFlowPropellant=zeros(n,1);
electricPropulsionParameters.powerJet=zeros(n,1);
electricPropulsionParameters.propellantMass=zeros(n,1);
electricPropulsionParameters.inputPower=zeros(n,1);
electricPropulsionParameters.powerDissipated=zeros(n,1);
electricPropulsionParameters.deltaV=deltaV;
electricPropulsionParameters.massDelivered=20;

for ii=1:length(electricPropulsionParameters.isp)
    g=9.8;
    electricPropulsionParameters.electricalEfficiency(ii)=(randn/3+1)
        ↪ /2;
    electricPropulsionParameters.isp(ii)=100000*((randn/3+1)/2)+11000;
    electricPropulsionParameters.thrust(ii)=(randn/3+1)/2+1; %Newtons
    electricPropulsionParameters.totalEfficiency(ii)=(randn/3+1)/2;
    electricPropulsionParameters.velocityExhaust(ii)=
        ↪ electricPropulsionParameters.isp(ii)*g;
    electricPropulsionParameters.massFlowPropellant(ii)=
        ↪  electricPropulsionParameters.thrust(ii)/
        ↪ electricPropulsionParameters.velocityExhaust(ii);
    electricPropulsionParameters.powerJet(ii)=0.5*
        ↪ electricPropulsionParameters.massFlowPropellant(ii)*
        ↪ electricPropulsionParameters.velocityExhaust(ii)^2;
    electricPropulsionParameters.propellantMass(ii)=
        ↪ electricPropulsionParameters.massDelivered*(exp(
        ↪ electricPropulsionParameters.deltaV/
        ↪ electricPropulsionParameters.velocityExhaust(ii))-1);
    electricPropulsionParameters.inputPower(ii)=
        ↪ electricPropulsionParameters.powerJet(ii)/
        ↪ electricPropulsionParameters.totalEfficiency(ii);
    electricPropulsionParameters.powerDissipated(ii)=
        ↪ electricPropulsionParameters.inputPower(ii)*(1-
        ↪ electricPropulsionParameters.electricalEfficiency(ii));
end
counter=1;
for jj=1:length(electricPropulsionParameters.isp)
    if((electricPropulsionParameters.propellantMass(jj)<=
        ↪ electricPropulsionParameters.massDelivered)&&(
        ↪ electricPropulsionParameters.electricalEfficiency(jj)>=0.75)
        ↪ &&(electricPropulsionParameters.totalEfficiency(jj)>=0.50)
        ↪ &&(electricPropulsionParameters.totalEfficiency(jj)<=
        ↪  electricPropulsionParameters.electricalEfficiency(jj))&&(
        ↪electricPropulsionParameters.propellantMass(jj)>0))
        electricPropulsionParameters2.electricalEfficiency(counter,1)=
            ↪ electricPropulsionParameters.electricalEfficiency(jj);
        electricPropulsionParameters2.isp(counter,1)=
            ↪ electricPropulsionParameters.isp(jj);
        electricPropulsionParameters2.thrust(counter,1)=
            ↪ electricPropulsionParameters.thrust(jj);
        electricPropulsionParameters2.totalEfficiency(counter,1)=
            ↪ electricPropulsionParameters.totalEfficiency(jj);
        electricPropulsionParameters2.velocityExhaust(counter,1)=
            ↪ electricPropulsionParameters.velocityExhaust(jj);
        electricPropulsionParameters2.massFlowPropellant(counter,1)=
            '→ electricPropulsionParameters.massFlowPropellant(jj);
        electricPropulsionParameters2.powerJet(counter,1)=
            '→ electricPropulsionParameters.powerJet(jj);
        electricPropulsionParameters2.propellantMass(counter,1)=
            '→ electricPropulsionParameters.propellantMass(jj);
        electricPropulsionParameters2.inputPower(counter,1)=
            '→ electricPropulsionParameters.inputPower(jj);
        electricPropulsionParameters2.powerDissipated(counter,1)=
            '→ electricPropulsionParameters.powerDissipated(jj);
        counter=counter+1;
    end
end

figure(2)
histogram(electricPropulsionParameters2.inputPower)
xlabel(' InputuPoweru(W)' )
title(' DistributionuofuInputuPower' )
figure(3)
histogram(electricPropulsionParameters2.propellantMass)
xlabel(' PropellantuMassu(kg)' )
title(' DistributionuofuPropellantuMass' )
figure(4)
histogram(electricPropulsionParameters2.totalEfficiency)
xlabel(' TotaluEfficiency' )
```

```
title(' DistributionuofuTotaluEfficiency' )
figure(5)
histogram(electricPropulsionParameters2.thrust)
xlabel(' Thrustu(N)' )
title(' DistributionuofuThrust' )
numberOfConfig=length(electricPropulsionParameters2.isp);

while(numberOfConfig>=10 && numberOfConfig~=0)
    counter=1;
    temp=electricPropulsionParameters2;
    avgPropMass=mean(temp.propellantMass);
    stdPropMass=std(temp.propellantMass);
    avgElectricalEfficiency=mean(temp.electricalEfficiency);
    stdElectricalEfficiency=std(temp.electricalEfficiency);
    avgTotEfficiency=mean(temp.totalEfficiency);
    stdTotEfficiency=std(temp.totalEfficiency);
    electricPropulsionParameters3= struct(' isp' , [], ' thrust' , [], '
        '→ electricalEfficiency' , [], ' totalEfficiency' , [], '
        '→ velocityExhaust' , []...
        , ' massFlowPropellant' , [], ' powerJet' , [], ' propellantMass' ,
            '→ [], ' inputPower' , [], ' powerDissipated' , []);
    for kk=1:length(temp.isp)
        if(temp.electricalEfficiency(kk)>0 && temp.isp(kk)>0 && temp.
        '→ totalEfficiency(kk)>0 && temp.thrust(kk)>0 && temp.
        '→ velocityExhaust(kk)>0 && temp.massFlowPropellant(kk)>0 &&
        ↪ temp.powerJet(kk)>0 && temp.inputPower(kk)>0 && temp.
        ↪ powerDissipated(kk)>0 && temp.propellantMass(kk) < (
        ↪ avgPropMass)&& temp.electricalEfficiency(kk) > (
        ↪ avgElectricalEfficiency) &&...
        temp.totalEfficiency(kk) > (avgTotEfficiency))
        electricPropulsionParameters3.electricalEfficiency(counter,1)=
            ↪ temp.electricalEfficiency(kk);
        electricPropulsionParameters3.isp(counter,1)=temp.isp(kk);
        electricPropulsionParameters3.thrust(counter,1)=temp.thrust(kk)
            ↪ ;
        electricPropulsionParameters3.totalEfficiency(counter,1)=temp.
            ↪ totalEfficiency(kk);
        electricPropulsionParameters3.velocityExhaust(counter,1)=temp.
            ↪ velocityExhaust(kk);
        electricPropulsionParameters3.massFlowPropellant(counter,1)=
            ↪ temp.massFlowPropellant(kk);
        electricPropulsionParameters3.powerJet(counter,1)=temp.powerJet
            ↪ (kk);
        electricPropulsionParameters3.propellantMass(counter,1)=temp.
            ↪ propellantMass(kk);
        electricPropulsionParameters3.inputPower(counter,1)=temp.
            ↪ inputPower(kk);
        electricPropulsionParameters3.powerDissipated(counter,1)=temp.
            ↪ powerDissipated(kk);
        counter=counter+1;
```

```
            numberOfConfig=length(electricPropulsionParameters3.isp);
        end
    end
    electricPropulsionParameters2=electricPropulsionParameters3;
end

%% The one at a time parametric study of the optimal values


% change one of the value that was being randomized by a small
    '→ amount to
% determine the significance of each parameter will be observing the
% propellant mass, the input power and mass flow propellant and will
    '→  be
% varying the thrust and the isp


THRUST_VAR=0.001;
INITIAL_THRUST_PERCENT=0.8;
initialThrust=mean(electricPropulsionParameters2.thrust);
initialIsp=mean(electricPropulsionParameters2.isp);
initTotEff=mean(electricPropulsionParameters2.totalEfficiency);
initEleEff=mean(electricPropulsionParameters2.electricalEfficiency);
newThrust=initialThrust*INITIAL_THRUST_PERCENT;
newIsp=initialIsp*INITIAL_THRUST_PERCENT;
newEleEff=initEleEff*INITIAL_THRUST_PERCENT;
ii=1;
```

```
while(newThrust <= 1.2*initialThrust)
    g=9.8;
    elePropThrusSens.electricalEfficiency(ii)=initEleEff;
    elePropThrusSens.isp(ii)=initialIsp;
    elePropThrusSens.thrust(ii)=newThrust; %Newtons
    elePropThrusSens.totalEfficiency(ii)=initTotEff;
    elePropThrusSens.velocityExhaust(ii)=elePropThrusSens.isp(ii)*g;
    elePropThrusSens.massFlowPropellant(ii)=newThrust/elePropThrusSens
        ↪ .velocityExhaust(ii);
    elePropThrusSens.powerJet(ii)=0.5*elePropThrusSens.
        ↪   massFlowPropellant(ii)*elePropThrusSens.velocityExhaust(ii)
        ↪ ^2;
    elePropThrusSens.propellantMass(ii)=electricPropulsionParameters.
        ↪    massDelivered*(exp(deltaV/elePropThrusSens.velocityExhaust(
        ↪ ii))-1);
    elePropThrusSens.inputPower(ii)=elePropThrusSens.powerJet(ii)/
        ↪ elePropThrusSens.totalEfficiency(ii);
    elePropThrusSens.powerDissipated(ii)=elePropThrusSens.inputPower(
        ↪ ii)*(1-elePropThrusSens.electricalEfficiency(ii));
    newThrust=newThrust+THRUST_VAR*initialThrust;
    ii=ii+1;
end

figure(6)
plot(elePropThrusSens.thrust, elePropThrusSens.propellantMass)
```

```
ii=1;
while(newIsp <= 1.2*initialIsp)
    g=9.8;
    elePropIspSens.electricalEfficiency(ii)=initEleEff;
    elePropIspSens.isp(ii)=newIsp;
    elePropIspSens.thrust(ii)=initialThrust; %Newtons
    elePropIspSens.totalEfficiency(ii)=initTotEff;
    elePropIspSens.velocityExhaust(ii)=elePropIspSens.isp(ii)*g;
    elePropIspSens.massFlowPropellant(ii)=elePropIspSens.thrust(ii)/
        ↪ elePropIspSens.velocityExhaust(ii);
    elePropIspSens.powerJet(ii)=0.5*elePropIspSens.massFlowPropellant(
        ↪ ii)*elePropIspSens.velocityExhaust(ii)^2;
    elePropIspSens.propellantMass(ii)=electricPropulsionParameters.
        ↪ massDelivered*(exp(deltaV/elePropIspSens.velocityExhaust(ii)
        ↪ )-1);
    elePropIspSens.inputPower(ii)=elePropIspSens.powerJet(ii)/
        ↪ elePropIspSens.totalEfficiency(ii);
    elePropIspSens.powerDissipated(ii)=elePropIspSens.inputPower(ii)
        ↪ *(1-elePropIspSens.electricalEfficiency(ii));
    newIsp=newIsp+THRUST_VAR*initialIsp;
    ii=ii+1;
end

figure(7)
```

```
plot(elePropIspSens.isp, elePropIspSens.propellantMass)



ii=1;
while(newEleEff <= 1.2*initEleEff && newEleEff<=1)
    g=9.8;
    elePropEleEffSens.electricalEfficiency(ii)=newEleEff;
    elePropEleEffSens.isp(ii)=initialIsp;
    elePropEleEffSens.thrust(ii)=initialThrust; %Newtons
    elePropEleEffSens.totalEfficiency(ii)=initTotEff;
    elePropEleEffSens.velocityExhaust(ii)=elePropEleEffSens.isp(ii)*g;
    elePropEleEffSens.massFlowPropellant(ii)=elePropEleEffSens.thrust(
        ↪ ii)/elePropEleEffSens.velocityExhaust(ii);
    elePropEleEffSens.powerJet(ii)=0.5*elePropEleEffSens.
        ↪   massFlowPropellant(ii)*elePropEleEffSens.velocityExhaust(ii)
        ↪ ^2;
    elePropEleEffSens.propellantMass(ii)=electricPropulsionParameters.
        ↪ massDelivered*(exp(deltaV/elePropEleEffSens.velocityExhaust(
        ↪ ii))-1);
    elePropEleEffSens.inputPower(ii)=elePropEleEffSens.powerJet(ii)/
        ↪ elePropEleEffSens.totalEfficiency(ii);
    elePropEleEffSens.powerDissipated(ii)=elePropEleEffSens.inputPower
        ↪ (ii)*(1-elePropEleEffSens.electricalEfficiency(ii));
    newEleEff=newEleEff+THRUST_VAR*initEleEff;
    ii=ii+1;
```

```
end


figure(8)
plot(elePropEleEffSens.electricalEfficiency, elePropEleEffSens.
    ↪ propellantMass)
ispOverall=horzcat(elePropIspSens.isp,elePropThrusSens.isp,
    ↪ elePropEleEffSens.isp);
thrustOverall=horzcat(elePropIspSens.thrust,elePropThrusSens.thrust,
    ↪ elePropEleEffSens.thrust);
inputPowerOverall=horzcat(elePropIspSens.inputPower,elePropThrusSens.
    ↪ inputPower,elePropEleEffSens.inputPower);
propMassOverall=horzcat(elePropIspSens.propellantMass,elePropThrusSens
    ↪ .propellantMass,elePropEleEffSens.propellantMass);
eleEffOverall=horzcat(elePropIspSens.electricalEfficiency,
    ↪ elePropThrusSens.electricalEfficiency,elePropEleEffSens.
    ↪ electricalEfficiency);
tri = delaunay(ispOverall,thrustOverall);
figure(9)
trisurf(tri, ispOverall, thrustOverall, propMassOverall);
xlabel('ispu(s)');ylabel('thrustu(N)'); zlabel('propellantumassu(kg)')
    ↪ ;
shading interp;
figure(10)
trisurf(tri, ispOverall,thrustOverall,inputPowerOverall)
xlabel('ispu(s)');ylabel('thrustu(N)'); zlabel('inputupoweru(W)');
```

```
shading interp;
quad=delaunay(ispOverall,eleEffOverall);
figure(11)
trisurf(quad,ispOverall,eleEffOverall,propMassOverall);
xlabel(' ispu(s)' );ylabel(' electricaluefficiency' ); zlabel(' propellantu
    '→ massu(kg)' );
shading interp;
figure(12)
trisurf(quad,ispOverall,eleEffOverall,inputPowerOverall);
xlabel(' ispu(s)' );ylabel(' electricaluefficiency' ); zlabel(' inputupower
    '→ u(W)' );
shading interp;
squad=delaunay(thrustOverall,eleEffOverall);
figure(13)
trisurf(squad,thrustOverall,eleEffOverall,propMassOverall);
xlabel(' thrustu(N)' );ylabel(' electricaluefficiency' ); zlabel('
    '→ propellantumassu(kg)' );
shading interp;
figure(14)
trisurf(squad,thrustOverall,eleEffOverall,inputPowerOverall);
xlabel(' thrustu(N)' );ylabel(' electricaluefficiency' ); zlabel(' inputu
    '→ poweru(W)' );
shading interp;
```

The Lambert Solver Used in this script:

```
function [v1,v2,capFOfZ,yOfZ,z] = lambertSolver2(r1_vec, r2_vec,
    '→ deltaT, trajectory, MassOfCentralBody)
    %% Finding the magnitude of the position vectors
    r1 = norm(r1_vec);
    r2 = norm(r2_vec);

    %% Determining the trajectory of the spacecraft and solving
        '→ delta_theta
    crossOfRadVectors = cross(r1_vec, r2_vec);
    if strcmp(trajectory,' prograde' )
        if crossOfRadVectors(3) >= 0
            deltaTheta = acosd(dot(r1_vec, r2_vec)/(r1*r2));
        else
            deltaTheta =360 - acosd(dot(r1_vec, r2_vec)/(r1*r2));
        end
    elseif strcmp(trajectory, ' retrograde' )
        if crossOfRadVectors(3) >= 0
            deltaTheta =360 - acosd(dot(r1_vec, r2_vec)/(r1*r2));
        else
            deltaTheta =acosd(dot(r1_vec, r2_vec)/(r1*r2));
        end
    end
```

```
%% Finding the value of A
A = sind(deltaTheta)*sqrt(r1*r2/(1-cosd(deltaTheta)));

%% Finding a function of z to iterate to find the z-value
mu = 6.67E-20*MassOfCentralBody;
z = linspace(-100,100,100000);
numberOfIterations =  4;
for  kk  =  1:numberOfIterations
    sOfZ   =   zeros(1,length(z));
    cOfZ   =   zeros(1,length(z));
    for ii=1:length(z)
        if(z(ii)>0)
            sOfZ(ii)=(sqrt(z(ii))-sin(sqrt(z(ii))))/(sqrt(z(ii)))
                ↪ ^3;
            cOfZ(ii)=(1-cos(sqrt(z(ii))))/z(ii);
        elseif(z(ii)<0)
            sOfZ(ii) =  (sinh(sqrt(-z(ii)))-sqrt(-z(ii)))/(sqrt(-z(
                ↪ ii)))^3;
            cOfZ(ii) = (cosh(sqrt(-z(ii)))-1)/(-z(ii));
        else
            sOfZ(ii) = 1/6;
            cOfZ(ii) =  1/2;
        end
    end
    yOfZ = r1 + r2 + A*(z.*sOfZ-1)./(sqrt(cOfZ));
```

```
        capFOfZ = real(((yOfZ./cOfZ).^1.5).*sOfZ + A*sqrt(yOfZ) - sqrt(
            ↪ mu)*deltaT);
        jj = 1;
        while(capFOfZ(jj)*capFOfZ(jj+1) > 0)
            jj =  jj+1;
        end
        z =  linspace(z(jj),z(jj+1),length(z));

    end


%% Solving for Y and Lagrangian  Coefficients
    y = yOfZ(jj);
    f = 1-y/r1;
    g = A*sqrt(y/mu);
    gDot = 1-y/r2;


%% Finding velocity vectors
    v1 = 1/g*(r2_vec  -  f*r1_vec);
    v2 =  1/g*(gDot*r2_vec-r1_vec);
```

Julian Day Function:

```
function  J  =  JulianDay(year,  month,  day,  UT)
    J_0 = 367*year-floor(7*(year + floor((month+9)/12))/4) + floor
        ↪ (275*month/9)...
        +day + 1721013.5;
    J = J_0 + UT/24.0;
```

Propagator:

```
function [pos,vel] = propagator(rx,  ry,  rz,  vx,  vy,  vz,  delT,  totT,  M)
    mu = 6.67E-20*M;
    r = [rx;ry;rz];
    v =  [vx;vy;vz];
    h =  delT;
    N = totT/delT +  1;
    temppos = zeros(3, N+1);
    tempvel = zeros(3, N+1);
    temppos(:,1) = r;
    tempvel(:, 1) = v;
% time = linspace(0, totT, N);
    w = [rx; ry; rz; vx; vy;  vz];

    for ii  = 1:N
        k1 = h*rkf45op(w, mu);
        k2  =  h*rkf45op(w+k1/2,  mu);
        k3  =  h*rkf45op(w+k2/2,  mu);
        k4 = h*rkf45op(w+k3,mu);
        w = w+(1/6)*(k1+2*k2+2*k3+k4);
        temppos(:, ii+1) = w(1:3,1);
        tempvel(:,   ii+1)   = w(4:6,1);
    end
    pos = temppos;
    vel =  tempvel;
end
```

RK4 Scheme:

```
function [ f ] = rkf45op(x, mu  )
    f = zeros(6,1);
    f(1)  =  x(4);
    f(2)  =  x(5);
    f(3)  =  x(6);
    f(4)   =   -mu*x(1)/(norm(x(1:3)))^3;
    f(5)   =   -mu*x(2)/(norm(x(1:3)))^3;
    f(6)   =   -mu*x(3)/(norm(x(1:3)))^3;
end
```

Stumpff Function C:

```
function c = stumpC(z)
if z > 0
    c = (1 - cos(sqrt(z)))/z;
elseif z < 0
    c = (cosh(sqrt(-z)) - 1)/(-z);
else
    c = 1/2;
end
```

Stumpff Function S:

```
function s = stumpS(z)
if z >  0
    s = (sqrt(z) - sin(sqrt(z)))/(sqrt(z))^3;
elseif z < 0
    s = (sinh(sqrt(-z)) - sqrt(-z))/(sqrt(-z))^3;
else
    s = 1/6;
end
```

Current C++ code:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <math.h>
#include  <string>
#include <sstream>
#include <time.h>
using namespace std;


struct electricPropulsion {
  double electricEff;
  double isp;
  double thrust;
  double totalEff;
  double velEx;
  double massFlowProp;
  double  powerJet;
  double propellantMass;
  double inputPower;
  double  powerDiss;
};

double randNorm(){
  double randVal;
  randVal=rand();
  randVal=randVal/RAND_MAX;
  return  randVal;
}

double julianDay(int year, int month, int day,  double  univTime){
  double  julNot  =  367*year-floor(7*(year  +  floor((month+9)/12))/4)  +
      ↪floor(275*month/9)+day + 1721013.5;
  double jul = julNot +  univTime/24.0;
  return  jul;
}

double magnitude(double vector[3]){
  double c=sqrt(pow(vector[0],2) + pow(vector[1],2) + pow(vector[2],2)
      ↪ );
  return c;
```

```
}

double dot(double vectorOne[3],double vectorTwo[3]){
  double c = vectorOne[0]*vectorTwo[0] + vectorOne[1]*vectorTwo[1] +
      ↪ vectorOne[2]*vectorTwo[2];
  return c;
}

double * cross(double vectorOne[3], double vectorTwo[3]){
  static double newVec[3];
  newVec[0]=vectorOne[1]*vectorTwo[2]-vectorOne[2]*vectorTwo[1];
  newVec[1]=vectorOne[2]*vectorTwo[0]-vectorOne[0]*vectorTwo[2];
  newVec[2]=vectorOne[0]*vectorTwo[1]-vectorOne[1]*vectorTwo[0];
  return newVec;
}

// struct lambPar {
// double velocityOne [3];
//double velocityTwo [3];
// double  zValue;
// }
//
// lambPar lambertSolver(double firstRVector [3], double
    ↪ secondRVector [3], double deltaT, string trajectory, double
    ↪ massOfCenBody){
```

```
//
// }
int main()
{
/*This portion of the project will begin to initialize the beginning
    ↪ of the
initial position from the start of the Earth trajectory, then the
    ↪ Mars entry
position, Mars departure position, and Earth arrival  position*/
  // Earth Initial Positon to Mars Entry Position
  double earthInitPos[3]={149598000*-2.244542935534042e
      ↪ -1,149598000*-9.898623255903916e-1,149598000*4.560701155303384
      ↪ e-5};
  double earthInitVel[3]={149598000/86400*1.649999873155679e
      ↪ -2,149598000/86400*-3.875737397365565e
      ↪ -3,149598000/86400*4.005872971272586e-7};
  double marsInsPos[3]={149598000*4.000337515036055e
      ↪ -1,149598000*-1.366767680356064,149598000*-3.845438967555528e
      ↪ -2};
  double marsInsVel[3]={149598000/86400*1.395777605470139e
      ↪ -2,149598000/86400*5.132183704069709e
      ↪ -3,149598000/86400*-2.349802301687601e-4};

  // Mars Departure Position to Earth Orbit Reentry
  double marsDepPos
```

```
    ↪ [3]={149598000*1.192813351251525,149598000*-6.969502737632471E
    ↪ -1,149598000*-4.387304975491351E-2};
double marsDepVel[3]={149598000/86400*7.591205446206280E
    ↪ -3,149598000/86400*1.327897806399470E
    ↪ -2,149598000/86400*9.197681741173623E-5};
double  earthRetPos[3]={149598000*9.751904220415768E
    ↪ -1,149598000*2.205338163864697E-1,149598000*-1.485898354345033
    ↪ E-5};
double earthRetVel[3]={149598000/86400*-4.072134829623000E
    ↪ -3,149598000/86400*1.672285385762414E
    ↪ -2,149598000/86400*-1.282508017134359E-6};

//Constants
double mu=1.32712440042e+11;
double mSun=1.989e+30;
double  radiusSun=695508.0;
double deltaT1 =  24*(3600*(julianDay(2018,7,18,7.5)-julianDay
    ↪ (2018,6,8,7.5)));
double deltaT2 =  24*(3600*(julianDay(2018,9,28,7.5)-julianDay
    ↪ (2018,7,18,7.5)));

clock_t t;
t = clock();
double deltaV = 3.464e+4;
double massDelivered = 20;
```

```
srand((unsigned)time(NULL));
double g = 9.8;
int numIter=10000;
electricPropulsion initialRandom[numIter];
double foo [numIter];
int counter = 0;
int indexTrack [numIter];
for(int i=0; i<numIter;i++)
{
  initialRandom[i].electricEff = randNorm();
  initialRandom[i].isp = 100000*randNorm()+11000;
  initialRandom[i].thrust = 2*randNorm();
  initialRandom[i].totalEff = randNorm();
  initialRandom[i].velEx = initialRandom[i].isp*g;
  initialRandom[i].massFlowProp =  initialRandom[i].thrust/
      ↪ initialRandom[i].velEx;
  initialRandom[i].powerJet = 0.5*initialRandom[i].massFlowProp*pow(
      ↪  initialRandom[i].velEx,2);
  initialRandom[i].propellantMass = massDelivered*(exp(deltaV/
      ↪initialRandom[i].velEx)-1);
  initialRandom[i].inputPower =  initialRandom[i].powerJet/
      ↪ initialRandom[i].totalEff;
  initialRandom[i].powerDiss = initialRandom[i].inputPower*(1-
      ↪ initialRandom[i].electricEff);
  if((initialRandom[i].propellantMass<=massDelivered)&&(
      ↪ initialRandom[i].electricEff>=0.75)&&(initialRandom[i].
      ↪ totalEff>=0.50)&&(initialRandom[i].totalEff<=initialRandom[i
      ↪ ].electricEff)&&(initialRandom[i].propellantMass>0)){
    indexTrack[counter] = i;
    counter++;
  }
}
int indexTrackSec [counter];
electricPropulsion newEleSys[counter];
for(int i = 0; i < counter; i++){
  newEleSys[i]=initialRandom[indexTrack[i]];
}
electricPropulsion finalSystem[10];
while(counter>10){
  electricPropulsion tempSys[counter];
  int indexTrackTemp[counter];
  int tempCount = 0;
  double sumElEf = 0.0;
  double sumIsp = 0.0;
  double sumThrust =  0.0;
  double sumTotEf = 0.0;
  double sumVelEx =0.0;
  double sumMassFlRt = 0.0;
  double sumPowJet = 0.0;
  double sumPropMass = 0.0;
```

```
double sumInPow = 0.0;
double sumPowDiss = 0.0;
for(int i = 0; i < counter; i++){
   newEleSys[i]=initialRandom[indexTrack[i]];
   sumElEf += newEleSys[i].electricEff;
   sumIsp += newEleSys[i].isp;
   sumThrust   +=    newEleSys[i].thrust;
   sumTotEf  +=   newEleSys[i].totalEff;
   sumVelEx += newEleSys[i].velEx;
   sumMassFlRt += newEleSys[i].massFlowProp;
   sumPowJet +=  newEleSys[i].powerJet;
   sumPropMass += newEleSys[i].propellantMass;
   sumInPow += newEleSys[i].inputPower;
   sumPowDiss += newEleSys[i].powerDiss;
}
double avgElEf =  sumElEf/counter;
double  avgIsp   =   sumIsp/counter;
double avgThrust = sumThrust/counter;
double avgTotEf = sumTotEf/counter;
double avgVelEx =  sumVelEx/counter;
double avgMassFlRt = sumMassFlRt/counter;
double avgPowJet = sumPowJet/counter;
double avgPropMass = sumPropMass/counter;
double avgInPow  =  sumInPow/counter;
double avgPowDiss = sumPowDiss/counter;
```

```
  for(int i=0; i<counter; i++){
      if(newEleSys[i].propellantMass < (avgPropMass)&& newEleSys[i].
        ↪ electricEff > (avgElEf) &&newEleSys[i].totalEff > (
        ↪ avgTotEf)){
      newEleSys[tempCount] = newEleSys[i];
      tempCount++;
    }
    else{
      newEleSys[i]={};
    }
  }
  counter  =  tempCount;
}
for(int i = 0; i<counter; i++){
  finalSystem[i] = newEleSys[i];
  cout<<"Theuparametersuforuconfigurationu" << i+1 << "uare:" <<
      ↪ endl;
  cout<<"ElectricaluEfficiencyuisu" << finalSystem[i].electricEff  <<
      ↪  "." << endl;
  cout<<"SpecificuImpulseuisu" << finalSystem[i].isp << "s." << endl
      ↪ ;
  cout<<"Thrustuisu" << finalSystem[i].thrust << "N." << endl;
  cout<<"TotaluEfficiencyuisu" << finalSystem[i].totalEff << "."  <<
      ↪ endl;
  cout<<"ExhaustuVelocityuisu" << finalSystem[i].velEx << "." <<
```

```
        '→ endl;
    cout<<"PropellantuMassuFlowuRateuisu" << finalSystem[i].
        '→ massFlowProp << "kg/s." << endl;
    cout<<"JetuPoweruisu" << finalSystem[i].powerJet << "W." << endl;
    cout<<"TotaluMassuisu" << finalSystem[i].propellantMass + 20 << "
        '→ kg." << endl;
    cout<<"InputuPoweruisu" << finalSystem[i].inputPower << "W." <<
        '→ endl;
    cout<<"DissipateduPoweruisu" << finalSystem[i].powerDiss << "W."
        '→ << endl;
  }
  cout<<counter<<endl;
  t=clock() - t;
  t=double(t);
  printf ("Thisuprocessutooku%duclicksu(%fuseconds).\n",t,((float)t)/
      '→ CLOCKS_PER_SEC);
  // for(int i=0; i<numIter;i++){
  // sumIsp += initialRandom[i].isp;
  // }
  // double ispAvg = sumIsp/numIter;
  // cout<<ispAvg<<endl;
  return 0;
}
// To run the code in cmd prompt follow these steps
// 1) Type in the following command g++ helloworld.cpp -o helloworld
```

```
    '→ .exe
// 2) Then type in the .exe file you have Created
```

References

[1] Choueiri, E.Y., "A Critical History of Electric Propulsion: The First 50 Years (1906-1956)," *Journal of Propulsion and Power*, Vol. 20, No.2, 2004, pp. 193-203.

[2] Goebel, D.M., & Katz, I (2008). *Fundamentals of Electric Propulsion: Ion and Hall Thrusters*.California: Jet Propulsion Laboratory, California Institute of Technology.

[3] Mikellides, G.P, & Turchi, P.J., & Mikellides, "I.G. Design of a Fusion Propulsion System-Part 1: Gigawatt-Level Magnetoplasmadynamic Source." *Journal of Propulsion and Power* Vol 18, Issue No. 1, January-February 2002.

[4] Jordan, I. J. (2000). *Spacecraft? Electric Propulsion: Which One for my Spacecraft?* Baltimore: JHU, Whiting School of Engineering.

[5] Quarta, A.A., and Mengali, G., "Minimum-time space missions with solar electric propulsion," *Aerospace Science and Technology*, Vol. 15, No. 5, 2011, pp. 381-392.

[6] Kluever, C.A., "Efficient Computation of Optimal Interplanetary Trajectories Using Solar Electric Propulsion," *Journal of Guidance, Control, and Dynamics*, Vol. 38, No. 5, 2015, pp. 821-30.

[7] Genta, G., and Maffione, P.F., "Optimal low-thrust trajectories for nuclear and solar electric propulsion," *Acta Astronautica*, Vol. 118, 2016, pp. 251-261.

[8] Yam, C.H., McConaghy, T.T., Chen, K.J., "Preliminary design of nuclear electric propulsion missions to the outer planets," *Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference*, August 16, 2004 - August 19, Vol. 3, American Institute of Aeronautics and Astronautics Inc, Providence, RI, United states, 2004, pp. 1542-1561.

[9] Sutton, G. P., & Biblarz, O. (2001). *Rocket Propulsion Elements. In Rocket Propulsion Elements*(pp. 660-709). New York: John Wiley & Sons, Inc.

[10] Patel, P., Scheeres, D., and Gallimore, A., "Maximizing payload mass fractions of spacecraft for interplanetary electric propulsion missions," *Journal of Spacecraft and Rockets*, Vol. 43, No. 4, 2006, pp. 822-827.

[11] Li, M., Liu, H., Ning, Z., "Design optimization of a magnetoplasmadynamic thruster by numerical methods," *High Temperature Material Processes*, Vol. 18, No. 1-2, 2014, pp. 83-90.

[12] Petukhov, V.G., and Wook, W.S., "Joint Optimization of the Trajectory and the Main Parameters of an Electric Propulsion System," *6th Russian-German Conference on Electric Propulsion and Their Application*, RGCEP 2016, August 28, 2016 - September 2, Vol. 185, Elsevier Ltd, Samara, Russia, 2017, pp. 312-318.

[13] Walter, J.C., Barkema, G.T., "An introduction to Monte Carlo methods", *Physica A: Statistical Mechanics and its Applications*, Volume 418, 2015, Pages 78-87.

[14] "Examining the Stack, *Internet Article TIME International* Available: http: //kirste.userpage.fu-berlin.de/chemnet/use/info/gdb/gdb_7.html.

[15] Sims, J.A., Finlayson, P.A., Rinderle, E.A., "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," *AIAA/AAS Astrodynamics Specialist Conference*, 2006, August 21, 2006 - August 24, Vol. 3, American Institute of Aeronautics and Astronautics Inc, Keystone, CO, United states, 2006, pp. 1872-1881.

[16] Okutsu, M., Landau, D.F., Rogers, B.A., “Low-thrust roundtrip trajectories to Mars with one-synodic-period repeat time,” *Acta Astronautica*, Vol. 110, 2015, pp. 191-205.

[17] Curtis, H., *Orbital Mechanics: For Engineering Students*, Butterworth-Heinemann, 2015.

[18] Burden, R. L., and Faires, J. D., *Numerical analysis*, Boston: Prindle, Weber & Schmidt, 1986.